pdp11

RT-11

System Reference Manual

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

digital

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 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.

CONTENTS

SYSMAC

TABLES

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 Release 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	O	(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*** (39H150)
> GT44 User's Guide*** (39H250)
> 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-11-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 RP11C/RP02 and RP11E/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 O, 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 DEVELOPMENT

Computer systems such as RT-11 are often used extensively for program development. The programmer makes 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 include 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.

If 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 is 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 I) 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 O) is a collection of FORTRAN callable routines that make the programmed requests and various utility functions available to the FORTRAN programmer. SYSLIB also provides a complete string manipulation package and two-word integer package for RT-11 FORTRAN.

BASIC and FORTRAN 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 1-1. 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 devices 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 interfaced 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	RS11		
	RC11			RS64
	RH11	RJS03	RJS04	
Removable Pack	RP11	RP02	RP03	
Diskette	RX11	RX01	RX01 (second controller)	
DECTAPE	TC11	TU56		
MAGTAPE	TM11/TMA11 RH11	TU10,TS03 TJU16		
CASSETTE	TA11	TU60		
HIGH-SPEED PAPER TAPE READER/PUNCH	PC11 PR11	PC11 (both) PR11 (reader only)		
LINE PRINTER	LS11 LV11 LP11	LS11, LA180 LV11 (printer only) all LP11 controlled printers		
CARD READER	CR11 CM11		CR11 CM11	
TERMINAL	DL11	LT33, LT35 LA30P, LA36, 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. However, when the foreground job reaches a state in which no more processing 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.) Thus, the user can run FORTRAN, 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 include:

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 commands. 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 PROCEDURE

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.

System Communication

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 ADRS 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	Disk (RK11,RK05)	(RK Disk other than Unit 0)	(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	5	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
5		5007				6203
105710						103402
100376						112711
5007						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 ADRS 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 WRITE 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:RKMNFB.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 <u>RT-11 System Generation Manual</u> 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.

Files in ASCII format conform to the American National Standard Code for Information Interchange, in which each character is represented by a 7-bit code. Files in ASCII format include program source files created by the Editor, 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.

Files 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 foreground. 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 image (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 Characters

The following table summarizes the characters typed by RT-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 2-1
Prompting Characters

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 prompts 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 is to print output, F> is typed first. B> and F> are also printed as a result of the CTRL 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 format of a file. It is a good practice to conform to the standard filename extensions for RT-11. If an extension is not specified for an input or output file, most system programs assign appropriate default extensions. Table 2-3 lists the standard extensions used in RT-11.

Table 2-3
File Name Extensions

Extension	Meaning
.BAD	Files 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	Directory listing file
.DMP	DUMP output file.
.FOR	FORTRAN IV source file (FORTRAN input).
.LDA	Absolute binary file (optional Linker output).
.LLD	Library listing file.
.LOG	BATCH 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, FORTRAN 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 given, 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 COMPONENTS

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 2-1 helpful while reading the following descriptions.

2.3.1 Resident Monitor (RMON)

The Resident Monitor is the only permanently 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 Keyboard 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 console terminal page indicates that the Keyboard Monitor is in memory and is 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.)

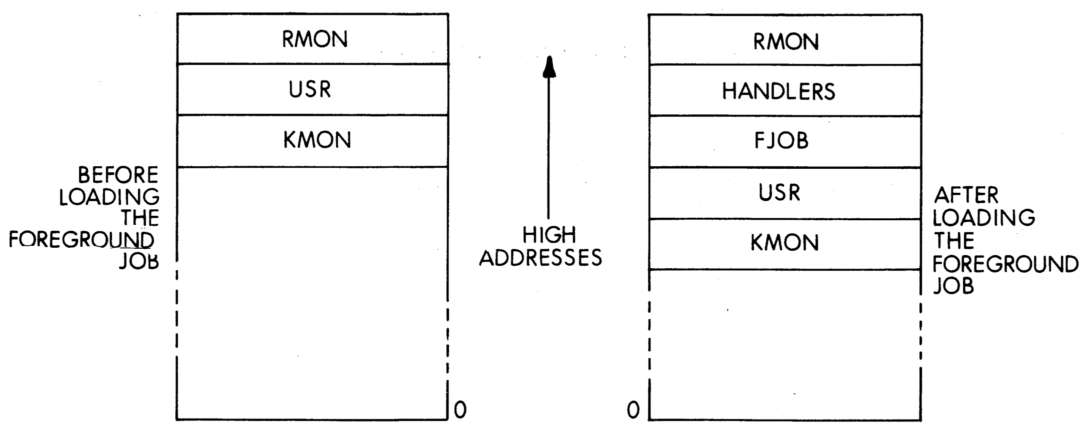


Figure 2-1
RT-11 System Memory Maps

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 commands:

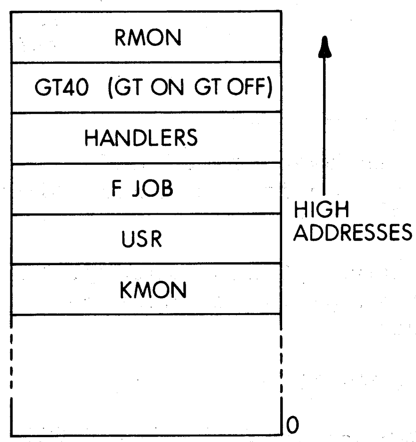


Figure 2-2
RT-11 Memory Map (GT40)

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 following 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

where:

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 number of blocks (n) desired 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 number or one to three alphanumeric characters (the first of which must be alphabetic) which will be converted to radix-50 (see Section 5.5.4 of the 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,LP:=TASK.1,RK1:TASK.2,TASK.3
```

it is interpreted as though all devices had been indicated as follows:

```
*DT1:FIRST.OBJ,LP:=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 Keys

Key	Function
CTRL A	Valid when the monitor GT ON command has been typed and the display is in use. The command does not echo on the terminal. It is used after a CTRL S has been typed to effectively page output. Console output is permitted to resume until the screen is completely filled; text previously displayed is scrolled upward off the screen. CTRL 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).
CTRL B	Under 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). CTRL B has no special meaning when used under a Single-Job Monitor or when a SET TTY NOFB command has been issued (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 necessary. Note that under the F/B Monitor, the job which is currently receiving input will be the job that is stopped (determined by whether a CTRL F or CTRL B was most recently 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 display 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 exists, 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 terminates. 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 suppression of teleprinter output while continuing program execution. Teleprinter output is re-enabled when one of the following occurs: 1. A second CTRL O is typed, 2. A return to the monitor occurs, or 3. The running program issues a .RCTRLO directive (see Chapter 9). (RT-11 system programs reset CTRL O 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 command 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 terminals 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 return 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 from 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 is printed by another. The user may be in the process of typing input to one job when the other job is 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 redirected to this job, however, unless a CTRL B or CTRL F is explicitly typed. If 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 is 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 example:

```
.R PIP
*DT1:TAPE=PR:
DT1:/L
*13-FEB-74
TAPE          78 13-FEB-74
 486 FREE 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 is 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. If a job is terminated by typing two CTRL C's, any unprocessed type-ahead is discarded.

NOTE

> If 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 lines are terminated by a carriage return.

All commands, 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 on 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}

where:

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

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

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

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

12" screen (GT40, DEClab)	/L:37	(31 decimal)
	/T:1350	(744 decimal)
17" screen (GT44)	/L:50	(40 decimal)
	/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? message 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 is:

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.

Examples:

.DATE 21-FEB-74	Enter the date 21-FEB-74 as the current system date.
.DAT 21-FEB-74	Print the current date.

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

TIME

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 form 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). If any of the arguments are omitted, 0 is assumed. If no argument is given, the current time of day is output.

Examples:

.TIM 8:15:23	Sets the time of day to 8 hours, 15 minutes and 23 seconds.
.TIM 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 is 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 O, 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 is:

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 maximum) 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 is:

$$\text{ASS}\left\{\text{IGN}\right\} \left\{\left\{\text{dev}\right\}\text{:udev}\right\}$$

where:

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 logical device names.

: — is a delimiter character (can be a colon, equal sign, and, if separating 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:

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

Assignment of logical names to logical names is not allowed.

If 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 files in the background job to become permanent files. If 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 is:

CLO{SE}

The CLOSE command makes temporary directory entries permanent.

Example:

`R EDIT`
`*EWTEXT$$`
`*IABCD$$`
`*^C`
	The Editor has a temporary file open (TEXT), which is preserved by .CLOSE.

`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,...}

where:

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/Background system. 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 the 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 foreground 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 command:

```
.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 is:

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

where:

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 RK. (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:

.UNLOAD LP,PP	The lineprinter and paper tape punch handlers are released 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 example:

```
.UNL FG
```

FG can be mixed with other device names.

However, if, for example, DT2 has been assigned the name FG and loaded into memory as follows:

```
.ASSIGN DT2:FG

.LOAD FG
```

the command:

```
.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 dev:{NO}option=value {,{NO}option=value,...}

where:

dev:	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 number required in some cases.

A space may be used in place of or in addition to the colon, equal sign, or comma. 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 may 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.)

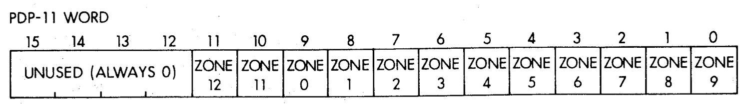
(continued on next page)

Table 2-5 (Cont.)
SET Command Options

Device	Option	Alteration
LP	NOCTRL	Ignores nonprinting 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. 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 is not ready at the start of a transfer. Note that the handler will wait regardless of how the option is set if the reader becomes "not ready" during a transfer (i.e., the input hopper is empty, but an end-of-file card has not yet been read).

 (continued on next page)

Table 2-5 (Cont.)
SET Command Options

<table>
<tr><th>Device</th><th>Option</th><th>Alteration</th></tr>
<tr><td>CR</td><td>IMAGE</td><td>Causes each card column to be stored as a 12-bit binary number, one column per word. The CODE option has no effect in IMAGE mode. The format of the 12-bit binary number is:


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.</td></tr>
<tr><td>CR</td><td>NOIMAGE</td><td>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.</td></tr>
<tr><td>CR</td><td>TRIM</td><td>Causes trailing blanks to be removed from each card read. It is not recommended that TRIM and NOCRLF be used together since card boundaries will be difficult to find. This is the normal mode.</td></tr>
<tr><td>CR</td><td>NOTRIM</td><td>Transfers a full 80 characters per card.</td></tr>
<tr><td>CT</td><td>RAW</td><td>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.</td></tr>
<tr><td>CT</td><td>NORAW</td><td>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.</td></tr>
<tr><td colspan="3">The following 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 <u>RT-11 Software Support Manual</u>. (Note that the device specification is TTY, not TT, because the handler itself is not changed.)</td></tr>
<tr><td>TTY</td><td>COPY</td><td>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 <u>VT50 Video Terminal Programmer's Manual</u> for more information.</td></tr>
<tr><td>TTY</td><td>NOCOPY</td><td>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 is the normal mode.</td></tr>
</table>

(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 CTRL F to have no special meaning. NOTE 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 VT50, 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 is 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 CTRL 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 is capable of executing hardware tabs.
TTY	NOTAB	Causes the monitor to simulate tab stops every eight positions. The normal mode is NOTAB. VT05/VT50 terminals 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 is 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 set: 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 form of the GET command is:

GE{T} dev:filnam.ext

where:

dev:	represents any legal RT-11 device name. If a device is not specified, DK: is 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 is not specified, the extension .SAV is assumed.

The GET command is typically used to load a program into memory for modification 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. Thus:

.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. If identical locations are required by any of the programs, the later 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 address of the location to be referenced. This command 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 is:

B {location}

where:

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 is odd, it is rounded down by one to an even address.

The base is cleared whenever user program execution is 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 is:

E location m{-location n}

where:

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 is 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 specified, the contents of location 0 are printed. Examination of locations outside the background area is 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}

where:

location — represents an octal address which is added to the relocation base value to get the actual address 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.

If multiple values are specified (valuel,...,valuen), they are deposited beginning at the location specified. The DEPOSIT command accepts word or byte addresses but executes the command as though a word address was specified. An odd address is 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 zeroed.

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	Puts 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}

where:

dev: — represents one of the standard RT-11 block-replaceable device names. If no device is specified, DK is assumed.

file.ext represents the name to be assigned to the file being saved. If the file name is omitted, an error message is output. If no extension is specified, the extension .SAV is 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). If 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. If the user wants to change these or any of the following addresses, he must first use the DEPOSIT command to alter them and then SAVE the correct areas:

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, is 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 initialized to some part of memory where these 10 words will not modify it.

The form of the command is:

RU{N} dev:filnam.ext

where:

dev:	is any standard device name specifying a block-replaceable device. If dev: is not specified, the device is 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	is the file to be executed. If an extension is not specified, the extension .SAV is 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 .RUN PROG2	Loads PROG1.SAV from device DK without executing it. Then combines PROG1 and 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) is similar to the RUN command except that the file specified must be on the system device (SY:).

The form of the command is:

```
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 form of the command is:

$$\mathrm{ST}\left\{\mathrm{ART}\right\}\ \left\{\mathrm{address}\right\}$$

where:

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.

If 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 start address from location 40 is used.

Examples:

```
.GET FILE.1
.START 1000
```

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

```
.GET FILEA

.GET FILEB

.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 starts the program at its reentry address (the start address minus two). REENTER 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 stopped with a CTRL C.

The form of the command is:

RE{ENTER}

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

For most system programs, the REENTER command restarts 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
   ↑C typed)
.
 REENTER
*
```

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> (↑B typed) B> R PIP *^C .SUSPEND	Foreground job is running. Control is redirected to the background job and PIP is called (the foreground is still active). CTRL C stops PIP and starts KMON. The foreground job is suspended. (See Section 2.7.5.2.)

FRUN

2.7.5.1 FRUN Command - The FRUN command is used to initiate foreground jobs. FRUN will only run relocatable 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 is:

$$\text{FRU}\left\{\text{N}\right\}\ \text{dev:file.ext}\left\{\text{/N:n}\right\}\left\{\text{/S:n}\right\}\left\{\text{/P}\right\}$$

where:

dev: represents a block replaceable RT-11 device. If dev: is not specified, DK: is 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 the program. 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 program is started when the RSUME command is given, allowing the programmer to examine or modify the program before starting it.

If another foreground job is active when the FRUN command is given, an error message is printed. If a terminated foreground job is 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 F1	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 is:

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 completion 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 scheduled while the foreground was suspended are entered at this time.

The form of the command is:

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 ENOUGH 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 and no 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 is expected.

Message	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 CLOCK?	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. Possible 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 unrecoverable 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 is:

?M-text PC	where PC is 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 listed 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 occurred, and the user program had no .SFPA exception routine active (see Chapter 9).
?M-ILL ADDR	Under the F/B Monitor, an address 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 the 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 the recovery procedures described in section 4.2.12.1 of Chapter 4.
?M-TRAP TO 4 ?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 performing critical operations from which it cannot recover. If the F/B Monitor halts, 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) is 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 modes: 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 loaded 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 3-1. 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, `*GMOV A,B$-1D$$`
CTRL 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 REENTER command may be used to restart the Editor, but the contents of the text buffers are lost.

(continued on next page)

Table 3-1 (cont.)
EDIT Key Commands

Key	Explanation
CTRL O	Echoes ↑O and a carriage return. Inhibits printing on the terminal until completion of the current command string. Typing a second CTRL O resumes output.
CTRL U	Echoes ↑U and a carriage return. Deletes all the characters on the current terminal input line. (Equivalent to 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:
	
	`*IABCD`
	`EFGH^X`
	`*`
	
	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 Mode 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 format, 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 performed. 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 -. If 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 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 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,

```
         text object           text object
*BGMOV PC,R0$-2CR1$5KGCLR @R2$$
  second      third       fifth
  command     command     command
first command       fourth
                     command
```

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$$
```

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 commands.

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 (↑ 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     #VEC↑T,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 command 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     #VEC↑T,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 typing:

```
*BADADVADADVADADV$$
```

The following bracket structures are examples of legal usage:

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

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

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

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

```
*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:

$$\texttt{*EBSAMP\$5}\underbrace{\texttt{<N4}\underbrace{\texttt{<BGMOV \#200,R0\$=J\$3}\underbrace{\texttt{<G0\$=C4\$>}}_{C}\texttt{>}}_{B}\texttt{>}}_{A}\texttt{EX\$\$}$$

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

Execution initially 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:

MACRO BUFFER	High Memory
SAVE BUFFER	
FREE MEMORY	
COMMAND INPUT BUFFER	
TEXT BUFFER	Low Memory

The Text Buffer contains the current 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 commands 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 0M or 0U (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 Buffer 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. However, 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 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; 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 is:

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 must 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 .BAK extension; any previous file with this filename and a .BAK 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$$	Opens BAS1.MAC on SY. When editing is complete, the old BAS1.MAC becomes BAS1.BAK and the new file becomes BAS1.MAC. Any previous version of BAS1.BAK is deleted.
*EBBAS2.BAS[15]$$	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 command (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. A form 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 is 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 printed 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 ending 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 pointer 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 0LL (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 commands until the end of the input file is reached, then closes both the input and output files.

The command format 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 is 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.

Examples:

`*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

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 command:

```
*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. If 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) must 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.

Example:

`*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 format 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 commands; 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 is:

```
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 I command will cause the text entered to be executed as commands.

EDIT automatically protects against overflowing the Text Buffer during an Insert. If the I command is the first command in a multiple command line, EDIT ensures that there will be enough space for the Insert to be executed at least once. If repetition of the command exceeds the available memory, an error message 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 is:

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 of:

```
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 command replaces n characters, starting at the pointer, with the specified text string and leaves the pointer positioned immediately following the changed text.

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 of:

```
CLR      @R2
MOV↑     5(R1),@R2
```

The command:

```
*0CADDB$$
```

leaves the buffer with:

```
CLR      @R2
ADDB↑    5(R1),@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 specifed 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)+        Exchanges the two lines to
CLR     @R2             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. If the Save command causes an overflow of the Save Buffer, an error message is printed.

Example:

Assume the Text Buffer contains the following assembly language subroutine:

```
;SUBROUTINE MSGTYP
;WHEN CALLED, EXPECTS R0 TO POINT TO AN
;ASCII MESSAGE THAT ENDS IN A ZERO BYTE,
;TYPES THAT MESSAGE ON THE USER TERMINAL

                .ASECT
                .=1000
MSGTYP:         TSTB (%0)               ;DONE?
                BEQ MDONE               ;YES-RETURN
MLOOP:          TSTB @#177564           ;NO-IS TERMINAL READY?
                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 is:

U	Insert in the Text Buffer the contents of the Save Buffer.
0U	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 0U 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 command string into the EDIT Macro Buffer. The Macro command is of the form:

M/command string/	Store the command string in the Macro Buffer.
0M or M//	Clear the Macro Buffer and reclaim the area for text.

/ 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 ALTMODEs to terminate text commands. All commands, except the M and EM commands, are legal in a command string macro.

In addition to the 0M 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 IN MACRO*?
*
```

Executes the MACRO stored in the previous example. An error message is returned when the end of buffer is reached. (This macro effectively changes all occurrences of R0 in the Text Buffer to R1.)

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

In a new program, inserts MOV PC,R1 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 is:

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 two commands are equivalent:

```
*GTEXT$=Cnew text$V$$

*gTEXT$=cnew text$v$$
```

The Editor automatically translates (internally) all commands to upper-case independent of EL or EU.

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 VT-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 3-1:

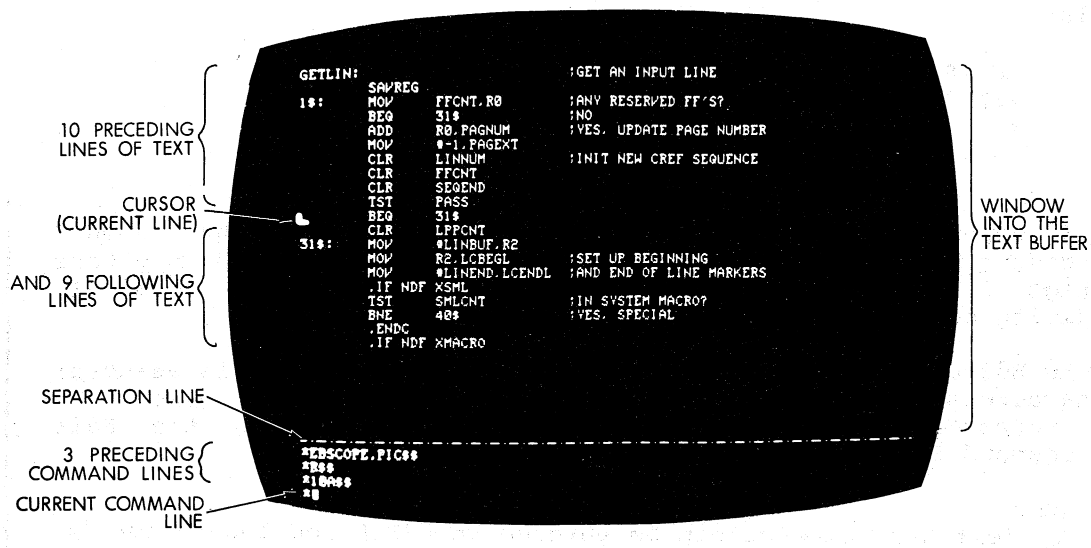


Figure 3-1
Display Editor Format

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 section 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
```

However, 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. If 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 3-1.

   The Edit Console command can be used to return the scroller to its normal mode so that text and commands 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 3-1.

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 display 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 VT-11 display hardware is active and the Editor is running; it is entered by typing two ALTMODES (only) in response to the Command Mode asterisk:

*$$

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

Once Immediate 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 Immediate Mode, type CTRL C.

Table 3-3
Immediate Mode Commands

Command	Meaning
CTRL N	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).
CTRL V	Move the pointer (cursor) back by one character (equivalent to -J).
RUBOUT	Delete the character immediately preceding the pointer (cursor) (equivalent to -D).
CTRL C	Return control to the monitor.
ALTMODE (one only) (two)	Return control to Command Mode. Direct control to Immediate 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 the device DK. Sections of the terminal output are coded by letter and corresponding explanations follow the example.

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

     START:  MOV #1000,%6      ;INITIALIZE STACK
             MOV #MSG,%0       ;POINT R0 TO MESSAGE
B            JSR %7,MSGTYP     ;PRINT IT
             HALT              ;STOP
     MSG:    .ASCII/IT WORKS/
             .BYTE 15
             .BYTE 12
             .BYTE 0

C    *B 1J 5D$$
     *GPROGRAM$$
D    *0L$$
     ;PROGRAM*I TO TEST SUBROUTINE MSGTYP. TYPES
E    ;"THE TEST PROGRAM WORKS"
     ;ON THE TEMI\IM\RMINAL$$
F    *F.ASCII/$$
     *8CTHE TEST PROGRAM WORKS$$
     *P.BYTE^X
G    *F.BYTE 0$V$$
             .BYTE 0
     *I
             .END
     $B/L$$
     ;PROGRAM TO TEST SUBROUTINE MSGTYP. TYPES
     ;"THE TEST PROGRAM WORKS"
     ;ON THE TERMINAL

     START:  MOV #1000,%6      ;INITIALIZE STACK
H            MOV #MSG,%0       ;POINT R0 TO MESSAGE
             JSR %7,MSGTYP     ;PRINT IT
             HALT              ;STOP
     MSG:    .ASCII/THE TEST PROGRAM WORKS/
             .BYTE 15
             .BYTE 12
             .BYTE 0
             .END

     *EX$$
I
     .
```

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 text. 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.

I Close input and output files after copying the current text 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.). If 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.
?ILL CMD?	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 text available.
?*SRCH FAIL*?	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 compress 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 exception: the asterisk character can be 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 rename 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 performed 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.

Examples:

`**.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 now 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 performed by PIP are summarized in Table 4-1. If no switch is specified, PIP assumes the operation is a file transfer in image (/I) mode. Detailed explanations of the switches follow 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) treated as 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 the 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 represents the numeric position of the file to be accessed in relation 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 the nth file is found; if n is negative, the tape is rewound first, and then it spaces 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 so that 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 specified 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 ACTION? 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 writes a volume label followed by a dummy file followed by double tape marks indicating logical end-of-tape.

4.2.1 Operations 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 so 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 deletions) 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 file-name (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 is:

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

and the last PIP operation involved FILE.4, leaving the cassette positioned at the end 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 spaces forward from its current 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 number of files are to be created on a magtape or cassette. 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 cassette) is to rewind, then search the tape for the logical end. If a file with the same 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 tape; 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 is 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 example:

```
.R PIP
*MT0:/E                               Extended directory: 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, 57 BLOCKS

*MT0:/L                               Normal directory; does
11-SEP-74                             not accurately display
A      .MAC  40 11-SEP-74             file positions.
B      .MAC  15 11-SEP-74
D      .MAC   2 11-SEP-74
3 FILES, 57 BLOCKS
```

If the user wished to access file A.MAC on the magtape in the example above, /M:-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 blocks. 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 operations involving cassette, if the cassette is full before the transfer has finished, the message:

```
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 message 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 NEW 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 CT0. The cassette runs out of room and requests 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 is the second part of a continued file.

```
*CTØ:/L
23-MAY-74
TRA    .BIN    Ø 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 abort the process at any time (using two CTRL Cs) since continuation files may not be completed and no sentinel file will be written on the cassette.

To read multiple files from a cassette, use 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:1000
```

Repeat the process as often as necessary to copy all files. Do not abort the process at any time (using two CTRL Cs) since continuation files may not be completed.

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 is typed during any output operation 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 followed:

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

   ```
   *dev1:*.*/X=dev0:file1,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 loss of information can occur. For example:

```
*CTØ:FILE1.MAC=CTØ: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 Operations

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 than ASCII or formatted binary. 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 follows:

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

Nulls and rubouts are ignored in an ASCII mode file transfer. CTRL Z (32 octal) is treated as logical end-of-file if encountered 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 assembler or FORTRAN 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 error 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=DT2: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 ASCII 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 size (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 fi'

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 less than 200 blocks. Transfers in this situation can be accomplished in either of two ways:

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 PIP
*DT1:A=A
?OUT ER?                File longer than 100 blocks.

*DT1:A[150]=A
or                      Either command causes a correct
*DT1:A=A/X              transfer.
```

4.2.3 Multiple Copy Operations

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. If /X is not indicated, all output files but the first will be ignored.

Examples:

`*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 F1 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 transferred 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. If there is a third, it is ignored and a fourth causes an ?OUT FIL? error.

```
*DT0:*.SYS=*.SYS/X/Y
```

Copies all system files from device DK to device DT0.

File transfers performed via normal operations 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 accommodate it. Therefore, the /X switch 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 is:

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

To copy the file PP.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
MONITR.SYS    32   5-MAY-74
< UNUSED >     2
PR    .SYS     2   5-MAY-74
PP    .SYS     2   9-MAY-74
< UNUSED >   526
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:

```
 9-MAY-74
MONITR.SYS    32  5-MAY-74
PP    .SYS     2  9-MAY-74
PR    .SYS     2  5-MAY-74
< UNUSED >   528
3 FILES, 36 BLOCKS
 528 FREE BLOCKS
```

4.2.4 The Extend and Delete Operations

The /T switch is used to increase the number of blocks 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 blocks to be allocated to the file at the completion of the extend operation.

The format of the /T switch is:

```
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 is 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 command 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]</T	Assigns 100 blocks to the file named XYZ on device DT1.

The /D switch is used to delete one or more files from the specified 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 is not used).

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

```
*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 form:

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

where filena.ext is the name desired and n is the length of the deleted file. For example:

```
*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 like:

```
*DT1:/E
 4-JUN-74
< UNUSED >    33
A      .MAC    18   3-JUN-74
< UNUSED >    17
C      .MAC    19   3-JUN-74
< UNUSED >   477
2 FILES, 37 BLOCKS
 527 FREE BLOCKS
```

A recovery of B.MAC would require:

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

If the 33 block unused area was not named prior to B.MAC, the first 17 blocks of the 33 block area would become B.MAC. Note that magtape and cassette files cannot be recovered once deleted.

Examples:

`*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 permanently 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.

Examples:

`*DT1:F1,X1<DT1:F0,X0/R`	Renames F0 to F1 and X0 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 Operations

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 blocks on the device if disk or DECtape, the number of files listed, and number of blocks 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   32  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 .MAC     7 22-JUL-74
FILE2 .MAC     1
2 FILES, 8 BLOCKS
3728 FREE BLOCKS
*
```

Lists on the terminal a directory of files on device DK with the extension .MAC.

```
*CT1:/L
10-SEP-74
PAT1  .FOR    0 10-SEP-74
PAT2  .FOR    0 10-SEP-74
IMUL  .OBJ    0 10-SEP-74
SQRT  .FTN    0 10-SEP-74
4 FILES, 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.

Examples:

```
*/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 lists only filenames, omitting the file lengths and associated dates.

Examples:

```
*DT0:/F
TRACE .MAC
CARGO .REL
BMAP  .OBJ
AAA
```

Outputs a filename directory of the device DT0 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.)

Using 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
3710 FREE BLOCKS
*
```

to be output, providing the 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 follows:

device/Z ARE YOU SURE ?

For example:

```
*DT1:/Z
DT1:/Z   ARE 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 number of directory blocks in a two-volume compress (that is, 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 performed 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 compress 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 signify 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?
```

Compresses the files on the system device SY:

```
*DT1:A<DT2:/S
```

Transfers and compresses the files from device DT2 to device DT1. 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 present; 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 filename.

```
*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.

If a boot switch is specified on an illegal device, the message:

```
?BAD BOOT?
```

is printed. Legal devices are DT0, RK0-RK7, RF, SY, DK, DP0-DP7, DX0-DX1, and DS0-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 BLOCK 140 IS BAD	Scan disk drive 2 for bad blocks.
*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. If 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 1-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 free area, create a file of length n-1 with the /T switch. Remember that there must be no spaces larger than n-1 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 1-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    4 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 reads:

```
 .
 .
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
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
< UNUSED >   150
A              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     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
   .
   .
```

Continuing with PIP:

```
*RK1:FILE.BAD[1]=/Y/T                    Create a bad file.
```

The directory now contains:

```
   .
   .
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.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
PIP   .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
< 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 on a volume.

4.3 PIP ERROR MESSAGES

The following error messages are output on the terminal when PIP is used incorrectly:

Errors	Meaning
?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 rename 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. If 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 the /N:n switch was 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 specified sequence 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 directives
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 preceded 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.**

5.1.1 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,R2
```

An assembly language 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 formatted with EDIT so that use of the TAB character causes the statement fields to be aligned. For example:

Label Field	Operator Field	Operand Field	Comment 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 independently 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. Subsequent 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 user 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 are 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 the 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 following examples:

```
MOV A,B     (space terminates the operator MOV)
MOV@A,B     (@ terminates the operator MOV)
```

When the statement line does not contain an operand or comment, the operator 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 expressions, 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 label 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 terminates 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.

5.1.2 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 written:

```
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-case 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 Section 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 expression indicator
+	plus sign	arithmetic addition operator or auto increment indicator
-	minus sign	arithmetic subtraction operator or auto decrement indicator
*	asterisk	arithmetic multiplication operator
/	slash	arithmetic division operator
&	ampersand	logical AND 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 5-1 and following.

Table 5-1
Legal Separating Characters

Character	Definition	Usage
space	one or more spaces and/or tabs	A space is a legal separator only for argument operands. Spaces within expressions are ignored.
,	comma	A comma is a legal separator for both expressions and argument operands.
<...>	paired angle brackets	Paired angle brackets are used to enclose an argument,

(Continued on next page)

Table 5-1 (cont.)
Legal Separating Characters

Character	Definition	Usage
		particularly when that argument contains separating characters. Paired angle brackets may be used anywhere in a program to enclose an expression for treatment as a term. (The angle bracket construction should be used when the argument contains unary operators.)
↑\...\	Up arrow construction where the up arrow character is followed by an argument bracketed by any paired printing characters.	This construction is equivalent in function to the paired angle brackets and is generally used only where the argument contains angle brackets.

Macro arguments may appear in several forms to allow for special cases. The rules to observe when separating arguments are:

1. If an argument string contains only non-separating characters (those not defined in Table 5-1) and no spaces, then it may appear in the argument list separated, if necessary, from the other arguments by commas.

2. If an argument string contains separating characters or spaces, but does not contain the characters < or > (left or right angle brackets), then the argument may appear enclosed in paired angle brackets (e.g., <argument string>). The paired angle brackets are removed before the argument string is used. Successive pairs of angle brackets may be used to enclose an argument; only the outermost pair is removed.

3. If an argument string contains separating characters or spaces (possibly including the left or right angle bracket characters), then it may appear in the following form: ↑\argument string\ where the backslashes may be replaced by any character not appearing in the argument string. The uparrow and backslashes (or other character) are removed before the argument string is substituted into the text.

Note that regardless of the method used to specify an argument, it must be separated from any other arguments by commas.

5.2.1.2 Illegal Characters - A character can be illegal in one of two ways:

1. A character which is not recognized as an element of the MACRO character set is always an illegal character and causes immediate termination of the current line at that point, plus the output of an error flag in the assembly listing. For example:

```
LABEL←*A:  MOV A,B
```

   Since the backarrow is not a recognized character, the entire line is treated as a:

```
.WORD LABEL
```

   statement and is flagged in the listing.

2. A legal MACRO character may be illegal in context. Such a character generates a Q error on the assembly listing.

5.2.1.3 Operator Characters - Under MACRO, legal unary operators (operators applying to only one operand) are as follows:

Unary Operator	Explanation	Example	
+	plus sign	+A	(positive value of A, equivalent to A)
-	minus sign	-A	(negative, 2's complement, value of A)
↑	uparrow, universal unary operator (this usage is described in greater detail in Sections 5.5.4.2 and 5.5.6.2).	↑F3.0	(interprets 3.0 as a 1-word floating-point number)
		↑C24	(interprets the one's complement of the binary representation of 24(8))
		↑D127	(interprets 127 as a decimal number)
		↑O34	(interprets 34 as an octal number)
		↑B11000111	(interprets 11000111 as a binary value)

The unary operators described above can be used adjacent to each other in a term. For example:

```
↑C↑O12
-↑O5
```

Legal binary operators under MACRO are as follows:

Binary Operator	Explanation	Example
+	addition	A+B
-	subtraction	A-B
*	multiplication	A*B (16-bit product returned)
/	division	A/B (16-bit quotient returned)
&	logical AND	A&B
!	logical inclusive OR	A!B

All binary operators have the same priority. Division and multiplication are signed operations. Items can be grouped for evaluation within an expression by enclosure in angle brackets. Terms in angle brackets are evaluated first, and remaining operations are performed left to right. For example:

```
.WORD    1+2*3    ;IS 11 OCTAL
.WORD    1+<2*3>  ;IS 7 OCTAL
```

5.2.2 Symbols

There are three types of symbols: permanent, user-defined and macro. MACRO maintains three types of symbol tables: the Permanent Symbol Table (PST), the User Symbol Table (UST) and the Macro Symbol Table (MST). The PST contains all the permanent symbols and is part of the MACRO Assembler load module. The UST and MST are constructed as the source program is assembled; user-defined symbols are added to the table as they are encountered.

5.2.2.1 Permanent Symbols -

Permanent symbols consist of the instruction mnemonics (Appendix C) and assembler directives and macro directives (sections 5.5 and 5.6, Appendix C). These symbols are a permanent part of the assembler and need not be defined before being used in the source program.

5.2.2.2 User-Defined and Macro Symbols -

User-defined symbols are those used as labels or defined by direct assignment (Section 5.2.3). These symbols are added to the User Symbol Table as they are encountered during the first pass of the assembly. Macro symbols are those symbols used as macro names in the operator field (Section 5.6.1). These symbols are added to the Macro Symbol Table as they are encountered during the assembly.

User-defined and macro symbols can be composed of alphanumeric characters, dollar signs, and periods only; any other character is illegal.

The $ and . characters are reserved for system software symbols (for example, the system macro symbol .READ); it is recommended that $ and . not be inserted in user-defined or macro symbols.

The following rules apply to the creation of user-defined and macro symbols:

1. The first character must not be a number (except in the case of local symbols, see Section 5.2.5).

2. Each symbol must be unique within the first six characters.

3. A symbol can be written with more than six legal characters, but the seventh and subsequent characters are only checked for legality, and are not otherwise recognized by the assembler.

4. Spaces, tabs, and illegal characters must not be embedded within a symbol.

The value of a symbol depends upon its use in the program. A symbol in the operator field may be any one of the three symbol types. To determine the value of the symbol, the assembler searches the three symbol tables in the following order:

1. Macro Symbol Table

2. Permanent Symbol Table

3. User-Defined Symbol Table

A symbol found in the operand field is sought in the:

1. User-Defined Symbol Table

2. Permanent Symbol Table

in that order. The assembler never expects to find a macro name in an operand field.

These search orders allow redefinition of Permanent Symbol Table entries as user-defined or macro symbols. The same name can be assigned to both a macro and a label.

User-defined symbols are either internal or external (global). All user-defined symbols are internal unless explicitly defined as being global with the .GLOBL directive (see Section 5.5.10).

Global symbols provide links between object modules. A global symbol defined as a label is generally called an entry point (to a section of code). Such symbols are referenced from other object modules to transfer control throughout the load module (which may be composed of a number of object modules).

Since MACRO provides program sectioning capabilities (Section 5.5.9), two types of internal symbols must be considered:

1. Symbols that belong to the current program section, and

2. Symbols that belong to other program sections.

In both cases, the symbol must be defined within the current assembly; the significance of the distinction is critical in evaluating expressions involving type (2) above (see Section 5.2.9).

5.2.3 Direct Assignment

A direct assignment statement associates a symbol with a value. When a direct assignment statement defines a symbol for the first time, that symbol is entered into the user symbol table and the specified value is associated with it. A symbol may be redefined by assigning a new value to a previously defined symbol. The latest assigned value replaces any previous value assigned to a symbol.

The general format for a direct assignment statement is:

symbol = expression

Symbols take on the relocatable or absolute attribute of their defining expression. However, if the defining expression is global, the symbol is not global unless explicitly defined as such in a .GLOBL directive. For example:

```
        A=1             ;THE SYMBOL A IS EQUATED TO THE
                        ;VALUE 1

        B='A-1&MASKLOW  ;THE SYMBOL B IS EQUATED TO THE
                        ;VALUE OF THE EXPRESSION

C:      D=3             ;THE SYMBOL D IS EQUATED TO 3
```

```
E:      MOV     #1,ABLE ;LABELS C AND E ARE EQUATED TO THE
                        ;LOCATION OF THE MOV COMMAND
```

The following conventions apply to direct assignment statements:

1. An equal sign (=) must separate the symbol from the expression defining the symbol value.

2. A direct assignment statement is usually placed in the operator field and may be preceded by a label and followed by a comment.

NOTE

If the program jumps to or references the label of a direct assignment statement, it is actually referencing the following instruction statement. For example:

```
        .=.+1000
C:      D=3
E:      MOV #D,ABLE

         .
         .
        JMP C
```

This code causes a jump to the label E.

3. Only one symbol can be defined by any one direct assignment statement.

4. Only one level of forward referencing is allowed. That is, the following arrangement is illegal:

   X = Y

   Y = Z

   Z = 1

   X and Y are both undefined throughout pass 1. X is undefined throughout pass 2 and causes an error flag in the assembly listing.

5.2.4 Register Symbols

The eight general registers of the PDP-11 are numbered 0 through 7 and can be expressed in the source program as:

```
%0
%1
 .
 .
 .
%7
```

The digit indicating the specific register can be replaced by any legal term which can be evaluated during the first assembly pass.

It is recommended that the programmer create and use symbolic names for all register references. A register symbol may be defined in a direct assignment statement among the first statements in the program. A register symbol cannot be defined after the statement which uses it. The defining expression of a register symbol must be absolute. For example:

```
R0=%0                   ;REGISTER DEFINITION
R1=%1
R2=%2
R3=%3
R4=%4
R5=%5
SP=%6
PC=%7
```

The symbolic names assigned to the registers in the example above are the conventional names used in all PDP-11 system programs. Since these names are fairly mnemonic, it is suggested the user follow this convention. Registers 6 and 7 are given special names because of their special functions, while registers 0 through 5 are given similar names to denote their status as general purpose registers.

All register symbols must be defined before they are referenced. A forward reference to a register symbol causes phase errors in an assembly.

The % character can be used with any term or expression to specify a register. (A register expression less than 0 or greater than 7 is flagged with an R error code.) For example:

```
CLR %3+1
```

is equivalent to:

```
CLR %4
```

and clears the contents of register 4, while:

```
CLR 4
```

clears the contents of memory address 4.

In certain cases a register can be referenced without the use of a register symbol or register expression; these cases are recognized through the context of the statement. An example is shown below:

```
JSR 5,SUBR      ;FIRST OPERAND FIELD MUST ALWAYS
                ;BE A REGISTER
```

5.2.5 Local Symbols

Local symbols are specially formatted symbols used as labels within a given range.

Local symbols provide a convenient means of generating labels to be referenced by branch instructions. Use of local symbols reduces the possibility of multiply-defined symbols within a user program and separates entry point symbols from local references. Local symbols, then, are not referenced from other object modules or even from outside their local symbol block.

Local symbols are of the form n$, where n is a decimal integer from 1 to 127, inclusive, and can only be used on word boundaries. Local symbols include:

```
  1$
 27$
 59$
104$
```

Within a local symbol block, local symbols can be defined and referenced. However, a local symbol cannot be referenced outside the block in which it is defined. There is no conflict with labels of the same name in other local symbol blocks.

Local symbols 64$ through 127$ can be generated automatically as a feature of the macro processor (see Section 5.6.3.5 for further details). When using local symbols the user is advised to first use the range from 1$ to 63$.

A local symbol block is delimited in one of the following ways:

1. The range of a single local symbol block can consist of those statements between two normally constructed symbolic labels. (Note that a statement of the form:

   ```
   LABEL=.
   ```

   is a direct assignment, does not create a label in the strict sense, and does not delimit a local range.)

2. The range of a local symbol block is terminated upon encountering a .CSECT directive.

3. The range of a single local symbol block can be delimited with .ENABL LSB and the first symbolic label or .CSECT directive following the .DSABL LSB directives. The default for LSB is off.

For examples of local symbols and local symbol blocks, see Figure 5-1.

The maximum offset of a local symbol from the base of its local symbol block is 128 decimal words. Symbols beyond this range are flagged with an A error code.

5.2.6 Assembly Location Counter

The period (.) is the symbol for the assembly location counter. When used in the operand field of an instruction, it represents the address of the first word of the instruction. When used in the operand field of an assembler directive, it represents the address of the current byte or word. For example:

```
A:      MOV     #.,R0   ;. REFERS TO LOCATION A,
                        ;I.E., THE ADDRESS OF THE
                        ;MOV INSTRUCTION
```

(# is explained in Section 5.4.9).

At the beginning of each assembly pass, the assembler clears the location counter. Normally, consecutive memory locations are assigned to each byte of object data generated. However, the location where the object data is stored may be changed by a direct assignment statement altering the location counter:

```
.=expression
```

The expression defining the location counter must not contain forward references or symbols that vary from one pass to another. If an expression is assigned to the current location counter in a relocatable CSECT, an error flag is generated. (The construction .=.+expression must be used.)

Similar to other symbols, the location counter symbol has a mode associated with it, either absolute or relocatable; the mode cannot be external. The existing mode of the location counter cannot be changed by using a defining expression of a different mode.

```
 Line      Octal
Number   Expansion          Source Code                      Comments

  1
  2
  3
  4
  5
  6                         .MCALL  .REGDEF,..V2..
  7 000000                  .REGDEF
  8 000000                  ..V2..
  9        000000           R0=%0
 10                         .SBTTL  SECTOR INITIALIZATION
 11        000000'          .CSECT  IMPURE          ;IMPURE STORAGE AREA
 12 00000          IMPURE:
 13        000000'                  .CSECT  IMPPAS  ;CLEARED EACH PASS
 14 00000          IMPPAS:
 15        000000'          .CSECT  IMPLIN          ;CLEARED EACH LINE
 16 00000          IMPLIN:
 17        000000'          .CSECT  XCTPRG          ;PROGRAM
 18                                                 ;INITIALIZATION
 19 00000          XCTPRG:
 20 00000  012700           MOV     #IMPURE,R0
           000000'
 21 00004  005020  1$:      CLR     (R0)+           ;CLEAR IMPURE AREA
 22 00006  022700           CMP     #IMPTOP,R0
           000000'
 23 00012  101374           BHI     1$
 24
 25        000000'          .CSECT  XCTPAS          ;PASS INITIALIZATION
 26 00000          XCTPAS:
 27 00000  012700           MOV     #IMPPAS,R0
           000000'
 28 00004  005020  1$:      CLR     (R0)+           ;CLEAR IMPURE PART
 29 00006  022700           CMP     #IMPTOP,R0
           000000'
 30 00012  101374           BHI     1$
 31
 32        000000'          .CSECT  XCTLIN          ;LINE INITIALIZATION
 33 00000          XCTLIN:
 34 00000  012700           MOV     #IMPLIN,R0
           000000'
 35 00004  005020  1$:      CLR     (R0)+
 36 00006  022700           CMP     #IMPTOP,R0
           000000'
 37 00012  101374           BHI     1$
 38
 39        000000'          .CSECT  MIXED           ;MIXED MODE SECTOR
 40 00000  000000  IMPTOP:  .WORD 0
 41        000001'          .END
```

Figure 5-1
Assembly Source Listing of MACRO Code Showing Local Symbol Blocks

The mode of the location counter symbol can be changed by the use of the .ASECT or .CSECT directive as explained in Section 5.5.9.

Examples:

```
          .ASECT

  .=500                           ;SET LOCATION COUNTER TO
                                  ;ABSOLUTE 500

  FIRST:  MOV .+10,COUNT          ;THE LABEL FIRST HAS THE VALUE
                                  ;500(8)
                                  ;.+10 EQUALS 510(8).  THE
                                  ;CONTENTS OF THE LOCATION
                                  ;510(8) WILL BE DEPOSITED
  COUNT:  .WORD 0                 ;IN LOCATION COUNT.

  .=520                           ;THE ASSEMBLY LOCATION COUNTER
                                  ;NOW HAS A VALUE OF
                                  ;ABSOLUTE 520(8).

  SECOND: MOV .,INDEX             ;THE LABEL SECOND HAS THE
                                  ;VALUE 520(8)
                                  ;THE CONTENTS OF LOCATION
                                  ;520(8), THAT IS, THE BINARY
                                  ;CODE FOR THE INSTRUCTION
                                  ;ITSELF WILL BE DEPOSITED IN
  INDEX:  .WORD 0                 ;LOCATION INDEX.

          .CSECT

          .=.+20                  ;SET LOCATION COUNTER TO
                                  ;RELOCATABLE 20 OF THE
                                  ;UNNAMED PROGRAM SECTION.

  THIRD:  .WORD 0                 ;THE LABEL THIRD HAS THE
                                  ;VALUE OF RELOCATABLE 20.
```

Storage area may be reserved by advancing the location counter. For example, if the current value of the location counter is 1000, the direct assignment statement:

```
        .=.+100
```

reserves 100(octal) bytes of storage space in the program. The next instruction is stored at 1100. (The .BLKW and .BLKB directives can also be used to reserve blocks of storage; see Section 5.5.5.3.)

5.2.7 Numbers

The MACRO Assembler assumes all numbers in the source program are to be interpreted in octal radix unless otherwise specified. The assumed radix can be altered with the .RADIX directive or individual numbers can be treated as being of decimal, binary, or octal radix (see Section 5.5.4.2).

Octal numbers consist of the digits 0 through 7 only. A number not specified as a decimal number and containing an 8 or 9 is flagged with an N error code and treated as a decimal number.

Negative numbers are preceded by a minus sign (the assembler translates them into two's complement form). Positive numbers may be preceded by a plus sign, although this is not required.

A number which is too large to fit into 16 bits (177777<n) is truncated from the left and flagged with a T error code in the assembly listing.

Numbers are always considered absolute quantities (that is, not relocatable).

The single-word floating-point numbers which can be generated with the ↑F operator (see Section 5.5.6.2) are stored in the following format:

15	14 — 7	6 — 0
SIGN BIT	8-BIT EXPONENT	7-BIT MANTISSA

Refer to <u>PDP-11/45 Processor Handbook</u> for details of the floating-point format.

5.2.8 Terms

A term is a component of an expression. A term may be one of the following:

1. A number whose 16-bit value is used.

2. A symbol that is interpreted according to the following hierarchy:

   a. a period that causes the value of the current location counter to be used

   b. a permanent symbol whose basic value is used and whose arguments (if any) are ignored

   c. user defined symbols

   d. an undefined symbol that is assigned a value of zero and inserted in the user-defined symbol table

3. An ASCII conversion using either an apostrophe followed by a single ASCII character or a double quote followed by two ASCII characters, which results in a word containing the 7-bit ASCII value of the character(s). (This construction is explained in greater detail in Section 5.5.3.3.)

4. An expression enclosed in angle brackets. Any quantity enclosed in angle brackets is evaluated before the remainder of the expression in which it is found. Angle brackets are used to alter the left to right evaluation of expressions (for example, to differentiate between A*B+C and A*<B+C>) or to apply a unary operator to an entire expression (-<A+B>).

5.2.9 Expressions

Expressions are combinations of terms that are joined together by binary operators and that reduce to a 16-bit value. The operands of a .BYTE directive are evaluated as word expressions before truncation to the low-order eight bits. Prior to truncation, the high-order byte must be zero or all ones (when the byte value is negative, the sign bit is propagated). The evaluation of an expression includes the evaluation of the mode of the resultant expression--that is, absolute, relocatable or external. Expression modes are defined further below.

Expressions are evaluated left to right with no operator hierarchy rules except that unary operators take precedence over binary operators. A term preceded by a unary operator can be considered as containing that unary operator. (Terms are evaluated, where necessary, before their use in expressions.) Multiple unary operators are valid and are treated as follows:

```
-+-A
```

is equivalent to:

```
-<+<-A>>
```

The value of an external expression is the value of the absolute part of the expression; e.g., EXT+A has a value of A. This is modified by the Linker to become EXT+A.

Expressions, when evaluated, are either absolute, relocatable, or external. For the programmer writing position independent code, the distinction is important.

1. An expression is absolute if its value is fixed. An expression whose terms are numbers and ASCII conversions has an absolute value. A relocatable expression minus a relocatable term, where both items belong to the same program section, is also absolute.

2. An expression is relocatable if its value is fixed relative to a base address but will have an offset value added when linked. Expressions whose terms contain labels defined in relocatable sections and the assembly location counter (in relocatable sections) have a relocatable value.

3. An expression is external (or global) if its value is only partially defined during assembly and is completed at link

time. An expression whose terms contain a global symbol not defined in the current program is an external expression. External expressions have relocatable values at execution time if the global symbol is defined as being relocatable or absolute if the global symbol is defined as absolute.

An example of the three expression types follows:

```
        .ASECT
.=100
ABSSYM=.                           ;THE VALUE OF ABSSYM IS
                                   ;NOT RELOCATABLE, BECAUSE
                                   ;WE ARE IN AN ASECT

        .CSECT MAIN                ;START RELOCATABLE
                                   ;PROGRAM SECTION

        .GLOBL EXTVAL              ;EXTVAL IS DEFINED ELSEWHERE,
                                   ;ITS VALUE WILL NOT BE KNOWN
                                   ;UNTIL LINK TIME

BEGSYM: .BLKW 4                    ;THE VALUES OF BEGSYM
        .ASCII /ABCD/              ;AND ENDSYM ARE
        .EVEN                      ;RELOCATABLE, BECAUSE
ENDSYM=.                           ;THE ADDRESS AT WHICH
                                   ;"MAIN" WILL BE LOADED
                                   ;IS NOT DETERMINED UNTIL
                                   ;LINK TIME

SIZE = ENDSYM-BEGSYM               ;HOWEVER, THE
                                   ;VALUE OF SIZE IS KNOWN
                                   ;(IT IS 12.)AT ASSEMBLY
                                   ;TIME AND IS ABSOLUTE

RELEXP = ENDSYM-BEGSYM+.           ;RELEXP (=.+12.) IS
                                   ;RELOCATABLE


EXTEXP: .WORD  EXTVAL+4            ;THE EXPRESSION "EXTVAL+4"
                                   ;IS EXTERNAL (OR GLOBAL)
                                   ;BECAUSE EXTVAL IS DEFINED
                                   ;IN ANOTHER PROGRAM UNIT.

CHARA='A                           ;THE VALUE OF CHARA
                                   ;IS ABSOLUTE
```

5.3 RELOCATION AND LINKING

The output of the MACRO Assembler is an object module which must be processed by LINK before loading and execution (refer to Chapter 6 for details). The Linker essentially fixes (i.e., makes absolute) the values of external or relocatable symbols and turns the object module into a load module.

To enable the Linker to fix the value of an expression, the assembler issues certain directives to the Linker together with required parameters. In the case of relocatable expressions, the Linker adds the base of the associated relocatable section (the location in memory of relocatable 0) to the value of the relocatable expression provided by the assembler. In the case of an external expression, the value of the external term in the expression is determined by the Linker (since the external symbol must be defined in one of the other object modules which are being linked together) and adds it to the value of the external expression provided by the assembler.

All words that are to be modified (as described in the previous paragraph) are marked with an apostrophe in the assembly listing. A G in the listing indicates that the value is external, or that a global is added to that value. Thus, the binary text output looks as follows:

```
005065          CLR     EXTERNAL(R5)    ;VALUE OF EXTERNAL SYMBOL
000000G
                                        ;ASSEMBLED ZERO; WILL BE
                                        ;MODIFIED BY THE LINKER.

005065          CLR     EXTERNAL+6(R5)  ;THE ABSOLUTE PORTION OF THE
000006G
                                        ;EXPRESSION (000006) IS ADDED
                                        ;BY THE LINKER TO THE VALUE OF
                                        ;THE EXTERNAL SYMBOL

005065          CLR RELOCATABLE(R5)     ;ASSUMING WE ARE IN A
000040
                                        ;RELOCATABLE SECTION
                                        ;AND THE VALUE OF RELOCATABLE
                                        ;IS RELOCATABLE 40
```

5.4 ADDRESSING MODES

The program counter (PC, register 7 of the eight general registers) always contains the address of the next word to be fetched; i.e., the address of the next instruction to be executed, or the second or third word of the current instruction.

In order to understand how the address modes operate and how they assemble, the action of the program counter must be understood. The key rule is:

> Whenever the processor implicitly uses the program counter to fetch a word from memory, the program counter is automatically incremented by two after the fetch.

That is, when an instruction is fetched, the PC is incremented by two so that it is pointing to the next word in memory; if an instruction uses indexing (Sections 5.4.7, 5.4.9 and 5.4.11) the processor uses the program counter to fetch the base from memory. Hence, using the rule above, the PC increments by two, and now points to the next word.

The following conventions are used in this section:

1. Let E be any expression as defined in Section 5.2.

2. Let R be a register expression. This is any expression containing a term preceded by a % character or a symbol previously equated to such a term.

Examples:

```
R0=%0   ;GENERAL REGISTER 0
R1=R0+1 ;GENERAL REGISTER 1
R2=1+%1 ;GENERAL REGISTER 2
```

3. Let ER be a register expression or an expression in the range 0 to 7 inclusive.

4. Let A be any general address specification which produces a 6-bit mode address field as described in Sections 3.1 and 3.2 of the PDP-11 PROCESSOR HANDBOOK (both 11/20 and 11/45 versions).

The addressing specifications, A, can be explained in terms of E, R, and ER as defined above. Each is illustrated with the single operand instruction CLR or double operand instruction MOV.

5.4.1 Register Mode

The register contains the operand.

Format for A: R

Examples:

```
R0=%0           ;DEFINE R0 AS REGISTER 0
CLR     R0      ;CLEAR REGISTER 0
```

5.4.2 Register Deferred Mode

The register contains the address of the operand.

Format for A: @R or (ER)

Examples:

```
CLR     @R1     ;BOTH INSTRUCTIONS CLEAR
                ;THE WORD AT THE ADDRESS
CLR     (R1)    ;CONTAINED IN REGISTER 1
```

5.4.3 Autoincrement Mode

The contents of the register are incremented immediately after being used as the address of the operand. (See NOTE below.)

Format for A: (ER)+

Examples:

```
        CLR     (R0)+   ;EACH INSTRUCTION CLEARS
        CLR     (R0+3)+ ;THE WORD AT THE ADDRESS
        CLR     (R2)+   ;CONTAINED IN THE SPECIFIED
                        ;REGISTER AND INCREMENTS
                        ;THAT REGISTER'S CONTENTS
                        ;BY TWO.

        CLRB    (R4)+   ;CLEARS THE BYTE AT THE
                        ;ADDRESS SPECIFIED BY THE
                        ;CONTENTS OF R4 AND
                        ;INCREMENTS R4 BY ONE.
```

NOTE

Both JMP and JSR instructions using non-deferred autoincrement mode, autoincrement the register before its use on the PDP-11/20 and 11/05 (but not on the PDP-11/40 or 11/45). In double operand instructions of the addressing form %R,(R)+ or %R,-(R) where the source and destination registers are the same, the source operand is evaluated as the autoincremented or autodecremented value, but the destination register, at the time it is used, still contains the originally intended effective address.

In the following two examples, as executed on the PDP-11/20, R0 originally contains 100.

```
MOV     R0,(R0)+    ;THE QUANTITY 102 IS MOVED
                    ;TO LOCATION 100

MOV     R0,-(R0)    ;THE QUANTITY 76 IS MOVED
                    ;TO LOCATION 76
```

The use of these forms should be avoided as they are not compatible with the PDP-11/05, 11/40 and 11/45.

A Z error code is printed with each instruction which is not compatible among all members of the PDP-11 family. This is merely a warning code.

5.4.4 Autoincrement Deferred Mode

The register contains the pointer to the address of the operand. The contents of the register are incremented after being used.

Format for A: @(ER)+

```
Example:      CLR     @(R3)+   ;CONTENTS OF REGISTER 3 POINT
                               ;TO ADDRESS OF WORD TO BE
                               ;CLEARED, AND REGISTER 3 IS
                               ;THEN INCREMENTED BY TWO
```

5.4.5 Autodecrement Mode

The contents of the register are decremented before being used as the address of the operand (see NOTE under autoincrement mode).

Format for A: -(ER)

```
Examples:     CLR     -(R0)    ;DECREMENT CONTENTS OF
              CLR     -(R0+3)  ;0, 3, AND 2 BY TWO
              CLR     -(R2)    ;BEFORE USING AS ADDRESSES
                               ;OF WORDS TO BE CLEARED.
```

5.4.6 Autodecrement Deferred Mode

The contents of the register are decremented before being used as the pointer to the address of the operand.

Format for A: @-(ER)

```
Example:      CLR     @-(R2)   ;DECREMENT CONTENTS OF
                               ;REGISTER 2 BY TWO BEFORE
                               ;USING AS A POINTER
                               ;TO ADDRESS OF WORD TO BE
                               ;CLEARED.
```

5.4.7 Index Mode

The value of an expression E is stored as the second or third word of the instruction. The effective address is calculated as the value of E plus the contents of register ER. The value E is called the base.

Format for A: E(ER)

```
Examples:     CLR     X+2(R1)  ;EFFECTIVE ADDRESS IS X+2 PLUS
                               ;THE CONTENTS OF REGISTER 1
              CLR     -2(R3)   ;EFFECTIVE ADDRESS IS -2 PLUS
                               ;THE CONTENTS OF REGISTER 3.
```

5.4.8 Index Deferred Mode

An expression plus the contents of a register gives the pointer to the address of the operand.

Format for A: @E(ER)

```
Example:     CLR     @14(R4) ;IF REGISTER 4 HOLDS 100 AND
                             ;LOC 114 HOLDS 2000,
                             ;LOCATION 2000 IS CLEARED.
```

5.4.9 Immediate Mode

The immediate mode allows the operand itself to be stored as the second or third word of the instruction. It is assembled as an autoincrement of register 7, the PC.

Format for A: #E

```
Examples: MOV     #100,R0 ;MOVE AN OCTAL 100 TO
                          ;REGISTER 0
          MOV     #X,R0   ;MOVE THE VALUE OF THE SYMBOL X TO
                          ;REGISTER 0
```

The operation of this mode can be explained by the following example. The statement MOV #100,R3 assembles as two words. These are:

```
012703
000100
```

Just before this instruction is fetched and executed, the PC points to the first word of the instruction. The processor fetches the first word and increments the PC by two. The source operand mode is 27 (autoincrement the PC). Thus, the PC is used as a pointer to fetch the operand (the second word of the instruction) before being incremented by two, to point to the next instruction.

5.4.10 Absolute Mode

Absolute mode is the equivalent of immediate mode deferred. @#E specifies an absolute address which is stored in the second or third word of the instruction. Absolute mode is assembled as an autoincrement deferred of register 7, the PC.

Format for A: @#E

```
Examples: MOV     @#100,R0     ;MOVE THE VALUE OF CONTENTS
                               ;OF LOCATION 100 TO
                               ;REGISTER 0.
          CLR     @#X          ;CLEAR THE CONTENTS OF THE
                               ;LOCATION WHOSE ADDRESS IS X.
```

5.4.11 Relative Mode

Relative mode is the normal mode for memory references.

Format for A: E

```
Examples:   CLR     100     ;CLEAR LOCATION 100
            MOV     X,Y     ;MOV THE CONTENTS OF LOCATION X
                            ;TO LOCATION Y.
```

Relative mode is assembled as index mode, using register 7, the PC, as the index register. The base of the address calculation, which is stored in the second or third word of the instruction, is not the address of the operand (as in index mode), but the number which, when added to the PC, becomes the address of the operand. Thus, the base is X-PC, which is called an offset. The operation is explained as follows:

If the statement MOV 100,R3 is assembled at absolute location 20, the assembled code is:

```
Location 20:        016703
Location 22:        000054
```

The processor fetches the MOV instruction and adds two to the PC so that it points to location 22. The source operand mode is 67, that is, indexed by the PC. To pick up the base, the processor fetches the word pointed to by the PC and adds two to the PC. The PC now points to location 24. To calculate the address of the source operand, the base is added to the designated register, that is, BASE+PC=54+24=100, the operand address.

Since the assembler considers "." as the address of the first word of the instruction, an equivalent index mode statement would be:

```
MOV 100-.-4(PC),R3
```

This mode is called relative because the operand address is calculated relative to the current PC. The base is the distance or offset (in bytes) between the operand and the current PC. If the operator and its operand are moved in memory so that the distance between the operator and data remains constant, the instruction will operate correctly anywhere in memory.

5.4.12 Relative Deferred Mode

Relative deferred mode is similar to relative mode, except that the expression, E, is used as the pointer to the address of the operand.

Format for A: @E

```
Example:    MOV @X,R0       ;MOVE THE CONTENTS OF THE
                            ;LOCATION WHOSE ADDRESS IS IN
                            ;X INTO REGISTER 0
```

5.4.13 Table of Mode Forms and Codes

Each instruction assembles into at least one word. Operands of the first six forms listed below do not increase the length of an instruction. Each operand in one of the other modes, however, increases the instruction length by one word.

Form	Mode	Meaning
R	0n	Register mode
@R or (ER)	1n	Register deferred mode
(ER)+	2n	Autoincrement mode
@(ER)+	3n	Autoincrement deferred mode
-(ER)	4n	Autodecrement mode
@-(ER)	5n	Autodecrement deferred mode

n represents the register number.

Any of the following forms adds one word to the instruction length:

Form	Mode	Meaning
E (ER)	6n	Index mode
@E(ER)	7n	Index deferred mode
#E	27	Immediate mode
@#E	37	Absolute memory reference mode
E	67	Relative mode
@E	77	Relative deferred reference mode

n represents the register number. Note that in the last four forms, register 7 (the PC) is referenced.

NOTE

> An alternate form for @R is (ER). However, the form @(ER) is equivalent to @0(ER).
>
> The form @#E differs from the form E in that the second or third word of the instruction contains the absolute address of the operand rather than the relative distance between the operand and the PC. Thus, the instruction CLR @#100 clears absolute location 100 even if the instruction is moved from the point at which it was assembled. See the description of the .ENABL AMA function in Section 5.5.2, which directs the assembly of all relative mode addresses as absolute mode addresses.

5.4.14 Branch Instruction Addressing

The branch instructions are 1-word instructions. The high byte contains the op code and the low byte contains an 8-bit signed offset which specifies the branch address relative to the PC. Upon execution of a branch instruction, the hardware calculates the branch address as follows:

1. Extend the sign of the offset through bits 8-15.
2. Multiply the result by 2. This creates a word offset rather than a byte offset.

3. Add the result to the PC to form the final branch address.

The assembler performs the reverse operation to form the byte offset from the specified address. Remember that when the offset is added to the PC, the PC is pointing to the word following the branch instruction; hence the term -2 in the calculation.

Byte offset = (E-PC)/2 truncated to eight bits.

Since PC = .+2, we have:

Byte offset = (E-.-2)/2 truncated to eight bits.

NOTE

It is illegal to branch to a location specified as an external symbol, or to a relocatable symbol from within an absolute section, or to an absolute symbol or a relocatable symbol or another program section from within a relocatable section.

5.4.15 EMT and TRAP Addressing

The EMT and TRAP instructions do not use the low-order byte of the word. This allows information to be transferred to the trap handlers in the low-order byte. If EMT or TRAP is followed by an expression, the value is put into the low-order byte of the word. However, if the expression is too big (>377(8)) it is truncated to eight bits and a T error flag is generated.

5.5 ASSEMBLER DIRECTIVES

Directives are statements which cause the assembler to perform certain processing operations.

Assembler directives can be preceded by a label, subject to restrictions associated with specific directives, and followed by a comment. An assembler directive occupies the operator field of a MACRO source line. Only one directive can be placed on any one line. Zero, one, or more operands can occupy the operand field; legal operands differ with each directive and may be either symbols, expressions, or arguments.

5.5.1 Listing Control Directives

5.5.1.1 .LIST and .NLIST

Listing options can be specified in the text of a MACRO program through the .LIST and .NLIST directives. These are of the form:

```
.LIST arg
.NLIST arg
```

where arg represents one or more optional arguments. When used without arguments, the listing directives alter the listing level count. The listing level count causes the listing to be suppressed when it is negative. The count is initialized to zero, incremented for each .LIST and decremented for each .NLIST. For example:

```
        .LIST ME
        .MACRO LTEST     ;LIST TEST
;A=THIS LINE SHOULD LIST
        .NLIST
;B=THIS LINE SHOULD NOT LIST
        .NLIST
;C=THIS LINE SHOULD NOT LIST
        .LIST
;D=THIS LINE SHOULD NOT LIST (LEVEL NOT BACK TO ZERO)
        .LIST
;E=THIS LINE SHOULD LIST (LEVEL BACK TO ZERO)
        .ENDM
        LTEST                  ;CALL THE MACRO

;A=THIS LINE SHOULD LIST
;E=THIS LINE SHOULD LIST (LEVEL BACK TO ZERO)
```

The primary purpose of the level count is to allow macro expansions to be selectively listed and yet exit with the level returned to the status current during the macro call.

The use of arguments with the listing directives does not affect the level count; however, .LIST and .NLIST can be used to override the current listing control. For example:

```
.MACRO XX
 .
 .
.LIST    ;LIST NEXT LINE
X=.
.NLIST   ;DO NOT LIST REMAINDER
         ;OF MACRO EXPANSION
 .
 .
.ENDM
.NLIST ME      ;DO NOT LIST MACRO EXPANSIONS
XX
X=.
```

Allowable arguments for use with the listing directives are as follows (these arguments can be used singly or in combination):

Argument	Default	Function
SEQ	list	Controls the listing of source line sequence numbers.
LOC	list	Controls the listing of the location counter (this field would not normally be suppressed).
BIN	list	Controls the listing of generated binary code (supersedes BEX).

BEX	list	Controls listing of binary extensions; that is, prevents listing those locations and binary contents beyond the first line of an expansion. This is a subset of the BIN argument.
SRC	list	Controls the listing of the source code.
COM	list	Controls the listing of comments. This is a subset of the SRC argument and can be used to reduce listing time and/or space where comments are unnecessary.
MD	list	Controls listing of macro definitions and repeat range expansions (has no effect in ASEMBL).
MC	list	Controls listing of macro calls and repeat range expansions (has no effect in ASEMBL).
ME	no list	Controls listing of macro expansions (supersedes MEB; has no effect in ASEMBL).
MEB	no list	Controls listing of macro expansion binary code. A .LIST MEB causes only those macro expansion statements producing binary code to be listed. This is a subset of the ME argument (has no effect in ASEMBL).
CND	list	Controls the listing of unsatisfied conditions and all .IF and .ENDC statements. This argument permits conditional assemblies to be listed without including unsatisfied code.
LD	no list	Controls listing of all listing directives having no arguments (those used to alter the listing level count).
TOC	list	Controls listing of table of contents on pass 1 of the assembly (see Section 5.5.1.4 describing the .SBTTL directive). The full assembly listing is printed during pass 2 of the assembly.
TTM	Terminal mode	Controls listing output format (has no effect in ASEMBL). The TTM argument (the default case) causes output lines to be truncated to 72 characters. Binary code is printed with the binary extensions below the first binary word. The alternative (.NLIST TTM) to Terminal mode is line printer mode, which is shown in Figure 5-2.
SYM	list	Controls the listing of the symbol table for the assembly.

An example of an assembly listing as sent to a 132-column line printer is shown in Figure 5-2. Notice that binary extensions for statements generating more than one word are spread horizontally on the source line. An example of an assembly listing as sent to an 80-column line printer is shown in Figure 5-3 (this is the same format as a terminal listing). Notice that binary extensions for statements generating more than one word are printed on subsequent lines.

Figure 5-4 illustrates a symbol table listing. With the exception of local symbols and macro names, all user-defined symbols are listed in the symbol table. The characters following the symbols listed have special meanings as follows:

=	the symbol is assigned in a direct assignment statement
%	the symbol is a register symbol
R	the symbol is relocatable
G	the symbol is global

The final value of the symbol is expressed in octal. If the symbol is undefined six asterisks are printed in place of the octal number.

CSECT numbers are listed if the symbol is in a named CSECT. All CSECTs are listed at the end of the table with their lengths and corresponding number.

```
RTEXEC  RT-11 MACRO VM02-09   7-SEP-74 17:26:14 PAGE 21
GET PHYSICAL SOURCE LINE

     1                                      .SBTTL  GET PHYSICAL SOURCE LINE
     2
     3           104240                     WINST=EMT+240
     4 001756                        GETPLI:
     5 001756                               $READW  SRC
     6 001766    005000                     CLR     R0
     7 001770    032737  000004  000012'    BIT     #IO.EOF,IOFTBL+SRCCHN   ;END OF FILE?
     8 001776    001424                     BEQ     2$                      ;NO
     9 002000    013700  002362'            MOV     CHAN+SRCCHN,R0          ;GET CURRENT INPUT CHAN
    10 002004    005200                     INC     R0                      ;MOVE TO NEXT CHAN
    11 002006    020027  000010             CMP     R0,#8.                  ;LAST CHAN?
    12 002012    101017                     BHI     1$                      ;YES, FLAG END OF INPUT
    13 002014    005037  000026'            CLR     RECNUM+SRCCHN           ;RESET RECORD (BLK) NUMBER
    14 002020    013737  002310' 002306'    MOV     BLKTBL+<SRCCHN*4>,PTRTBL+<SRCCHN*4>
    15 002026    010037  002362'            MOV     R0,CHAN+SRCCHN
    16 002032    052700  104240             BIS     #WINST,R0               ;CREATE A WAIT CALL FOR NEXT CHAN
    17 002036    010017                     MOV     R0,@PC                  ;AND STORE IN NEXT LOCATION
    18 002040                               .WAIT   0
    19 002042    103403                     BCS     1$                      ;BRANCH IF NO MORE INPUT
    20 002044    012700  177777             MOV     #-1,R0                  ;FLAG END OF FILE
    21 002050                        2$:    RETURN
    22 002052    012700  000001      1$:    MOV     #1,R0                   ;FLAG END OF INPUT
    23 002056                               RETURN
```

Figure 5-2
Example of MACRO Line Printer Listing
(132-column Line Printer)

```
RTEXEC  RT-11 MACRO VM02-09   5-SEP-74 22:30:23 PAGE 21
GET PHYSICAL SOURCE LINE

1                           .SBTTL  GET PHYSICAL SOURCE LINE
2
3          104240           WINST=EMT+240
4 001756           GETPLI:
5 001756                    $READW  SRC
6 001766 005000             CLR     R0
7 001770 032737             BIT     #IO.EOF,IOFTBL+SRCCHN   ;END OF FILE?
         000004
         000012'
8 001776 001424             BEQ     2$               ;NO
9 002000 013700             MOV     CHAN+SRCCHN,R0   ;GET CURRENT INPUT CHAN
         002362'
10 02004 005200             INC     R0               ;MOVE TO NEXT CHAN
11 02006 020027             CMP     R0,#8.           ;LAST CHAN?
         000010
12 02012 101017             BHI     1$               ;YES, FLAG END OF INPUT
13 02014 005037             CLR     RECNUM+SRCCHN    ;RESET RECORD (BLK) NUMBER
         000026'
14 02020 013737             MOV     BLKTBL+<SRCCHN*4>,PTRTBL+<SRCCHN*4>
         002310'
         002306'
15 02026 010037             MOV     R0,CHAN+SRCCHN
         002362'
16 02032 052700             BIS     #WINST,R0        ;CREATE A WAIT CALL FOR NEXT CHA
         104240
17 02036 010017             MOV     R0,@PC           ;AND STORE IN NEXT LOCATION
18 02040                    .WAIT   0
19 02042 103403             BCS     1$               ;BRANCH IF NO MORE INPUT
20 02044 012700             MOV     #-1,R0           ;FLAG END OF FILE
         177777
21 02050           2$:      RETURN
22 02052 012700    1$:      MOV     #1,R0            ;FLAG END OF INPUT
         000001
23 02056                    RETURN
```

Figure 5-3
Example of Page Heading from MACRO 80-Column Line Printer
(same format as Terminal Listing)

```
RTEXEC  RT-11 MACRO VM02-09   5-SEP-74 22:30:23 PAGE 29+
SYMBOL TABLE

ABSEXP= ****** G        ARGCNT= ****** G        ASSEM = ****** G
BINCHN= 000004          BINDAT  002322R    004  BLKTBL= 002310R    004
BPMB  = 000020          BUFTBL  000374RG   003  CHAN    002362R    004
CHNSPC  000312R    003  CHRPNT= ****** G        CLK50 = 000040
CMILEN= 000123          CNTTBL  000360RG   003  CONFIG= 000300
CONT    000040RG   010  CORERR  001726R    010  CPL   = 000120
CR    = 000015          CRFBUF  002076RG   004  CRFC  = 000040
CRFCHN= 000012          CRFCNT  000004RG   007  CRFDAT  002352R    004
CRFE  = 000100          CRFFLG  000000R    007  CRFLEN= 000204
CRFM  = 000010          CRFP  = 000020          CRFPNT  000064R    003
CRFR  = 000004          CRFS  = 000002          CRFSPC  000114R    003
CRFTAB  000026R    003  CRFTST  000002RG   007  CSIERR  000214R    003
CTLTBL  000000R    003  DATE    001000R    010  DATTIM  001004RG   004
DEFEXT  000104R    003  DEVFUL  000252R    003  DIV60   001240R    010
DNC   = ****** G        EDMASK= ****** G        ED.ABS= ****** G
EMTERR= 000052          ENDP1 = ****** G        ENDP2 = ****** G
ENDSWT  000434R    010  ERR     001662R    010  ERRB    000102R    010
ERRBTS= ****** G        ERRCNT= ****** G        EXMFLG= ****** G
FF    = 000014          FILNF   000264R    003  FIN     001434RG   010
FINCL   001636R    010  FINMSG  001030R    004  FINMS1  001052R    004
FINMS2  001070R    004  FINP1   000776R    010  FINP2   000776R    010
FINSML  002124RG   010  FRECOR  000006R    007  GETPLI  001756RG   010
GETR50= ****** G        GSARG = ****** G        HDRTTL  001102RG   004
HIGHAD= 000050          ILLCMD  000226R    003  ILLDEV  000240R    003
IMPURT  000042R    007  IMPUR$  000000R    007  INIOF   000106R    010
        .
        .
        .
RTEXEC  RT-11 MACRO VM02-09   5-SEP-74 22:30:23 PAGE 29+
SYMBOL TABLE

TIME    000210R    003  TIMTIM  001016R    004  TIMWRD  000204R    003
TMPCNT= 000014          TSTSTK  001704RG   010  TTLBRK= ****** G
TTLBUF= ****** G        TTLLEN= 000040          TTYBUF= 000616
USRLOC= 000046          VT    = 000013          WINST = 104240
WRTERR  002306R    010  XBAW  = 000000          XEDPIC= 000000
XMIT0 = ****** G        $CLOUT  003006RG   010  $EDABL= ****** G
$FLUSH  002732RG   010  $NLIST= ****** G        $READ   002422RG   010
$READW  002422RG   010  $WAIT   002730RG   010  $WRITE  002134RG   010
$WRITW  002134RG   010
. ABS.  000000     000
        000000     001
DPURE   000000     002
DPURE$  000410     003
MIXED$  002376     004
SWTSE$  000000     005
SWTSEC  000000     006
IMPUR$  000042     007
MAIN$   003024     010
ERRORS DETECTED: 0
FREE CORE: 13431. WORDS

,LP:/C/L:BEX=RP4:RTPAR,RPARAM,RCIOCH,RTEXEC
```

Figure 5-4
Symbol Table

5.5.1.2 Page Headings - The MACRO Assembler outputs each page in the format shown in Figure 5-3. On the first line of each listing page the assembler prints (from right to left):

1. title taken from .TITLE directive (most recent one encountered)

2. assembler version identification

3. the date and time of day if entered

4. page number

The second line of each listing page contains the subtitle text specified in the last encountered .SBTTL directive.

5.5.1.3 .TITLE - The .TITLE directive is used to print a heading in the output listing and to assign a name to the object module. The heading printed on the first line of each page of the listing is taken from the first 31 characters of the argument in the .TITLE directive. The first six characters (symbol name) of this same line are also used as the name of the object module. These six characters must be Radix-50 characters (any characters beyond the first six are ignored). Non-Radix-50 characters are not acceptable.

For example:

```
.TITLE PROG TO PERFORM DAILY ACCOUNTING
```

causes PROG TO PERFORM DAILY ACCOUNTIN to be printed in the heading for each page and causes the object module of the assembled program to be PROG (this name is distinguished from the filename of the object module specified in the command string to the assembler).

If there is no TITLE statement, the default name assigned to the first object module is:

```
.MAIN.
```

The first tab or space following the .TITLE directive is not considered part of the object module name or header text, although subsequent tabs and spaces are significant.

If there is more than one .TITLE directive, the last .TITLE directive in the program conveys the name of the object module.

5.5.1.4 .SBTTL - The .SBTTL directive is used to provide the elements for a printed table of contents of the assembly listing. The text following the directive is printed as the second line of each of the following assembly listing pages until the next occurrence of a .SBTTL directive.

For example:

```
.SBTTL CONDITIONAL ASSEMBLIES
```

The text:

```
        CONDITIONAL ASSEMBLIES
```

is printed as the second line of each of the following assembly listing pages.

During pass 1 of the assembly process, MACRO automatically prints a table of contents for the listing containing the line sequence number and text of each .SBTTL directive in the program. Such a table of contents is inhibited by specifying the .NLIST TOC directive within the source.

An example of a table of contents is shown in Figure 5-5.

```
.MAIN.  RT-11 MACRO VM02-09   5-SEP-74 22:30:23
TABLE OF CONTENTS

    1- 29        RT-11 MACRO PARAMETER FILE
    1- 37        COMMON PARAMETER FILE
    2-  1                ASSEMBLY OPTIONS
    3-  1                VARIABLE PARAMETERS
    4-  1                GLOBALS
    5-  1                SECTOR INITIALIZATION
    7-  1                SUBROUTINE CALL DEFINITIONS
   10-  1                MISCELLANEOUS MACRO DEFINITIONS
   11-  2        MCIOCH - I/O CHANNEL ASSIGNMENTS
   12-  2        ****EXEC****
   13-  1        PROGRAM START
   14-  1        INIT OUTPUT FILES
   15-  1        SWITCH HANDLERS
   16-  1        END-OF-PASS ROUTINES
   17-  1        SWITCH AND DATE DATA AREAS
   18-  1        INIT OUTPUT FILES (CONTINUED)
   19-  1        FINISH ASSEMBLY AND RESTART
   20-  1        MEMORY MANAGEMENT
   21-  1        GET PHYSICAL SOURCE LINE
   22-  1        SYSTEM MACRO HANDLERS
   23-  1        WRITE ROUTINES
   24-  1        READ ROUTINE
   25-  1        COMMON I/O ROUTINES
   26-  1        MESSAGES
   27-  1        I/O TABLES
   29-  1        FINIS
```

Figure 5-5
Assembly Listing Table of Contents

Table of Contents text is taken from the text of each .SBTTL directive. The associated numbers are the page and line numbers of the .SBTTL directives.

5.5.1.5 .IDENT - The .IDENT directive is not used or supported by the RT-11 system, but is handled by MACRO for compatibility with other systems. .IDENT provides a means of labeling the object module produced as a result of a MACRO assembly. In addition to the name assigned to the object module with the .TITLE directive, a character string (up to six characters, treated like a .RAD50 string) can be specified between paired delimiters. For example:

```
.IDENT /V005A/
```

The character string:

```
V005A
```

is converted to Radix-50 notation and output to the global symbol directory of the object module.

When more than one .IDENT directive is found in a given program, the last .IDENT found determines the symbol which is passed as part of the object module identification.

5.5.1.6 Page Ejection (.PAGE Directive) - There are several means of obtaining a page eject in a MACRO assembly listing:

1. After a line count of 58 lines, MACRO automatically performs a page eject to skip over page perforations on line printer paper and to formulate terminal output into pages.

2. A form feed character used as a line terminator (or as the only character on a line) causes a page eject. Used within a macro definition a form feed character causes a page eject. A page eject is not performed when the macro is invoked.

3. More commonly, the .PAGE directive is used within the source code to perform a page eject at that point. The format of this directive is:

   ```
   .PAGE
   ```

   This directive takes no arguments and causes a skip to the top of the next page.

   Used within a macro definition, the .PAGE is ignored, but the page eject is performed at each invocation of that macro.

5.5.2 Functions: .ENABL and .DSABL Directives

Several functions are provided by MACRO through the .ENABL and .DSABL directives. These directives use 3-character symbolic arguments to designate the desired function and are of the forms:

```
.ENABL arg
.DSABL arg
```

where arg is one of the legal symbolic arguments defined below.

The following list describes the symbolic arguments and their associated functions in the MACRO language:

Symbolic Argument	Function
ABS	Enabling of this function (has no effect in ASEMBL) produces absolute binary output; (i.e., for input to the Paper Tape Software System absolute binary loader using a .BIN extension instead of .OBJ). The default case is .DSABL ABS.
AMA	Enabling of this function directs the assembly of all relative addresses (address mode 67) as absolute addresses (address mode 37). This switch is useful during the debugging phase of program development.
CDR	The statement .ENABL CDR (has no effect in ASEMBL) causes source columns 73 and greater to be treated as comments. This accommodates sequence numbers in card columns 72-80.
FPT	Enabling of this function (has no effect in ASEMBL) causes floating point truncation, rather than rounding as is otherwise performed. .DSABL FPT returns to floating point rounding mode.
LC	Enabling of this function causes the assembler to accept lower case ASCII input instead of converting it to upper case (has no effect in ASEMBL).
LSB	Enable or disable a local symbol block (has no effect in ASEMBL). While a local symbol block is normally entered by encountering a new symbolic label or .CSECT directive, .ENABL LSB forces a local symbol block which is not terminated until a label or .CSECT directive following the .DSABL LSB statement is encountered. The default case is .DSABL LSB.
PNC	The statement .DSABL PNC (has no effect in ASEMBL) inhibits binary output until an .ENABL PNC is encountered. The default case is .ENABL PNC.

An incorrect argument causes the directive containing it to be flagged as an error.

5.5.3 Data Storage Directives

A wide range of data and data types can be generated with the following directives and assembly characters:

```
.BYTE
.WORD
'
"
.ASCII
.ASCIZ
.RAD50
↑B
↑D
↑O
```

These facilities are explained in the following sections.

5.5.3.1 .BYTE - The .BYTE directive is used to generate successive bytes of data. The directive is of the form:

```
.BYTE exp           ;WHICH STORES THE OCTAL
                    ;EQUIVALENT OF THE EXPRESSION
                    ;EXP IN THE NEXT BYTE

.BYTE exp1,exp2,    ;WHICH STORES THE OCTAL
                    ;EQUIVALENTS OF THE LIST OF
                    ;EXPRESSIONS IN SUCCESSIVE BYTES.
```

A legal expression must have an absolute value (or contain a reference to an external symbol) and must result in eight bits or less of data. The 16-bit value of the expression must have a high-order byte (which is truncated) that is either all zeros or all ones. Each operand expression is stored in a byte of the object program. Multiple operands are separated by commas and stored in successive bytes. For example:

```
SAM=5
.=.+410
.BYTE   ↑D48,SAM          ;060 (OCTAL EQUIVALENT OF 48
                          ;DECIMAL) IS STORED IN LOCATION
                          ;411 - 005 IS STORED IN
                          ;LOCATION 411  412?
```

If the high-order byte of the expression equates to a value other than 0 or -1, it is truncated to the low-order eight bits and flagged with a T error code. If the expression is relocatable, an A-type warning flag is given.

At link time it is likely that relocation will result in an expression of more than eight bits, in which case, the Linker prints an error message. For example:

```
            .
            .
            .BYTE   23        ;STORES OCTAL 23 IN NEXT BYTE
B:
            .BYTE   B         ;RELOCATABLE VALUE CAUSES AN "A"
            .                 ;ERROR FLAG
            .
```

Here, X has an absolute value,

```
.GLOBL  X
X=3
.BYTE   X       ;STORES 3 IN NEXT BYTE
```

and can be linked later with another program:

```
.GLOBL  X
.BYTE   X
```

If an operand following the .BYTE directive is null, it is interpreted as a zero. For example (assume assembly begins at relocatable 0):

```
.=.+420
.BYTE   ,,      ;ZEROS ARE STORED IN BYTES
                ;420, 421, AND 422.
```

5.5.3.2 .WORD - The .WORD directive is used to generate successive words of data. The directive is of the form:

```
.WORD exp               ;WHICH STORES THE OCTAL
                        ;EQUIVALENT OF THE EXPRESSON
                        ;EXP IN THE NEXT WORD

.WORD exp1,exp2...      ;WHICH STORES THE OCTAL
                        ;EQUIVALENTS OF THE LIST OF
                        ;EXPRESSIONS IN SUCCESSIVE
                        ;WORDS
```

where a legal expression must result in 16 bits or less of data. Each operand expression is stored in a word of the object program.

Multiple operands are separated by commas and stored in successive words. For example:

```
SAL=0
.=.+500
.WORD   177535,.+4,SAL  ;STORES 177535, 506, AND 0
                        ;IN WORDS 500, 502, AND 504.
```

If an expression equates to a value of more than 16 bits, it is truncated and flagged with a T error code.

If an operand following the .WORD directive is null, it is interpreted as zero. For example:

```
.=.+500
.WORD   ,5,     ;STORES 0,5,0 IN LOCATIONS
                ;500, 502, AND 504
```

A blank operator field (any operator not recognized as a macro call, op-code, directive or semicolon) is interpreted as an implicit .WORD directive. Use of this convention is discouraged. The first term of the first expression in the operand field must not be an instruction mnemonic or assembler directive unless preceded by a + or - operator.

For example:

```
          .=.+440           ;THE OP-CODE FOR MOV, WHICH IS
LABEL:    +MOV,LABEL        ;010000, IS STORED IN LOCATION
                            ;440, 440 IS STORED IN
                            ;LOCATION 442.
```

Note that the default .WORD directive occurs whenever there is a leading arithmetic or logical operator, or whenever a leading symbol is encountered which is not recognized as a macro call, an instruction mnemonic or assembler directive. Therefore, if an instruction mnemonic, macro call, or assembler directive is misspelled, the .WORD directive is assumed and errors will result. Assume that MOV is spelled incorrectly as MOR:

```
        MOR     A,B
```

Two error codes result: A and U. Two words are then generated, one for MOR A and one for B.

5.5.3.3 ASCII Conversion of One or Two Characters - The ' and " characters are used to generate text characters within the source text. A single apostrophe followed by a character results in a term in which the 7-bit ASCII representation of the character is placed in the low-order byte and zero is placed in the high-order byte. For example:

```
        MOV     #'A,R0
```

results in the following 16 bits being moved into R0:

0000000001000001

The ' character is never followed by a carriage return, null, RUBOUT, line feed, or form feed. (For another use of the ' character, see Section 5.6.3.6.)

```
STMNT:
        GETSYM
        BEQ     4$
        CMPB    @CHRPNT,#':         ;COLON DELIMITS LABEL FIELD
        BEQ     LABEL
        CMPB    @CHRPNT,#'=         ;EQUAL DELIMITS
        BEQ     ASGMT               ;ASSIGNMENT PARAMETER
```

A double quote followed by two characters results in a term in which the 7-bit ASCII representations of the two characters are placed. For example:

```
        MOV #"AB,R0
```

results in the following binary word being moved into R0:

0100001001000001

Note that the first character is placed in the low-order byte and the second character in the high-order byte.

The " character is never followed by a carriage return, null, rubout, line feed, or form feed. For example:

```
;DEVICE NAME TABLE

DEVNAM: .WORD   "RF      ;RF DISK
        .WORD   "RK      ;RK DISK
DEVNKB: .WORD   "TT      ;TERMINAL KEYBOARD
        .WORD   "DT      ;DECTAPE
        .WORD   "LP      ;LINE PRINTER
        .WORD   "PR      ;PAPER TAPE READER
        .WORD   "PP      ;PAPER TAPE PUNCH
        .WORD   0        ;TABLE'S END
```

5.5.3.4 .ASCII - The .ASCII directive translates character strings into their 7-bit ASCII equivalents for use in the source program. The format of the .ASCII directive is:

```
.ASCII    /character string/
```

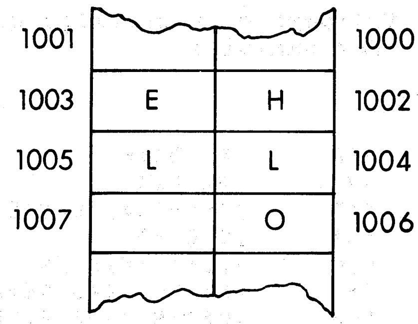
where: character string — is a string of any acceptable printing ASCII characters including spaces. The string may not include null characters, rubout, return, line feed, vertical tab, or form feed. Nonprinting characters can be expressed in digits of the current radix and delimited by angle brackets. (Any legal, defined expression is allowed between angle brackets.)

/ / — are delimiting characters and may be any printing characters other than ; < and = characters and any character within the string.

As an example:

```
A:      .ASCII /HELLO/  ;STORES ASCII REPRESENTATION OF
                        ;THE LETTERS H E L L O IN
                        ;CONSECUTIVE BYTES
```

The order of the characters as they are stored in memory is illustrated below.




```
.ASCII /ABC/<15><12>/DEF/
                    ;STORES
                    ;101,102,103,15,12,104,105,106
                    ;IN CONSECUTIVE BYTES

.ASCII /<AB>/       ;STORES 74,101,102,76 IN
                    ;CONSECUTIVE BYTES
```

The ; and = characters are not illegal delimiting characters, but are preempted by their significance as a comment indicator and assignment operator, respectively. For other than the first group, semicolons are treated as beginning a comment field. For example:

	Directive	Result	Explanation
.ASCII	;ABC;/DEF/	A B C D E F	Acceptable, but not recommended procedure.
.ASCII	/ABC/;DEF;	A B C	;DEF; is treated as a comment and ignored.
.ASCII	/ABC/=DEF=	A B C D E F	Acceptable, but not recommended procedure.
.ASCII	=DEF=		The assignment .ASCII=DEF is performed and an error generated upon encountering the second =.

5.5.3.5 .ASCIZ - The .ASCIZ directive is equivalent to the .ASCII directive with a zero byte automatically inserted as the final character of the string. For example:

When a list or text string has been created with a .ASCIZ directive, a search for the null character can determine the end of the list as follows:

```
CR=15
LF=12
.
.
.
        MOV     #HELLO,R1
```

```
                MOV     #LINBUF,R2
        X:      MOVB    (R1)+,(R2)+        ;MOVE A CHARACTER OF THE
                                           ;MESSAGE STRING INTO THE
                                           ;OUTPUT BUFFER
                BNE X                      ;BRANCH BACK IF BYTE
                                           ;NOT EQUAL TO 0
        .
        .
        .

        HELLO:  .ASCIZ  <CR><LF>/MACRO-11 V001A/<CR><LF>
                                           ;INTRO MESSAGE
```

5.5.3.6 .RAD50 - The .RAD50 directive allows the user the capability to handle symbols in Radix-50 coded form (this form is sometimes referred to as MOD40 and is used in PDP-11 system programs). Radix-50 form allows three characters to be packed into sixteen bits; therefore, any 6-character symbol can be held in two words. The form of the directive is:

.RAD50 /string/

where: / / delimiters can be any printing characters other than the =, <, and ; characters.

string is a list of the characters to be converted (three characters per word) and may consist of the characters A through Z, 0 through 9, dollar ($), dot (.) and space ( ). If there are fewer than three characters (or if the last set is fewer than three characters) they are considered to be left justified and trailing spaces are assumed. Illegal nonprinting characters are replaced with a ? character and cause an I error flag to be set. Illegal printing characters set the Q error flag.

The trailing delimiter may be a carriage return, semicolon, or matching delimiter. (A warning code is printed if it is not a matching delimiter, however.) For example:

```
****** A
20 00040 003223        .RAD50  /ABC
21                                  ;PACK ABC INTO ONE WORD
22 00042 003220        .RAD50  /AB/ ;PACK AB (SPACE) INTO ONE WORD.
23 00044 000000        .RAD50  //   ;PACK THREE SPACES INTO ONE WORD
24 00046 003223        .RAD50  /ABCD/ ;PACK ABC INTO FIRST WORD AND
   00050 014400
                                    ;D (SPACE)(SPACE) INTO SECOND WORD
```

Each character is translated into its Radix-50 equivalent as indicated:

Character	Radix-50 Equivalent (octal)	ASCII (octal)
(space)	0	40
A-Z	1-32	101-132
$	33	44
.	34	56
undefined	35	undefined
0-9	36-47	60-71

Note that another character could be defined for code 35, which is currently unused.

The Radix-50 equivalents for three characters (C1,C2,C3) are combined in one 16-bit word as follows:

    Radix-50 value = ((C1*50)+C2)*50+C3

For example:

    Radix-50 value of ABC is ((1*50)+2)*50+3 or 3223

See Appendix E for a table to quickly determine Radix-50 equivalents.

Use of angle brackets is encouraged in the .ASCII, .ASCIZ, and .RAD50 statements whenever leaving the text string to insert special codes. For example:

```
.ASCII  <101>    ;EQUIVALENT TO .ASCII/A/

.RAD50  /AB/<35>;STORES 3255 IN NEXT WORD.

CHR1=1
CHR2=2
CHR3=3
  .
  .
  .
.RAD50  <CHR1><CHR2><CHR3>       ;EQUIVALENT TO RAD50/ABC/
```

5.5.4 Radix Control

5.5.4.1 .RADIX

Numbers used in a MACRO source program are initially considered to be octal numbers. However, the programmer has the option of declaring the following radices:

    2, 4, 8, 10

This is done via the .RADIX directive of the form:

    .RADIX n

where n is one of the acceptable radices.

The argument to the .RADIX directive is always interpreted in decimal radix. Following any radix directive, that radix is the assumed base for any number specified until the following .RADIX directive.

The default radix at the start of each program, and the argument assumed if none is specified, is 8 (octal). For example:

```
        .RADIX  10              ;BEGINS SECTION OF CODE WITH
                                ;DECIMAL RADIX
          .
          .
          .
        .RADIX                  ;REVERTS TO OCTAL RADIX
```

In general it is recommended that macro definitions not contain or rely on radix settings from the .RADIX directive. The temporary radix control characters should be used within a macro definition. (↑D, ↑O, and ↑B are described in the following section.) A given radix is valid throughout a program until changed. Where a possible conflict exists within a macro definition or in possible future uses of that code module, it is suggested that the user specify values using the temporary radix controls.

5.5.4.2 Temporary Radix Control: ↑D, ↑O, and ↑B - Once the user has specified a radix for a section of code, or has determined to use the default octal radix, he may discover a number of cases where an alternate radix is more convenient (particularly within macro definitions). For example, the creation of a mask word might best be done in the binary radix.

MACRO has three unary operators to provide a single interpretation in a given radix within another radix as follows:

```
↑Dx  (x is treated as being in decimal radix)
↑Ox  (x is treated as being in octal radix)
↑Bx  (x is treated as being in binary radix)
```

For example:

```
↑D123
↑O 47
↑B 00001101
↑O<A+3>
```

Notice that while the uparrow and radix specification characters may not be separated, the radix operator can be physically separated from the number by spaces or tabs for formatting purposes. Where a term or expression is to be interpreted in another radix, it should be enclosed in angle brackets.

These numeric quantities may be used any place where a numeric value is legal.

A temporary radix change from octal to decimal may be made by specifying a decimal radix number with a "decimal point". For example:

```
     100.     (144(8))
    1376.     (2540(8))
     128.     (200(8))
```

5.5.5 Location Counter Control

The four directives that control movement of the location counter are .EVEN and .ODD which move the counter a maximum of one byte, and .BLKB and .BLKW which allow the user to specify blocks of a given number of bytes or words to be skipped in the assembly.

5.5.5.1 .EVEN -

The .EVEN directive ensures that the assembly location counter contains an even memory address by adding one if the current address is odd. If the assembly location counter is even, no action is taken. Any operands following a .EVEN directive are ignored.

The .EVEN directive is used as follows:

```
.ASCIZ /THIS IS A TEST/
.EVEN                       ;ASSURES NEXT STATEMENT
                            ;BEGINS ON A WORD BOUNDARY
.WORD XYZ
```

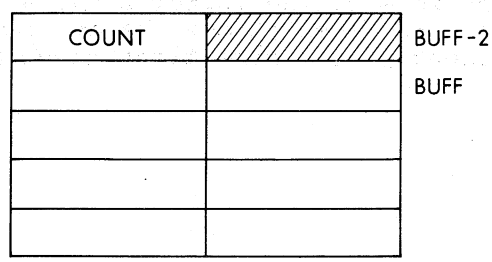
5.5.5.2 .ODD -

The .ODD directive ensures that the assembly location counter is odd by adding one if it is even. For example:

```
;CODE TO MOVE DATA FROM AN INPUT LINE
;TO A BUFFER

        N=5                 ;BUFFER HAS 5 WORDS
        .
        .
        .
        .ODD
        .BYTE N*2           ;COUNT=2N BYTES
BUFF:   .BLKW N             ;RESERVE BUFFER OF N WORDS
        .
        .
        MOV #BUFF,R2        ;ADDRESS OF EMPTY BUFFER IN R2
        MOV #LINE,R1        ;ADDRESS OF INPUT LINE IS IN R1
        MOVB -1(R2),R0      ;GET COUNT STORED IN BUFF-1 IN R0
AGAIN:  MOVB (R1)+,(R2)+            ;MOVE BYTE FROM LINE INTO BUFFER
        BEQ DONE            ;WAS NULL CHARACTER SEEN?
        DEC R0              ;DECREMENT COUNT
        BNE AGAIN           ;NOT=0, GET NEXT CHARACTER
        .
        .
        CLRB -(R2)          ;OUT OF ROOM IN BUFFER, CLEAR LAST
DONE:                       ;WORD
        .
        .
LINE:   .ASCIZ /TEXT/
```

In this case, .ODD is used to place the buffer byte count in the byte preceding the buffer, as follows:



5.5.5.3 .BLKB and .BLKW - Blocks of storage can be reserved using the .BLKB and .BLKW directives. .BLKB is used to reserve byte blocks and .BLKW reserves word blocks. The two directives are of the form:

```
.BLKB     exp
.BLKW     exp
```

where exp is the number of bytes or words to reserve. If no argument is present, 1 is the assumed default value. Any legal expression which is completely defined at assembly time and produces an absolute number is legal. For example:

```

                    ;

          000000'           .CSECT IMPURE
  000000          PASS:     .BLKW
                                            ;NEXT GROUP MUST STAY TOGETHER
  000002          SYMBOL:   .BLKW 2         ;SYMBOL ACCUMULATOR
  000006          MODE:
  000006          FLAGS:    .BLKB 1         ;FLAG BITS
   00007          SECTOR:   .BLKB 1         ;SYMBOL/EXPRESSION TYPE
   00010          VALUE:    .BLKW 1         ;EXPRESSION VALUE
   00012          RELLVL:   .BLKW 1
                            .BLKW 2         ;END OF GROUPED DATA
   00020          CLCNAM:   .BLKW 2         ;CURRENT LOCATION COUNTER
   00024          CLCFGS:   .BLKB 1
   00025          CLCSEC:   .BLKB 1
   00026          CLCLOC:   .BLKW 1
   00030          CLCMAX:   .BLKW 1
          000001'           .END
```

The .BLKB directive has the same effect as:

```
.=.+exp
```

but is easier to interpret in the context of source code.

5.5.6 Numeric Control

Several directives are available to provide software complements to the floating-point hardware on the PDP-11.

A floating-point number is represented by a string of decimal digits. The string (which can be a single digit in length) may optionally contain a decimal point, and may be followed by an optional exponent indicator in the form of the letter E and a signed decimal exponent. The list of number representations below contains seven distinct, valid representations of the same floating-point number:

```
3
3.
3.0
3.0E0
3E0
.3E1
300E-2
```

As can be quickly inferred, the list could be extended indefinitely (e.g., 3000E-3, .03E2, etc.). A leading plus sign is ignored (e.g., +3.0 is considered to be 3.0). Leading minus signs complement the sign bit. No other operators are allowed (e.g., 3.0+N is illegal).

Floating-point number representations are valid only in the contexts described in the remainder of this section.

Floating-point numbers are normally rounded. That is, when a floating-point number exceeds the limits of the field in which it is to be stored, the high-order excess bit is added to the low-order retained bit. For example, if the number were to be stored in a 2-word field, but more than 32 bits were needed for its value, the highest bit carried out of the field would be added to the least significant position. In order to enable floating-point truncation, the .ENABL FPT directive is used and .DSABL FPT is used to return to floating-point rounding.

5.5.6.1 .FLT2 and .FLT4 - Like the .WORD directive, the two floating-point storage directives cause their arguments to be stored in-line with the source program (have no effect in ASEMBL). These two directives are of the form:

```
.FLT2     arg1,arg2,...
.FLT4     arg1,arg2,...
```

where arg1,arg2, etc. represent one or more floating point numbers separated by commas.

.FLT2 causes two words of storage to be generated for each argument while .FLT4 generates four words of storage.

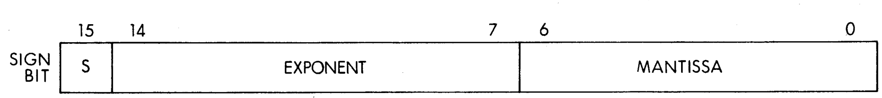
The following code shows the use of the .FLT4 directive:

```
000000 037314 ATOFTB: .FLT4 1.E-1      ;10^-1
000002 146314
000004 146314
000006 146315
000010 036443         .FLT4 1.E-2      ;10^-2
000012 153412
000014 036560
000016 121727
000020 034721         .FLT4 1.E-4      ;10^-4
000022 133427
000024 054342
000026 014545
000030 031453         .FLT4 1.E-8      ;10^-8
000032 146167
000034 010604
000036 060717
000040 022746         .FLT4 1.E-16     ;10^-16
000042 112624
000044 137304
000046 046741
000050 005517         .FLT4 1.E-32     ;10^-32
000052 130436
000054 126505
000056 034625
```

5.5.6.2 Temporary Numeric Control: ↑F and ↑C - Like the temporary radix control operators, operators are available to specify either a 1-word floating-point number (↑F--not available in ASEMBL) or the one's complement of a 1-word number (↑C). For example:

```
FL3.7:    ↑F3.7
```

creates a 1-word floating-point number at location FL3.7 containing the value 3.7 as follows:



This 1-word floating-point number is the first word of the 2- or 4-word floating-point number format shown in the PDP-11 Processor Handbook, and the statement:

```
CMP151:   ↑C151
```

stores the one's complement of 151 in the current radix (assume current radix is octal) as follows:

15	0
	177626

Since these control operators are unary operators, their arguments may be integer constants or symbols, and the operators may be expressed successively. For example:

```
↑C↑D25     or    ↑C31       or    177746
```

The term created by the unary operator and its argument is then a term which can be used by itself or in an expression. For example:

```
↑C2+6
```

is equivalent to:

```
<↑C2>+6    or    177775+6   or    000003
```

For this reason, the use of angle brackets is advised. Expressions used as terms or arguments of a unary operator must be explicitly grouped.

An example of the importance of ordering with respect to unary operators is shown below:

```
 ↑F1.0     = 040200
 ↑F-1.0    = 140200
-↑F1.0     = 137600
-↑F-1.0    = 037600
```

The argument to the ↑F operator must not be an expression and should be of the same format as arguments to the .FLT2 and .FLT4 directives.

5.5.7 Terminating Directives

5.5.7.1 .END - The .END directive indicates the physical end of the source program. The .END directive is of the form:

```
.END exp
```

where exp is an optional argument which, if present, indicates the program entry point, i.e., the transfer address.

When the load module is loaded, program execution begins at the transfer address indicated by the .END directive. In a runtime system (the load module output of the Linker) a .END exp statement should terminate the first object module and .END statements should terminate any other object modules.

5.5.7.2 .EOT - Under the RT-11 System, the .EOT directive is ignored. The physical end file allows several physically separate tapes to be assembled as one program.

5.5.8 Program Boundaries Directive: .LIMIT

The .LIMIT directive reserves two words into which the Linker puts the low and high addresses of the load module's relocatable code (the load module is the result of the link). The low address (inserted into the first word) is the address of the first byte of code. The high address is the address of the first free byte following the relocated code. These addresses are always even since all relocatable sections are loaded at even addresses. (If a relocatable section consists of an odd number of bytes, the Linker adds one to the size to make it even.)

5.5.9 Program Section Directives

The assembler provides for 255(10) program sections: an absolute section declared by .ASECT, an unnamed relocatable program section declared by .CSECT, and 253(10) named relocatable program sections declared by .CSECT symbol, where symbol is any legal symbolic name. These directives allow the user to:

1. Create his program (object module) in sections:

   The assembler maintains separate location counters for each section. This allows the user to write statements which are not physically contiguous but will be loaded contiguously. The following examples will clarify this:

```
        .CSECT            ;START THE UNNAMED RELOCATABLE SECTION
A:      0                 ;ASSEMBLED AT RELOCATABLE 0,
B:      0                 ;RELOCATABLE 2 AND
C:      0                 ;RELOCATABLE 4
ST:     CLR A             ;ASSEMBLE CODE AT
        CLR B             ;RELOCATABLE ADDRESS
        CLR C             ;6 THROUGH 21
        .ASECT            ;START THE ABSOLUTE SECTION
        .=4               ;ASSEMBLE CODE AT
        .WORD .+2,HALT    ;ABSOLUTE 4 THROUGH 7
        .CSECT            ;RESUME THE UNNAMED RELOCATABLE
                          ;SECTION
        INC A             ;ASSEMBLE CODE AT
        BR ST             ;RELOCATABLE 22 THROUGH 27
        .END
```

The first appearance of .CSECT or .ASECT assumes the location counter is at relocatable or absolute zero, respectively. The scope of each directive extends until a directive to the contrary is given. Further occurrences of the same .CSECT or .ASECT resume assembling where the section was left off.

```
        .CSECT COM1              ;DECLARE SECTION COM1
A:      0                        ;ASSEMBLED AT RELOCATABLE 0
B:      0                        ;ASSEMBLED AT RELOCATABLE 2
C:      0                        ;ASSEMBLED AT RELOCATABLE 4
        .CSECT COM2              ;DECLARE SECTION COM2
X:      0                        ;ASSEMBLED AT RELOCATABLE 0
Y:      0                        ;ASSEMBLED AT RELOCATABLE 2
        .CSECT COM1              ;RETURN TO COM1
D:      0                        ;ASSEMBLED AT RELOCATABLE 6
        .END
```

The assembler automatically begins assembling at relocatable zero of the unnamed .CSECT if not instructed otherwise; that is, the first statement of an assembly is an implied .CSECT.

All labels in an absolute section are absolute; all labels in a relocatable section are relocatable. The location counter symbol, ".", is relocatable or absolute when referenced in a relocatable or absolute section, respectively. Undefined internal symbols are assigned the value of relocatable or absolute zero in a relocatable or absolute section, respectively. Any labels appearing on a .ASECT or .CSECT statement are assigned the value of the location counter before the .ASECT or .CSECT takes effect. Thus, if the first statement of a program is:

```
    A:      .ASECT
```

then A is assigned to relocatable zero and is associated with the unnamed relocatable section (because the assembler implicitly begins assembly in the unnamed relocatable section).

Since it is not known at assembly time where the program sections are to be loaded, all references between sections in a single assembly are translated by the assembler to references relative to the base of that section. The assembler provides the Linker with the necessary information to resolve the linkage. Note that this is not necessary when making a reference to an absolute section (the assembler knows all load addresses of an absolute section).

Examples:

```
        .ASECT
        .=1000
A:      CLR X           ;ASSEMBLED AS CLR BASE OF UNNAMED
                        ;RELOCATABLE SECTION +12
        JMP Y           ;ASSEMBLED AS JMP BASE OF UNNAMED
                        ;RELOCATABLE SECTION + 10

        .CSECT
        MOV R0,R1
        JMP A           ;ASSEMBLED AS JMP 1000
Y:      HALT
X:      0
        .END
```

In the above example the references to X and Y were translated into references relative to the base of the unnamed relocatable section.

2. Share code and/or data between object modules (separate assemblies):

   Named relocatable program sections operate as FORTRAN labeled COMMON; that is, sections of the same name from different assemblies are all loaded at the same location by LINK. The unnamed relocatable section is the exception to this as all unnamed relocatable sections are loaded in unique areas by LINK.

   Note that there is no conflict between internal symbolic names and program section names; that is, it is legal to use the same symbolic name for both purposes. In fact, considering FORTRAN again, this is necessary to accommodate the FORTRAN statement:

   ```
   COMMON    /X/A,B,C,X
   ```

   where the symbol X represents the base of this program section and also the fourth element of this program section.

   Program section names should not duplicate .GLOBL names. In FORTRAN language, COMMON block names and SUBROUTINE names should not be the same.

The .ASECT and .CSECT program section directives are provided in MACRO to allow the user to specify an unnamed absolute or relocatable section. These directives are formatted as follows:

```
.ASECT
.CSECT
.CSECT  symbol
```

The single absolute section can be declared with an:

    .ASECT

directive. No name can be associated with the absolute section specified by means of the .ASECT directive. The single unnamed relocatable program section can be declared with a:

    .CSECT

directive.

All named relocatable sections are loaded in unique areas by LINK. Up to 253(10) named relocatable program sections can be declared with:

    .CSECT symbol

directives, where symbol is any legal symbolic name.

The assembler automatically begins assembling at relocatable zero of the unnamed .CSECT if not instructed otherwise; that is, the first statement of an assembly is an implied .CSECT.

5.5.10 Symbol Control: .GLOBL

If a program is created in segments which are assembled separately, global symbols are used to allow reference to one symbol by the different segments.

A global symbol must be declared in a .GLOBL directive. The form of the .GLOBL directive is:

    .GLOBL   sym1,sym2,...

where:

sym1,sym2, etc. — are legal symbolic names, separated by commas, tabs, or spaces where more than one symbol is specified.

Symbols appearing in a .GLOBL directive are either defined within the current program or are external symbols, in which case they are defined in another program which is to be linked with the current program, by LINK, prior to execution.

A .GLOBL directive line may contain a label in the label field and comments in the comment field.

```
;DEFINE A SUBROUTINE WITH 2 ENTRY POINTS WHICH
;CALLS AN EXTERNAL SUBROUTINE

        .CSECT          ;DECLARE THE CONTROL SECTION
        .GLOBL  A,B,C   ;DECLARE A,B,C AS GLOBALS
A:      MOV     @(R5)+,R0       ;ENTRY A IS DEFINED
        MOV     #X,R1
X:      JSR     PC,C    ;CALL EXTERNAL SUBROUTINE C
        RTS     R5      ;EXIT
B:      MOV     @(R5)+,R1       ;DEFINE ENTRY B
        CLR     R1
        BR      X
```

In the previous example, A and B are entry symbols (entry points), C is an external symbol and X is an internal symbol.

A global symbol is defined only when it appears in a .GLOBL directive. A symbol is not considered a global symbol if it is assigned the value of a global expression in a direct assignment statement.

References to external symbols can appear in the operand field of an instruction or assembler directive in the form of a direct reference, i.e.:

```
CLR     EXT
.WORD   EXT
CLR     @EXT
```

or a direct reference plus or minus a constant, i.e.:

```
D=6
CLR     EXT+D
.WORD   EXT-2
CLR     @EXT+D
```

A global symbol defined within the program can be used in the evaluation of a direct assignment statement, but an external symbol cannot. Since MACRO determines at the end of pass 1 whether a given global symbol is defined within the program or is expected to be external, a construction such as the following will cause errors at link time:

```
        .GLOBL FREE
        .GLOBL LIMITS
FREE=LIMITS+2              ;FREE WILL NOT BE
        .                  ;DEFINED UNTIL PASS 2
        .
        .


LIMITS: .LIMIT
        .
        .
        .
```

FREE will be flagged as an undefined global at link time. To allow correct linking, define FREE after LIMITS:.

5.5.11 Conditional Assembly Directives

Conditional assembly directives provide the programmer with the capability to conditionally include or ignore blocks of source code in the assembly process. This technique is used extensively to allow several variations of a program to be generated from the source program.

The general form of a conditional block is as follows:

```
.IF   cond,argument(s)     ;START CONDITIONAL BLOCK
        .                  ;STATEMENTS IN RANGE OF
        .                  ;CONDITIONAL
        .                  ;BLOCK
.ENDC                      ;END CONDITIONAL BLOCK
```

where: cond — is a condition which must be met if the block is to be included in the assembly. These conditions are defined below.

argument(s)	are a function of the condition to be tested. If more than one argument is specified, they must be separated by commas.
range	is the body of code which is included in the assembly or ignored depending upon whether the condition is met.

The following are the allowable conditions:

Conditions Positive	Complement	Arguments	Assemble Block If
EQ	NE	expression	expression=0 (or $\neq$ 0)
GT	LE	expression	expression>0 (or $\leq$ 0)
LT	GE	expression	expression<0 (or $\geq$ 0)
DF	NDF	symbolic argument	symbol is defined (or undefined)
B	NB	macro-type argument	argument is blank (or nonblank)
IDN	DIF	two macro-type arguments separated by a comma	arguments identical (or different)
Z	NZ	expression	same as EQ/NE
G		expression	same as GT/LE
L		expression	same as LT/GE

IF DIF and IF IDN are not available in ASEMBL.

NOTE

A macro-type argument is enclosed in angle brackets or within an up-arrow construction (as described in Section 5.2.1.1). For example:

```
<A,B,C>
↑/124/
```

For example:

```
ALPHA==1
        .IF     EQ,ALPHA+1      ;ASSEMBLE IF ALPHA+1=0
        .
        .
        .ENDC
```

Within the conditions DF and NDF the following two operators are allowed to group symbolic arguments:

&	logical AND operator
!	logical inclusive OR operator

For example:

```
.IF     DF,SYM1 & SYM2  ;ASSEMBLE IF BOTH SYM1
 .                      ;AND SYM2 ARE DEFINED
 .
.ENDC
```

5.5.11.1 Subconditionals - Subconditionals may be placed within conditional blocks to indicate:

1. assembly of an alternate body of code when the condition of the block indicates that the code within the block is not to be assembled,

2. assembly of a non-contiguous body of code within the conditional block depending upon the result of the conditional test to enter the block,

3. unconditional assembly of a body of code within a conditional block.

There are three subconditional directives, as follows:

Subconditional	Function
.IFF	The code following this statement up to the next subconditional or end of the conditional block is included in the program if the value of the condition tested upon entering the conditional block is false.
.IFT	The code following this statement up to the next subconditional or end of the conditional block is included in the program if the value of the condition tested upon entering the conditional block is true.
.IFTF	The code following this statement up to the next subconditional or the end of the conditional block is included in the program regardless of the value of the condition tested upon entering the conditional block.

The implied argument of the subconditionals is the value of the condition upon entering the conditional block. Subconditionals are used within outer level conditional blocks. Subconditionals are ignored within nested, unsatisfied conditional blocks.

For example:

```
        .IF     DF,SYM  ;ASSEMBLE BLOCK IF SYM IS DEFINED
        .IFF            ;ASSEMBLE THE FOLLOWING CODE ONLY IF
                        ;SYM IS UNDEFINED
         .
         .
        .IFT            ;ASSEMBLE THE FOLLOWING CODE ONLY IF
                        ;SYM  IS DEFINED
         .
         .
        .IFTF           ;ASSEMBLE THE FOLLOWING CODE
                        ;UNCONDITIONALLY
         .
         .
         .
        .ENDC

        .IF     DF,X    ;ASSEMBLY TESTS FALSE
        .IF     DF,Y    ;TESTS FALSE
        .IFF            ;NESTED CONDITIONAL
                        ;IGNORED WITHIN NESTED, UNSATISFIED
         .              ;CONDITIONAL BLOCK
         .
         .
        .IFT            ;NOT SEEN
         .
         .
        .ENDC
        .ENDC
```

However,

```
        .IF     DF,X    ;TESTS TRUE
        .IF     DF,Y    ;TESTS FALSE
        .IFF            ;IS ASSEMBLED
                        ;OUTER CONDITIONAL SATISFIED.
         .
         .
        .IFT            ;NOT ASSEMBLED
         .
         .
        .ENDC
        .ENDC
```

5.5.11.2 Immediate Conditionals - An immediate conditional directive is a means of writing a 1-line conditional block. In this form, no .ENDC statement is required and the condition is completely expressed on the line containing the conditional directive. Immediate conditions are of the form:

.IIF cond, arg, statement

where: cond   is one of the legal conditions defined for conditional blocks in Section 5.5.11.

arg — is the argument associated with the conditional specified, that is, either an expression, symbol, or macro-type argument, as described in Section 5.5.11.

statement — is the statement to be executed if the condition is met.

For example:

```
.IIF    DF,FOO,BEQ ALPHA
```

This statement generates the code:

```
BEQ    ALPHA
```

if the symbol FOO is defined.

A label must not be placed in the label field of the .IIF statement. Any necessary labels may be placed on the previous line, as in the following example:

```
LABEL:
        .IIF    DF,FPP  BEQ ALPHA
```

or included as part of the conditional statement:

```
.IIF    DF,FOO  LABEL:  BEQ ALPHA
```

5.5.11.3 PAL-11R and PAL-11S Conditional Assembly Directives - In order to maintain compatibility with programs developed under PAL-11R and PAL-11S, the following conditionals remain permissible under MACRO. It is advisable that future programs be developed using the format for MACRO conditional assembly directives.

Directive	Arguments	Assemble Block if
.IFZ or .IFEQ	expression	expression=0
.IFNZ or.IFNE	expression	expression≠0
.IFL or .IFLT	expression	expression<0
.IFG or .IFGT	expression	expression>0
.IFGE	expression	expression=>0
.IFLE	expression	expression<=0
.IFDF	logical expression	expression is true(defined)
.IFNDF	logical expression	expression is false(undefined)

The rules governing the usage of these directives are now the same as for the MACRO conditional assembly directives previously described. Conditional assembly blocks must end with the .ENDC directive and are limited to a nesting depth of 16(10) levels (instead of the 127(10) levels allowed under PAL-11R).

5.6 MACRO DIRECTIVES

5.6.1 Macro Definition

It is often convenient in assembly language programming to generate a recurring coding sequence with a single statement. In order to do this, the desired coding sequence is first defined with dummy arguments as a macro. Once a macro has been defined, a single statement calling the macro by name with a list of real arguments (replacing the corresponding dummy arguments in the definition) generates the correct sequence or expansion.

5.6.1.1 .MACRO - The first statement of a macro definition must be a .MACRO directive (not available in ASEMBL). The .MACRO directive is of the form:

.MACRO name, dummy argument list

where:

name	is the name of the macro. This name is any legal symbol. The name chosen may be used as a label elsewhere in the program.
,	represents any legal separator (generally a comma or space).
dummy argument list	zero, one, or more legal symbols which may appear anywhere in the body of the macro definition, even as a label. These symbols can be used elsewhere in the user program with no conflicts of definition. Where more than one dummy argument is used, they are separated by any legal separator (generally a comma).

A comment may follow the dummy argument list in a statement containing a .MACRO directive. For example:

```
        .MACRO  ABS     A,B     ;DEFINE MACRO ABS WITH TWO ARGUMENTS
```

A label must not appear on a .MACRO statement. Labels are sometimes used on macro calls, but serve no function when attached to .MACRO statements.

5.6.1.2 .ENDM - The final statement of every macro definition must be an .ENDM directive (not available in ASEMBL) of the form:

.ENDM name

where name is an optional argument and is the name of the macro being terminated by the statement.

For example:

```
.ENDM          (terminates the current macro definition)

.ENDM ABS      (terminates the definition of the macro ABS)
```

If specified, the symbolic name in the .ENDM statement must correspond to that in the matching .MACRO statement. Otherwise the statement is flagged and processing continues. Specification of the macro name in the .ENDM statement permits the assembler to detect missing .ENDM statements or improperly nested macro definitions.

The .ENDM statement may contain a comment field, but must not contain a label.

An example of a macro definition is shown below:

```
.MACRO  TYPMSG  MESSGE  ;TYPE A MESSAGE
JSR     R5,TYPMSG
.WORD   MESSGE
.ENDM
```

5.6.1.3 .MEXIT - In order to implement alternate exit points from a macro (particularly nested macros), the .MEXIT directive is provided. .MEXIT (not available in ASEMBL) terminates the current macro as though an .ENDM directive were encountered. Use of .MEXIT bypasses the complications of conditional nesting and alternate paths. For example:

```
.MACRO  ALTR    N,A,B
 .
 .
.IF     EQ,N    ;START CONDITIONAL BLOCK
 .
 .
.MEXIT          ;EXIT FROM MACRO DURING CONDITIONAL
                ;BLOCK
.ENDC           ;END CONDITIONAL BLOCK
 .
 .
.ENDM           ;NORMAL END OF MACRO
```

In an assembly where N=0, the .MEXIT directive terminates the macro expansion.

Where macros are nested, a .MEXIT causes an exit to the next higher level. A .MEXIT encountered outside a macro definition is flagged as an error.

5.6.1.4 Macro Definition Formatting - A form feed character used as a line terminator in a macro source statement (or as the only character on a line), causes a page eject when the source program is listed. Used within a macro definition, a form feed character also causes a page eject. A page eject is not performed, however, when the macro is invoked.

Used within a macro definition, the .PAGE directive is ignored, but a page eject is performed at invocation of that macro.

5.6.2 Macro Calls

A macro must be defined prior to its first reference. Macro calls are of the general form:

label: name, real arguments

where:

label — represents an optional statement label.

name — represents the name of the macro specified in the .MACRO directive preceding the macro definition.

, — represents any legal separator (comma, space, or tab). No separator is necessary where there are no real arguments. (Refer to Section 5.2.1.1.)

real arguments — are those symbols, expressions, and values which replace the dummy arguments in the .MACRO statement. Where more than one argument is used, they are separated by any legal separator.

Where a macro name is the same as a user label, the appearance of the symbol in the operation field designates a macro call, and the occurrence of the symbol in the operand field designates a label reference. For example:

```
ABS:    MOV     @R0,R1          ;ABS IS USED AS A LABEL
        .
        .
        BR      ABS             ;ABS IS CONSIDERED A LABEL
        .
        .
        ABS     #4,ENT,LAR      ;CALL MACRO WITH 3 ARGUMENTS
```

Arguments to the macro call are treated as character strings whose usage is determined by the macro definition.

5.6.3 Arguments to Macro Calls and Definitions

Arguments within a macro definition or macro call are separated from other arguments by any of the separating characters described in Section 5.2.1.1. For example:

```
.MACRO  REN     A,B,C   ;MACRO DEFINITION
.
.
.
REN     ALPHA,BETA,<C1,C2>      ;MACRO CALL
```

Arguments which contain separating characters are enclosed in paired angle brackets. An up-arrow construction is provided to allow angle brackets to be passed as arguments.

For example:

```
REN     <MOV    X,Y>,#44,WEV
```

This call would cause the entire statement:

```
MOV     X,Y
```

to replace all occurrences of the symbol A in the macro definition. Real arguments within a macro call are considered to be character strings and are treated as a single entity until their use in the macro expansion.

The up-arrow construction could have been used in the above macro call as follows:

```
REN     ^/MOV   X,Y/,#44,WEV
```

which is equivalent to:

```
REN     <MOV    X,Y>,#44,WEV
```

Since spaces are ignored preceding an argument, they can be used to increase legibility of bracketed constructions.

5.6.3.1 Macro Nesting - Macro nesting (nested macro calls), where the expansion of one macro includes a call to another macro, causes one set of angle brackets to be removed from an argument with each nesting level. The depth of nesting allowed is dependent upon the amount of memory space used by the program. To pass an argument containing legal argument delimiters to nested macros, the argument should be enclosed in one set of angle brackets for each level of nesting, as shown below:

```
R0=%0
R1=%1
X=10

        .MACRO  LEVEL1  DUM1,DUM2
        LEVEL2  DUM1
        LEVEL2  DUM2
        .ENDM
        .MACRO  LEVEL2  DUM3
        DUM3
        ADD     #10,R0
        MOV     R0,(R1)+
        .ENDM
```

A call to the LEVEL1 macro:

```
LEVEL1  <<MOV   X,R0>>,<<CLR    R0>>
```

causes the following expansion:

```
MOV     X,R0

ADD     #10,R0

MOV     R0,(R1)+
CLR     R0
ADD     #10,R0

MOV     R0,(R1)+
```

Where macro definitions are nested (that is, a macro definition is entirely contained within the definition of another macro) the inner definition is not defined as a callable macro until the outer macro has been called and expanded. For example:

```
.MACRO  LV1     A,B
 .
 .
.MACRO  LV2     A
 .
 .
.ENDM
.ENDM
```

The LV2 macro cannot be called by name until after the first call to the LV1 macro. Likewise, any macro defined within the LV2 macro definition cannot be referenced directly until LV2 has been called.

5.6.3.2 Special Characters - Arguments may include special characters without enclosing the argument in a bracket construction if that argument does not contain spaces, tabs, semicolons, or commas. For example:

```
.MACRO  PUSH    ARG
MOV     ARG,-(SP)
.ENDM
 .
 .
 .
PUSH    X+3(%2)
```

generates the following code:

```
MOV     X+3(%2),-(SP)
```

5.6.3.3 Numeric Arguments Passed as Symbols - When passing macro arguments, a useful capability is to pass a symbol which can be treated by the macro as a numeric string. An argument preceded by the unary operator backslash (\) is treated as a number in the current radix. (\ is not available in ASEMBL.) The ASCII characters representing the number are inserted in the macro expansion; their function is defined in context (see Section 5.6.3.6 for an explanation of single-quote usage). For example:

```
                .MACRO   CNT      A,B
        A'B:    .WORD
                .ENDM
                C=0
                .MACRO   INC      A,B
                CNT      A,\B
                B=B+1
                .ENDM
                 .
                 .
                INC      X,C
```

The macro call would expand to:

```
        X0:     .WORD
```

A subsequent identical call to the same macro would generate:

```
        X1:     .WORD
```

and so on for later calls. The two macros are necessary because the dummy value of B cannot be updated in the CNT macro. In the CNT macro, the number passed is treated as a string argument. (Where the value of the real argument is 0, a single 0 character is passed to the the macro expansion.)

The number being passed can also be used to make source listings somewhat clearer. For example, versions of programs created through conditional assembly of a single source can identify themselves as follows:

```
        .MACRO  IOT     SYM      ;ASSUME THAT THE SYMBOL ID TAKES
        .IDENT  /SYM/            ;ON A UNIQUE 2 DIGIT VALUE FOR
        .ENDM   IOT              ;EACH POSSIBLE CONDITIONAL
                                 ;ASSEMBLY OF THE PROGRAM
        .MACRO  OUT     ARG
        IOT     005A'ARG
        .ENDM
         .
         .
         .
        OUT     \ID              ;WHERE 005A IS THE UPDATE VERSION
                                 ;OF THE PROGRAM AND ARG INDICATES
                                 ;THE CONDITIONAL ASSEMBLY VERSION
```

The above macro call expands to:

```
        .IDENT   /005AXX/
```

where XX is the conditional value of ID.

Two macros are necessary since the text delimiting characters in the .IDENT statement would inhibit the concatenation of a dummy argument.

5.6.3.4 Number of Arguments - If more arguments appear in the macro call than in the macro definition, the excess arguments are ignored. If fewer arguments appear in the macro call than in the definition, missing arguments are assumed to be null (consist of no characters). The conditional directives .IF B and .IF NB can be used within the macro to detect null arguments.

A macro can be defined with no arguments.

5.6.3.5 Automatically Created Symbols Within User-Defined Macros - MACRO can be made to create symbols of the form n$ where n is a decimal integer number such that 64<=n<=127. Created symbols are always local symbols between 64$ and 127$. Such local symbols are created by the assembler in numerical order, i.e.:

```
64$
65$
 .
 .
 .
126$
127$
```

Created symbols are particularly useful where a label is required in the expanded macro. Such a label must otherwise be explicitly stated as an argument with each macro call or the same label is generated with each expansion (resulting in a multiply-defined label). Unless a label is referenced from outside the macro, there is no reason for the programmer to be concerned with that label.

The range of these local symbols extends between two explicit labels. Each new explicit label causes a new local symbol block to be initialized.

The macro processor creates a local symbol on each call of a macro whose definition contains a dummy argument preceded by the ? character. For example:

```
        .MACRO  ALPHA   A,?B
        TST     A
        BEQ     B
        ADD     #5,A
B:
        .ENDM
```

Local symbols are generated only where the real argument of the macro call is either null or missing. If a real argument is specified in the macro call, the generation of a local symbol is inhibited and normal replacement is performed. Consider the following expansions of the macro ALPHA above.

Generate a local symbol for missing argument:

```
                    ALPHA   %1
                    TST     %1
                    BEQ     64$
                    ADD     #5,%1
        64$:
```

Do not generate a local symbol:

```
                    ALPHA   %2,XYZ
                    TST     %2
                    BEQ     XYZ
                    ADD     #5,%2
        XYZ:
```

These assembler-generated symbols are restricted to the first sixteen (decimal) arguments of a macro definition.

5.6.3.6 Concatenation - The apostrophe or single quote character (') operates as a legal separating character in macro definitions. An ' character which precedes and/or follows a dummy argument in a macro definition is removed, and the substitution of the real argument occurs at that point. For example:

```
                    .MACRO  DEF     A,B,C
        A'B:        .ASCIZ  /C/
                    .WORD   ''A'''B
                    .ENDM
```

When this macro is called:

```
        DEF     X,Y,<MACRO-11>
```

it expands as follows:

```
        XY:         .ASCIZ  /MACRO-11/
                    .WORD   'X'Y
```

In the macro definition, the scan terminates upon finding the first ' character. Since A is a dummy argument, the ' is removed. The scan resumes with B, notes B as another dummy argument and concatenates the two dummy arguments. The third dummy argument is noted as going into the operand of the .ASCIZ directive. On the next line (this example is for purely illustrative purposes) the argument to .WORD is seen as follows: The scan begins with a ' character. Since it is neither preceded nor followed by a dummy argument, the ' character remains in the macro definition. The scan then encounters the second ' character which is followed by a dummy argument and is discarded. The scan of the argument A terminates upon encountering the second ' which is also discarded since it follows a dummy argument. The next ' character is neither preceded nor followed by a dummy argument and remains in the macro expansion. The last ' character is followed by another dummy argument and is discarded. (Note that the five ' characters were necessary to generate two ' characters in the macro expansion.)

Within nested macro definitions, multiple single quotes can be used, with one quote removed at each level of macro nesting.

5.6.4 .NARG, .NCHR, and .NTYPE

These three directives allow the user to obtain the number of arguments in a macro call (.NARG), the number of characters in an argument (.NCHR), or the addressing mode of an argument (.NTYPE). (They are not available in ASEMBL.) Use of these directives permits selective modifications of a macro depending upon the nature of the arguments passed.

The .NARG directive enables the macro being expanded to determine the number of arguments supplied in the macro call and is of the form:

```
label:    .NARG     symbol
```

where:

	label	is an optional statement label
	symbol	is any legal symbol which is equated to the number of arguments in the macro call currently being expanded. The symbol can be used by itself or in expressions.

This directive can occur only within a macro definition. An example of the use of .NARG follows.

```
.MACRO  ARGS    A1,A2,A3,A4
.NARG   NUM
.IF EQ  NUM-1   ;IF A2, A3, A4 WERE
                ;NOT SPECIFIED
.WORD   A1
.IFF            ;IF ALL ARGS WERE GIVEN
.WORD   A1
.ASCII  /A2/
.WORD   A3
.ASCII  /A4/
.ENDC
.ENDM

ARGS    ALPHA
.WORD   ALPHA                     generated

ARGS    ALPHA,BETA,GAMMA,DELTA
.WORD   ALPHA
.ASCII  /BETA/                    generated
.WORD   GAMMA
.ASCII  /DELTA/
```

The .NCHR directive enables a program to determine the number of characters in a character string, and is of the form:

```
label:    .NCHR     symbol, <character string>
```

where:

	label	is an optional statement label
	symbol	is any legal symbol which is equated to the number of characters in the specified character string. The symbol is separated from the character string argument by any legal separator.

<character string>
: is a string of printing characters which should only be enclosed in angle brackets if it contains a legal separator. A semicolon also terminates the character string.

This directive can occur anywhere in a MACRO program. For example:

```
.MACRO  CHARS    A
.NCHR   NUM,A
.ASCII  /A/
.IF EQ  NUM&1              ;IF THE STRING CONTAINS
                           ;AN EVEN NUMBER OF
                           ;CHARACTERS
.WORD   -1
.IFF                       ;IF STRING LENGTH IS ODD
.BYTE   -2
.ENDC
.ENDM
```

For example, using the above definition, the code:

```
CHARS ALPHA
```

expands to:

```
.ASCII  /ALPHA/
.BYTE   -2
```

and

```
CHARS BETA
```

expands to:

```
.ASCII  /BETA/
.WORD   -1
```

The .NTYPE directive enables the macro being expanded to determine the addressing mode and register of any argument, and is of the form:

label: .NTYPE symbol, arg

where:

label
: is an optional statement label

symbol
: is any legal symbol, the low order 6-bits of which is equated to the 6-bit addressing mode of the argument. The symbol is separated from the argument by a legal separator. This symbol can be used by itself or in expressions.

arg
: is any legal macro argument (dummy argument) as defined in Section 5.6.3.

This directive can occur only within a macro definition. An example of .NTYPE usage in a macro definition is shown below:

```
        .MACRO  SAVE    ARG
        .NTYPE  SYM,ARG
        .IF     EQ,SYM&70
        MOV     ARG,TEMP        ;REGISTER MODE
        .IFF
        MOV     #ARG,TEMP       ;NON-REGISTER MODE
        .ENDC
        .ENDM
```

Using this definition, the code:

```
        SAVE    R2
```

expands to:

```
        MOV     #R2,TEMP
```

and

```
        SAVE    ALPHA
```

expands to:

```
        MOV     #ALPHA,TEMP
```

5.6.5 .ERROR and .PRINT

The .ERROR directive (not available in ASEMBL) is used to output messages to the listing file during assembly pass 2. A common use is to provide diagnostic announcements of a rejected or erroneous macro call. The form of the .ERROR directive is as follows:

label: .ERROR expr;text

where:

- label — is an optional statement label
- expr — is an optional legal expression whose value is output to the listing file when the .ERROR directive is encountered. Where expr is not specified, the text only is output to the listing file.
- ; — denotes the beginning of the text string to be output.
- text — is the string to be output to the listing file. The text string is terminated by a line terminator.

Upon encountering a .ERROR directive anywhere in a MACRO program, the assembler outputs a single line containing:

1. the sequence number of the .ERROR directive line,
2. the current value of the location counter,
3. the value of the expression if one is specified, and,

4. the text string specified.

For example, assume the following error macro occurs:

```
.MACRO STORE SRC,DEST
.NTYPE A,DEST
.IF EQ,<A&7>=6   ;IF STACK POINTER USED
.ERROR           A;UNACCEPTABLE MACRO ARGUMENT
.IFF
MOV     SRC,DEST
.ENDC
.ENDM


STORE R3,@4(SP)
```

and the following line is output:

```
****** P
 00000 000076          .ERROR          A;UNACCEPTABLE MACRO ARGUMENT
```

This message is used to indicate an inability of the subject macro to cope with the argument DEST which is detected as being indexed deferred addressing mode (mode 70) with the stack pointer (%6) used as the index register. The line is flagged on the assembly listing with a P error code.

The .PRINT directive is identical to .ERROR except that it is not flagged with a P error code. (.PRINT is not available in ASEMBL.)

5.6.6 Indefinite Repeat Block: .IRP and .IRPC

An indefinite repeat block (not available in ASEMBL) is a structure very similar to a macro definition. An indefinite repeat is essentially a macro definition which has only one dummy argument and is expanded once for every real argument supplied. An indefinite repeat block is coded in-line with its expansion rather than being referenced by name as a macro is referenced. An indefinite repeat block is of the form:

```
label:   .IRP arg,<real arguments>
           .
           .
           .
         (range of the indefinite repeat)
           .
           .
           .
         .ENDM
```

where:

label — is an optional statement label. A label may not appear on any .IRP statement within another macro definition, repeat range or indefinite repeat range, or on any .ENDM statement.

arg — is a dummy argument which is successively replaced with the real arguments in the .IRP statement.

<real arguments>
: is a list of arguments to be used in the expansion of the indefinite repeat range and enclosed in angle brackets. Each real argument is a string of zero or more characters or a list of real arguments (enclosed in angle brackets). The real arguments are separated by commas.

range
: is the code to be repeated once for each real argument in the list. The range may contain macro definitions, repeat blocks, or other indefinite repeat blocks. Note that only created symbols should be used as labels within an indefinite repeat range.

An indefinite repeat block can occur either within or outside macro definitions, repeat ranges, or indefinite repeat ranges. The rules for creating an indefinite repeat block are the same as for the creation of a macro definition (for example, the .MEXIT statement is allowed in an indefinite repeat block). Indefinite repeat arguments follow the same rules as macro arguments. A second type of indefinite repeat block is available which handles character substitution rather than argument substitution. The .IRPC directive is used as follows:

```
label:    .IRPC arg,string
            .
            .
            .
          (range of indefinite repeat)
            .
            .
            .
          .ENDM
```

On each iteration of the indefinite repeat range, the dummy argument (arg) assumes the value of each successive character in the string. Terminators for the string are: space, comma, tab, carriage return, line feed, and semicolon.

Figure 5-6 is an example of .IRP and .IRPC usage.

```
IRPTST  RT-11 MACRO VM02-09    11:04:49 PAGE 1





                         .TITLE  IRPTST
                         .LIST   MD,MC,ME
        000000           R0=%0
 000000 012700           MOV     #TABLE,R0
        000056'
```

```
                                .IRP     X,<A,B,C,D,E,F>

                                MOV      X,(R0)+
                                .ENDM

   00004 016720                 MOV      A,(R0)+
         000032

   00010 016720                 MOV      B,(R0)+
         000030

   00014 016720                 MOV      C,(R0)+
         000026

   00020 016720                 MOV      D,(R0)+
         000024

   00024 016720                 MOV      E,(R0)+
         000022

   00030 016720                 MOV      F,(R0)+
         000020
                                .IRPC    X,ABCDEF
                                .ASCII   /X/
                                .ENDM
   00034    101                 .ASCII   /A/
   00035    102                 .ASCII   /B/
   00036    103                 .ASCII   /C/
   00037    104                 .ASCII   /D/
   00040    105                 .ASCII   /E/
   00041    106                 .ASCII   /F/

   00042 041101 A:              .WORD    "AB
   00044 041502 B:              .WORD    "BC
   00046 042103 C:              .WORD    "CD
   00050 042504 D:              .WORD    "DE
   00052 043105 E:              .WORD    "EF
   00054 043506 F:              .WORD    "FG
   00056        TABLE:          .BLKW    6
         000001'                .END
```

Figure 5-6
.IRP and .IRPC Example

5.6.7 Repeat Block: .REPT

Occasionally it is useful to duplicate a block of code a number of times in line with other source code. (.REPT is not available in ASEMBL.) This is performed by creating a repeat block of the form:

```
          label:    .REPT expr
                      .
                      .
                      .
                    (range of repeat block)
                      .
```

```
        .
        .
        .ENDM     ;OR  .ENDR
```

where: label — is an optional statement label. The .ENDR or .ENDM directive may not have a label. A .REPT statement occurring within another repeat block, indefinite repeat block, or macro definition may not have a label associated with it.

expr — is any legal expression controlling the number of times the block of code is assembled. Where expr<0, the range of the repeat block is not assembled.

range — is the code to be repeated expr number of times. The range may contain macro definitions, indefinite repeat blocks, or other repeat blocks. Note that no statements within a repeat range can have a label.

The last statement in a repeat block can be an .ENDM or .ENDR statement. The .ENDR statement is provided for compatibility with previous assemblers.

The .MEXIT statement is also legal within the range of a repeat block.

5.6.8 Macro Libraries: .MCALL

All macro definitions must occur prior to their referencing within the user program. MACRO provides a selection mechanism for the programmer to indicate in advance those system macro definitions required by his program.

The .MCALL directive is used to specify the names of all system macro definitions not defined in the current program but required by the program (not available in ASEMBL). The .MCALL directive must appear before the first occurrence of a macro call for an externally defined macro. The .MCALL directive is of the form:

```
.MCALL arg1,arg2,...
```

where arg1,arg2, etc. are the names of the macro definitions required in the current program.

When this directive is encountered, MACRO searches the system library file, SYSMAC.SML, to find the requested definition(s). MACRO searches for SYSMAC.SML on the system device (SY:).

See Appendix D for a listing of the system macro file (SYSMAC.SML) stored on the system device.

5.7 CALLING AND USING MACRO

The MACRO Assembler assembles one or more ASCII source files containing MACRO statements into a single relocatable binary object file. Assembler output consists of this binary object file and an optional assembly listing followed by the symbol table listing. CREF (Cross Reference) listings may also be specified as part of the assembly output by means of switch options.

MACRO is executed using the RT-11 Monitor R command as follows:

```
.R MACRO
```

The assembler responds by typing an asterisk (*) to indicate readiness to accept command string input. In response to the * printed by the assembler, the user types the output file specification(s), followed by an equal sign or left angle bracket, followed by the input file specification(s) in a command line as follows:

```
*dev:obj,dev:list/s:arg=dev:sourcel,..,dev:sourcen/s:arg
```

where:

	dev:	is any legal RT-11 device for output; must be file-structured for input
	obj	is the binary object file
	list	is the assembly listing file containing the assembly listing and symbol table
	sourcel, ..,sourcen	are the ASCII source files containing the macro source program(s); a maximum of six source files is allowed
	/s:arg	represents a switch and argument as explained in Section 5.7.1

A null specification in either of the output file fields signifies that the associated output file is not desired.

One or more switches can be indicated with the appropriate file specification to provide MACRO with information about that file.

The default case for each file specification is noted below:

file	device	filename	extension
object	DK:	-	.OBJ
listing	device used for object output	-	.LST
sourcel	DK:	-	.MAC
source2 . . . sourcen	device used for last source file specified	-	.MAC
system macro file	system device SY:	SYSMAC	.SML

Type CTRL C to halt MACRO at any time and return control to the monitor. To restart the assembler type R MACRO or the REENTER command in response to the monitor's dot.

NOTE

> If ↑C was typed while a CREF listing was being produced, the REENTER command may not be accepted. In this case, type R MACRO to restart the assembler.

5.7.1 Switches

There are three types of switch options: listing control switches, function switches, and CREF specification switches. The listing control switches (/L,/N) provide capabilities similar to those described in detail in section 5.5.1.1. The function control switches (/D,/E) provide function control as described in Section 5.5.2; arguments for these switches are summarized in Section 5.7.1.2. CREF control switches allow the user to obtain a detailed cross-referenced listing of his assembled file, and are described in detail in Section 5.7.1.3. Multiple arguments may be specified for a particular switch, if desired, by separating each switch value from the next by a colon. For example:

```
/N:TTM:CND
```

These switches turn off teleprinter mode and suppress printing of unsatisfied conditionals (as described in the next section). Also, the switches are not restricted to appearing near a particular file in the command string; /N:TTM, for example, is legal in all of the following places:

```
*,LP:/N:TTM=source
*,LP:=source/N:TTM
*/N:TTM,LP:=source
```

and they are all equivalent in function.

5.7.1.1 Listing Control Switches - A listing control switch (/L for list or /N for nolist) is indicated in a command line as follows:

```
*dev:obj.ext,dev:list.ext/s:arg=dev:source.ext
```

where s:arg represents /L or /N; the remainder of the command line abbreviations are as described in Section 5.7.

The /N switch with no argument causes only the symbol table, table of contents and error listings to be produced. The /L switch with no arguments causes .LIST and .NLIST directives that appear in the source program but have no arguments to be ignored. A summary of the arguments which are valid for the listing control switches follows (refer to Section 5.5.1.1 for details):

Argument	Default	Controls listing of
SEQ	list	Source line sequence numbers
LOC	list	Location counter
BIN	list	Generated binary code
BEX	list	Binary extensions

SRC	list	Source code
COM	list	Comments
MD	list	Macro definitions, repeat range expansions
MC	list	Macro calls, repeat range expansions
ME	nolist	Macro expansions
MEB	nolist	Macro expansion binary code
CND	list	Unsatisfied conditionals, .IF and .ENDC statements
LD	nolist	Listing directives with no arguments
TOC	list	Table of Contents
TTM	terminal mode	Listing output format
SYM	list	Symbol table

For example, a command line in the following format allows binary code to be listed throughout the assembly using the 132-column line printer format:

```
*,LP:/L:MEB/N:TTM=FILE
```

5.7.1.2 Function Switches - The function control switches (/D for disable and /E for enable) are used to enable or disable certain functions in source input files and are indicated in the command line as follows:

```
*dev:obj.ext,dev:list.ext=dev:source/s:arg
```

/s:arg here represents either /D:arg or /E:arg. A summary of the arguments which are valid for use with the function control switches follows (refer to Section 5.5.2 for details):

Argument	Default	Enables or disables
ABS	disable	Absolute binary output
AMA	disable	Assembly of all absolute addresses as relative addresses
CDR	disable	Source columns 73 and greater to be treated as comments
FPT	disable	Floating point truncation
LC	disable	Accepts lower case ASCII input
LSB	disable	Local symbol block
PNC	enable	Binary output

For example, the following commands assemble a file allowing all 80 columns of each card to be used as input (note that since MACRO is a two-pass assembler, the cards cannot be read directly from the card reader; input from any nonfile-structured device must first be transferred to a file-structured device before assembly):

```
.R PIP
*CARDS.MAC=CR:/A
*^C

.R MACRO
*,LP:=CARDS.MAC/E:CDR
```

Use of either the function control or listing control switches and arguments at assembly-time will override any corresponding listing or function control directives and arguments in the source program. For example, assume the following appears in the source program:

```
.NLIST MEB
  .
  .          MACRO References
  .    .
.LIST MEB
```

The "MEB" printing will be disabled for the block indicated; however, if /L:MEB is indicated in the assembly command line, both the .NLIST MEB and the .LIST MEB will be ignored and the "MEB" printing will be enabled everywhere in the program.

5.7.1.3 Cross Reference Table Generation (CREF) - A cross reference table of all or a subset of all symbols used in the source program and the statements where they were defined or used can be obtained automatically following an assembly by specifying /C:arg with the assembly listing file specification (and any listing or function control specifications) as follows:

```
*dev:obj.ext,dev:list.ext/s:arg/C:arg=dev:source.ext
```

/s:arg represents /L:arg, /N:arg, /E:arg, or /D:arg. (If the listing device is magtape or cassette, and a CREF listing is desired, the handler must first be loaded, using the monitor LOAD command.)

There are six sections to a complete cross reference listing:

1. Cross reference of program symbols (i.e., labels used in the program and symbols used on the left of the "=" operator).

2. Cross reference of register-equate symbols (those symbols which are defined in the program by a "SYMBOL=%N", 0<=N<=7, construct. (Normally this consists of the symbols R0, R1, R2, R3, R4, R5, SP, and PC.)

3. Cross reference of MACRO symbols (names of macros as defined by a .MACRO directive, or as specified in a .MCALL directive).

4. Cross reference of permanent symbols (all operation mnemonics and assembler directives).

5. Cross reference of control sections (those names specified as the operand of a .CSECT directive, plus the blank .CSECT and the absolute section ". ABS." which are always defined by MACRO).

6. Cross reference of errors (all errors flagged on the listing are grouped by error type).

Any or all of the above sections may be included in the cross reference listing as desired. The associated switch options and their arguments are listed below:

Switch Argument	Section Type
/C:S	User-defined symbols
/C:R	Register symbols
/C:M	Macro symbolic names
/C:P	Permanent symbols (instructions, directives)
/C:C	Control sections (.CSECT symbolic names)
/C:E	Error codes
/C<no arg>	Equivalent to /C:S:M:E

The specification of a /C switch in a command string causes a temporary file, "DK:CREF.TMP", to be generated. If device DK: is write-locked or contains insufficient free space for the temporary file, the user may allocate the temporary file on another device. To do so, a third output file specification is given in the MACRO command string; this file is then used instead of DK:CREF.TMP, and is purged after use. For example, a command string of this type:

```
*,LP:,RK2:TEMP.TMP=SOURCE/C
```

causes "RK2:TEMP.TMP" to be used as the temporary file.

Figure 5-7 illustrates assembled source code and Figure 5-8 contains the CREF output. The command line used to produce these listings was:

```
*,LP:/C:S:M:R:P:C:E/N:BEX=EXAMPL
```

An explanation of the CREF output follows the figures.

```
EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09   5-SEP-74 22:11:59 PAGE 1

1                               .TITLE  EXAMPLE OF CROSS-REFERENCE LISTING
2
3          000000 R0=           %0                              ;DEFINE THE REGISTER SYMBOLS
4          000001 R1=           %1
5          000002 R2=           %2
6          000003 R3=           %3
7          000004 R4=           %4
8          000005 R5=           %5
9          000006 SP=           %6
10         000007 PC=           %7
11
12         000012 LF=           012                             ;SYMBOL FOR LINE FEED
13
14                              .MCALL  .TTYIN, .EXIT
15
16                              .MACRO  CALL    NAME            ;DEFINE A USER MACRO
17                              JSR     PC,NAME
18                              .ENDM
19
20                              .GLOBL  SUBR1,  SUBR2           ;TWO EXTERNAL SUBROUTINES
21
22         000000'              .CSECT  PROG                    ;DEFINE A CSECT
23 00000   012702 START:        MOV     #BUFFER,R2              ;R2 = ADRS(BUFFER)
24 00004          1$:           .TTYIN                          ;READ A CHAR INTO R0
25 00010   110022               MOVB    R0,(R2)+                ;AND STORE IN BUFFER
26 00012   120027               CMPB    R0,#LF                  ;WAS IT A LINE FEED?
27 00016   001372               BNE     1$                      ;NOPE - KEEP READING
28 00020   105022               CLRB    (R2)+                   ;ELSE FLAG END OF LINE WITH ZERO
29 00022   012703               MOV     #BUFFER,R3              ;R3 = ADRS(BUFFER) FOR SUBR1
30 00026                        CALL    SUBR1                   ;INVOKE CALL MACRO
31 00032   103762               BCS     START                   ;GET A NEW LINE IF CARRY SET
32 00034                        CALL    SUBR2                   ;ELSE CALL OTHER SUBR.
33 00040   010067               MOV     R0,ANSWER               ;AND STORE IN ANSWER
34 00044                        .EXIT                           ;RETURN TO RT-11
35
36 00046          ANSWER:       .BLKW                           ;DEFINE ANSWER STORAGE
37 00050          BUFFER:       .BLKB   72.                     ;INPUT LINE BUFFER
38
39         000000'              .END    START
```

Figure 5-7
MACRO Source Code

```
EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09   5-SEP-74 22:11:59 PAGE 1+
SYMBOL TABLE

ANSWER  000046R    002  BUFFER  000050R    002  LF      = 000012
PC      =%000007        R0      =%000000        R1      =%000001
R2      =%000002        R3      =%000003        R4      =%000004
R5      =%000005        SP      =%000006        START     000000R    002
SUBR1 = ****** G        SUBR2 = ****** G
. ABS.  000000     000
        000000     001
PROG    000160     002
ERRORS DETECTED: 0
FREE CORE: 16248. WORDS

,LP:/C:S:M:R:P:C:E/N:BEX=EXAMPL
```

Figure 5-7 (Cont.)
MACRO Source Code

```
EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09   5-SEP-74 22:11:59 PAGE S-1
CROSS REFERENCE TABLE (CREF V01-02 )


.          1-24
ANSWER     1-33*    1-36#
BUFFER     1-23     1-29     1-37#
LF         1-12#    1-26
START      1-23#    1-31     1-39
SUBR1      1-20#    1-30
SUBR2      1-20#    1-32
```

Figure 5-8
CREF Listing Output

```
EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09   5-SEP-74 22:11:59 PAGE R-1
CROSS REFERENCE TABLE (CREF V01-02 )

PC        1-10#    1-30*    1-32*
R0        1-3#     1-25     1-26     1-33
R1        1-4#
R2        1-5#     1-23*    1-25*    1-28*
R3        1-6#     1-29*
R4        1-7#
R5        1-8#
SP        1-9#
```

```
EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09   5-SEP-74 22:11:59 PAGE M-1
CROSS REFERENCE TABLE (CREF V01-02 )

.EXIT     1-14#    1-34
.TTYIN    1-14#    1-24
CALL      1-16#    1-30     1-32
```

Figure 5-8 (Cont.)
CREF Listing Output

```
EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09    5-SEP-74 22:11:59 PAGE P-1
CROSS REFERENCE TABLE (CREF V01-02 )

.BLKB     1-37
.BLKW     1-36
.CSECT    1-22
.END      1-39
.GLOBL    1-20
.IF       1-24
.MACRO    1-16
.MCALL    1-14
.TITLE    1-1
BCS       1-24      1-31
BNE       1-27
CLRB      1-28
CMPB      1-26
EMT       1-24      1-34
JSR       1-30      1-32
MOV       1-23      1-29      1-33
MOVB      1-25
```

```
EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09    5-SEP-74 22:11:59 PAGE C-1
CROSS REFERENCE TABLE (CREF V01-02 )

           0-0
. ABS.     0-0
PROG       1-22
```

Figure 5-8 (Cont.)
CREF Listing Output

Cross reference tables, if requested, are generated at the end of a MACRO assembly listing. Each table begins on a new page (the tables in Figure 5-8 have been consolidated due to space considerations). Symbols, control sections, and error codes are listed at the left margin of the page; corresponding references are indicated next to them across the page from left to right. A reference is of the form p-l, where p is the page on which the symbol, control section, or error code appears, and l is the line number within the page. A number sign (#) appears next to a reference wherever a symbol has been defined. An asterisk appears next to a reference wherever a destructive reference has been made to the symbol (i.e., the contents of the location defined by that symbol has been altered at that point).

The CREF output requested in the preceding figures included user defined symbols, macro symbolic names, control sections, error codes, register symbols, and permanent symbols. Since no errors were generated in this assembly, no CREF output for error codes was produced.

5.8 MACRO ERROR MESSAGES

MACRO error messages enclosed in question marks are output on the terminal. The single-letter error codes are printed in the assembly listing.

In terminal mode these error codes are printed following a field of six asterisk characters and on the line preceding the source line containing the error. For example:

```
****** A
26 00236 000002' .WORD REL1+REL2
```

Error Code	Meaning
A	Addressing error. An address within the instruction is incorrect. Also may indicate a relocation error. The addition of two relocatable symbols is flagged as an A error. May also indicate that a local symbol is being defined more than 128 words from the beginning of a local symbol block.
B	Bounding error. Instructions or word data would be assembled at an odd address in memory. The location counter is updated by +1.
D	Multiply-defined symbol referenced. Reference was made to a label (not a local label) that is defined more than once.
E	End directive not found. (A .END is generated.)
I	Illegal character detected. Illegal characters which are also non-printing are replaced by a ? on the listing. The character is then ignored.
L	Line buffer overflow, i.e., input line greater than 132 characters. Extra characters on a line are ignored in terminal mode.

M	Multiple definition of a label. A label was encountered which was equivalent (in the first six characters) to a previously encountered label.
N	Number containing 8 or 9 has decimal point missing.
O	Opcode error. Directive out of context.
P	Phase error. A label's definition or value varies from one pass to another or a local symbol occurred twice within a local symbol block.
Q	Questionable syntax. There are missing arguments or the instruction scan was not completed or a carriage return was not immediately followed by a line feed or form feed.
R	Register-type error. An invalid use of or reference to a register has been made.
T	Truncation error. A number generated more than 16 bits of significance or an expression generated more than 8 bits of significance during the use of the .BYTE directive.
U	Undefined symbol. An undefined symbol was encountered during the evaluation of an expression. Relative to the expression, the undefined symbol is assigned a value of zero.
Z	Instruction which is not compatible among all members of the PDP-11 family.

Error Message	Explanation
?BAD SWITCH?	The switch specified was not recognized by the program.
?INSUFFICIENT CORE?	There are too many symbols in the program being assembled. Try dividing program into separately-assembled subprograms.
?I/O ERROR ON CHANNEL n?	A hardware error occurred while attempting to read from or write to the device on the channel specified in the message. (Channel numbers (0<=n<=10 octal) are assigned to files in the manner described in Section 9.4.7, Chapter 9.) Note that the CREF temporary file is on channel 2 even if it was not specified in the command string (i.e., if the default file DK:CREF.TMP is used).

?NO INPUT FILE?	No input file was specified and there must be at least one input file.
?OUTPUT DEVICE FULL?	No room to continue writing output. Try to compress device with PIP.
TOO MANY OUTPUT FILES	Too many output files were specified.

All CREF error messages begin with C- to distinguish them from MACRO error messages. When a CREF error occurs, the error message is printed on the console terminal and CREF chains back to MACRO; MACRO prints an asterisk, at which time another command line may be entered.

Error Message	Explanation
?C-CHAIN-ONLY-CUSP?	An attempt was made either to "R CREF" or to "START" a copy of CREF which was in memory. CREF can only be "chained" to.
?C-CRF FILE ERROR?	An output error occurred while accessing "DK:CREF.TMP", the temporary file passed to CREF.
?C-DEVICE?	An invalid device was specified to CREF.
?C-LST FILE ERROR?	An output error occurred while attempting to write the cross-reference table to the listing file.

CHAPTER 6

LINKER

6.1 INTRODUCTION

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 processed by the Linker to:

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. If 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 job of an F/B System).

The Linker runs in a minimal RT-11 system of 8K; 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 is 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 format:

```
*dev:binout,dev:mapout=dev:obj1,dev:obj2,.../s1/s2/s3
```

where:

dev:	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's save image, LDA format, 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.
obj1,...	are files of one or more object modules to be input to the Linker (these may be library files).
/s1/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 as:

```
*objm,objn,.../s1/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.

If an output file is not specified, the Linker assumes that the associated output is not desired. For example, if the load module and load map are not specified, only error messages (if any) are printed by the Linker.

The default values for each specification are:

	Device	Filename	Extension
Load Module	DK:	none	SAV, REL(/R), LDA(/L)
Map Output	Same as load module	none	MAP
Object Module	DK: or same as previous object module	none	OBJ

If a syntax error is made in a command string, an error message is printed. A new command string can then be typed following the asterisk.

If a nonexistent file is specified a fatal error occurs; control is returned to the command string interpreter, an asterisk is printed and a new command string may be entered.

6.2.2 Switches

The switches associated with the Linker are listed in Table 6-1. The letter representing each switch is always preceded by the slash character. Switches must appear on the line indicated if the command is continued on more than one line. They may be positioned anywhere on the line. (A more detailed explanation of each switch is provided in Section 6.8.)

Table 6-1
Linker Switches

Switch Name	Command Line	Meaning
/A	1st	Alphabetizes the entries in the load map.
/B:n	1st	Bottom address of program is indicated as n (illegal for foreground links).
/C	any	Continues input specification on another command line. Used also with /O.
/F	1st	Instructs the Linker to use the default FORTRAN library, FORLIB.OBJ; note that FORLIB does not have to be specified in the command line.
/I	1st	Includes the global symbols to be searched from the library.
/L	1st	Produces an output file in LDA format (illegal for foreground links).

(Continued on next page)

Table 6-1 (Cont.)
Linker Switches

Switch Name	Command Line	Meaning
/M or /M:n	1st	Stack address is to be specified at the terminal keyboard or via n.
/O:n	any but the 1st	Indicates that the program will be an overlay structure; n specifies the overlay region to which the module is assigned.
/R	1st	Produces output in REL format; only files in REL format will run in the foreground (REL format files may not be run under a Single-Job system)
/S	1st	Allows the maximum amount of space in memory to be available for the Linker's symbol table. (This switch should only be used when a particular link stream causes a symbol table overflow.)
/T or /T:n	1st	Transfer address is to be specified at terminal keyboard or via n.

6.3 ABSOLUTE AND RELOCATABLE PROGRAM SECTIONS

A program produced by one of the RT-11 assemblers or FORTRAN IV can consist of an absolute program section, declared by the .ASECT assembler directive, and relocatable program sections declared by the .CSECT assembler directive. A .CSECT directive is assumed at the beginning of the source program. The instructions and data in relocatable sections are normally assigned locations beginning at 1000(octal) or 0 for a foreground link. The assignment of addresses can be influenced by command string switches and the size of the absolute section (.ASECT, if present). Each control section is assigned a memory address; the Linker then appropriately modifies all instructions and/or data as necessary to account for the relocation of the control sections.

NOTE

Foreground programs cannot use .ASECTs beyond 1000 (octal); as a general practice, they should be avoided under a Foreground/Background system.

The RT-11 Linker handles the absolute section as well as the named and unnamed control sections. The unnamed control section is internal to each object module. That is, every object module can have an unnamed control section but the Linker treats each control section independently. Each is assigned an absolute address such that it occupies an exclusive area of memory. Named control sections, on the other hand, are treated globally; if different object modules have control sections with the same name, they are all assigned the same absolute load address and the size of the area reserved for loading of the section is the size of the largest. Thus, named control sections allow for the sharing of data and/or instructions among object modules. This is the same as the handling and function of COMMON in FORTRAN IV. The names assigned to control sections are global and can be referenced as any other global symbol.

NOTE

If relocatable code is to be linked for the foreground, no location may be filled more than once (using location counter arithmetic); any such location may be improperly relocated during the FRUN and may cause program or system failure.

For example, the following code illustrates a program using location counter arithmetic that is illegal if linked for the foreground. Note that the code at line 15 starts at 0 due to location counter modification. To correct this program for foreground linking, remove all .=.- instructions.

```
.MAIN.  RT-11 MACRO VM02-12    PAGE 1

1          000000'         .CSECT TEST
2                          .GLOBL A,B,C
3                          .MCALL ..V2..,.REGDEF
4  000000                  ..V2..
5  000000                  .REGDEF
6  000000 000000G          .WORD A
7  000002 016701  START:   MOV A+6,R1
          000006G
8  000006 060127           ADD R1,(PC)+
9  000010 000000G          .WORD B
10         000002'         .=.-10
11  00002 000004G          .WORD A+4
12         000002'         .=.-2
13  00002 000002G          .WORD B+2
14         000000'         .=.-4
15  00000 000006G          .WORD C+6
16         000002'         .END START
```

This page intentionally blank.

6.4 GLOBAL SYMBOLS

Global symbols provide the link, or communication, between object modules. Global symbols are created with the .GLOBL assembler directive (see Chapter 5). If the global symbol is defined in an object module (as a label or by direct assignment), it is called an entry symbol and other object modules can reference it. If the global symbol is not defined in the object module, it is an external symbol and is assumed to be defined (as an entry symbol) in some other object module.

As the Linker reads the object modules it keeps track of all global symbol definitions and references. It then modifies the instructions and/or data which reference the global symbols. Undefined globals are printed on the console terminal after pass 1 (or pass 2 if a library file is also linked).

6.5 INPUT AND OUTPUT

Linker input and output is in the form of modules; one or more input modules (object files produced by either assembler or FORTRAN IV) are used to produce a single output (load) module.

6.5.1 Object Modules

Object files, consisting of one or more object modules, are the input to the Linker (the Linker ignores files which are not object modules). Object modules are created by the RT-11 assemblers or FORTRAN IV. The Linker reads each object module at least twice (three times if library files are linked). During the first pass each object module is read to construct a global symbol table and to assign absolute values to the control section names and global symbols. If library files are linked, a second pass is needed to resolve the undefined globals from the library files and force their associated object modules into the root; on the final pass, the Linker reads the object modules, links and relocates the modules and outputs the load module.

6.5.2 Load Module

The primary output of the Linker is a load module which may be loaded and run under RT-11. The load module is output as a save image file (SAV) for use under a Single-Job system or the background job. Under an F/B System, the /R switch must be used to produce a REL (reloctable) format foreground load module if the job is to run in the foreground. An absolute load module (LDA) is produced if the module is to be loaded by the Absolute Loader.

The load module for a save image file is arranged as follows:

Root Segment	Overlay Segments (optional)

For a REL image file, the load modules are arranged as:

Root Segment	Overlay Segments (optional)	Resident REL Blocks	Overlay REL Blocks (optional)

The first 256-word block of the root segment (main program) contains the memory usage map and the locations used by the Linker to pass program control parameters. The memory usage map outlines the blocks of memory used by the load module and is located in locations 360 to 377.

The control parameters are located in locations 40-50 and contain the following information when the module is loaded:

Address	Information
40:	Start Address of program
42:	Initial setting of R6 (stack pointer)
44:	Job Status Word
46:	USR Swap Address (0 implies normal location)
50:	Highest Memory Address in user's program

For a foreground link the following additional parameters contain information:

34,36:	TRAP Vector
52:	Size of Resident (words)
54:	Sum of the Resident and largest Overlay Region (words)
56:	F/B Identification
60:	Address of Resident REL Block

Memory locations 0-476 (comprising the interrupt vectors and system communication area) may be assigned initial values by using an .ASECT assembler statement and will appear in block 0 of the load module, but there are restrictions on the use of .ASECTs in this region. The Linker does not permit an .ASECT of location 54 or of locations 360-377 (the memory usage map is passed in those locations). In addition, for foreground links, an .ASECT of words 40-60 is not permitted (additional parameters are passed to FRUN in those locations).

Any location which is not restricted may be set with an .ASECT, but caution should be used in changing the system communication area. Restricted areas, such as the region 360-377, must be initialized by the program itself. There are no restrictions on .ASECTs if the output format is LDA.

Locations in the region 0-476 which are initialized by an .ASECT in a program may never be loaded when the program is executed. There are two reasons for this. For background tasks (or the Single-Job system) the R, RUN, and GET commands will not load an address protected by the monitor's memory protection map. The addresses normally protected include such important areas as the system device and console device vectors, but protection may be extended dynamically (e.g., by a foreground task issuing a .PROTECT call). For foreground tasks, the FRUN command will load only locations 34-50 (34 is the TRAP instruction vector) and all other .ASECTs are ignored. The procedure for loading these locations is to do so at run-time using MOV instructions.

6.5.3 Load Map

If requested, a load map is produced following the completion of the initial pass(es) of the Linker. This map, shown in Figure 6-1, diagrams the layout of memory for the load module.

Each .CSECT included in the linking process is listed in the load map. The entry for a .CSECT includes the name and low address of the section and its size (in bytes). The remaining columns contain the entry points (or globals) found in the section and their addresses.

The map begins with the name of the load module and the date of creation. The modules located in the root segment of the load module are listed next, followed by those modules which were assigned to overlays in order by their region number (see Section 6.6). Any undefined global symbols are then listed. The map ends with the transfer address (start address) and high limit of relocatable code.

NOTE

> The load map will not reflect the absolute addresses for a REL file created to be run as a foreground job; the base relocation address specified at FRUN time must be added to obtain the absolute addresses.

For example, assume the FRUN command is used to run the program RELTST:

```
.FRUN RELTST/P
LOADED AT 137150
```

When linked, the following load map is produced:

```
RT-11 LINK    X03-16      LOAD MAP
RELTST.REL                03-SEP-74

SECTION ADDR    SIZE      ENTRY   ADDR     ENTRY   ADDR     ENTRY   ADDR

. ABS.  000000  000350    $TKSIZ  000012   $FVEC   000320   $CRPOS  001750
TSTVT1  000350  000326
$GTB    000676  001116    $DSINT  000676   $STOPF  000710   $NR     000752
                          $TACKP  000764   $TATBF  001106   $DPC    001142
                          $LINKF  001146   $LCDIS  001150   $LCNT   001152
                          $DSVEC  001154   $TACKE  001202   $DFILE  001226
                          $YS     001244   $BLANK  001246   $LINK   001252
                          $BYPAS  001256   $CTRAK  001260   $LSRA   001276
                          $CUSER  001302   $NULL   001316   $XT     001326
                          $YT     001330   $LPINT  001404   $LPBUF  001434
                          $NRBUF  001446   $TRAKC  001562   $XSTOR  001572
                          $YSTOR  001574   $SOINT  001670   $VSTIN  001676
                          $VSTP   001722   $PDV1   001756   $PEXIT  001764
```

```
OVERLAY REGION 000001   SEGMENT 000001
SGT1     002016  000210  $VINIT  002016  $VFDEL  002100  $VSTOP  002114
                         $VNSRT  002130  $VRMOV  002174  $VSTRT  002214
SGT2     002226  000102  $VBLNK  002226  $VRSTR  002256  $VSRCH  002300
SGT3     002330  000130  $VTRAK  002330  $VLPEN  002410  $VSTPM  002420
                         $NOSYN  002430  $SYNC   002440  $NAME   002450

OVERLAY REGION 000001   SEGMENT 000002
SGT4     002016  000602  $VRTLK  002016  $VUNLK  002300  $VLSET  002404
                         $VSCRL  002512

TRANSFER ADDRESS = 000350
HIGH LIMIT = 002620
```

To determine the address of TSTVT1, 137150 must be added to 000350; thus 137520 is the absolute address of TSTVT1. The transfer address is 137150 plus 350, or 137520.

6.5.4 Library Files

The RT-11 Linker has the capability of automatically searching libraries. Libraries are composed of library files--specially formatted files produced by the Librarian program (Chapter 7) which contain one or more object modules. The object modules provide routines and functions to aid the user in meeting specific programming needs. (For example, FORTRAN has a special library containing all necessary computational functions--TAN, ATAN, etc.) By using the Librarian, libraries can be created and updated so that routines which are used more than once, or routines which are used by more than one program, may be easily accessed. Selected modules from the appropriate library file are linked as needed with the user program to produce one load module. Libraries are further described in Section 6.7 and in Chapter 7.

NOTE

> Library files that have been combined under PIP are illegal as input to both the Linker and the Librarian.

```
RT-11 LINK    V03-01      LOAD MAP
SQRT   .SAV               19-SEP-74

SECTION ADDR    SIZE      ENTRY   ADDR      ENTRY   ADDR      ENTRY   ADDR

. ABS.  000000  001000    $USRSW  000000    $V005A  000001    $NLCHN  000006
                          $LRECL  000210    $TRACE  004737
        001000  000220
        001220  001364    $OTI    001246
        002604  002300    OCI$    002604    ICI$    002612    $GET    002772
                          RCI$    003006    OCO$    003712    ICO$    003720
                          GCO$    004144    FCO$    004152    ECO$    004156
                          DCO$    004164
        005104  000160    ISN$    005104    $ISNTR  005110    LSN$    005124
                          $LSNTR  005130
        005264  000102    MOI$SS  005264    MOL$SS  005264    MOI$SM  005270
                          MOI$SA  005274    MOI$IS  005300    MOL$IS  005300
                          REL$    005300    MOI$IM  005304    MOI$IA  005310
                          MOI$MS  005314    MOI$MM  005320    MOI$MA  005324
                          MOI$0S  005330    MOI$0M  005334    MOI$0A  005340
                          MOI$1S  005344    MOI$1M  005352    MOI$1A  005360
        005366  000020    IFR$    005366    IFW$    005400
        005406  000046    EOL$    005406
        005454  000062    TVL$    005454    TVF$    005462    TVD$    005470
                          TVQ$    005476    TVP$    005504    TVI$    005512
        005536  000036    CAI$    005536    CAL$    005544
        005574  000174    SQRT    005574
        005770  000026    MOF$RS  005770    MOF$RM  005776    MOF$RA  006006
                          MOF$RP  006012
        006016  000044    NMI$1I  006016    NMI$1I  006026    BLE$    006034
                          BEQ$    006036    BGT$    006044    BGE$    006046
                          BRA$    006050    BNE$    006054    BLT$    006056
        006062  000072    FOO$    006062    EXIT    006074    STP$    006074
        006154  000002    $AOTS   006154
$ERRTB  006156  000100
$ERRS   006256  002637
        011116  001534    $FIO    011600
        012652  000202    $FMTDR  012652    $FMTDW  012702    $INITI  012750
        013054  000416    $CLOSE  013054
        013472  000106    LCI$    013472    LCO$    013540
        013600  000302    $GETRE  013600    $TTYIN  013722
        014102  000262    $PUTRE  014102
        014364  000106    $FCHNL  014364
        014472  000674    $OPEN   014472
        015366  000110    $DUMPL  015366
        015476  000414    $PUTBL  015476    $GETBL  015676    $EOFIL  016046
        016112  000042    $WAIT   016112

TRANSFER ADDRESS = 001000
HIGH LIMIT = 016154
```

Figure 6-1
Linker Load Map for Background Job

6.6 USING OVERLAYS

The RT-11 program overlay facility enables the user to have virtually unlimited memory space for an assembly language or FORTRAN program. A program using the overlay facility can be much larger than would normally fit in the available memory space, since portions of the program (called overlay segments) reside on a backup storage device (disk or DECtape).

The RT-11 overlay scheme is a strict multi-region arrangement; it is not tree-structured. Figure 6-2 diagrams this scheme. The overlay system which the user constructs from his completed program is composed of a root segment, memory-resident overlay regions, and the overlay segments stored on the backup storage device. The root segment is a required part of every overlay program and contains all transfer addresses; it must therefore never be overlaid. An overlay region corresponds to a run-time area of memory that is shared by two or more subroutines (called co-resident subroutines); there is a distinct memory area for each overlay region. Overlay segments are portions of the save image or REL format file from which the user's program is run; these are brought into memory as needed.

```
A=A/C        = Root
B/O:1/C      = Segment 1
                            = Region 1
C/O:1/C      = Segment 2
D/O:2/C      = Segment 3
                            = Region 2
E/O:2        = Segment 4
```

Figure 6-2
Overlay Scheme

Overlay regions are specified to the Linker via the /O switch as described in Section 6.8.8. The size of the overlay region is calculated by the Linker to be the size of the largest group of subroutines that can occupy the region at one time. The Linker creates the overlay regions and edits the program to produce the desired overlays at run-time.

Figure 6-3 shows a diagram of memory for a program which has an overlay structure and Figure 6-4 is a listing of the run-time overlay handler.

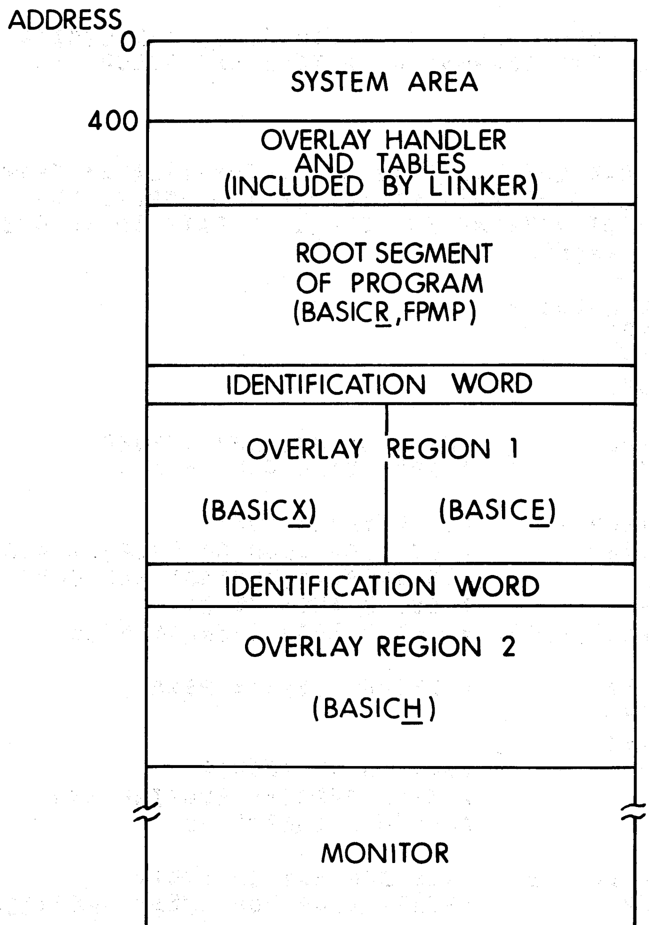


Figure 6-3
Memory Diagram Showing BASIC Link With Overlay Regions

```
        .SBTTL  THE RUN-TIME OVERLAY HANDLER

; THE FOLLOWING CODE IS INCLUDED IN THE USER'S PROGRAM BY THE
; LINKER WHENEVER OVERLAYS ARE REQUESTED BY THE USER.
; 56.8 MICROSECONDS (APPROX) IS ADDED TO EACH REFERENCE OF
; A RESIDENT OVERLAY SEGMENT.

; THE RUN-TIME OVERLAY HANDLER IS CALLED BY A DUMMY
; SUBROUTINE OF THE FOLLOWING FORM:

;               JSR     R5,$OVRH        ;CALL TO COMMON CODE
;               .WORD   <OVERLAY #>     ;# OF DESIRED SEGMENT
;               .WORD   <ENTRY ADDR>    ;ACTUAL CORE ADDR

; ONE DUMMY ROUTINE OF THE ABOVE FORM IS STORED IN THE RESIDENT
; PORTION OF THE USER'S PROGRAM FOR EACH ENTRY POINT TO
; AN OVERLAY SEGMENT.  ALL REFERENCES TO THE ENTRY POINT ARE
; MODIFIED BY THE LINKER TO INSTEAD BE REFERENCES TO THE APPRO-
; PRIATE DUMMY ROUTINE.  EACH OVERLAY SEGMENT IS CALLED INTO
; CORE AS A UNIT AND MUST BE CONTIGUOUS IN CORE.  AN OVERLAY
```

```
; SEGMENT MAY HAVE ANY NUMBER OF ENTRY POINTS, TO THE LIMITS
; OF CORE MEMORY.  ONLY ONE SEGMENT AT A TIME MAY OCCUPY AN
; OVERLAY REGION.

; RESTRICTIONS:
; SINCE REFERENCES TO OVERLAY SEGMENTS ARE AUTOMATICALLY TRANS-
; LATED BY THE LINKER INTO REFERENCES TO DUMMY SUBROUTINES,
; THE PROGRAMMER MUST NOT ATTEMPT TO REFERENCE DATA IN AN OVER-
; LAY BY USING GLOBAL SYMBOLS.

        $OVTAB=1000+$OVRHE-$OVRH
$OVRH:  MOV     R0,-(SP)
        MOV     R1,-(SP)
        MOV     R2,-(SP)
$OVRHB:
;       MOV     (R5)+,R0            ;PICK UP OVERLAY NUMBER
        BR      $FIRST              ;FIRST CALL ONLY * * *
        MOV     R0,R1
$OVRHA: ADD     #$OVTAB-6,R1        ;CALC TABLE ADDR
        MOV     (R1)+,R2            ;GET CORE ADDR OF OVERLAY REGION
        CMP     R0,@R2              ;IS OVERLAY ALREADY RESIDENT?
        BEQ     $ENTER              ;YES, BRANCH TO IT
        .READW  17,R2,(R1)+,(R1)+ ;READ FROM OVERLAY FILE
        BCS     $ERR
$ENTER: MOV     (SP)+,R2            ;RESTORE USER'S REGS
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        MOV     @R5,R5              ;GET ENTRY ADDRESS
        RTS     R5                  ;ENTER OVERLAY ROUTINE AND
                                    ;RESTORE USER'S R5

$FIRST: MOV     #12500,$OVRHB       ;RESTORE SWITCH INSTR
        MOV     (PC)+,R1            ;START ADDR FOR CLEAR OPERATION
$HROOT: .WORD   0                   ;HIGH ADDR OF ROOT SEGMENT
        MOV     (PC)+,R2            ;COUNT
$HOVLY: .WORD   0                   ;HIGH LIMIT OF OVERLAYS
1$:     CLR     (R1)+               ;CLEAR ALL OVERLAY REGIONS
        CMP     R1,R2
        BLO     1$
        BR      $OVRHB              ;AND RETURN TO CALL IN PROGRESS
$ERR:   EMT     376                 ;GENERATE ALWAYS FATAL ERROR
        .BYTE   0,373               ;AND DISREGARD SOFT ERROR
$OVRHE:


; OVERLAY SEGMENT TABLE FOLLOWS:
; $OVTAB:       .WORD   <CORE ADDR>,<RELATIVE BLK>,<WORD COUNT>
; THREE WORDS PER ENTRY, ONE ENTRY PER OVERLAY SEGMENT.

; ALSO, THERE IS ONE WORD PREFIXED TO EACH OVERLAY REGION
; THAT IDENTIFIES THE SEGMENT CURRENTLY RESIDENT IN THAT REGION.
; THIS WORD IS AN INDEX INTO THE $OVTAB TABLE.
```

Figure 6-4
The Run-Time Overlay Handler

There is no special code or function call needed to use overlays but the following rules must be observed when referencing parts of the user program which might be overlaid.

1. Calls or branches to overlay segments must be made directly to entry points in the segment. Entry points are locations tagged with a global symbol (refer to Chapter 5, Section 5.5.10). For example, if ENTER is a global tag in an overlay segment:

```
JMP ENTER            is legal, but
JMP ENTER+6          is illegal.
```

2. Entries in overlay segments can be used only for transfer of control and not for referencing data within an overlay section (e.g., MOV ENTER,R4 is illegal if ENTER is in an overlay segment, but MOV #ENTER,R7 is legal because it is used for transfer of control). A violation of this rule cannot be detected by the assembler or Linker so no error is issued; however, it can cause the program to use incorrect data.

3. When calls are made to overlays, the entire return path must be in memory. This will happen if these rules are followed:

   Calls (with expected return) may be made from an overlay segment only to entries in the same segment, the root segment, or an overlay segment with a greater region number.

   Calls to entries in the same region as the call must be entirely within the same segment, not another segment in the same region.

   Jumps (with no expected return) can be made from an overlay segment to any entry in the program. However, jumps should not reference an overlay region whose number is lower than the region from which the last unreturned call was made (e.g., if a call was made from region 3, then no jumps should reference regions 1, 2 or 3 until the call has returned).

   Subroutines in the root segment may be called from overlay segments; in turn, they may call entries from the same overlay segment which called them, or from the root segment, or from another overlay segment with a greater region number. Such subroutines are considered part of the overlay segment which called them.

4. A .CSECT name cannot be used to pass control to an overlay. It will not cause the appropriate segment to be loaded into memory (e.g., JSR PC,OVSEC is illegal if OVSEC is used as a .CSECT name in an overlay). As stated in 1 above, a global symbol must be used to pass control from one segment to the next.

5. Channel 17(octal) cannot be used by the user program because overlays are read on that channel.

6. Object modules acquired from a library file cannot be placed into overlays.

7. Library files may not be specified on the same command line as an overlay.

8. Overlay regions must be specified in ascending order and are read-only. Unlike USR swapping, an overlay segment does not save the segment it is overlaying. Any tables, variables, or instructions that are modified within a given overlay segment are re-initialized to their original values in the SAV or REL file if that segment has been overlaid by another segment. Any variables or tables whose values must be maintained across overlays should be placed in the root segment.

9. .ASECTs of any size in an overlay foreground link are illegal; the error message ?ILL ASECT? is printed and the line is aborted.

The following information should be noted when writing FORTRAN overlays.

1. When dividing a FORTRAN program into a root segment and overlay regions (and subsequently dividing each overlay region into overlay segments), routine placement should be carefully considered. The user should always remember that it is illegal to call a routine located in a different overlay segment in the same overlay region, or an overlay region with a lower numeric value (as specified by the Linker overlay switch, /O:n) from the calling routine. The user should divide each overlay region into overlay segments which never need to be resident simultaneously (i.e., if segments A and B are assigned to region X, they cannot call each other because they occupy the same locations in memory).

2. The FORTRAN main program unit must be placed in the root segment.

3. In an overlay environment, subroutine calls and function subprogram references may refer only to one of the following:

   A FORTRAN library routine (e.g., ASSIGN, DCOS)

   A FORTRAN or assembly language routine contained in the root segment

   A FORTRAN or assembly language routine contained in the same overlay segment as the calling routine

   A FORTRAN or assembly language routine contained in a segment whose region number is greater than that of the calling routine

4. In an overlay environment, COMMON blocks must be placed so that they are resident when referenced. Blank COMMON is always resident since it is always placed in the root segment. All named COMMON must be placed either in the root segment, or into the segment whose region number is lowest of all segments which reference the COMMON block. A named COMMON block cannot be referenced by two segments in the same region unless the COMMON block appears in a segment of a lower region number. The Linker automatically places a COMMON block into the root segment if it is referenced by the FORTRAN main program or by a subprogram that is located in

the root segment. Otherwise the Linker places a COMMON block in the first segment encountered in the Linker command string that references that COMMON block.

5. All COMMON blocks which are initialized (by use of DATA statements) must be so initialized in the segment in which they are placed.

Refer to the RT-11/RSTS-11 FORTRAN IV User's Guide for more details.

The .ASECT never takes part in overlaying in any way (i.e., if part of an .ASECT is destroyed by overlay operations, it is not restored by the overlay handler).

The aforementioned sets of rules apply only to communications among the various modules that make up a program. Internally, each module must only observe standard programming rules for the PDP-11 (as described in the PDP-11 Processor Handbook and in Chapter 5).

It should be noted that the condition codes set by a user program are not preserved across overlay segment boundaries.

The Linker provides overlay services by including a small resident overlay handler (Figure 6-4) in the same file with the user program to be used at program run-time. This overlay handler plus some tables are inserted into the user's program beginning at the bottom address computed by the Linker. The Linker moves the user's program up in memory by an appropriate amount to make room for the overlay handler and tables, if necessary.

6.7 USING LIBRARIES

Libraries are specified in a command string in the same fashion as normal modules; they may be included anywhere in the command string, with the exception of overlay lines. If a global symbol is undefined at the time the library is encountered in the input stream and a module is included in the library which includes that global definition, that module is pulled from the library and linked into the load image. Only the modules needed to resolve references are pulled from the library; unused modules are not linked.

NOTE

Modules in one library may call modules from another library; however, the libraries must appear in the command string in the order in which they are called. For example, assume module X in library ALIB calls SQRT from the FORTRAN library. To correctly resolve all globals, the order of ALIB and the FORTRAN library should appear in the command line as:

```
    *Z=B,ALIB/F
or  *Z=B,ALIB,FORLIB
```

Module B is the root. It calls X from ALIB and brings X into the root. X in turn calls SQRT which is brought from FORLIB into the root.

FORTRAN libraries cannot precede their root segment in a command line as this creates a bad transfer address. For example:

```
*X=ROOT/F
*X=ROOT,FORLIB
```

are legal, but:

```
*X=FORLIB,ROOT
```

is not. Unpredictable results will occur.

6.7.1 User Library Searches

Object modules from the named user libraries built by the Librarian are relocated selectively and linked by the Linker. The RT-11 Linker searches a specified library file during the library pass as follows (refer to Figure 6-5 for a flowchart representation of this process):

1. If there are any undefined globals in the Linker's table when a library is encountered in the command string, proceed to step 2; otherwise skip this library (go to step 5).

2. Read the library directory.

3. If any of the undefined globals can be defined by a module in this library, include the relevant module into the linked output file; otherwise, go to step 5.

4. If any undefined globals remain in the Linker's table and they have not been looked for in the library, return to step 2; otherwise go to step 5.

5. Close the library file.

6. Go to the next element in the command string.

This search method allows modules to appear in any order in the library. Any number of libraries may be specified in a link, and they may be positioned anywhere, with the exception of overlay segments and the restrictions noted in Section 6.7.

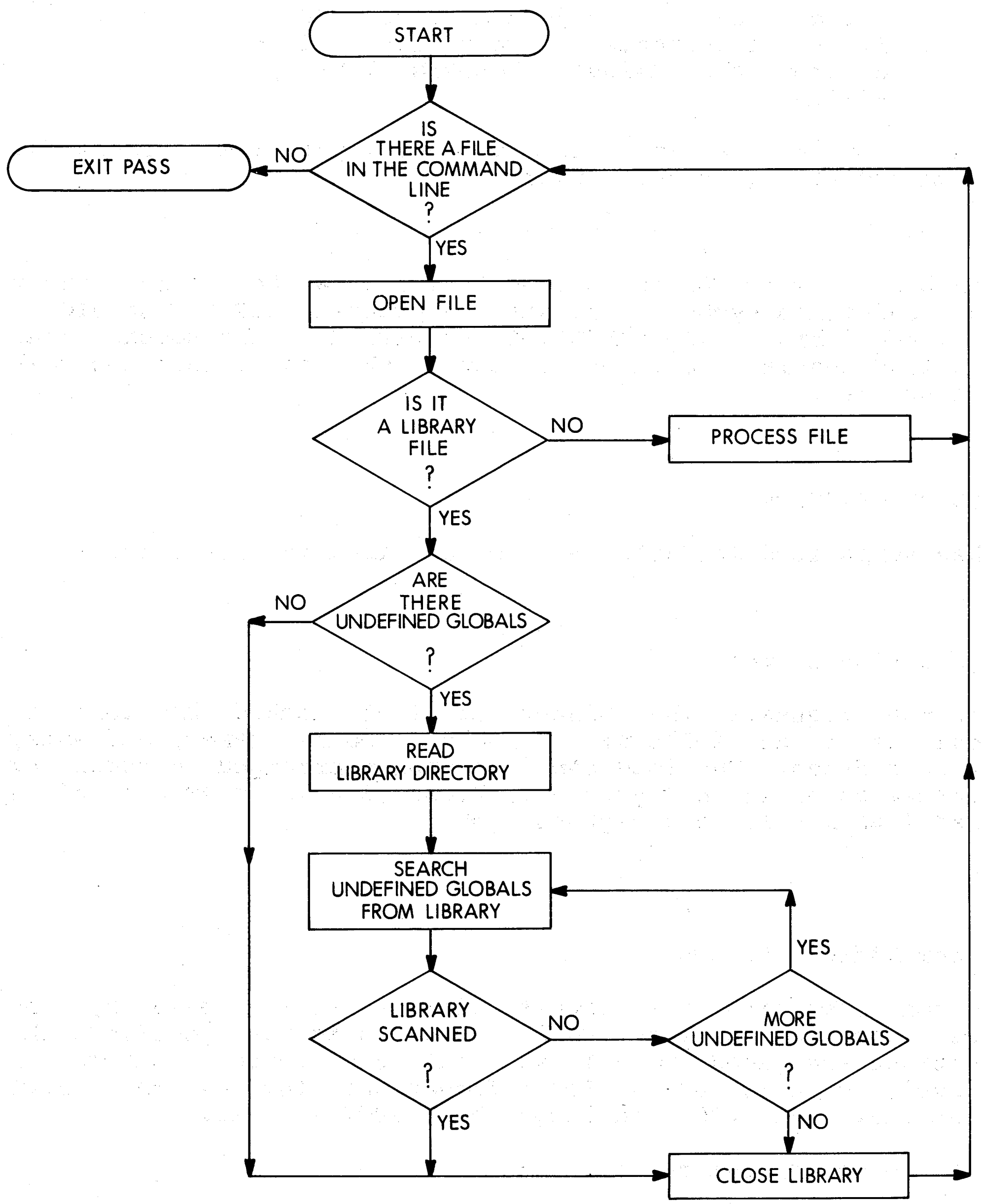


Figure 6-5
Library Searches

NOTE

For faster Linker performance, the user should specify all object files before library files, and all user library files before the system library files. For example:

```
*A=A,B,USELIB/F
```

where A and B are object modules, USELIB is a user-created library file, and /F denotes the default FORTRAN library, FORLIB.

Libraries are input to the Linker as any other input file. Assume the following command string to the Linker:

```
*TASK01.SAV,LP:=MAIN.OBJ,MEASUR.OBJ
```

This causes program MAIN.OBJ to be read from DK: as the first input file. Any undefined symbols generated by program MAIN.OBJ should be satisfied by the library file MEASUR.OBJ specified in the second input file. The load module, TASK01.SAV is put on DK: and a load map goes to the line printer.

6.8 SWITCH DESCRIPTION

The switches summarized in Table 6-1 are described in detail below.

6.8.1 Alphabetize Switch

The /A switch requests the Linker to list linked modules in alphabetical order as follows: .CSECTs, module names, and entry points within modules. The load map is normally arranged in order by module address as shown in Figure 6-1. Figure 6-6 is an example of an alphabetized load map for a background job.

6.8.2 Bottom Address Switch

The /B switch specifies the lowest address to be used by the relocatable code in the load module. When /B is not specified, the Linker positions the load module so that the lowest address is location 1000 (octal), or 0 for a foreground link. If the .ASECT length is greater than 1000, the length of .ASECT is used.

The form of the bottom switch is:

```
/B:n
```

n is a six-digit unsigned octal number which defines the bottom address of the program being linked. An error message results if n is not specified as part of the /B command.

If more than one /B switch is specified during the creation of a load module, the first /B switch specification is used.

NOTE

The bottom value must be an unsigned even octal number. If the value is odd, an error message is generated.

```
RT-11 LINK    V03-01     LOAD MAP
SQRT  .SAV               19-SEP-74

SECTION ADDR    SIZE     ENTRY   ADDR     ENTRY   ADDR     ENTRY   ADDR

. ABS.  000000  001000   $LRECL  000210   $NLCHN  000006   $USRSW  000000
                         $TRACE  004737   $V005A  000001
        001000  000220
        001220  001364   $OTI    001246
        002604  002300   DCO$    004164   ECO$    004156   FCO$    004152
                         GCO$    004144   ICI$    002612   ICO$    003720
                         OCI$    002604   OCO$    003712   RCI$    003006
                         $GET    002772
        005104  000160   ISN$    005104   LSN$    005124   $ISNTR  005110
                         $LSNTR  005130
        005264  000102   MOI$IA  005310   MOI$IM  005304   MOI$IS  005300
                         MOI$MA  005324   MOI$MM  005320   MOI$MS  005314
                         MOI$SA  005274   MOI$SM  005270   MOI$SS  005264
                         MOI$0A  005340   MOI$0M  005334   MOI$0S  005330
                         MOI$1A  005360   MOI$1M  005352   MOI$1S  005344
                         MOL$IS  005300   MOL$SS  005264   REL$    005300
        005366  000020   IFR$    005366   IFW$    005400
        005406  000046   EOL$    005406
        005454  000062   TVD$    005470   TVF$    005462   TVI$    005512
                         TVL$    005454   TVP$    005504   TVQ$    005476
        005536  000036   CAI$    005536   CAL$    005544
        005574  000174   SQRT    005574
        005770  000026   MOF$RA  006006   MOF$RM  005776   MOF$RP  006012
                         MOF$RS  005770
        006016  000044   BEQ$    006036   BGE$    006046   BGT$    006044
                         BLE$    006034   BLT$    006056   BNE$    006054
                         BRA$    006050   NMI$1I  006026   NMI$1M  006016
        006062  000072   EXIT    006074   FOO$    006062   STP$    006074
        006154  000002   $AOTS   006154
$ERRTB  006156  000100
$ERRS   006256  002637
        011116  001534   $FIO    011600
        012652  000202   $FMTDR  012652   $FMTDW  012702   $INITI  012750
        013054  000416   $CLOSE  013054
        013472  000106   LCI$    013472   LCO$    013540
        013600  000302   $GETRE  013600   $TTYIN  013722
        014102  000262   $PUTRE  014102
        014364  000106   $FCHNL  014364
        014472  000674   $OPEN   014472
        015366  000110   $DUMPL  015366
        015476  000414   $EOFIL  016046   $GETBL  015676   $PUTBL  015476
        016112  000042   $WAIT   016112

TRANSFER ADDRESS = 001000
HIGH LIMIT = 016154
```

Figure 6-6
Alphabetized Load Map for a Background Job

/B is illegal with foreground links. (0 is assumed.)

Example:

*OUTPUT,LP:=INPUT/B:500	Causes the input file to be linked starting at location 500 (octal).

6.8.3 Continue Switch

The Continue switch (/C) is used to allow additional lines of command string input. The /C switch is typed at the end of the current line and may be repeated on subsequent command lines as often as necessary to specify all input modules for which memory is available. If memory is exceeded, an error message is output. A /C switch is not entered on the last line of input.

Example:

*OUTPUT,LP:=INPUT/C *	Input is to be continued on the next line; the Linker prints an asterisk.

6.8.4 Default FORTRAN Library Switch

By indicating the /F switch in the command line, the FORTRAN library, FORLIB.OBJ on the default device (SY:), is linked with the other object modules specified; the user does not need to specify FORLIB. For example:

```
*FILE,LP:=AB.OBJ/F
```

The object module AB.OBJ and the FORTRAN library FORLIB are linked together to form a load module called FILE.SAV. (Note that the FORLIB default is SY:FORLIB.OBJ, not DK:FORLIB.OBJ.)

6.8.5 Include Switch

The /I switch allows subsequent entry at the keyboard of global symbols to be taken from any library and included in the linking process. When the /I switch is specified, the Linker prints:

LIBRARY SEARCH:

Reply with the list of global symbols to be included in the load module; type a carriage return to enter each symbol in the list. A carriage return alone terminates the list of symbols. This provides a method for forcing modules (which are not called by other modules) to be loaded from the library.

Example:

```
*OUTPUT,LP:=INPUT,XLIB/I
```

LIBRARY SEARCH:	Linker prints LIBRARY SEARCH:
A <CR> GETSYM <CR> CHAR <CR> CHFLG <CR> <CR>	User enters A, GETSYM, etc. which are to be included in the linking process. Each symbol is entered by typing a carriage return; the list is terminated by an additional carriage return.

6.8.6 LDA Format Switch

The LDA format switch (/L) causes the output file to be in LDA format instead of save image format. The LDA format file can be output to any device, including devices that are not block-replaceable such as paper tape or cassette, and is useful for files which are to be loaded with the Absolute Loader. The default extension .LDA is assigned when the /L switch is used.

The /L switch cannot be used in conjunction with the overlay switch (/O) or in foreground links (/R).

Example:

`*DK:OUT,LP:=IN,IN2/L`	Links disk files IN and IN2; outputs an LDA format file OUT.LDA to the system device and a load map to the line printer.

6.8.7 Modify Stack Address

The stack address, location 42, is the address which contains the user's stack pointer. The /M switch allows terminal keyboard specification of the user's stack address.

The form of the switch is:

/M:n

n is an even unsigned 6-digit octal number which defines the stack address. If n is not specified, the Linker prints the message:

STACK ADDRESS =

In this case, specify the global symbol whose value is the stack address. A number cannot be specified, and if a nonexistent symbol is specified, an error message is printed and the stack address is set to the system default (1000 for save files, 0 for REL).

Direct assignment (via .ASECT) of the stack address within the program takes precedence over assignment with the /M switch.

Example:

```
*OUTPUT=INPUT/M

STACK ADDRESS = BEG
```

6.8.8 Overlay Switch

The Overlay switch (/O) is used to segment the load module so that the entire program is not memory resident at one time (overlay feature). This allows programs larger than the available memory to be executed. The switch has the form:

/O:n

where n is an unsigned octal number (up to six digits in length) specifying the overlay region to which the module is assigned. The /O switch must follow (on the same line) the specification of the object modules to which it applies, and only one overlay region can be specified on a command line. Overlay regions cannot be specified on the first command line as this is the root segment. Therefore, the /C continuation switch must be used.

Co-resident overlay routines (a group of subroutines which occupy the overlay region and segment at the same time) are specified as follows:

```
*OBJA,OBJB,OBJC/O:n/C
*OBJD,OBJE/O:m/C
.
.
.
```

All modules mentioned until the next /O switch will be co-resident overlay routines. If at a later time the /O switch is given with the same value previously used (same overlay region), then the corresponding overlay area is opened for a new group of subroutines. The new group of subroutines will occupy the same locations in memory as the first group, but not at the same time. For example, if subroutines in object modules R and S are to be in memory together, but are never needed at the same time as T, then the following commands to the Linker make R and S occupy the same memory as T (but at different times):

```
*MAIN,LP:=ROOT/C
*R,S/O:1/C
*T/O:1
```

The above could also be written as:

```
*MAIN,LP:=ROOT/C
*R/O:1/C
*S/C
*T/O:1
```

Example:

```
*OUTPUT,LP:=INPUT/C          Establishes two overlay
*OBJA/O:1/C                  regions
*OBJB/O:2
```

Overlays must be specified in order of increasing region number. For example:

```
.R LINK
*A=A/C
*B/O:1/C
*C/O:1/C
*D/O:1/C
*E,F/O:2/C
*G/O:3
```

The following overlay specification is illegal since the overlay regions are given in a random numerical order (an error message is printed in each case):

Linker

```
.R LINK
*A=A/C
*D/O:2/C
*B/O:1/C
/O IGNORED
*C/O:1/C
/O IGNORED
*G/O:3/C
*H/O:3/C
*E,F/O:2
/O IGNORED
```

6.8.9 REL Format Switch

The REL format switch (/R) causes the output file to be in REL format for use as a foreground job under the F/B Monitor. REL format files must be used in a foreground job (they may not be used under a Single-Job system). The /R switch assigns the default extension .REL to the output file.

Example:

`*DT2:FILEO,LP:=FILEI,NEXT/R`	Disk files FILEI and NEXT are linked and output to DT2 as FILEO.REL; a load map is output to the line printer.

The /B and /L switches cannot be used with /R since a foreground REL job has no bottom address and is always relocated by FRUN. A ?BAD SWITCH? error message is generated if this is attempted.

6.8.10 Symbol Table Switch

Use of the symbol table switch in the command line instructs the Linker to allow the largest possible memory area for its symbol table at the expense of making the link process slower. With the /S switch, library directories are not made resident in memory, but are left on disk. For example:

```
*OUTF,LP:=INPUT.OBJ,LIBR1.OBJ,LIBR2.OBJ/S
```

The directories of the library files LIBR1 and LIBR2 are not brought into memory, resulting in more room in the symbol table but longer link time.

If the /S switch is not used and the memory available to the Linker is approximately 10K or larger, the library directory is brought into memory (providing there is room); the directory is kept there until the library has been completely processed, thus reducing the size of the Linker's symbol table. If there is not enough room in memory for the directory (as is the case in an 8K system), the Linker determines this and leaves the directory on disk regardless of whether the /S switch was used or not.

The /S switch should be used only if an attempt to link a program failed because of symbol table overflow. Often, use of /S will allow the program to link.

6.8.11 Transfer Address Switch

The transfer address is the address at which a program is to be started when executed via an R, RUN, or FRUN command. The Transfer Address switch (/T) allows terminal keyboard specification of the start address of the load module to be executed. This switch has the form:

```
/T:n
```

where n is a six-digit unsigned octal number which defines the transfer address. If n is not specified, the message:

```
TRANSFER ADDRESS =
```

is printed. In this case, specify the global symbol whose value is the transfer address of the load module, followed by a carriage return. A number cannot be specified in answer to this message. When a nonexistent symbol is specified, an error message is printed and the transfer address is set to 1 (so that the program cannot be executed).

If the transfer address specified is odd, the program does not start after loading and control returns to the monitor.

Direct assignment (.ASECT) of the transfer address within the program takes precedence over assignment with the /T switch. The transfer address assigned with a /T has precedence over that assigned with a .END assembly directive.

Example:

```
*PROG=PROG1,PROG2,ODT/T
TRANSFER ADDRESS =
O.ODT
```

The files PROG1.OBJ,PROG2.OBJ and ODT.OBJ are linked together and started at ODT's transfer address.

6.9 LINKER ERROR HANDLING AND MESSAGES

The following error messages can be output by the Linker. The messages enclosed in question marks are output to the terminal; the other messages are only warnings and are included in the load map. If a load map is not requested in the command string, all messages are output to the terminal.

Message	Meaning
ADDITIVE REF OF xxxxxx AT SEGMENT # yyyyyy	Rule 1 of overlay rules explained in Section 6.6 has been violated. xxxxxx represents the entry point; yyyyyy represents the segment number.
?/B NO VALUE?	The /B switch requires an unsigned even octal number as an argument.
?/B ODD VALUE?	The argument to the /B switch was not an unsigned even octal number.
?BAD GSD?	There is an error in the global symbol directory (GSD). The file is probably not a

legal object module. This error message occurs on pass 1 of the Linker.

BAD OVERLAY AT SEG # yyyyyy	Overlay tries to store text outside its region; check for a .ASECT in overlay. yyyyyy represents the segment number.
?BAD RLD?	There is an invalid relocation directory (RLD) command in the input file; the file is probably not a legal object module. The message occurs on pass 2 of the Linker.
?BAD SWITCH?	LINK did not recognize a switch specified on the first command line. On a subsequent command line, a bad switch causes this warning message but does not restart the Linker.
?BAD x SWITCH IGNORED?	LINK did not recognize a switch (x) specified in the command line. The switch is ignored and linking continues.
BYTE RELOCATION ERROR AT xxxxxx	Linker attempted to relocate and link byte quantities but failed. xxxxxx represents the address at which the error occurred. Failure is defined as the high byte of the relocated value (or the linked value) not being all zero. In such a case, the value is truncated to 8 bits and the Linker continues processing (for save image and LDA files only; byte relocation is completely illegal for REL files).
?CORE?	There is not enough memory to accommodate the command or the resultant load module.
?ERROR ERROR?	An error occurred while the Linker was in the process of recovering from a previous system or user error.
?ERROR IN FETCH?	The device is not available.
?FILE NOT FND?	Input file was not found.
?FORLIB NOT FND?	The user indicated via the /F switch that the FORTRAN library, FORLIB, was to be linked with the other object modules in the command line, and the Linker could not find FORLIB.OBJ on the system device.
?HARD I/O ERROR?	A hardware error occurred; try the operation again.
?ILL ASECT?	The user has attempted to place an .ASECT above 1000 in a foreground link or to place an .ASECT into an overlay foreground link.

?LDA FILE ERROR?	There was a hardware problem with the device specified for LDA output or the device was full.
?/M ODD VAL?	An odd value has been specified for the stack address. Control returns to the Linker and another command line may be indicated.
?MAP FILE ERROR?	There was a hardware problem with the device specified for map output or the device is full.
MULT DEF OF xxxxxx	The symbol, xxxxxx, was defined more than once.
?NO INPUT?	No input files were specified.
/O IGNORED	Overlays have been specified in the wrong order (see overlay restrictions); the overlay switch is ignored.
?OUTPUT FULL?	The output device was full.
?REL FILE ERR?	The Linker encountered a problem writing the REL file; try the operation again.
?SAV FILE ERR?	The Linker encountered a problem writing the save image file; try the operation again.
?STACK ADDRESS UNDEFINED OR IN OVERLAY?	The stack address specified by the /M switch was either undefined or in an overlay. The stack address is set to the system default.
?SYMBOL TABLE OVERFLOW?	There were too many global symbols used in the program. Retry the link using the /S switch. If the error still occurs, the link cannot take place in the available memory.
?TOO MANY OUTPUT FILES?	The Linker allows specification of only two output files.
TRANSFER ADDRESS UNDEFINED OR IN OVERLAY?	The transfer address was not defined or was in an overlay.
UNDEF GLBLS	A load map was requested and undefined globals existed.
UNDEFINED GLOBALS xxxxxx xxxxxx . . .	The globals listed (xxxxxx) were undefined. If a load map is requested, this condition also causes the warning message, UNDEF GLBLS, to be printed on the terminal.

CHAPTER 7

LIBRARIAN

The RT-11 system provides the user with the capability of maintaining libraries which may be composed of functions and routines of his choice. Each library is a file containing a library header, library directory (or entry point table), and one or more object modules. The object modules in a library file may be routines which are repeatedly used in a program, routines which are used by more than one program, or routines which are related and simply gathered together for ease in usage--the contents of the library file are determined by the user's needs. An example of a typical library file is the FORTRAN library, FORLIB.OBJ. This library is provided with the FORTRAN package and contains all the mathematical functions needed for normal usage.

Object modules in a library file are accessed from another program via calls to their entry points; the object modules are linked with the program which uses them by the Linker (Chapter 6) to produce a single load module.

The RT-11 Librarian (LIBR) allows the user to create, update, modify, list, and maintain library files.

7.1 CALLING AND USING LIBR

The RT-11 Librarian is called from the system device by entering the command:

```
R LIBR
```

in response to the dot printed by the Keyboard Monitor. The Command String Interpreter prints an asterisk at the left margin on the console terminal when it is ready to accept a command line.

Type CTRL C to halt the Librarian at any time and return control to the monitor. To restart the Librarian, type R LIBR or the REENTER command in response to the monitor's dot.

7.2 USER SWITCH COMMANDS AND FUNCTIONS

The user maintains library files through the use of switch commands. Functions which can be performed include object module deletion, insertion and replacement, library file creation, and listing of a library file's contents.

7.2.1 Command Syntax

LIBR accepts command strings in the following general format:

```
*dev:lib,dev:list=dev:input/s1/s2/s3
```

where:

dev:	represents a legal RT-11 device specification
lib	represents the library file to be created or updated
list	represents a listing file for the library's contents
input	represents the filenames of the input object modules
/s1,...	represents one or more of the switches listed in Table 7-1

Devices and filenames are specified by the user in the standard RT-11 command string syntax, with default extensions assigned as follows:

File	Extension
list file:	.LLD
library file:	.OBJ
input files:	.OBJ

If no device is specified, the default device (DK:) is assumed.

Each input file is made up of one or more object modules, and is stored on a given device under a specific filename and extension. Once an object module has been inserted into a library file, the module is no longer referenced by the name of the file of which it was a part, but by its individual module name. (This module name has been assigned by the assembler either via a .TITLE statement in the assembly source program, or, if no .TITLE statement is present, with the default name .MAIN.; see Chapter 5.) Thus, for example, the input file FORT.OBJ may exist on DT2: and may contain an object module called ABC. Once the module is inserted into a library file, reference is made only to ABC (not FORT.OBJ).

7.2.2 LIBR Switch Commands

Table 7-1 summarizes the switches available for use under RT-11 LIBR. Switches are explained in detail following the table.

Table 7-1
LIBR Switches

Switch	Position In Command String	Meaning
/C	Any line but last	Command continuation; the command is too long for the current line and is continued on the next line
/D	1st line only	Delete; delete modules from a library file
/G	1st line only	Global deletion; delete entry points from the library directory
/R	1st line only	Replace; replace modules in a library file
/U	1st line only	Update; insert and replace modules in a library file

There is no switch to indicate module insertion. The function of inserting a module into a library file is assumed in the absence of other switches.

7.2.2.1 Command Continuation Switch - The Command Continuation switch is necessary whenever there is not enough room to enter a command string on one line and additional lines are needed. The /C switch is typed at the end of the current line and may be repeated at the end of subsequent command lines as often as necessary as long as memory is available; if memory is exceeded, an error message is output. A /C switch is not entered on the last line of input.

Command Format:

```
*dev:lib,dev:list=dev:input1,dev:input2,...,/C
*dev:inputn
```

where:

dev: represents a device specification

lib represents the filename of the library to be created or updated

list represents the filename of a listing file containing the library file's contents

input represents the filenames of the input modules to be inserted into the library

/C represents the Continuation switch, indicating that the command is to be continued on the following line

Examples:

```
*ALIB,LIBLST=DT1:MAIN,TEST,FXN/C
*DT1:TRACK
```

In this example, a library file is created on the default device (DK:) under the filename ALIB.OBJ; a listing of the library file's contents is created as LIBLST.LLD also on the default device; the filenames of the input modules are MAIN.OBJ, TEST.OBJ, FXN.OBJ, and TRACK.OBJ, all from DT1.

```
*BLIB=MAIN/C
*RK1:TEST/C
*RK0:FXN/C
*DT1:TRACK
```

A library file is created on the default device, (DK:) under the name BLIB. No listing is produced. Input files are MAIN from the default device, TEST from RK1:, FXN from RK0: and TRACK from DT1.

Another way of writing this command line is:

```
*BLIB=MAIN,RK1:TEST,RK0:FXN/C
*DT1:TRACK
```

7.2.2.2 Creating a Library File - A library file is created whenever a filename is indicated on the output side of a command line which does not represent a list file.

Command Format:

```
*dev:lib=dev:input1,...,dev:inputn
```

where:

dev:	represents a device specification
lib	represents the filename of the library to be created
input	represents the filenames of the input modules to be inserted into the new library

Example:

```
*NEWLIB=FIRST,SECOND
```

A new library called NEWLIB.OBJ is created on the default device (DK:). The modules which will make up this library file are in the files FIRST.OBJ and SECOND.OBJ, both on the default device.

Assume this command line is next entered:

```
*NEWLIB,LIST=THIRD,FOURTH
```

The already existing library file NEWLIB is destroyed when the new library file is created. A listing of the library file's contents is created under the filename LIST, and the object modules in the files THIRD and FOURTH are inserted into the library file NEWLIB.

7.2.2.3 Inserting Modules Into a Library - The Insert function is assumed whenever an input file does not have an associated switch; the modules in the file are inserted into the library file named on the output side of the command string. Any number of input files are allowed. If an attempt is made to insert a file which contains an entry point or .CSECT having the same name as an entry point or .CSECT already existing in the library file, a warning message is printed. However, the library file is updated, ignoring the entry point or .CSECT in error, and control returns to the CSI; the user may enter another command string.

Although the user may insert object modules which exist under the same name (as assigned by the .TITLE statement) this practice is not recommended because of the difficulty involved when replacing or updating these modules (refer to Sections 7.2.2.4 and 7.2.2.7).

NOTE

> The library operations of module insertion, replacement, deletion, merge, and update are actually performed in conjunction with the library file creation operation. Therefore, the library file to which the operation is directed must be indicated on both the input and output sides of the command line, since effectively a "new" output library file is created each time the operation is performed. The library file must be specified first in the input field.

Command Format:

```
*dev:lib=dev:lib,dev:input1,...,dev:inputn
```

where:

dev:	represents a device specification
lib	represents the filename of an existing library file
input	represents the filenames of the modules to be inserted into the library file

Example:

```
*DXY=DXY,DT1:FA,FB,FC
```

The modules included in the files FA.OBJ, FB.OBJ, and FC.OBJ on DT1: are inserted into a library file named DXY.OBJ on the default device. The library header and Entry Point Table of the library file are updated accordingly (see Section 7.4).

7.2.2.4 Replace Switch - The Replace function is used to replace modules in a library file. All modules contained in the file(s) indicated as input will replace existing modules of the same names in the library file specified as output.

An error message is printed and no modules are replaced if an old module does not exist under the same name as an input module, or if the user specifies the /R switch on a library file. /R must follow each input filename containing modules for replacement.

Command Format:

```
*dev:lib=dev:lib,input1/R,...,dev:inputn/R
```

where:

dev:	represents a device specification
lib	represents the filename of an existing library file
input	represents the names of the files containing modules to be replaced
/R	represents the Replace switch

Examples:

```
*TFIL=TFIL,INA,INB/R,INC
```

This command line indicates that the modules in the file INB.OBJ are to replace existing modules of the same names in the library file TFIL.OBJ. The object modules in the files INA.OBJ and INC.OBJ are to be added. All files are stored on the default device DK:.

```
*XFIL=TFIL,INA,INB/R,INC
```

The same operation occurs here as in the preceding example, except that this updated library file is assigned the new name XFIL.

7.2.2.5 Delete Switch - The Delete switch deletes modules and all their associated entry points from the library.

Command Format:

```
*dev:lib=dev:lib/D
```

where:

dev:	represents the device on which the library file exists
lib	represents the filename of an existing library file
/D	represents the Delete switch; may be positioned anywhere on the input side of the command line

When the /D switch is used, the Librarian prints:

```
MOD NAME:
```

The user should respond with the name of the module to be deleted followed by a carriage return; he may continue until all modules to be deleted have been entered. Typing only a carriage return (either on a line by itself or immediately after the MOD NAME: message) terminates input and initiates execution of the command line.

Examples:

```
*DT3:TRAP=DT3:TRAP/D


MOD NAME:
SGN <CR>
TAN <CR>
<CR>
```

The modules SGN.OBJ and TAN.OBJ are deleted from the library file TRAP.OBJ on DT3:.

```
*LIBFIL=LIBFIL/D,ABC/R,DEF


MOD NAME:
FIRST <CR>
<CR>
```

The module FIRST.OBJ is deleted from the library (LIBFIL); the module ABC.OBJ replaces an old module of the same name in the library, and the modules in the file DEF.OBJ are inserted into the library.

```
*LIBFIL=LIBFIL/D


MOD NAME:
X<CR>
X<CR>
<CR>
```

Two modules of the same name are deleted from the library file LIBFIL (module names are assigned with the .TITLE statement as described in Section 7.2.1).

7.2.2.6 Delete Global Switch - The Delete Global switch gives the user the ability to delete a specific entry point from a library file's Entry Point Table.

Command Format:

```
*dev:lib=dev:lib/G
```

where:

dev:	represents the device on which the library file exists
lib	represents the filename of an existing library file

/G — represents the Delete Global switch; may be positioned anywhere on the input side of the command line

When the /G switch is used, the Librarian prints:

ENTRY POINT:

The user should respond with the name of the entry point to be deleted followed by a carriage return; he may continue until all entry points to be deleted have been entered. Typing only a carriage return (either on a line by itself or immediately after the ENTRY POINT: message) terminates input and initiates execution of the command line.

Example:

```
*ROLL=ROLL/G

ENTRY POINT:
NAMEA <CR>
NAMEB <CR>
<CR>
```

This command instructs LIBR to delete the entry points NAMEA and NAMEB from the entry point table found in the library file ROLL.OBJ on DK:.

Since entry points are only deleted from the Entry Point Table (and not from the library itself) whenever a library file is updated, all entry points that were previously deleted are restored unless the /G switch is again used to delete them. This feature allows the user to recover from inadvertently deleting the wrong entry point.

7.2.2.7 Update Switch - The Update switch allows the user to update a library file by combining the insert and replace functions. If the object modules included in an input file in the command line already exist in the library file, they are replaced; if not, they are inserted. (No error messages are printed when using the Update function as might occur under the Insert and Replace functions.) /U must follow each input file containing modules to be updated.

Command Format:

```
*dev:lib=dev:lib,dev:input1/U,...,dev:inputn/U
```

where:

dev:	represents a device specification
lib	represents the filename of an existing library file
input	represents the names of files containing object modules to be updated.
/U	represents the Update switch

Examples:

```
*BALIB=BALIB,FOLT/U,TAL,BART/U
```

This command line instructs LIBR to update the library file BALIB.OBJ on the default device. First the modules in FOLT.OBJ and BART.OBJ replace old modules of the same names in the library file, or if none already exist under their names, the modules are inserted. Then the modules from the file TAL.OBJ are inserted; an error message is printed if the name of the module in TAL.OBJ already exists.

```
*XLIB=XLIB/D,Z/U/G

MOD NAME:
 X <CR>
 X <CR>
<CR>

ENTRY POINT:
SEC <CR>
SEC1 <CR>
<CR>
```

There are two object modules of the same name (X) in both Z and XLIB; these are first deleted from XLIB. This ensures that both modules X in file Z are correctly placed into the library. Entry points SEC and SEC1 are also deleted from the Entry Point Table, but automatically return when the libary (XLIB) is updated.

7.2.2.8 Listing the Directory of a Library File - The user may specify that a listing of the contents of a library file be output by indicating both the library file and a list file in the command line. Since a library file is not being created or updated, it is not necessary to indicate the filename on the output side of the command line; however a comma must be used to designate a null output library file.

Command Formats:

```
*,LP:=dev:lib
```

or

```
*,dev:list=dev:lib
```

where:

dev:	represents a device specification
lib	represents the file name of an existing library file
LP:	indicates the listing is to be sent directly to the line printer
list	represents a list file of the library file's contents

Examples:

```
*,DT2:LIST=LIBFIL
```

This command line outputs to DECtape 2 as LIST.LLD a listing of the contents of the library file LIBFIL.OBJ on the default device.

```
*,LP:=FLIB
```

This command outputs on the line printer a listing of all modules in the library file FLIB.OBJ stored on the default device. Assuming this library is composed of modules STOP, WAIT, and IMUL, is 2 blocks long, was created on September 6, 1974, and the listing was requested on September 6, 1974, the directory format appears as follows:

```
RT-11 LIBRARIAN X02-05          6-SEP-74
FLIB            6-SEP-74        2 BLOCKS

MODULE          ENTRY/CSECT     ENTRY/CSECT     ENTRY/CSECT

STOP            STP$
WAIT            $WAIT
IMUL            MUI$IS          MUI$MS          MUI$PS
                MUI$S$          $MLI
```

7.2.2.9 Merging Library Files - Two or more library files may be merged under one filename by indicating all the library files to be merged in a single command line. The individual library files are not deleted following the merge.

Command Format:

```
*dev:lib=dev:inputl,...,dev:inputn
```

where:

dev:	represents a device specification
lib	represents the name of the library file which will contain all the merged files (if a library file

already exists under this name, it must also be indicated in the input side of the command line in order to be included in the merge)

input — represents the library files to be merged together

Thus, the command:

```
*MAIN=MAIN,TRIG,STP,BAC
```

combines library files MAIN.OBJ, TRIG.OBJ, STP.OBJ, and BAC.OBJ under the existing library file name MAIN.OBJ; all files are on the default device DK:.

```
*FORT=A,B,C
```

This command creates a library file named FORT.OBJ and merges existing library files A.OBJ, B.OBJ, and C.OBJ under the filename FORT.OBJ.

NOTE

Library files that have been combined under PIP are illegal as input to both the Librarian and the Linker.

7.3 COMBINING LIBRARY SWITCH FUNCTIONS

Two or more library functions may be requested in the same command line. The Librarian performs functions in the following order:

1. /C
2. /D
3. /G
4. /U
5. /R
6. Insertions
7. Listing

Example:

```
*FILE,LP:=FILE/D,MODX,MODY/R


MOD NAME:
XYZ<CR>
A<CR>
<CR>
```

Functions in this example are performed in order, as follows:

1. Delete modules XYZ.OBJ and A.OBJ from the library file FILE.OBJ

2. Replace any duplicate of the module in the file MODY.OBJ

3. Insert the modules in the file MODX.OBJ

4. List the contents of FILE.OBJ on the line printer

7.4 FORMAT OF LIBRARY FILES

A library file is a contiguous file consisting of a header, an Entry Point Table (library directory) and one or more library object modules, as illustrated in Figure 7-1:

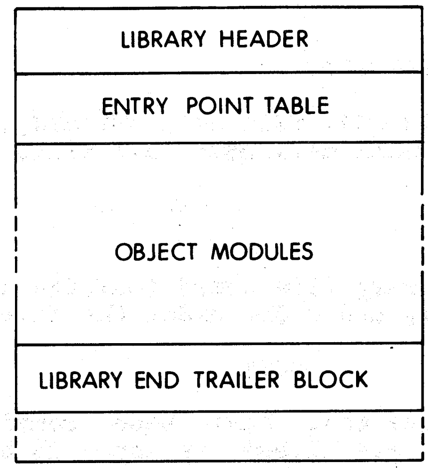


Figure 7-1
General Library File Format

The following paragraphs describe in detail each component of a library file.

7.4.1 Library Header

The header section of a library file contains 17 (decimal) words which describe the current status of the file (refer to Figure 7-2). This includes information relating to the version of the Librarian in use, the date and time of file creation or update, the relative starting address of the Entry Point Table (EPT), the number of EPT entries available and in use, and the placing of the next module to be inserted into the library file. The contents of the library header are updated as the library file is modified, so that LIBR can always quickly and easily access the information it needs to perform its functions. Figure 7-2 illustrates the header format.

Word	Description
1	FORMATTED BINARY BLOCK HEADER
56_8	
7	LIBRARIAN CODE
x	VERSION NUMBER
0	RESERVED
x	YEAR-MONTH-DAY
0	RESERVED
0	
0	
0	
0	
12_8	EPT RELATIVE START ADDRESS
x1	EPT ENTRIES ALLOCATED IN BYTES
0	EPT ENTRIES AVAILABLE (NOT USED IN VERSION 1)
x2	NEXT INSERT RELATIVE BLOCK NUMBER
x3	NEXT BYTE WITHIN BLOCK
0	NOT USED (MUST BE ZERO)

Figure 7-2
Library Header Format

7.4.2 Entry Point Table (Library Directory)

The Entry Point Table is located immediately after the library header. It is composed of four-word entries which include the names, addresses, and entry points of all object modules in the library file. The first two words of an entry in the EPT contain the Radix 50 name by which an entry point, CSECT, or module is referenced. The third word provides a pointer to the object module where an entry point is defined. The fourth word contains the total number of CSECTs in the object module (information needed by the Linker), and the relative byte within the block pointing to the object module's starting point, as shown in Figure 7-3.

Offset		
0	SYMBOL (RAD 50)	
2	SYMBOL (RAD 50)	
4	(bit 15)	ADDRESS OF BLOCK
6	# OF CSECTS IN OBJECT MODULE	RELATIVE BYTE IN BLOCK

BIT 15=1-MODULE NAME
=Ø-CSECT OR ENTRY POINT NAME
RELATIVE BYTE MAXIMUM=777_8
CSECTS MAXIMUM =177_8

Figure 7-3
Format of Entry Point Table

7.4.3 Object Modules

Object modules follow the Entry Point Table. An object module consists of three main types of data blocks: a global symbol directory, text blocks, and a relocation directory. The information contained in these data blocks is used by the Linker during creation of a load module.

7.4.4 Library End Trailer

Following all object modules in a library file is a specially coded library end trailer which signifies the end of the file. This trailer is illustrated in Figure 7-4.

	1	FORMATTED BINARY HEADER
	10	FORMATTED BINARY LENGTH
	10	TYPE CODE
	0	NOT USED (MUST BE ZERO)
	357	CHECKSUM BYTE

Figure 7-4
Library End Trailer

7.5 LIBR ERROR MESSAGES

The following error messages are printed following incorrect use of LIBR; if any errors result during library processing, the user must reenter the command.

Message	Meaning
?BAD LIBR?	The user has attempted to build a library file containing no directory entries or he has given an illegally constructed library file to the Librarian as input.
?BAD OBJ?	A bad object module was detected during input.
?CSECT ERROR?	The user has extended beyond the allowable .CSECT space for an object module to be placed in the library (i.e., the object module contains greater than 127(decimal) .CSECTs).

Message	Meaning
?DEV FULL?	The device is full; LIBR is unable to create or update the indicated library file. The CSI prints an asterisk and waits for the user to enter another command line.
?FIL NOT FND?	One of the input files indicated in the command line was not found. The CSI prints an asterisk; the command may be reentered.
?ILL CMD?	An illegal command was used in the command line. The CSI prints an asterisk; the command may be reentered.
xxxxxx ILL DEL	An attempt was made to delete from the library's directory a module or an entry point that does not exist; xxxxxx represents the module or entry point name. The name is ignored and processing continues.
?ILL DEV?	An illegal device was specified in the command line. The CSI prints an asterisk; the command may be reentered.
xxxxxx ILL INS	An attempt was made to insert a module into a library which contains the same entry point as an existing module. xxxxxx represents the entry point name. The entry point is ignored but the module is still inserted into the library.
xxxxxx ILL REPL	An attempt was made to replace in the library file a module which does not already exist. xxxxxx represents the module name. The module is ignored and the library is built without it.
?IN ERR?	An unrecoverable hardware/software error has occurred while processing an input file. The CSI prints an asterisk and waits for another command to be entered.
?LIBR FIL ILL REPL?	The user has specified that a library file be replaced by another library file. Only object modules can be replaced.
?NO CORE?	Available free memory has been used up. The current command is aborted and the CSI prints an asterisk; a new command may be entered.
?OUT ERR?	An unrecoverable hardware/software error has occurred while processing an output file. This may indicate that there is not enough space left on the device to create the library file, even though there may be enough directory entries. The CSI prints an asterisk and waits for the user to enter another command.

CHAPTER 8

ON-LINE DEBUGGING TECHNIQUE

RT-11 On-line Debugging Technique (ODT) is a system program that aids in debugging assembled and linked object programs. From the keyboard, the user interacts with ODT and the object program to:

1. Print the contents of any location for examination or alteration.

2. Run all or any portion of an object program using the breakpoint feature.

3. Search the object program for specific bit patterns.

4. Search the object program for words which reference a specific word.

5. Calculate offsets for relative addresses.

6. Fill a single word, block of words, byte or block of bytes with a designated value.

The assembly listing of the program to be debugged should be readily available when ODT is being used. Minor corrections to the program can be made on-line during the debugging session, and the program may then be run under control of ODT to verify any changes made. Major corrections, however (such as a missing subroutine), should be noted on the assembly listing and incorporated in a subsequent updated program assembly.

8.1 CALLING AND USING ODT

ODT is supplied as a relocatable object module. It can be linked with the user program (using the RT-11 Linker) for an absolute area in memory and loaded with the user program. When linked with the user program, ODT should reside in low memory, starting at 1000, to accommodate its stack.

Once loaded in memory with the user program, ODT has three legal start or restart addresses. The lowest (O.ODT) is used for normal entry, retaining the current breakpoints. The next (O.ODT+2) is a restart address which clears all breakpoints and re-initializes ODT saving the general registers and clearing the relocation registers. The last address (O.ODT+4) is used to reenter ODT. A reenter saves the Processor Status and general registers and removes the breakpoint instructions from the user program. ODT prints the Bad Entry (BE) error message. Breakpoints which were set are reset on the next ;G command. (;P is illegal after a BE message.) The ;G and ;P commands are used to run a program and are explained in Section 8.3.7.

The absolute address used is the address of the entry point O.ODT shown in the Linker load map. O.ODT is always the lowest address of ODT+172, i.e., O.ODT is relative location 172 in ODT.

NOTE

> If linked with an overlay structured file, ODT should reside in the root segment so it is always in memory. A breakpoint inserted in an overlay will be destroyed if it is overlaid during program execution.

If ODT is being used in a Foreground/Background environment with another job running, ODT's priority bit must be set to 0 as follows:

```
*$P/000007 0 <CR>
```

This puts ODT into the wait state at level 0, not 7. If this is not done, all interrupts (including clock) will be locked out while ODT is waiting for terminal input.

Examples:

1. ODT Linked with the User Program:

.GET USER.SAV	User program previously linked to ODT is brought into memory.
.START 1172	Value (1172) of entry point O.ODT (determined from Linker load map) is used to start ODT.
ODT V01-01 *	

2. Loading ODT with the User Program:

.GET ODT.SAV	ODT is loaded into memory.
.GET USER.SAV	User program is loaded into memory.
.START 40172	Assuming ODT has been linked for a bottom address of 40000, ODT starts.
ODT V01-02 *	

3. Restarting ODT Clearing Breakpoints:

.START 1174 *	Assuming ODT was originally linked for a bottom address of 1000, this command (O.ODT+2) re-initializes ODT and clears any previous breakpoints.

4. Reenteringg ODT:

```
.START 1176

BE001212
*
```

Assuming ODT was linked for a bottom address of 1000, the value of O.ODT 1172+4 is used as the start address.

5. Using ODT with Foreground/Background Jobs:

It is possible to use ODT to debug programs written as either background or foreground jobs. In the background or under the Single-Job Monitor, ODT can be linked with the program as described in Example 1 above.

To debug a program in the foreground area, it is recommended that ODT be run in the background while the program to be debugged is in the foreground. The sequence of commands to do this is:

```
.FRUN PROG/F
LOADED AT xxxxxx
.RUN ODT

ODT  V01-01
*xxxxxx;0R
*$F/000000 0<CR>
*0;G

.RSU
```

Load the foreground program. The first address of the job is printed (xxxxxx) Run ODT in the background and set a relocation register to the start of the job. $F is the format register. It should be cleared to enable proper address print out. 0;G starts the Keyboard Monitor again, and .RSU starts the foreground job.

The copy of ODT used must be linked low enough so that it will fit in memory along with the foreground job.

NOTE

Since ODT uses its own terminal handler, it cannot be used with the display hardware. If GT ON has been typed, ODT will ignore it and direct I/O only to the console terminal.

8.1.1 Return to Monitor, CTRL C

If ODT is awaiting a command, a CTRL C from the keyboard calls the RT-11 Keyboard Monitor. The monitor responds with a ↑C on the terminal and awaits a Keyboard Monitor command. (The monitor REENTER command may be used to reenter ODT only if the user program has set the reenter bit. Otherwise ODT is reentered at address O.ODT+4 as shown above.)

8.1.2 Terminate Search, CTRL U

If typed during a search printout, a CTRL U terminates the search and ODT prints an asterisk.

8.2 RELOCATION

When the assembler produces a binary object module, the base address of the module is taken to be location 000000, and the addresses of all program locations as shown in the assembly listing are indicated relative to this base address. After the module is linked by the Linker, many values within the program, and all the addresses of locations in the program, will be incremented by a constant whose value is the actual absolute base address of the module after it has been relocated. This constant is called the relocation bias for the module. Since a linked program may contain several relocated modules each with its own relocation bias, and since, in the process of debugging, these biases will have to be subtracted from absolute addresses continually in order to relate relocated code to assembly listings, RT-11 ODT provides an automatic relocation facility.

The basis of the relocation facility lies in eight relocation registers, numbered 0 through 7, which may be set to the values of the relocation biases at different times during debugging. Relocation biases should be obtained by consulting the memory map produced by the Linker. Once set, a relocation register is used by ODT to relate relocatable code to relocated code. For more information on the exact nature of the relocation process, consult Chapter 6, the RT-11 Linker.

8.2.1 Relocatable Expressions

A relocatable expression is evaluated by ODT as a 16-bit (6-digit octal) number and may be typed in any one of the three forms presented in Table 8-1. In this table, the symbol n stands for an integer in the range 0 to 7 inclusive, and the symbol k stands for an octal number up to six digits long, with a maximum value of 177777. If more than six digits are typed, ODT takes the last six digits, truncated to the low-order 16 bits. k may be preceded by a minus sign, in which case its value is the two's complement of the number typed. For example:

k (number typed)	Values
1	000001
-1	177777
400	000400
-177730	000050
1234567	034567

Table 8-1
Forms of Relocatable Expressions (r)

	r	Value of r
A)	k	The value of r is simply the value of k.
B)	n,k	The value of r is the value of k plus the contents of relocation register n. If the n part of this expression is greater than 7, ODT uses only the last octal digit of n.
C)	C or C,k or n,C or C,C	Whenever the letter C is typed, ODT replaces C with the contents of a special register called the Constant Register. This value has the same role as the k or n that it replaces (i.e., when used in place of n it designates a relocation register). The Constant Register is designated by the symbol $C and may be set to any value, as indicated below.

In the following examples, assume in each case that relocation register 3 contains 003400 and that the constant register contains 000003.

r	Value of r
5;C	000005
-17;C	177761
3,0;C	003400
3,150;C	003550
3,-1;C	003377
C;C	000003
3,C;C	003403
C,0;C	003400
C,10;C	003410
C,C;C	003403

NOTE

For simplicity most examples in this section use Form A. All three forms of r are equally acceptable, however.

8.3 COMMANDS AND FUNCTIONS

When ODT is started (as explained in Section 8.1) it indicates readiness to accept commands by printing an asterisk on the left margin of the terminal page. Most of the ODT commands can be issued in response to the asterisk. For example, a word can be examined and changed if desired, the object program can be run in its entirety or in segments, or memory can be searched for certain words or references to certain words. The discussion below explains these features. In the following examples, characters output by ODT are underlined to differentiate from user input.

8.3.1 Printout Formats

Normally, when ODT prints addresses (as with the commands ↓, ↑, ←, @, <, and >) it attempts to print them in relative form (Form B in Table

8-1). ODT looks for the relocation register whose value is closest but less than or equal to the address to be printed, and then represents the address relative to the contents of the relocation register. However, if no relocation register fits the requirement, the address is printed in absolute form. Since the relocation registers are initialized to -1 (the highest number) the addresses are initially printed in absolute form. If any relocation register subsequently has its contents changed, it may then, depending on the command, qualify for relative form.

For example, suppose relocation registers 1 and 2 contain 1000 and 1004 respectively, and all other relocation registers contain numbers much higher. Then the following sequence might occur (the slash command causes the contents of the location to be printed; the line feed command (<LF>) accesses the next sequential location):

```
*774/000000 <LF>
000776 /000000 <LF>
1,000000 /000000 <LF>        (absolute location 1000)
1,000002 /000000 <LF>        (absolute location 1002)
2,000000 /000000             (absolute location 1004)
```

The printout format is controlled by the format register, $F. Normally this register contains 0, in which case ODT prints addresses relatively whenever possible. $F may be opened and changed to a non-zero value, however, in which case all addresses will be printed in absolute form (see paragraph 8.3.4, Accessing Internal Registers).

8.3.2 Opening, Changing, and Closing Locations

An open location is one whose contents ODT prints for examination, making those contents available for change. In a closed location, the contents are no longer available for change. Several commands are used for opening and closing locations.

Any command used to open a location when another location is already open causes the currently open location to be closed. The contents of an open location may be changed by typing the new contents followed by a single-character command which requires no argument (i.e., <LF> , ↑, RETURN, ←, @, >, <).

The Slash, /

One way to open a location is to type its address followed by a slash:

```
*1000/012746
```

Location 1000 is open for examination and is available for change.

If the contents of an open location are not to be changed, type the RETURN key and the location is closed; ODT prints an asterisk and waits for another command. However, to change the word, simply type the new contents before giving a command to close the location:

```
*1000/012746 012345 <CR>
*
```

In the example above, location 1000 now contains 012345 and is closed since the RETURN key was typed after entering the new contents, as indicated by ODT's second asterisk.

Used alone, the slash reopens the last location opened:

```
*1000/012345 2340 <CR>
*/002340
```

In the example above, the open location was closed by typing the RETURN key. ODT changed the contents of location 1000 to 002340 and then closed the location before printing the *. The single slash command directed ODT to reopen the last location opened. This allowed verification that the word 002340 was correctly stored in location 1000.

Note again, that opening a location while another is open automatically closes the currently open location before opening the new location.

Also note that if an odd numbered address is specified with a slash, ODT opens the location as a byte, and subsequently behaves as if a backslash had been typed (see the following paragraph).

The Backslash, \

In addition to operating on words, ODT operates on bytes. One way to open a byte is to type the address of the byte followed by a backslash. (On the LT33 or LT35 terminal \ is typed by pressing the SHIFT key while typing the L key.) This causes not only the printing of the byte value at the specified address but also the interpreting of the value as ASCII code, and the printing of the corresponding character (if possible) on the terminal:

```
*1001\101 =A
```

A backslash typed alone reopens the last open byte. If a word was previously open, the backslash reopens its even byte:

```
*1002/000004 \004 =
```

The LINE FEED Key, <LF>

If the LINE FEED key is typed when a location is open, ODT closes the open location and opens the next sequential location:

```
*1000/002340 <LF>    ( <LF> denotes typing the LINE FEED key)
001002 /012740
```

In this example, the LINE FEED key caused ODT to print the address of the next location along with its contents, and to wait for further instructions. After the above operation, location 1000 is closed and 1002 is open. The open location may be modified by typing the new contents.

If a byte location was open, typing the LINE FEED key opens the next byte location.

The Up-Arrow, ↑ or ^

If the up-arrow (or circumflex) is typed when a location is open (up-arrow is produced on an LT33 or LT35 by typing SHIFT N), ODT closes the open location and opens the previous location. To continue from the example above:

```
*001002/012740  ↑
001000 /002340
```

Now location 1002 is closed and 1000 is open. The open location may be modified by typing the new contents.

If the opened location was a byte, then up-arrow opens the previous byte.

The Back-Arrow, ← or _

If the back-arrow (or underline) is typed (via SHIFT O on an LT33 or LT35 terminal) to an open word, ODT interprets the contents of the currently open word as an address indexed by the Program Counter (PC) and opens the addressed location:

```
*1006/000006 ←
001016 /000405
```

Notice in this example that the open location, 1006, was indexed by the PC as if it were the operand of an instruction with address mode 67 as explained in Chapter 5.

A modification to the opened location can be made before a line feed, up-arrow, or back-arrow is typed. Also, the new contents of the location will be used for address calculations using the back-arrow command. Example:

```
*100/000222 4 <LF>    (modify to 4 and open next location)
000102 /000111 6↑     (modify to 6 and open previous location)
000100 /000004 100←   (change to 100 and open location indexed
000202 /123456         by PC)
```

Open the Addressed Location, @

The at symbol @ (SHIFT P on the LT33 or LT35 terminal) may be used to optionally modify a location, close it, and then use its contents as the address of the location to open next.

```
*1006/001044 @          (open location 1044 next)
001044 /000500

*1006/001044 2100@      (modify to 2100 and open location
002100 /000167           2100)
```

Relative Branch Offset, >

The right-angle bracket, >, will optionally modify a location, close it, and then use its low-order byte as a relative branch offset to the next word to be opened. For example:

```
*1032/000407 301>   (modify to 301 and interpret as a
000636 /000010       relative branch)
```

Note that 301 is a negative offset (-77). The offset is doubled before it is added to the PC; therefore, 1034+(-176)=636.

Return to Previous Sequence, <

The left-angle bracket, <, allows the user to optionally modify a location, close it, and then open the next location of the previous sequence which was interrupted by a back-arrow, @, or right-angle bracket command. Note that back-arrow, @, or right-angle bracket causes a sequence change to the word opened. If a sequence change has not occurred, the left-angle bracket simply opens the next location as a LINE FEED does. This command operates on both words and bytes.

```
*1032/000407 301>          (> causes a sequence change)
000636 /000010 <           (return to original sequence)
001034 /001040 @           (@ causes a sequence change)
001040 /000405 \005 = <    (< now operates on byte)
001035 \002 = <            (< acts like <LF>)
001036 \004 =
```

8.3.3 Accessing General Registers 0-7

The program's general registers 0-7 are opened with a command in the following format:

```
*$n/
```

where n is the integer representing the desired register (in the range 0 through 7). When opened, these registers can be examined or changed by typing in new data as with any addressable location. For example:

```
*$0/000033 <CR>        (R0 was examined and closed)
*

*$4/000474 464<CR>     (R4 was opened, changed, and closed)
*
```

The example above can be verified by typing a slash in response to ODT's asterisk:

```
*/000464
```

The LINE FEED, up-arrow, back-arrow or @ command may be used when a register is open.

8.3.4 Accessing Internal Registers

The program's Status Register contains the condition codes of the most recent operational results and the interrupt priority level of the object program. It is opened by typing $S. For example:

```
*$S/000311
```

$S represents the address of the Status Register. In response to $S in the example above, ODT printed the 16-bit word, of which only the low-order eight bits are meaningful. Bits 0-3 indicate whether a carry, overflow, zero, or negative (in that order) has resulted, and bits 5-7 indicate the interrupt priority level (in the range 0-7) of the object program. (Refer to the PDP-11 Processor Handbook for the Status Register format.)

The $ is used to open certain other internal locations listed in Table 8-2:

Table 8-2
Internal Registers

Register	Function
$B	location of the first word of the breakpoint table (see Section 8.3.6).
$M	mask location for specifying which bits are to be examined during a bit pattern search (see Section 8.3.9).
$P	location defining the operating priority of ODT (see Section 8.3.15).
$S	location containing the condition codes (bits 0-3) and interrupt priority level (bits 5-7) (explained above).
$C	location of the Constant Register (see Section 8.3.10).
$R	location of Relocation Register 0, the base of the Relocation Register table (see Section 8.3.13).
$F	location of Format Register (see Section 8.3.1).

8.3.5 Radix 50 Mode, X

The Radix 50 mode of packing certain ASCII characters three to a word is employed by many DEC-supplied PDP-11 system programs, and may be employed by any programmer via the MACRO Assembler's ".RAD50" directive. ODT provides a method for examining and changing memory words packed in this way with the X command.

When a word is opened and the X command is typed, ODT converts the contents of the opened word to its 3-character Radix 50 equivalent and prints these characters on the terminal. One of the responses in Table 8-3 can then be typed:

Table 8-3
Radix 50 Terminators

Response	Effect
RETURN key <CR>	Closes the currently open location.
LINE FEED key <LF>	Closes the currently open location and opens the next one in sequence.
↑ key	Closes the currently open location and opens the previous one in sequence.
Any three characters whose octal code is 040 (space) or greater.	Converts the three specified characters into packed Radix 50 format.
Legal Radix 50 characters for this last response are:	
. $ Space 0 through 9 A through Z	

If any other characters are typed, the resulting binary number is unspecified (that is, no error message is printed and the result is unpredictable). Exactly three characters must be typed before ODT resumes its normal mode of operation. After the third character is typed, the resulting binary number is available to be stored in the opened location by closing the location in any one of the ways listed in Table 8-3. Example:

```
*1000/042431 X=KBI CBA <CR>
*1000/011421 X=CBA
```

NOTE

After ODT has converted the three characters to binary, the binary number can be interpreted in one of many different ways, depending on the command which follows. For example:

```
*1234/063337 X=PRO XIT/013704
```

Since the Radix 50 equivalent of XIT is 113574, the final slash in the example will cause ODT to open location 113574 if it is a legal address. (Refer to paragraph 8.5 for a discussion of command legality and detection of errors.)

8.3.6 Breakpoints

The breakpoint feature facilitates monitoring the progress of program execution. A breakpoint may be set at any instruction which is not referenced by the program for data. When a breakpoint is set, ODT replaces the contents of the breakpoint location with a trap instruction so that program execution is suspended when a breakpoint is encountered. The original contents of the breakpoint location are restored, and ODT regains control.

With ODT, up to eight breakpoints, numbered 0 through 7, can be set at any one time. A breakpoint is set by typing the address of the desired location of the breakpoint followed by ;B. Thus r;B sets the next available breakpoint at location r. (If all 8 breakpoints have been set, ODT ignores the r;B command.) Specific breakpoints may be set or changed by the r;nB command where n is the number of the breakpoint. For example:

```
*1020;B        (sets breakpoint 0)
*1030;B        (sets breakpoint 1)
*1040;B        (sets breakpoint 2)
*1032;1B       (resets breakpoint 1)
*
```

The ;B command removes all breakpoints. Use the ;nB command to remove only one of the breakpoints, where n is the number of the breakpoint. For example:

```
*;2B      (removes the second breakpoint)
*
```

A table of breakpoints is kept by ODT and may be accessed by the user. The $B/ command opens the location containing the address of breakpoint 0. The next seven locations contain the addresses of the other breakpoints in order, and can be sequentially opened using the LINE FEED key. For example:

```
*$B/001020 <LF>
nnnnnn /001032 <LF>
nnnnnn /nnnnnn       (nnnnnn=address internal to ODT)
```

In this example, breakpoint 2 is not set. The contents printed is an address internal to ODT and can be determined by checking the Linker Load Map (see Chapter 6).

It should be noted that a repeat count in a Proceed command refers only to the breakpoint that has most recently occurred. Execution of other breakpoints encountered is determined by their own repeat counts.

8.3.7 Running the Program, r;G and r;P

Program execution is under control of ODT. There are two commands for running the program: r;G and r;P. The r;G command is used to start execution (Go) and r;P to continue (Proceed) execution after halting at a breakpoint. For example:

```
*1000;G
```

Execution is started at location 1000. The program runs until a breakpoint is encountered or until program completion, unless it gets caught in an infinite loop, in which case it must be either restarted or reentered as explained in Section 8.1.

Upon execution of either the r;G or r;P command, the general registers 0-6 are set to the values in the locations specified as $0-$6 and the processor Status Register is set to the value in the location specified as $S.

When a breakpoint is encountered, execution stops and ODT prints Bn; (where n is the breakpoint number), followed by the address of the breakpoint. Locations can then be examined for expected data. For example:

```
*1010;3B        (breakpoint 3 is set at location 1010)
*1000;G         (execution started at location 1000)
B3;001010       (execution stopped at location 1010)
*
```

To continue program execution from the breakpoint, type ;P in response to ODT's last *.

When a breakpoint is set in a loop, it may be desirable to allow the program to execute a certain number of times through the loop before recognizing the breakpoint. This can be done by setting a proceed count using the k;P command; this command specifies the number of times the breakpoint is to be encountered before program execution is suspended (on the kth encounter). The count, k, refers only to the numbered breakpoint which most recently occurred. A different proceed count may be specified for the breakpoint when it is encountered. Thus:

```
B3;001010       (execution halted at breakpoint 3)
*1026;3B        (reset breakpoint 3 at location 1026)
*4;P            (set proceed count to 4 and
B3;001026       continue execution; loop through
*               breakpoint three times and halt on
                fourth occurrence of the breakpoint)
```

Following the table of breakpoints (as explained in Section 8.3.6) is a table of proceed command repeat counts for each breakpoint. These repeat counts can be inspected by typing $B/ and nine LINE FEEDs. The repeat count for breakpoint 0 is printed (the first seven LINE FEEDs cause the table of breakpoints to be printed; the eighth types the single instruction mode, explained in the next section, and the ninth LINE FEED begins the table of proceed command repeat counts). The repeat counts for breakpoints 1 through 7, and the repeat count for the single-instruction trap follow in sequence. Before a proceed count is assigned a value by the user, it is set to 0; after the count has been executed, it is set to -1. Opening any one of these provides an alternative way of changing the count as the location, once open, can have its contents modified in the usual manner by typing the new contents and then the RETURN key. For example:

```
        .
        .
nnnnnn /001036 <LF>       (address of breakpoint 7)
nnnnnn /006630 <LF>       (single instruction address)
nnnnnn /000000 15 <LF>    (count for breakpoint 0; change to 15)
nnnnnn /000000 <LF>       (count for breakpoint 1)
        .
        .
        .
nnnnnn /000000 <LF>       (count for breakpoint 7)
nnnnnn /nnnnnn            (repeat count for single instruction
                          mode; the single instrucion address
```

is an address internal to the user program if single instrucion mode is used)

The address indicated as the single instruction address and the repeat count for single instruction mode are explained next.

8.3.8 Single Instruction Mode

With this mode the number of instructions to be executed before suspension of the program run can be specified. The Proceed command, instead of specifying a repeat count for a breakpoint encounter, specifies the number of succeeding instructions to be executed. Note that breakpoints are disabled when single instruction mode is operative.

Commands for single instruction mode are:

;nS	Enables single instruction mode (n can have any non-zero value and serves only to distinguish this form from the form ;S). Breakpoints are disabled.
n;P	Proceeds with program run for next n instructions before reentering ODT (if n is missing, it is assumed to be 1). Trap instructions and associated handlers can affect the Proceed repeat count. See Section 8.4.2.
;S	Disables single instruction mode.

When the repeat count for single instruction mode is exhausted and the program suspends execution, ODT prints:

```
B8;nnnnnn
```

where nnnnnn is the address of the next instruction to be executed. The $B breakpoint table contains this address following that of breakpoint 7. However, unlike the table entries for breakpoints 0-7, direct modification has no effect.

Similarly, following the repeat count for breakpoint 7 is the repeat count for single instruction mode. This table entry may be directly modified and thus is an alternative way of setting the single-instruction mode repeat count. In such a case, ;P implies the argument set in the $B repeat count table rather than an assumed 1.

8.3.9 Searches

With ODT all or any specified portion of memory can be searched for any specific bit pattern or for references to a particular location.

Word Search, r;W

Before initiating a word search, the mask and search limits must be specified. The location represented by $M is used to specify the mask of the search. $M/ opens the mask register. The next two sequential locations (opened by LINE FEEDs) contain the lower and upper limits of the search. Bits set to 1 in the mask are examined during the search; other bits are ignored. Then the search object and the initiating command are given using the r;W command where r is the search object. When a match is found, (i.e., each bit set to 1 in the search object is set to 1 in the word being searched over the mask range) the matching word is printed. For example:

```
*$M/000000 177400 <LF>        (test high-order eight bits)
nnnnnn /000000 1000  <LF>     (set low address limit)
nnnnnn /000000 1040  <CR>     (set high address limit)
*400;W                        (initiate word search)
001010 /000770
001034 /000404
*
```

In the above example, nnnnnn is an address internal to ODT; this location varies and is meaningful only for reference purposes. In the first line above, the slash was used to open $M which now contains 177400; the LINE FEEDs opened the next two sequential locations which now contain the upper and lower limits of the search.

In the search process an exclusive OR (XOR) is performed with the word currently being examined and the search object, and the result is ANDed to the mask. If this result is zero, a match has been found and is reported on the terminal. Note that if the mask is zero, all locations within the limits are printed.

Typing CTRL U during a search printout terminates the search.

Effective Address Search, r;E

ODT provides a search for words which address a specified location. Open the mask register only to gain access to the low and high limit registers. After specifying the search limits (as explained for the word search), type the command r;E (where r is the effective address) to initiate the search.

Words which are either an absolute address (argument r itself), a relative address offset, or a relative branch to the effective address, are printed after their addresses. For example:

```
*$M/177400 <LF>              (open mask register only to gain
nnnnnn /001000 1010 <LF>     access to search limits)
nnnnnn /001040 1060 <CR>
*1034;E                      (initiating search)
001016 /001006               (relative branch)
001054 /002767               (relative branch)
*1020;E                      (initiating a new search)
001022 /177774               (relative address offset)
001030 /001020               (absolute address)
```

Particular attention should be given to the reported effective address references because a word may have the specified bit pattern of an effective address without actually being so used. ODT reports all possible references whether they are actually used as such or not.

Typing CTRL U during a search printout terminates the search.

8.3.10 The Constant Register, r;C

It is often desirable to convert a relocatable address into its value after relocation or to convert a number into its two's complement, and then to store the converted value into one or more places in a program. The Constant Register provides a means of accomplishing this and other useful functions.

When r;C is typed, the relocatable expression r is evaluated to its 6-digit octal value and is both printed on the terminal and stored in the Constant Register. The contents of the Constant Register may be invoked in subsequent relocatable expressions by typing the letter C. Examples follow:

*-4432;C=173346	(the two's complement of 4432 is placed in the Constant Register)
*6632/062701 C <CR>	(the contents of the Constant Register are stored in location 6632)
*1000;1R	(relocation Register 1 is set to 1000)
*1,4272;C=005272	(relative location 4272 is reprinted as an absolute location and stored in the Constant Register)

8.3.11 Memory Block Initialization, ;F and ;I

The Constant Register can be used in conjunction with the commands ;F and ;I to set a block of memory to a given value. While the most common value required is zero, other possibilities are plus one, minus one, ASCII space, etc.

When the command ;F is typed, ODT stores the contents of the Constant Register in successive memory words starting at the memory word address specified in the lower search limit, and ending with the address specified in the upper search limit.

When the command ;I is typed, the low-order 8 bits in the Constant Register are stored in successive bytes of memory starting at the byte address specified in the lower search limit and ending with the byte address specified in the upper search limit.

For example, assume relocation register 1 contains 7000, 2 contains 10000, and 3 contains 15000. The following sequence sets word locations 7000-7776 to zero, and byte locations 10000-14777 to ASCII spaces.

```
                                   (open mask register to gain access
*$M/000000 <LF>                    to search limits)
nnnnnn /000000 1,0  LF>            (set lower limit to 7000)
nnnnnn /000000 2,-2 <LF>           (set upper limit to 7776)
*0;C=000000                        (Constant Register set to zero)
*;F                                (Locations 7000-7776 set to zero)

*$M/000000  <LF>
nnnnnn/007000 2,0 <LF>             (set lower limit to 10000)
nnnnnn/007776 3,-1 <CR>            (set upper limit to 14777)
*40;C=000040                       (Constant Register set to 40
*;I                                (SPACE))
*                                  (Byte locations 10000-14777 are set
                                   to value in low-order 8 bits of
                                   Constant Register)
```

8.3.12 Calculating Offsets, r;O

Relative addressing and branching involve the use of an offset--the number of words or bytes forward or backward from the current location to the effective address. During the debugging session it may be necessary to change a relative address or branch reference by replacing one instruction offset with another. ODT calculates the offsets in response to the r;O command.

The command r;O causes ODT to print the 16-bit and 8-bit offsets from the currently open location to address r. For example:

```
*346/000034 414;O 000044 022 22 <CR>
*/000022
```

In the example, location 346 is opened and the offsets from that location to location 414 are calculated and printed. The contents of location 346 are then changed to 22 (the 8-bit offset) and verified on the next line.

The 8-bit offset is printed only if it is in the range -128(decimal) to 127(decimal) and the 16-bit offset is even, as was the case above. For example, the offset of a relative branch is calculated and modified as follows:

```
*1034/103421 1034;O 177776 377 \021 = 377 <CR>
*/103777
```

Note that the modified low-order byte 377 must be combined with the unmodified high-order byte.

8.3.13 Relocation Register Commands, r;nR, ;nR, ;R

The use of the relocation registers is defined in Section 8.2. At the beginning of a debugging session it is desirable to preset the registers to the relocation biases of those relocatable modules which will be receiving the most attention.

This can be done by typing the relocation bias, followed by a semicolon and the specification of relocation registers, as follows:

```
r;nR
```

r may be any relocatable expression and n is an integer from 0 to 7. If n is omitted it is assumed to be 0. As an example:

```
*1000;5R        (puts 1000 into relocation register 5)
*5,100;5R       (effectively adds 100 to the contents
*               of relocation register 5)
```

Once a relocation register is defined, it can be used to reference relocatable values. For example:

```
*2000;1R             (puts 2000 into relocation register 1)
*1,2176/002466       (examines contents of location 4176)
*1,3712;0B           (sets a breakpoint at location 5712)
```

In certain uses, programs may be relocated to an address below that at which they were assembled. This could occur with PIC code (Position Independent Code) which is moved without the use of the Linker. In this case the appropriate relocation bias would be the two's complement of the actual downward displacement. One method for easily evaluating the bias and putting it in the relocation register is illustrated in the following example.

Assume a program was assembled at location 5000 and was moved to location 1000. Then the sequence:

```
*1000;1R
*1,-5000;1R
*
```

enters the two's complement of 4000 in relocation register 1, as desired.

Relocation registers are initialized to -1, so that unwanted relocation registers never enter into the selection process when ODT searches for the most appropriate register.

To set a relocation register to -1, type ;nR. To set all relocation registers to -1, type ;R.

ODT maintains a table of relocation registers, beginning at the address specified by $R. Opening $R ($R/) opens relocation register 0. Successively typing a line feed opens the other relocation registers in sequence. When a relocation register is opened in this way, it may be modified like any other memory location.

8.3.14 The Relocation Calculators, nR and n!

When a location has been opened, it is often desirable to relate the relocated address and the contents of the location back to their relocatable values. To calculate the relocatable address of the opened location relative to a particular relocation bias, type n!, where n specifies the relocation register. This calculator works with opened bytes and words. If n is omitted, the relocation register whose contents are closest but less than or equal to the opened location is selected automatically by ODT. In the following example, assume that these conditions are fulfilled by relocation register 2, which contains 2000. To find the most likely module that a given opened byte is in:

```
*2500\011 = !=2,000500
```

Typing nR after opening a word causes ODT to print the octal number which equals the value of the contents of the opened location minus the contents of relocation register n. If n is omitted, ODT selects the relocation register whose contents are closest but less than or equal to the contents of the opened location. For example, assume the relocation bias stored in relocation register 1 is 7000; then:

```
*1,500/000000 1R=1,171000
```

The value 171000 is the content of 1,500, relative to the base 7000. An example of the use of both relocation calculators follows.

If relocation register 1 contains 1000, and relocation register 2 contains 2000, then to calculate the relocatable addresses of location 3000 and its contents, relative to 1000 and 2000, the following can be performed.

```
*3000/000410 1!=1,002000 2!=2,001000 1R=1,177410 2R=2,176410
```

8.3.15 ODT Priority Level, $P

$P represents a location in ODT that contains the interrupt (or processor) priority level at which ODT operates. If $P contains the value 377, ODT operates at the priority level of the processor at the time ODT is entered. Otherwise $P may contain a value between 0 and 7 corresponding to the fixed priority at which ODT operates.

To set ODT to the desired priority level, open $P. ODT prints the present contents, which may then be changed:

```
*$P/000006 377 <CR>
*
```

If $P is not specified, its value is seven.

ODT priority must be set to 0 if ODT is being used in an F/B environment with another job running.

Breakpoints may be set in routines which run at different priority levels. For example, a program running at a low priority may use a device service routine which operates at a higher priority level. If a breakpoint occurs from a low-priority routine, ODT operates at a low priority; if an interrupt occurs from a high priority routine, the breakpoints in the high priority routine will not be recognized since they were removed when the low priority breakpoint occurred. That is, interrupts set at a priority higher than the one at which ODT is running will occur and any breakpoints will not be recognized. ODT disables all breakpoints from the program whenever it gains control.

Breakpoints are enabled when ;P and ;G commands are executed. For example:

```
*$P/000007 5
*1000;B
*2000;B
*1000;G
B0;001000
*          (an interrupt occurs and is serviced)
```

If a higher level interrupt occurs while ODT is waiting for input the interrupt will be serviced, and no breakpoints will be recognized.

8.3.16 ASCII Input and Output, r;nA

ASCII text may be inspected and changed by the command:

r;nA

where r is a relocatable expression, and n is a character count. If n is omitted it is assumed to be 1. ODT prints n characters starting at location r, followed by a carriage return/line feed. Type one of the following:

<CR> ODT outputs a carriage return/line feed and an asterisk and waits for another command.

<LF> ODT opens the byte following the last byte output.

Up to n characters of text
ODT inserts the text into memory, starting at location r. If fewer than n characters are typed, terminate the command by typing CTRL U, causing a carriage return/line feed/asterisk to be output. However, if exactly n characters are typed, ODT responds with a carriage return/line feed, the address of the next available byte and a carriage return/line feed/asterisk.

ODT does not check the magnitude of n.

8.4 PROGRAMMING CONSIDERATIONS

Information in this section is not necessary for the efficient use of ODT. However, it does provide a better understanding of how ODT performs some of its functions and in certain difficult debugging situations, this understanding is necessary.

8.4.1 Functional Organization

The internal organization of ODT is almost totally modularized into independent subroutines. The internal structure consists of three major functions: command decoding, command execution, and various utility routines.

The command decoder interprets the individual commands, checks for command errors, saves input parameters for use in command execution, and sends control to the appropriate command execution routine.

The command execution routines take parameters saved by the command decoder and use the utility routines to execute the specified command. Command execution routines exit either to the object program or back to the command decoder.

The utility routines are common routines such as SAVE-RESTORE and I/O. They are used by both the command decoder and the command executers.

8.4.2 Breakpoints

The function of a breakpoint is to give control to ODT whenever the user program tries to execute the instruction at the selected address. Upon encountering a breakpoint, all of the ODT commands can be used to examine and modify the program.

When a breakpoint is executed, ODT removes all the breakpoint instructions from the user's code so that the locations may be examined and/or altered. ODT then types a message on the terminal of the form Bn;k where k is the breakpoint address (and n is the breakpoint number). The breakpoints are automatically restored when execution is resumed.

A major restriction in the use of breakpoints is that the word where a breakpoint was set must not be referenced by the program in any way since ODT altered the word. Also, no breakpoint should be set at the location of any instruction that clears the T-bit. For example:

```
        MOV #240,177776      ;SET PRIORITY TO LEVEL 5
```

NOTE

> Instructions that cause or return from traps (e.g., EMT, RTI) are likely to clear the T-bit, since a new word from the trap vector or the stack is loaded into the Status Register.

A breakpoint occurs when a trace trap instruction (placed in the user program by ODT) is executed. When a breakpoint occurs, the following steps are taken:

1. Set processor priority to seven (automatically set by trap instruction).
2. Save registers and set up stack.
3. If internal T-bit trap flag is set, go to step 13.
4. Remove breakpoints.
5. Reset processor priority to ODT's priority or user's priority.
6. Make sure a breakpoint or single-instruction mode caused the interrupt.
7. If the breakpoint did not cause the interrupt, go to step 15.
8. Decrement repeat count.
9. Go to step 18 if non-zero; otherwise reset count to one.
10. Save terminal status.
11. Type message about the breakpoint or single-instruction mode interrupt.

12. Go to command decoder.

13. Clear T-bit in stack and internal T-bit flag.

14. Jump to the Go processor.

15. Save terminal status.

16. Type BE (Bad Entry) followed by the address.

17. Clear the T-bit, if set, in the user status and proceed to the command decoder.

18. Go to the Proceed processor, bypassing the TT restore routine.

Note that steps 1-5 inclusive take approximately 100 microseconds during which time interrupts are not permitted (ODT is running at level 7).

When a proceed (;P) command is given, the following occurs:

1. The proceed is checked for legality.

2. The processor priority is set to seven.

3. The T-bit flags (internal and user status) are set.

4. The user registers, status, and Program Counter are restored.

5. Control is returned to the user.

6. When the T-bit trap occurs, steps 1, 2, 3, 13, and 14 of the breakpoint sequence are executed, breakpoints are restored, and program execution resumes normally.

When a breakpoint is placed on an IOT, EMT, TRAP, or any instruction causing a trap, the following occurs:

1. When the breakpoint occurs as described above, ODT is entered.

2. When ;P is typed, the T-bit is set and the IOT, EMT, TRAP, or other trapping instruction is executed.

3. This causes the current PC and status (with the T-bit included) to be pushed on the stack.

4. The new PC and status (no T-bit set) are obtained from the respective trap vector.

5. The whole trap service routine is executed without any breakpoints.

6. When an RTI is executed, the saved PC and PS (including the T-bit) are restored. The instruction following the trap-causing instruction is executed. If this instruction is not another trap-causing instruction, the T-bit trap occurs, causing the breakpoints to be reinserted in the user program,

or the single-instruction mode repeat count to be decremented. If the following instruction is a trap-causing instruction, this sequence is repeated starting at step 3.

NOTE

Exit from the trap handler must be via the RTI instruction. Otherwise, the T-bit is lost. ODT can not regain control since the breakpoints have not been reinserted yet.

Note that the ;P command is illegal if a breakpoint has not occurred (ODT responds with ?); ;P is legal, however, after any trace trap entry.

The internal breakpoint status words have the following format:

1. The first eight words contain the breakpoint addresses for breakpoints 0-7. (The ninth word contains the address of the next instruction to be executed in single-instruction mode.)

2. The next eight words contain the respective repeat counts. The following word contains the repeat count for single-instruction mode.)

These words may be changed at will, either by using the breakpoint commands or by direct manipulation with $B.

When program runaway occurs (that is, when the program is no longer under ODT control, perhaps executing an unexpected part of the program where a breakpoint has not been placed), ODT may be given control by pressing the HALT key to stop the computer,and restarting ODT (see Section 8.1). ODT prints *, indicating that it is ready to accept a command.

If the program being debugged uses the teleprinter for input or output, the program may interact with ODT to cause an error since ODT uses the teleprinter as well. This interactive error will not occur when the program being debugged is run without ODT.

Note the following rules concerning the ODT break routine:

1. If the teleprinter interrupt is enabled upon entry to the ODT break routine, and no output interrupt is pending when ODT is entered, ODT generates an unexpected interrupt when returning control to the program.

2. If the interrupt of the teleprinter reader (the keyboard) is enabled upon entry to the ODT break routine, and the program is expecting to receive an interrupt to input a character, both the expected interrupt and the character are lost.

3. If the teleprinter reader (keyboard) has just read a character into the reader data buffer when the ODT break routine is entered, the expected character in the reader data buffer is lost.

8.4.3 Searches

The word search allows the user to search for bit patterns in specified sections of memory. Using the $M/ command, the user specifies a mask, a lower search limit ($M+2), and an upper search limit ($M+4). The search object is specified in the search command itself.

The word search compares selected bits (where ones appear in the mask) in the word and search object. If all of the selected bits are equal, the unmasked word is printed.

The search algorithm is:

1. Fetch a word at the current address.

2. XOR (exclusive OR) the word and search object.

3. AND the result of step 2 with the mask.

4. If the result of step 3 is zero, type the address of the unmasked word and its contents. Otherwise, proceed to step 5.

5. Add two to the current address. If the current address is greater than the upper limit, type * and return to the command decoder, otherwise go to step 1.

Note that if the mask is zero, ODT prints every word between the limits, since a match occurs every time (i.e., the result of step 3 is always zero).

In the effective address search, ODT interprets every word in the search range as an instruction which is interrogated for a possible direct relationship to the search object. The mask register is opened only to gain access to the search limit registers.

The algorithm for the effective address search is (where (X) denotes contents of X, and K denotes the search object):

1. Fetch a word at the current address X.

2. If (X)=K [direct reference], print contents and go to step 5.

3. If (X)+X+2=K [indexed by PC], print contents and go to step 5.

4. If (X) is a relative branch to K, print contents.

5. Add two to the current address. If the current address is greater than the upper limit, perform a carriage return/line feed and return to the command decoder; otherwise, go to step 1.

8.4.4 Terminal Interrupt

Upon entering the TT SAVE routine, the following occurs:

1. Save the LSR status register (TKS).

2. Clear interrupt enable and maintenance bits in the TKS.

3. Save the TT status register (TPS).

4. Clear interrupt enable and maintenance bits in the TPS.

To restore the TT:

1. Wait for completion of any I/O from ODT.

2. Restore the TKS.

3. Restore the TPS.

NOTES

If the TT printer interrupt is enabled upon entry to the ODT break routine, the following may occur:

1. If no output interrupt is pending when ODT is entered, an additional interrupt always occurs when ODT returns control to the user.

2. If an output interrupt is pending upon entry, the expected interrupt occurs when the user regains control.

If the TT reader (keyboard) is busy or done, the expected character in the reader data buffer is lost.

If the TT reader (keyboard) interrupt is enabled upon entry to the ODT break routine, and a character is pending, the interrupt (as well as the character) is lost.

8.5 ODT ERROR DETECTION

ODT detects two types of error: illegal or unrecognizable command and bad breakpoint entry. ODT does not check for the legality of an address when commanded to open a location for examination or modification. Thus the command:

```
*177774/
?M-TRAP TO 4 003362
```

references nonexistent memory, thereby causing a trap through the vector at location 4. If this vector has not been properly initialized, unpredictable results occur.

Typing something other than a legal command causes ODT to ignore the command, print:

```
(echoes illegal command)?
*
```

and wait for another command. Therefore, to cause ODT to ignore a command just typed, type any illegal character (such as 9 or RUBOUT) and the command will be treated as an error, i.e., ignored.

ODT suspends program execution whenever it encounters a breakpoint, i.e., traps to its breakpoint routine. If the breakpoint routine is entered and no known breakpoint caused the entry, ODT prints:

```
BEnnnnnn
*
```

and waits for another command. BEnnnnnn denotes Bad Entry from location nnnnnn. A bad entry may be caused by an illegal trace trap instruction, setting the T-bit in the status register, or by a jump to the middle of ODT.

CHAPTER 9

PROGRAMMED REQUESTS

A number of services at the machine language level which the monitor regularly provides to system programs are also available to user-written programs. These include services for file manipulation, command interpretation, and facilities for input and output operations. User programs call these monitor services by means of "programmed requests", which are assembler macro calls written into the user program and interpreted by the monitor at program execution time.

NOTE

Programmed requests used in Version 2 differ from those used in Version 1; for example, the channel number in Version 1 was limited to the range 0-17, where it is not in Version 2; blank fields in macro calls were not allowed in Version 1, and are in Version 2; a .area argument points to an argument list in Version 2, where arguments were pushed on the stack in Version 1.

Programs written for use under Version 1 will assemble and execute properly when the ..V1.. macro call (explained in Section 9.3.1.5) is included, but it is to the user's advantage to convert these programs so they use the new Version 2 macro calls wherever possible. Only macro calls which are used with the current version of RT-11 (Version 2) are discussed in this chapter. See Section 9.5 for instructions on converting Version 1 macro calls to the Version 2 format.

The macro definitions for both Version 1 and Version 2 requests are included in the file SYSMAC.SML (in 8K systems, the system macro library is called SYSMAC.8K); Appendix D provides a listing of SYSMAC.SML. Refer to Chapter 5 for general information related to the use of macro calls.

The FORTRAN programmer should note that the system subroutine library (SYSLIB) gives him the same capability (under FORTRAN) to use the programmed requests which are available to the machine language programmer and described in this chapter. SYSLIB users should first read this chapter and then read Appendix O.

9.1 FORMAT OF A PROGRAMMED REQUEST

The basis of a programmed request is the EMT instruction, used to communicate information to the monitor. When an EMT is executed, control is passed to the monitor, which extracts appropriate information from the EMT and executes the function required. The low-order byte of the EMT instruction contains a code which is interpreted as:

Low-Order Byte of EMT	Meaning
377	Reserved; RT-11 ignores this EMT and returns control to the user program immediately.
376	Used internally by the RT-11 monitor; this EMT code should never be used by user programs.
375	Programmed request with several arguments: R0 must point to a list of arguments which designates the specific function.
374	Programmed request with one argument: R0 contains a function code in the high-order byte and a channel number (see Section 9.2.1) or 0 in the low-order byte.
360-373	Used internally by the RT-11 monitor; these EMT codes should never be used by user programs.
340-357	Programmed request with arguments on the stack and/or in R0.
0-337	Version 1 programmed request. These EMTs use arguments both on the stack and in R0. They are supported for binary compatability with Version 1 programs.

A programmed request consists of a macro call followed, where necessary, by one or more arguments. Arguments supplied to a macro call must be legal assembler expressions since arguments will be used as source fields in MOV instructions when the macros are expanded at assembly time. The following two formats are used:

1. PRGREQ ARG1,ARG2,...ARGN

2. PRGREQ AREA,ARG1,ARG2,...ARGN

Form 1 above contains the arguments ARG1 through ARGN; no argument list pointer is required. Macros of this form generate either an EMT 374 or one of the EMTs 340-357. Certain arguments for this form may be omitted; refer to the listing of SYSMAC.SML in Appendix D.

In form 2 above, AREA is a pointer to the argument list which contains the arguments ARG1 through ARGN. This form always causes an EMT 375 to be generated. Blank fields are permitted; however, if the AREA argument is blank, the macro assumes that R0 points to a valid argument block (see Section 9.2.3). If any of the fields ARG1 to ARGN are blank, the corresponding entries in the argument list are left untouched. Thus,

```
.PRGREQ AREA,A1,A2
```

points R0 to the argument block at AREA and fills in the first and second arguments, while:

```
.PRGREQ AREA
```

points R0 to the block, and fills in the first word but does not fill in any other arguments.

The call:

```
.PRGREQ ,A1
```

assumes R0 points to the argument block and fills in the A1 argument, but leaves the A2 argument alone. The call:

```
.PRGREQ
```

generates only an EMT 375 and assumes that both R0 and the block to which it points are properly set up.

The arguments to RT-11 programmed request macros all serve as the source field of a MOV instruction which moves a value into the argument block or R0. For example:

```
.PRGREQ  CHAR
```

expands into:

```
MOV  CHAR,R0
EMT  357
```

Care should be taken to make certain that the arguments specified are legal source fields and that the address accurately represents the value desired. If the value is a constant, immediate mode [#] should be used; if the value is in a register, the register mnemonic [Rn] should be used; if the value is indirectly addressed, the appropriate register convention is necessary [@Rn], and if the value is in memory, the label of the location whose value is the argument is used.

Following are some examples of both correct and incorrect macro calls. Consider the general request:

```
.PRGREQ .AREA,.ARG1,...ARGN
```

A more common way of writing a request of this form is:

```
.PRGREQ #AREA,#ARG1,...#ARGN
```

In this format, the address of AREA is put directly into the argument list. AREA is the tag which indicates the beginning of the argument block. For example:

```
        .PRGREQ #AREA,#4
             .
             .
             .
AREA:   .BLKW 3
```

When a direct numerical argument is required, the # causes the correct value to be put into the argument block. For example:

```
.PRGREQ #AREA,#4
```

is correct, while:

```
.PRGREQ #AREA,4
```

is not. This form interprets the 4 as meaning "move the contents of location 4 into the argument block", where the number 4 itself should be moved into the block.

If the request is written as:

```
.PRGREQ AREA,#4
```

it is interpreted as "use the contents of location AREA as the list pointer", when the address of AREA is actually desired. This expansion could be used with the following form:

```
          .PRGREQ LIST1,#4
             .
             .
             .
LIST1:    AREA
AREA:     .BLKW3
```

In this case, the content of location LIST1 is the address of the argument list. Similarly, this form is correct:

```
          .PRGREQ LIST1,NUMBER
LIST1:    AREA
NUMBER:   4
```

In this case, the contents of the locations LIST1 and NUMBER are the argument list pointer and data value, respectively.

NOTE

All registers except R0 are preserved across a programmed request. (In certain cases, R0 may contain information passed back by the monitor; however, unless the description of a request indicates that a specific value is returned in R0, it may be assumed that the contents of R0 are unpredictable upon return from the request). With the exception of calls to the CSI, the position of the stack pointer is also preserved across a programmed request.

9.2 SYSTEM CONCEPTS

Some basic operational characteristics and concepts of RT-11 are described below.

9.2.1 Channel Number (chan)

A channel number is a logical identifier in the range 0 to 377(octal) for a file or "set of data" used by the RT-11 monitor. Thus, when a file is opened on a particular device, a channel number is assigned to that file. To refer to an open file, it is only necessary to refer to the appropriate channel number for that file.

9.2.2 Device block (dblk)

A device block is a four-word block of radix-50 information which specifies a physical device and file name for an RT-11 programmed request. (Refer to Chapter 5 for an explanation of .RAD50 strings.) For example, a device block representing a file FILE.EXT on device DK: could be written as:

```
        .RAD50  /DK /
        .RAD50  /FIL/
        .RAD50  /E  /
        .RAD50  /EXT/
```

The first word contains the device name, the second and third words contain the file name, and the fourth contains the extension. Device, name, and extension must each be left-justified in the appropriate field. This string could also be written as:

```
        .RAD50  /DK FILE  EXT/
```

Note that spaces must be used to fill out each field. Note also that the colon and period separators do not appear in the actual RAD50 string. They are used only by the monitor keyboard interface to delimit the various fields.

9.2.3 EMT Argument Blocks

Programmed requests which call the monitor via EMT 375 use R0 as a pointer to an argument list. In general, this argument list appears as follows:

address	contents	
x	Function Code	Channel Number
x+2	argument1	
x+4	argument2	
.	.	.
.	.	.
.	.	.

R0 points to location x. The even (low-order) byte of location x contains the channel number named in the macro call. If no channel number is required, the byte is set to 0. The odd (high-order) byte of x is a code specifying the function to be performed. Locations x+2, x+4, etc. contain arguments to be interpreted. These are described in detail under each request.

Requests which use EMT 374 set up R0 with the channel number in the even byte and the function code in the odd byte. They require no other arguments.

9.2.4 Important Memory Areas

9.2.4.1 Vector Addresses (0-37, 60-477) - Certain areas of memory between 0 and 477 are reserved for use by RT-11. KMON does not load these locations from the save image file when it initiates a program, i.e., R, RUN, and GET will not load these words. However, no hardware memory protection is supplied. Thus, programs should never alter the contents of the indicated areas at run-time.

Locations	Contents
0,2	Monitor restart. Executes .EXIT request and returns control to KMON.
4,6	Time out or bus error trap; RT-11 sets this to point to its internal trap handler.
10,12	Reserved instruction trap; RT-11 sets this to point to its internal trap handler.
30,32	EMT trap vector and status.
40-57	RT-11 system communication area (see below).
60,62	TTY input interrupt vector and status.
64,66	TTY output interrupt vector and status.
100,102	KW11L vector and status.
204,206	RF11 vector and status.
214,216	TC11 vector and status.
220,222	RK05 vector and status.
330,332	GT40 shift out interrupt vector and status.

These areas are not replaced by RT-11. If they are destroyed by a program, the system must be re-bootstrapped, or the program must restore them.

9.2.4.2 Resident Monitor - Section 2.4 of Chapter 2 describes the placement of monitor components when either the Single-Job Monitor or F/B Monitor is brought into memory; included is the approximate size of each monitor component and the size of the area available for handlers and user programs.

9.2.4.3 System Communication Area - RT-11 uses bytes 40-57 to hold information about the program currently executing, as well as certain information used only by the monitor. A description of these bytes follows:

Bytes	Meaning and Use
40,41	Start address of job. When a file is linked into an RT-11 memory image, this word is set to the starting address of the job either with the Linker /T switch or as an argument in the .END statement of the program. When a foreground program is executed, the FRUN processor relocates this word to contain the actual starting address of the program.
42,43	Initial value of the stack pointer. If it is not set by the user program in an .ASECT, it defaults to 1000 or the top of the .ASECT in the background, whichever is larger. If a foreground program does not specify a stack pointer in this word, a default stack (128 decimal words) is allocated by FRUN immediately below the program. The initial stack pointer can also be set with the Linker /M switch option.
44,45	Job Status Word. Used as a flag word for the monitor. Certain bits are maintained by the monitor exclusively while others must be set or cleared by the user job. Those bits in the following list which are marked by an asterisk are bits which must be set by the user job.

Since the currently unassigned bits may be used in future releases of RT-11, user programs should not use these bits for internal flags.

Bit Number	Meaning
15	USR swap bit. (Unused in F/B.) The monitor sets this bit when programs do not require the USR to be swapped. See Section 9.2.5 for details on USR swapping.
14	Lower-case bit. When set (automatically by EDIT when the EL command is typed), disables conversion of lower-case to upper-case.
*13	Reenter bit. When set, this bit indicates that the program may be restarted from the terminal with the REENTER command.

*12 Special mode TT bit. When set, this bit indicates that the job is in a "special" keyboard mode of input. Refer to the explanation of the .TTYIN/.TTINR requests for details.

11-10 For F/B Monitor use only.

9 Overlay Bit. Set (by the Linker) if the job uses the Linker overlay structure.

8 CHAIN bit. If this bit is set in a job's save image, words 500-776 are loaded from the save file when the job is started even if the job is entered via CHAIN. (These words are normally used to pass parameters across CHAINs.) The bit is set when a job is running if and only if the job was actually entered with CHAIN.

*7 Error halt bit. When set, this bit indicates a halt on an I/O error. If the user desires to halt when any I/O device error occurs, this bit should be set. (Unused in F/B.)

*6 Inhibit TT wait bit. For use with the Foreground/Background system. When set, this bit inhibits the monitor from entering a console terminal wait state. Refer to the sections concerning .TTYIN/.TTINR, and .TTYOUT/.TTOUTR for more information.

5-0 Unused.

46,47 USR load address. Normally 0, this word may be set to any valid word address in the user's program. See Section 9.2.5, Swapping Algorithm, for details of use.

50,51 High memory address. The monitor maintains the highest address the user program can use in this word. The Linker sets it initially. It is modified only via the .SETTOP (Set Top of Memory) monitor request.

52 EMT error code. If a monitor request results in an error, the code number of the error is always returned in byte 52 and the carry bit is set. Each monitor call has its own set of possible errors. It is recommended that the user program reference byte 52 with absolute addressing, rather than relative addressing. For example:

```
ERRWRD = 52
TSTB ERRWRD           ;RELATIVE ADDRESSING
TSTB @#ERRWRD         ;ABSOLUTE ADDRESSING
```

NOTE

> Location 52 must always be addressed as a byte, never as a word, since byte 53 will be used in future releases of RT-11.

53 Reserved for future system use.

54,55 Address of the beginning of the Resident Monitor. RT-11 always loads the resident into the highest available memory locations; this word points to its first location. It must never be altered by the user. Doing so will cause RT-11 to malfunction.

56 Fill character (7-bit ASCII). Some high-speed terminals require filler (null) characters after printing certain characters. Byte 56 should contain the ASCII 7-bit representation of the character after which fillers are required.

57 Fill count. This byte specifies the number of fill characters required. If bytes 56 and 57=0, no fillers are required.

The required fill characters are:

Terminal	No. of fills	Value of Word 56
Serial LA30 @ 300 baud	10 after carriage return	5015
Serial LA30 @ 150 baud	4 after carriage return	2015
Serial LA30 @ 110 baud	2 after carriage return	1015
VT05 @ 2400 baud	4 after line feed	2012
VT05 @ 1200 baud	2 after line feed	1012
VT05 @ 600 baud	1 after line feed	412

9.2.5 Swapping Algorithm

Programmed requests are divided into two categories according to whether or not they require the USR to be in memory (see Table 9-2). Any request which requires the USR in memory may also require that a portion of the user program be saved temporarily on the system device scratch blocks (i.e., be "swapped out") to provide room for the USR. The USR will be read into the swapped region.

During most normal operations, this swapping is invisible to the user and he need not be concerned about it. However, it is possible to optimize programs so that they require little or no swapping. This is particularly useful when operating in an F/B environment, since under the F/B system, the USR will be swapped for both background and foreground jobs regardless of which job required it. If the USR is not swapped, neither the foreground nor the background job will be slowed down by the swapping process.

The following items should be considered if a swap operation is necessary:

1. The background job - If a .SETTOP request in a background job specifies an address beyond the point at which the USR

normally resides, a swap will be required when the USR is called. More details concerning the .SETTOP request are in Section 9.4.36.

2. The value of location 46 - If the user either assembles an address into word 46 or moves a value there while the program is running, RT-11 uses the contents of that word as an alternate place to swap the USR. If location 46 is 0, this indicates that the USR will be at its normal location in high memory.

NOTES

1. If the USR does not require swapping, the value in location 46 is ignored. Swapping is a relatively time-consuming operation and is avoided, if possible.

2. A foreground job should always have a value in location 46 unless it is certain that the USR will never be swapped. If the foreground job does not allow space for the USR and a swap is required, a fatal error occurs. (The SET USR NOSWAP command, explained in Chapter 2, ensures that the USR will be resident.)

3. Care should be taken when specifying an alternate address to location 46. The single-job system does not verify the legality of the USR swap address. Thus, if the area to be swapped overlays the Resident Monitor, the system is destroyed.

4. The user should also take care that the USR is never swapped over any of the following areas: the program stack; any parameter block for calls to the USR; any I/O buffers, device handlers, or completion routines being used when the USR is called.

The following is an example of the way a background program can avoid unnecessary USR swapping.

```
        .MCALL  ..V2..,.REGDEF,.SETTOP,.EXIT
        ..V2..
        .REGDEF
RMPTR=54                        ;POINTER TO RMON IS AT 54.
USRLOC=266                      ;POINTER TO USR LOCATION IS
                                ;AT 266 BYTES INTO RMON.
START:
        MOV     @#RMPTR,R1      ;R1 -> RESIDENT MONITOR
        MOV     USRLOC(R1),R0   ;R0 -> USR
        TST     -(R0)           ;POINT JUST BELOW
```

```
        CMP     R0,@#50         ;DOES USR SWAP OVER US?
        BHI     1$              ;NO, OK
        MOV     #-2,R0          ;YES, USR MUST SWAP
1$:     .SETTOP                 ;ASK FOR MEMORY UP TO USR
        MOV     R0,HILIM        ;R0 = HIGH LIMIT OF MEMORY
                                ;ACTUALLY GRANTED BY MONITOR.
        .EXIT
HILIM:  .WORD   0               ;CONTAINS HI LIMIT OF MEMORY
        .END    START
```

9.2.6 Offset Words

There are several words which always have fixed positions relative to the start of the Resident Monitor. It is often advantageous for user programs to be able to access these words. This is done with the code:

```
RMON = 54
MOV @#RMON,register
MOV OFFSET(register),register
```

Here, register is any general register and OFFSET is a number from the following list:

OFFSET (Bytes)	Contents
262	System date. (See .DATE request.)
266	Start of normal USR area. This is where the USR will reside when it is non-swapping. It is useful to be able to perform a .SETTOP in a background job such that the USR is always resident. (An example is in Section 9.2.5.)
270	Address of I/O exit routine for all devices. The exit routine is an internal queue management routine through which all device handlers exit once the I/O transfer is complete. Any new devices added to RT-11 must also use this exit location.
275	Unit number of system device (device from which system was last bootstrapped).
276	Monitor version number (2-377). The user can always access the version number to determine if the most recent monitor is in use.
277	Update number. Patches to the monitor always increment the update number. This provides a means of checking that all patches have been made. (This number should be accessed by MOVB rather than MOV).
300	Configuration word. This is a string of 16 bits used to indicate information about either the hardware configuration of the system, or a software condition. The bits and their meanings are:

Bit #	Meaning
0	0 = Single-Job Monitor 1 = F/B Monitor
2	1 = GT40 display hardware exists
3	1 = RT-11 BATCH is in control of the background
5	0 = 60-cycle clock 1 = 50-cycle clock
6	1 = 11/45 floating-point hardware exists
7	0 = No foreground job is in memory 1 = Foreground job is in memory
8	1 = User is linked to the GT40 scroller
9	1 = USR is permanently resident (via a SET USR NOSWAP)
11	1 = Processor is an 11/Ø3
15	1 = KW11L clock is present (always set if 11/Ø3)

The other bits are reserved for future use and should not be accessed by user programs.

304-313 These locations contain the addresses of the console terminal control and status registers. The order is:

304 Keyboard status
306 Keyboard buffer
310 Printer status
312 Printer buffer

These locations can be changed, for example, to reflect a second terminal; thus RT-11 can be made to run on any terminal present on the system which is connected to the machine via the DL11 multiple terminal interface. (Refer to the RT-11 Software Support Manual (DEC-11-ORPGA-B-D).

314 The maximum file size allowed in a 0 length .ENTER. This can be adjusted by the user program or by using the PATCH program to be any reasonable value. The default value is 177777 (decimal) blocks, allowing an essentially unlimited file size.

324 Address of .SYNCH entry. User interrupt routines may enter the monitor through this address to synchronize with the job they are servicing.

354 Address of VT11 display processor display stop interrupt vector.

9.2.7 File Structure

RT-11 uses a "contiguous" file structure. This type of structure implies that every file on the device is made up of a contiguous group of physical blocks. Thus, a file that is 9 blocks long occupies 9 contiguous blocks on the device.

A contiguous area on a device can be in one of the following categories:

1. Permanent file. This is a file which has been .CLOSEd on a device. Any named files which appear in a PIP directory listing are permanent files.

2. Tentative file. Any file which has been created via .ENTER, but not .CLOSEd, is a tentative file entry. When the .CLOSE request is given, the tentative entry becomes a permanent file. If a permanent file already exists under the same name, the old file is deleted. If a .CLOSE is never given, the tentative file is treated like an empty entry.

3. Empty entry. When disk space is unused or a permanent file is deleted, an empty entry is created. Empty entries appear in a PIP /E directory listing as <UNUSED> N, where N is the decimal block length of the empty area.

Since a contiguous structure does not automatically reclaim unused disk space, the device may eventually become "fragmented". A device is fragmented when there are many empty entries which are scattered over the device. RT-11 PIP has an option which allows the user to collect all empty areas so that they occur at the end of a device. Refer to Chapter 4 for details.

9.2.8 Completion Routines

Completion routines are user-written routines which are entered following an operation. On entry to a completion routine, R0 contains the channel status word for the operation; R1 contains the octal channel number of the operation. The carry bit is not significant.

Completion routines are handled differently in the Single-Job and the F/B versions of RT-11. In the Single-Job version, completion routines are totally asynchronous and can interrupt one another. In F/B, completion routines do not interrupt each other. Instead they are queued and made to wait until the correct job is running. For example, if a foreground job is running and an I/O transfer initiated by a background job completes and wants to go to a completion routine, the background routine is queued and will not execute until the foreground gives up control of the system. If the foreground is running and a foreground I/O transfer completes and wants a completion routine, that routine will be entered immediately if the foreground is not already inside a completion routine. If it is in a completion routine, that routine continues to termination, at which point any other completion routines are entered in a first in/first out manner. If the background is running and a foreground I/O transfer completes and needs a completion routine, the background is suspended and the foreground routine is entered immediately.

The restrictions which must be observed when writing completion routines are:

1. Completion functions cannot issue a request which would cause the USR to be swapped in. They are primarily used for issuing READ/WRITE commands, not for opening or closing files, etc. A fatal monitor error is generated if the USR is called from a completion routine.

2. Completion routines should never reside in the memory space which will be used for the USR, since the USR can be interrupted when I/O terminates and the completion routine is entered. If the USR has overlaid the routine, control passes to a random place in the USR, with a HALT or error trap the likely result.

3. The routine must be exited via an RTS PC, as it is called from the monitor via a JSR PC,ADDR where ADDR is the user-supplied address.

4. If a completion routine uses registers other than R0 or R1, it must save them upon entry and restore them before exiting.

9.2.9 Using the System Macro Library

User programs for RT-11 should always be written using the system macro library (SYSMAC.SML), supplied with RT-11. This ensures compatibility among all user programs and allows easy modification by redefining a macro. A listing of SYSMAC.SML appears in Appendix D.

The system macro library for 8K systems appears on the system device as SYSMAC.8K.

Suggestions for writing foreground programs are in Appendix H, F/B Programming and Device Handlers. This appendix should be read in conjunction with Chapter 9 before coding F/B programs.

9.3 TYPES OF PROGRAMMED REQUESTS

There are three types of services which the monitor makes available to the user through programmed requests. These are:

1. Requests for File Manipulation

2. Requests for Data Transfer

3. Requests for Miscellaneous Services

Table 9-1 summarizes the programmed requests in each of these categories alphabetically. Those marked with an asterisk function only in a F/B environment; they are ignored under the Single-Job Monitor. The EMT and function code for each request (where applicable) are included.

Table 9-1
Summary of Programmed Requests

Mnemonic	EMT & Code		Section	Purpose
File Manipulation Requests				
*.CHCOPY	375	13	9.4.3	Establishes a link and allows one job to access another job's channel.
.CLOSE	374	6	9.4.4	Closes the specified channel.
.DELETE	375	0	9.4.10	Deletes the file from the specified device.
.ENTER	375	2	9.4.13	Creates a new file for output.
.LOOKUP	375	1	9.4.21	Opens an existing file for input and/or output via the specified channel.
.RENAME	375	4	9.4.32	Changes the name of the indicated file to a new name.
.REOPEN	375	6	9.3.33	Restores the parameters stored via a SAVESTATUS request and reopens the channel for I/O.
.SAVESTATUS	375	5	9.4.34	Saves the status parameters of an open file in user memory and frees the channel for future use.
Data Transfer Requests				
*.RCVD *.RCVDW *.RCVDC	375	26	9.4.29	Receives data. Allows a job to read messages or data sent by another job in an F/B environment. The three modes correspond to the READ, .READC, and READW modes.
.READ	375	10	9.4.30	Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. No special action is taken upon completion of I/O.
.READC	375	10	9.4.30	Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the read, control transfers asynchronously to the routine specified in the .READC request.

 (continued on next page)

Table 9-1 (Cont.)
Summary of Programmed Requests

Mnemonic	EMT & Code		Section	Purpose
.READW	375	10	9.4.30	Transfers data via the specified channel to a memory buffer and returns control to the user program only after the transfer is complete.
*.SDAT *.SDATC *.SDATW	375	25	9.4.35	Allows the user to send messages or data to the other job in an F/B environment. The three modes correspond to the .WRITE, .WRITC and .WRITW modes.
.TTYIN .TTINR	340	--	9.4.43	Transfers one character from the keyboard buffer to R0.
.TTYOUT .TTOUTR	341	--	9.4.44	Transfers one character from R0 to the terminal input buffer.
.WRITE	375	11	9.4.47	Transfers data via the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. No special action is taken upon completion of the I/O.
.WRITC	375	11	9.4.47	Transfers data via the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the write, control transfers asynchronously to the routine specified in the .WRITC request.
.WRITW	375	11	9.4.47	Transfers data via the specified channel to a device and returns control to the user program only after the transfer is complete.
Miscellaneous Services				
.CDFN	375	15	9.4.1	Defines additional channels for doing I/O.
.CHAIN	374	10	9.4.2	Chains to another program (in the background job only).
*.CMKT	375	23	9.4.5	Cancels an unexpired mark time request.
*.CNTXSW	375	33	9.4.6	Requests that the indicated memory locations be part of the F/B context switch process.
.CSIGEN	344	--	9.4.7	Calls the Command String Interpreter (CSI) in general mode.
.CSISPC	345	--	9.4.8	Calls the CSI in special mode.

(continued on next page)

Table 9-1 (Cont.)
Summary of Programmed Requests

Mnemonic	EMT & Code		Section	Purpose
*.CSTAT	375	27	9.4.9	Returns the status of the channel indicated.
.DATE	---	--	9.3.1.1	Moves the current date information into R0.
*.DEVICE	375	14	9.4.11	Allows user to turn off device interrupt enable in F/B upon program termination.
.DSTATUS	342	--	9.4.12	Returns the status of a particular device.
.EXIT	350	--	9.4.14	Exits the user program and returns control to the Keyboard Monitor.
.FETCH	343	--	9.4.15	Loads device handlers into memory.
.GTIM	375	21	9.4.16	Gets time of day.
.GTJB	375	20	9.4.17	Gets parameters of this job.
.HERR	374	5	9.4.18	Specifies termination of the job on fatal errors.
.HRESET	357	--	9.4.19	Terminates I/O transfers and does a .SRESET operation.
.INTEN	---	--	9.3.1.2	Notifies monitor that an interrupt has occurred and to switch to "system state", and sets the processor priority to the correct value.
.LOCK	346	--	9.4.20	Makes the monitor User Service Routines (USR) permanently resident until .EXIT or .UNLOCK is executed. The user program is swapped out if necessary.
.MFPS	---	--	9.3.1.3	Reads the priority bits in the processor status word (does not read the condition codes).
*.MRKT	375	22	9.4.22	Marks time; i.e., sets asynchronous routine to occur after a specified interval.
.MTPS	---	--	9.3.1.3	Sets the priority bits, condition codes, and T bit in the processor status word.
*.MWAIT	374	11	9.3.23	Waits for messages to be processed.
.PRINT	351	--	9.4.24	Outputs an ASCII string to the terminal.
*.PROTECT	375	31	9.4.25	Requests that vectors in the area from 0-476 be given exclusively to this job.

(continued on next page)

Table 9-1 (Cont.)
Summary of Programmed Requests

Mnemonic	EMT & Code		Section	Purpose
.PURGE	374	3	9.4.26	Clears out a channel.
.QSET	353	--	9.4.27	Expands the size of the monitor I/O queue.
.RCTRLO	355	--	9.4.28	Enables output to the terminal.
.REGDEF	---	--	9.3.1.4	Defines the PDP-11 general registers.
.RELEAS	343	--	9.4.31	Removes device handlers from memory.
*.RSUM	374	2	9.4.39	Causes the main line of the job to be resumed where it was suspended with .SPND.
.SERR	374	4	9.4.18	Inhibits most fatal errors from causing the job to the aborted.
.SETTOP	354	--	9.4.36	Specifies the highest memory location to be used by the user program.
.SFPA	375	30	9.4.37	Sets user interrupt for floating point processor exceptions.
.SPFUN	375	32	9.4.38	Performs special functions on magtape and cassette units.
*.SPND	374	1	9.4.39	Causes the running job to be suspended.
.SRESET	352	--	9.4.40	Resets all channels and releases the device handlers from memory.
.SYNCH	---	--	9.3.1.5	Enables user program to perform monitor programmed requests from within an interrupt service routine.
*.TLOCK	374	7	9.4.41	Indicates if the USR is currently being used by another job and performs a .LOCK if available.
.TRPSET	375	3	9.4.42	Sets a user intercept for traps to locations 4 and 10.
*.TWAIT	375	24	9.4.45	Suspends the running job for a specified amount of time.
.UNLOCK	347	--	9.4.20	Releases USR if a LOCK was done. The user program is swapped in if required.
..V1..	---	--	9.3.1.6	Enables expansions to occur in Version 1 format.
..V2..	---	--	9.3.1.6	Enables expansions to occur in Version 2 format.
.WAIT	374	0	9.4.46	Waits for completion of all I/O on a specified channel.

Requests requiring the USR (as explained in Section 9.2.5) differ between the Single-Job and F/B Monitors. Table 9-2 indicates which requests require the USR to be in memory. Those requests marked by an asterisk are Version 2 macros only. The CLOSE request on non-file structured devices (LP, PP, TT, etc.) does not require the USR under either monitor.

This page intentionally blank.

Table 9-2
Requests Requiring the USR

Request	F/B	Single-Job
*.CDFN	No	Yes
*.CHAIN	No	No
*.CHCOPY	No	N/A
.CLOSE (see Note 1)	Yes	Yes
*.CMKT	No	N/A
*.CNTXSW	No	N/A
.CSIGEN	Yes	Yes
.CSISPC	Yes	Yes
*.CSTAT	No	N/A
.DELETE	Yes	Yes
*.DEVICE	No	N/A
.DSTATUS	Yes	Yes
.ENTER	Yes	Yes
.EXIT	No	No
.FETCH	Yes	Yes
*.GTIM	No	No
*.GTJB	No	No
*.HERR	No	No
.HRESET	No	Yes
.LOCK (see Note 2)	Yes	Yes
.LOOKUP	Yes	Yes
*.MRKT	No	N/A
*.MWAIT	No	N/A
.PRINT	No	No
*.PROTECT	No	N/A
*.PURGE	No	No
.QSET	Yes	Yes
.RCTRLO	No	No
*.RCVD/.RCVDC/.RCVDW	No	N/A
.READ/.READC/.READW	No	No
.RELEAS	Yes	Yes
.RENAME	Yes	Yes
.REOPEN	No	No
*.RSUM	No	N/A
.SAVESTATUS	No	No
*.SDAT/.SDATC/.SDATW	No	N/A
*.SERR	No	No
.SETTOP	No	No
*.SFPA	No	No
*.SPFUN	No	No
*.SPND	No	N/A
*.SRESET	No	Yes
*.TLOCK (see Note 3)	No	No
*.TRPSET	No	No
.TTINR/.TTYIN	No	No
.TTOUTR/.TTYOUT	No	No
*.TWAIT	No	N/A
.UNLOCK	No	No
.WAIT	No	No
.WRITE/.WRITC/.WRITW	No	No

Note 1: Only if channel was opened via .ENTER.

Note 2: Only if USR is in a swapping state.

Note 3: Only if USR is not in use by the other job.

9.3.1 System Macros

The following five macros are included in the system macro library, but are not programmed requests in that they cause no EMT instruction to be generated:

```
.DATE      .SYNCH
.INTEN     ..V2..
.REGDEF
```

They can be used in the same manner as the other macro calls; their explanations follow.

.DATE

9.3.1.1 .DATE

This request moves the current date information from the system date word into R0. The date word returned is in the following format:

Bit:		14 10	9 5	4 0
		MONTH (1-12.)	DAY (1-31.)	YEAR-72(DECIMAL)

Macro Call: .DATE

Errors:

No errors are returned. A zero result in R0 indicates that no DATE command was entered.

.INTEN

9.3.1.2 .INTEN

This request is used by user program interrupt service routines to:

1. Notify the monitor that an interrupt has occurred and to switch to "system state",

2. Set the processor priority to the correct value.

In Version 2 of RT-11, all external interrupts cause the processor to go to level 7 (see Appendix H). .INTEN is used to lower the priority to the value at which the device should be run. On return from .INTEN, the device interrupt can be serviced, at which point the interrupt routine returns via an RTS PC. It is very important to note that an RTI will not return correctly from an interrupt routine which specifies an .INTEN.

Macro Call: .INTEN .priority, pic

where: .priority is the processor priority at which the user wishes to run his interrupt routine.

pic is an optional argument which should be non-blank if the interrupt routine is written as a PIC (position independent code) routine. If the routine does not have to be PIC, it is recommended that the PIC field be left blank; the non-PIC version is slightly faster than the PIC version.

The user is advised to read Appendix H for more details concerning the use of .INTEN and .SYNCH.

Errors:

None.

Example:

Refer to Section 9.3.1.5, .SYNCH, for an example.

.MFPS/.MTPS

9.3.1.3 .MFPS/.MTPS

The .MFPS and .MTPS macro calls allow processor-independent user access to the processor status word.

The .MFPS call is used to read the priority bits only; condition codes are destroyed during the call and must be directly accessed (using conditional branch instructions) if they are to be read in a processor-independent manner.

Macro Call: .MFPS .addr

where:	.addr	is the address into which the processor status is to be stored; if .addr is not defined, the value is returned on the stack. Note that only the priority bits are significant.

The .MTPS call is used to set the priority, condition codes, and T bit with the value designated in the call.

Macro Call: .MTPS .addr

where:	.addr	is the address of the word to be placed in the processor status word; if .addr is not defined, the processor status word is taken from the stack. Note that the high byte on the stack is set to zero when .addr is present. If .addr is not present, the user should set the stack to the appropriate value. In either case, the whole word on the stack is put in the processor status word.

The contents of R0 are preserved across either call.

Errors:

None.

Example:

```
        .MCALL ..V2..,.REGDEF,.MFPS,.MTPS,.EXIT
        ..V2..
        .REGDEF

START:  JSR PC,PICKQ            ;PICK A QUEUE ELEMENT
                .
                .
                .
        .EXIT

PICKQ:  .MFPS                   ;SAVE PREVIOUS PRIORITY IN SP
        MOV #QHEAD,R4           ;POINT TO QUEUE HEAD
        .MTPS #340              ;RAISE PRIORITY TO 7
        MOV @R4,R5              ;R5 POINTS TO NEXT ELEMENT
        BEQ 10$                 ;NO MORE ELEMENTS AVAILABLE
        MOV @R5,@R4             ;RELINK THE QUEUE
        .MTPS                   ;RESTORE PREVIOUS PRIORITY
        CLZ                     ;FLAG SUCCESS
10$:    RTS PC
QHEAD:  .WORD Q1                ;QUEUE HEAD

;THREE QUEUE ELEMENTS
Q1:     .WORD Q2,0,0
Q2:     .WORD Q3,0,0
Q3:     .WORD 0,0,0

        .END START
```

.REGDEF

9.3.1.4 .REGDEF

This macro call defines the PDP-11 general registers as R0 through R5, SP, and PC.

Macro Call: .MCALL .REGDEF,...
.REGDEF

Errors:

None.

Example:

Refer to the example for the .SYNCH request. Appendix D shows the expansion of .REGDEF.

.SYNCH

9.3.1.5 .SYNCH

This macro call enables the user program to perform monitor programmed requests from within an interrupt service routine. Unless a .SYNCH is used, issuing programmed requests from interrupt routines is not supported by the system and should not be performed. .SYNCH, like .INTEN and .DATE, is not a programmed request and generates no EMT instructions.

Macro Call: .SYNCH .area

where: .area is the address of a seven-word area which the user must set aside for use by .SYNCH. The 7-word block appears as:

Word 1	RT-11 maintains this word; its contents should not be altered by the user.
Word 2	The current job's number. This can be obtained by a .GTJB call.
Word 3	Unused.

Word 4	Unused.
Word 5	R0 argument. When a successful return is made from .SYNCH, R0 contains this argument.
Word 6	Must be -1.
Word 7	Must be 0.

Note:

.SYNCH assumes that the user has not pushed anything on the stack between the .INTEN and .SYNCH calls. This rule must be observed for proper operation.

Errors:

The monitor returns to the location immediately following the .SYNCH if the .SYNCH was rejected. The routine is still unable to issue programmed requests, and R4 and R5 are available for use. Errors returned are due to one of the following:

1. Another .SYNCH which specified the same 7-word block is still pending.

2. An illegal job number was specified in the second word of the block. The only currently legal job numbers are 0 and 2.

3. If the job has been aborted or for some reason is no longer running, the .SYNCH will fail.

Normal return is to the word after the error return with the routine in user state and thus allowed to issue programmed requests. R0 contains the argument which was in word 5 of the block. R0 and R1 are free to be used without having to be saved. (R4 and R5 are not free.) Exit from the routine should be done via an RTS PC. (Refer to Appendix H, Section H.1.4, and to the <u>RT-11 Software Support Manual</u>, Section 6.4.)

Example:

```
        .MCALL  ..V2..,.REGDEF
        ..V2..
        .REGDEF
        .MCALL  .GTJB,.INTEN,.WRITC,.SYNCH,.EXIT,.PRINT
START:  MOV     #JOB,R5             ;OUTPUT OF .GTJB GOES HERE
        .GTJB   #AREA,R5            ;GET JOB NUMBER
        MOV     (R5),SYNBLK+2       ;STORE THE JOB NUMBER INTO SYNCH BLOCK
                                    ;IN HERE WE SET UP INTERRUPT
                                    ;PROCESSING, AND START UP THE
                                    ;INTERRUPTING DEVICE.


INTRPT: .INTEN  5                   ;GO INTO SYSTEM STATE
                                    ;RUN AT LEVEL FIVE
                                    ;INTERRUPT PROCESSING --
                                    ;NOTHING CAN GO ON STACK
        .SYNCH  #SYNBLK             ;TIME TO WRITE A BUFFER
        BR      SYNFAIL             ;SYNCH BLOCK IN USE
```

```
;RETURN HERE AT PRIORITY 0.  NOTE: .SYNCH DOES RTI

        .WRITC  #AREA,CHAN,BUFF,WCNT,#CRTN1,BLK
                                ;WRITE A BUFFER
        BCS     WTFAIL          ;FAILED SOMEHOW

                                ;RE-INITIALIZE FOR MORE
        RTS     PC              ;INTERRUPTS AND EXIT
SYNBLK: .WORD   0
        .WORD   0               ;JOB NUMBER
        .WORD   0
        .WORD   0
        .WORD   5               ;R0 CONTAINS 5 ON SUCCESSFUL
                                ;SYNCH
        .WORD   -1,0            ;SET UP FOR MONITOR
SYNFAIL:.
        .
        .
```

..V1../..V2..

9.3.1.6 ..V1../..V2..

Any program that uses system MACROs must specify the version format (Version 1 or Version 2) in which the macro calls are to be expanded. Assembly errors at macro calls will result if the proper version designation is not made.

The ..V1.. macro call enables all macro expansions to occur in Version 1 format. (Note that any requests marked with an asterisk in Table 9-1 are not valid as Version 1 requests, and thus will be flagged as errors if they are assembled in Version 1 form.)

```
Macro Call:   .MCALL ..V1..
              ..V1..
```

This causes all macros in the program to be assembled in Version 1 form and the symbol ...V1 to be defined. User programs should not use this symbol. The ..V1.. macro expands into:

```
...V1=1
```

To cause all macro expansions to occur in Version 2 format, the ..V2.. macro call is used. Using ..V2.. causes the symbol ...V2 to be defined. Likewise, user programs should not use this symbol.

```
Macro Call:   .MCALL ..V2..
              ..V2..
```

The ..V2.. macro expands into:

```
.MCALL ...CM1,...CM2,...CM3,...CM4
...V2=1
```

Note:

It is possible that user programs will exist in which both Version 1 and Version 2 macros are present. To allow proper assembly, the user should include the statements:

```
.MCALL ..V1..,...CM1,...CM2,...CM3,...CM4
..V1..
```

to define the utility macros (CM1, CM2, etc.) used by other Version 2 macros. This causes all macros which existed in Version 1 to assemble in Version 1 format, while those macros new to Version 2 are correctly generated as Version 2 macros. Note that in this case a macro which existed in Version 1 (such as .READ) will expand in the Version 1 format.

Run-time or assembly errors will occur if both the ..V1.. and ..V2.. macro calls are used in a program.

Example:

All examples in Chapter 9 illustrate the Version 2 format. Users are urged to use the ..V2.. macro call in their programs.

9.4 PROGRAMMED REQUEST USAGE

This section provides a description of each of the programmed requests alphabetically. The following parameters are commonly used as arguments in the various calls:

.addr	**an address, the meaning of which depends on the request being used**
.area	**a pointer to the EMT argument list (for those requests which require a list); see Section 9.2.3**
.blk	**a block number specifying the relative block in a file where an I/O transfer is to begin**
.buff	**a buffer address specifying a memory location into or from which an I/O transfer is to be performed**
.chan	**a channel number in the range 0-377(octal)**
.crtn	**the entry point of a completion routine; see Section 9.2.8**
.count	**file number for magtape/cassette operations (see Appendix H); if this argument is blank, a value of 0 is assumed**
.dblk	**the address of a four-word RAD50 descriptor of the file to be operated upon; see Section 9.2.2**
.num	**a number, the value of which depends on the request**
.wcnt	**a word count specifying the number of words to be transferred to or from the buffer during an I/O operation**

Additional information concerning these parameters (and others not defined here) is provided as necessary under each request.

.CDFN

9.4.1 .CDFN

The .CDFN request is used to redefine the number of I/O channels. Each job, whether foreground or background, is initially provided with 16(decimal) I/O channels, numbered 0-15. .CDFN allows the number to be expanded to as many as 255(decimal) channels.

Note that .CDFN defines new channels; the previously-defined channels are not used. Thus, a .CDFN for 20(decimal) channels (while the 16 original channels are defined) causes only 20 I/O channels; the space for the original 16 is unused.

Note that if a program is overlaid, channel 15 is used by the overlay handler and should not be modified. (Other channels can be defined and used as usual.)

Macro Call: .CDFN .area, .addr, .num

where: .addr is the address where the I/O channels begin

.num is the number of I/O channels to be created

Request Format:

R0 → .area:

15	0
.addr	
.num	

The space used to contain the new channels is taken from within the user program. Each I/O channel requires 5 words of memory. Thus, the user must allocate 5*N words of memory, where N is the number of channels to be defined.

It is recommended that the .CDFN request be used at the beginning of a program, before any I/O operations have been initiated. If more than one .CDFN request is used, the channel areas must either start at the same location or not overlap at all. The two requests .SRESET and .HRESET cause the user's channels to revert to the original 16 channels defined at program initiation. Hence, any .CDFNs must be reissued after using those directives.

Errors:

Code	Explanation
0	An attempt was made to define fewer channels than already exist.

Example:

```
        .MCALL  ..V2..,.REGDEF
        ..V2..
        .REGDEF
        .MCALL  .CDFN,.PRINT,.EXIT
START:  .CDFN   #R0LIST,#CHANL,#40.
        BCS     BADCDF
        .PRINT  #MSG1
        .EXIT
BADCDF: .PRINT  #MSG2
        .EXIT
MSG1:   .ASCIZ  /.CDFN O.K./
.EVEN
MSG2:   .ASCIZ  /BAD .CDFN/
```

```
        .EVEN
RØLIST: .BLKW   3               ;EMT ARGUMENT LIST
CHANL:  .BLKW   40.*5           ;ROOM FOR CHANNELS
        .END    START
```

The example defines 40 (decimal) channels to start at location CHANL. An error occurs if 40 or more channels are already defined.

This page intentionally blank.

.CHAIN

9.4.2 .CHAIN

This request allows a background program to pass control directly to another background program without operator intervention. Since this process may be repeated, a large "chain" of programs can be strung together.

The area from locations 500-507 contains the device name and file name (in RAD50) to be chained to, and the area from locations 510-777 is used to pass information between the chained programs.

Macro Call: .CHAIN

Notes:

1. No assumptions should be made concerning which areas of memory will remain intact across a .CHAIN. In general, 500-777 is the only area guaranteed to be preserved across a .CHAIN.

2. I/O channels are left open across a .CHAIN for use by the new program. However, I/O channels opened via a .CDFN request are not available in this way. Since the monitor reverts to to the original 16 channels during a .CHAIN, programs which leave files open across a .CHAIN should not use .CDFN. Furthermore, non-resident device handlers are released during a .CHAIN, and must be .FETCHed again by the new program.

3. A program can determine whether it was CHAINed to or RUN from the keyboard by examining bit 8 of the JSW. This bit is on during program execution only if the program was entered via CHAIN. If a program normally loads into area 500-777, bit 8 of the JSW should be set during program assembly. This causes the monitor to load the area properly. If the bit is not set, locations 500-777 are preserved from the chaining program, causing the new program to malfunction.

Errors:

.CHAIN is implemented by simulating the monitor RUN command (described in Chapter 2), and can produce any errors which RUN can produce. If an error occurs, the .CHAIN is abandoned and the Keyboard Monitor is entered.

When using .CHAIN, care should be taken for initial stack placement, since the program being "chained to" is started. The Linker normally defaults the initial stack to 1000(octal); if caution is not observed, the stack may destroy chain date before it can be used (see Chapter 2, the RUN command).

Example:

```
          .MCALL   ..V2..,.REGDEF
          ..V2..
          .REGDEF
          .MCALL   .CHAIN,.TTYIN
START:
          MOV      #500,R1          ;SET UP TO CHAIN
          MOV      #CHPTR,R2        ;DEVICE, FILE NAME TO 500-511
          .REPT    4
          MOV      (R2)+,(R1)+
          .ENDR
LOOP:     .TTYIN                    ;NOW GET A COMMAND LINE
          MOVB     R0,(R1)+         ;AND PASS IT TO THE JOB
          CMPB     R0,#12           ;IN LOCATIONS 512 AND UP
          BNE      LOOP             ;LOOP UNTIL LINE FEED
          CLRB     (R1)+            ;PUT IN A NULL BYTE
          .CHAIN
CHPTR:    .RAD50   /DK /
          .RAD50   /TECO  /
          .RAD50   /SAV/
          .END     START
```

.CHCOPY

9.4.3 .CHCOPY (F/B only)

The .CHCOPY request opens a channel for input, logically connecting it to a file which is currently open by the other job for either input or output. This request may be used by either the foreground or the background. .CHCOPY must be done before the first .READ or .WRITE.

Macro Call: .CHCOPY .area, .chan, .ochan

where: .chan — is the channel which the job will use to read the data.

.ochan — is the channel number of the other job which is to be copied

Request Format:

R0 ⇒ .area:

13	.chan
.ochan	

.CHCOPY is legal only on files which are on disk or DECtape; however, no errors are detected by the system if another device is used. (To close a channel following use of .CHCOPY, use either the .CLOSE or .PURGE request.)

Notes:

1. If the other job's channel was opened via an .ENTER in order to create a file, the copier's channel indicates a file which extends to the highest block that the creator of the file had written at the time the .CHCOPY was executed.

2. A channel which is open on a nonfile-structured device should not be copied, because intermixture of buffer requests may result.

3. A program can write to a file (which is being created by the other job) on a copied channel just as it could if it were the creator. When the copier's channel is closed, however, no directory update takes place.

Errors:

Code	Explanation
0	Other job does not exist, does not have enough channels defined, or does not have the specified channel (.ochan) open.
1	Channel (.chan) already open.

Example:

In this example, .CHCOPY is used to read data currently being written by the other job. The correct block number and channel to read is obtained by a .RCVDW command. The channel number will be in MSG+4.

```
        .MCALL  ..V2..,.REGDEF
        ..V2..
        .REGDEF
        .MCALL  .CHCOPY,.RCVDW,.PURGE,.READW,.EXIT,.PRINT
ST:
        .PURGE  #0                      ;MAKE SURE WE HAVE CLEAR
                                        ;CHANNEL
        .RCVDW  #AREA,#MSG,#2           ;READ TWO WORDS, BLOCK #
                                        ;AND CHANNEL
        BCS     NOJOB                   ;NO JOB THERE
        .CHCOPY #AREA,#0,MSG+4          ;CHANNEL # IS IN THERE
        BCS     BUSY                    ;BUT BUSY
        .READW  #AREA,#0,#BUFF,#256.,MSG+2 ;GET THE CORRECT BLOCK
        BCS     RDERR
        .PRINT  #OKMSG
        .EXIT
NOJOB:  .PRINT  #MSG1
        .EXIT
BUSY:   .PRINT  #MSG2
        .EXIT
RDERR:  .PRINT  #MSG3
        .EXIT
AREA:   .BLKW   5
MSG:    .BLKW   5
BUFF:   .BLKW   256.
MSG1:   .ASCIZ  /NO JOB!/
MSG2:   .ASCIZ  /BUSY!/
MSG3:   .ASCIZ  /READ ERROR/
OKMSG:  .ASCIZ  /READ OK/
.EVEN
        .EXIT
        .END    ST
```

.CLOSE

9.4.4 .CLOSE

The .CLOSE request terminates activity on the specified channel and frees it for use in another operation. The handler for the associated device must be in memory.

Macro Call: .CLOSE .chan

Request Format:

R0 ⇒ | 6 | .chan |

A .CLOSE is required on any channel opened for either input or output. A .CLOSE request specifying a channel that is not opened is ignored.

A .CLOSE performed on a file which was opened via .ENTER causes the device directory to be updated to make that file permanent. A file opened via .LOOKUP does not require any directory operations. If the device associated with the specified channel already contains a file with the same name and extension, the old copy is deleted when the new file is made permanent. When an entered file is .CLOSEd, its permanent length reflects the highest block written since it was entered; for example, if the highest block written is block number 0, the file is given a length of 1; if the file was never written, it is given a length of 0. If this length is less than the size of the area which was allocated at .ENTER time, the unused blocks are reclaimed as an empty area on the device.

Errors:

.CLOSE does not return any errors. If the device handler for the operation is not in memory, a fatal monitor error is generated.

Example:

An example which illustrates the .CLOSE request follows the discussion of the .WRITW request in Section 9.4.47.

.CMKT

9.4.5 .CMKT (F/B only)

The .CMKT request causes one or more outstanding mark time requests to be cancelled (mark time requests are discussed in Section 9.4.22).

Macro Call: .CMKT .area, .id, .time

where:	.id	is a number used to identify each mark time request to be cancelled. If more than one mark time request has the same .id, that with the earliest expiration time is cancelled. If .id = 0, all nonsystem mark time requests (i.e., in the range 1-177377) for the issuing job are cancelled.
	.time	is the pointer to a two-word area in which the Monitor will return the amount of time remaining in the cancelled request. The first word contains the high-order time, the second contains the low-order. If an address of 0 is specified, no value is returned. If .id = 0, the .time parameter is ignored and need not be indicated.

Request Format:

R0 ⇒ .area:

23	0
.id	
.time	

Notes:

1. Cancelling a mark time request frees the associated queue element for other uses.
2. A mark time request can be converted into a timed wait by issuing a .CMKT followed by a .TWAIT, and specifying the same .time area.

Errors:

Code	Explanation
0	The .id was not zero; a mark time with that identification number could not be found (implying that the request was never issued or that it has already expired).

Example:

See the example following the description of the .MRKT request.

.CNTXSW

9.4.6 .CNTXSW (F/B only)

A context switch is an operation performed when a transition is made from running one job to running the other. The .CNTXSW request is used to specify locations to be included in the context switch.

Macro Call: .CNTXSW .area, .addr

where: .addr is a list of addresses terminated by a zero word. The addresses in the list must be even and:

a. in the range 2-476, or
b. in the user job area, or
c. in the I/O page (addresses 160000-177776).

Request Format:

R0 ⇒ .area:

33	0
.addr	

The system always saves the parameters it needs to uniquely identify and execute a job, including all registers, and the locations:

34/36	Vector for TRAP instruction
40-52	System Communication Area

If an .SFPA request (Section 9.4.37) has been executed with a non-zero address, all floating point registers and the floating point status are also saved.

It is possible that both jobs may want to share the use of a particular location and that location is not included in normal context switch operations. For example, if a program uses the IOT instruction to perform some internal user function (such as print error messages), it must set up the vector at 20 and 22 to point to an internal IOT trap handling routine. If both foreground and background wish to use IOT, the IOT vector must always point to the proper location for the job which is executing. Including locations 20 and 22 in the .CNTXSW list for both jobs will accomplish this.

If .CNTXSW is issued more than once, only the latest list is used; the previous address list is discarded. Thus, all addresses to be switched must be included in one list. If the address (.addr) is zero, no extra locations are switched. The list may not be in an area into which the USR swaps, nor may it be modified while a job is running.

Errors:

Code	Explanation
0	One or more of the above conditions was violated.

Example:

In this example, .CNTXSW request is used to specify that locations 20 and 22 (IOT vector) and certain necessary EAE registers be context switched. This allows both jobs to use IOT and the EAE simultaneously yet independently.

```
        .MCALL  ..V2..,.REGDEF,.CNTXSW,.PRINT,.EXIT
        ..V2..                  ;CALL FOR V2 MACROS
        .REGDEF                 ;DEFINE REGISTERS
START:  MOV     #LIST,R0        ;SET R0 TO OUR OWN LIST
        .CNTXSW ,#SWAPLS        ;THE LIST OF ADDRS IS
                                ;AT SWAPLS.
        BCC     1$
        .PRINT  #ADDERR         ;ADDRESS ERROR(SHOULD NOT
                                ;OCCUR)
        .EXIT
1$:
        .PRINT  #CNTOK
        .EXIT
SWAPLS: .WORD   20              ;ADDRESSES TO INCLUDE IN LIST
        .WORD   22
        .WORD   177302
        .WORD   177304
        .WORD   177310
        .WORD   0
LIST:   .BYTE   0,33            ;FUNCTION CODE WORD
        .WORD   0               ;THE MACRO FILLS THIS ONE.
ADDERR: .ASCIZ  /ADDRESSING ERROR/
.EVEN
CNTOK:  .ASCIZ  /CONTEXT SWITCH O.K./
.EVEN
        .END    START
```

.CSIGEN

9.4.7 .CSIGEN

The .CSIGEN request calls the Command String Interpreter (CSI) in general mode to process a standard RT-11 command string (see Chapter 2 for the description of a standard command string). In general mode, all file .LOOKUPs and .ENTERs as well as handler .FETCHs are performed. When called in general mode, the CSI closes channels 0-10 (octal).

Macro Call: .CSIGEN .devspc, .defext, .cstring

where:

.devspc — is the address of the memory area where the device handlers (if any) are to be loaded.

.defext — is the address of a four-word block which contains the RAD50 default extensions. These extensions are used when a file is specified without an extension.

.cstring — is the address of the ASCIZ input string or a #0 if input is to come from the console terminal. (In a F/B environment only, if the input is from the console terminal, an .UNLOCK of the USR is automatically performed, even if the USR is locked at the time.) If the string is in memory, it must not contain a <CR><LF>, but must terminate with a zero byte. If the .cstring field is left blank, input is automatically taken from the console terminal.

.CSIGEN loads all necessary handlers and opens the files as specified. The area specified for the device handlers must be large enough to hold all the necessary handlers simultaneously. If the device handlers exceed the area available, the user program may be destroyed. The system, however, is protected from this.

When the EMT is complete, register 0 points to the first available location above the handlers.

The four-word block pointed to by .defext is arranged as:

Word 1: default extension for all input channels

Words 2,3,and 4: default extensions for output channels 0,1,2 respectively

If there is no default for a particular position, the associated word must contain a zero. All extensions are expressed in Radix 50. For example, the following block can be used to set up default extensions for a macro assembler:

```
DEFEXT: .RAD50   "MAC"
        .RAD50   "OBJ"
        .RAD50   "LST"
        .WORD    0
```

In the command string:

```
*DT0:ALPHA,DT1:BETA=DT2:INPUT
```

the default extension for input is MAC; for output, OBJ and LST. The following cases are legal:

```
*DT0:OUTPUT=
*DT2:INPUT
```

In other words, the equal sign is not necessary in the event that only input files are specified.

When control returns to the user program after a call to .CSIGEN, all the specified files have been opened for input and/or output. The association is as follows: the three possible output files are assigned to channels 0, 1, and 2; the six input slots are assigned to channels 3 through 10. A null specification causes the associated channel to remain inactive. For example, in the following string:

```
*,LP:=F1,F2
```

channel 0 is inactive since the first slot is null. Channel 1 is associated with the line printer, and channel 2 is inactive. Channels 3 and 4 are associated with two files on DK:, while channels 5 through 10 are inactive. The user program can determine whether a channel is inactive by issuing a .WAIT request on the associated channel, which returns an error if the channel is not open.

Switches and their associated values are returned on the stack; see Section 9.4.8.1 for a description of the way switch information is passed.

Errors:

If CSI errors occur and input was from the console terminal, an error message describing the fault is printed on the terminal and the CSI retries the command (these messages appear in Section 9.4.8.1). If the input was from a string, the carry bit is set and byte 52 contains the error code. The errors are:

Code	Explanation
0	Illegal command (bad separators, illegal filename, command too long, etc.).
1	A device specified is not found in the system tables.
2	Unused.
3	An attempt to .ENTER a file failed because of a full directory.
4	An input file was not found in a .LOOKUP.

Example:

This example uses the general mode of the CSI in a program to copy an input file to an output file. Command input to the CSI is from the console terminal.

```
        .MCALL  ..V2..,.REGDEF
        ..V2..
        .REGDEF
        .MCALL  .CSIGEN,.READW,.PRINT,.EXIT,.WRITW,.CLOSE,.SRESET
ERRWD=52

START:  .CSIGEN #DSPACE,#DEXT     ;GET STRING FROM TERMINAL
        MOV     R0,BUFF           ;R0 HAS FIRST FREE LOCATION
        CLR     INBLK             ;INPUT BLOCK #
        MOV     #LIST,R5          ;EMT ARGUMENT LIST
READ:   .READW  R5,#3,BUFF,#256.,INBLK ;READ CHANNEL 3
        BCC     2$                ;NO ERRORS
        TSTB    @#ERRWD           ;EOF ERROR?
        BEQ     EOF               ;YES
        MOV     #INERR,R0
```

```
1$:     .PRINT                          ;ERROR MESSAGE
        CLR     R0                      ;HARD EXIT
        .EXIT
2$:     .WRITW  R5,#0,BUFF,#256.,,INBLK ;WRITE THE BLOCK
        BCC     NOERR                   ;NO ERROR WRITING
        MOV     #WTERR,R0
        BR      1$                      ;HARD OUTPUT ERROR
NOERR:  INC     INBLK                   ;GET NEXT BLOCK
        BR      READ                    ;LOOP UNTIL DONE
EOF:    .CLOSE  #0                      ;CLOSE OUTPUT CHANNEL
        .CLOSE  #3                      ;AND INPUT CHANNEL
        .SRESET                         ;RELEASE HANDLER FROM MEMORY
        BR      START                   ;GO FOR NEXT COMMAND LINE
DEXT:   .WORD   0,0,0,0                 ;NO DEFAULT EXTENSIONS
BUFF:   .WORD   0                       ;I/O BUFFER START
INBLK:  .WORD   0                       ;RELATIVE BLOCK TO READ/WRITE
LIST:   .BLKW   5                       ;EMT ARGUMENT LIST
INERR:  .ASCIZ  /INPUT ERROR/
        .EVEN
WTERR:  .ASCIZ  /OUTPUT ERROR/
        .EVEN
DSPACE=.                                ;HANDLER SPACE
        .END    START
```

.CSISPC

9.4.8 .CSISPC

The .CSISPC request calls the Command String Interpreter in special mode to parse the command string and return file descriptors and switches to the program. In this mode, the CSI does not perform any handler fetches, .CLOSEs, .ENTERs, or .LOOKUPs.

Macro Call: .CSISPC .outspc, .defext, .cstring

where: .outspc is the address of the 39-word block to contain the file descriptors produced by .CSISPC. This area may overlay the space allocated to .cstring if desired.

.defext is the address of a four-word block which contains the RAD50 default extensions. These extensions are used when a file is specified without an extension.

.cstring — is the address of the ASCIZ input string or a #0 if input is to come from the console terminal. If the string is in memory, it must not contain a <CR><LF> but must terminate with a zero byte. If .cstring is blank, input is automatically taken from the console terminal.

The 39-word file description consists of nine file descriptor blocks (five words for each of three possible output files; four words for each of six possible input files) which correspond to the nine possible files (three output, six input). If any of the nine possible filenames are not specified, the corresponding descriptor block is filled with zeroes.

The five-word blocks hold four words of RAD50 representing dev:file.ext, and 1 word representing the size specification given in the string. (A size specification is a decimal number enclosed in square brackets [], following the output file descriptor.) For example,

```
*DT3:LIST.MAC[15]=PR:
```

Using special mode, the CSI returns in the first five word slot:

16101	.RAD50 for DT3
46173	.RAD50 for LIS
76400	.RAD50 for T
50553	.RAD50 for MAC
00017	Octal value of size request

In the fourth slot (starting at an offset of 36 (octal) bytes into .outspc), the CSI returns:

63320	.RAD50 for PR
0	No file name
0	Specified
0	

Since this is an input file, only four words are returned.

Switches and their associated values are returned on the stack. See Section 9.4.8.1.

Errors:

Errors are the same as in general mode. However, since .LOOKUPs and .ENTERs are not done, the error codes which are valid are:

Code	Explanation
0	Illegal command line
1	Illegal device

Example:

This example illustrates the use of the special mode of CSI. This example could be a program to read a file which is not in RT-11 format to a file under RT-11.

```
        .MCALL  ..V2..,.REGDEF
        ..V2..
        .REGDEF
        .MCALL  .CSISPC,.PRINT,.EXIT,.ENTER,.CLOSE

START:  .CSISPC #OUTSPC,#DEXT,#CSTRNG ;GET INPUT FROM A
                                      ;STRING IN MEMORY
        BCC     2$
        MOV     #SYNERR,R0       ;SYNTAX ERROR
1$:     .PRINT                   ;ERROR MESSAGE
        .EXIT
2$:     .ENTER  #LIST,#0,#OUTSPC,#64. ;ENTER FILE UNDER RT-11
        BCC     3$
        MOV     #ENMSG,R0        ;ENTER FAILED
        BR      1$
3$:     JSR     R5,INPUT         ;ROUTINE INPUT WILL USE
                                 ;THE INFORMATION AT
                                 ;#OUTSPC+36 TO READ INPUT
                                 ;FROM THE NON-RT11 DEVICE.
                                 ;INPUT IS PROCESSED AND
                                 ;WRITTEN VIA .WRITW REQUESTS
        .CLOSE  #0               ;MAKE OUTPUT FILE PERMANENT
        .EXIT                    ;AND EXIT PROGRAM
CSTRNG: .ASCIZ  "DT4:RTFIL.MAC=DT2:DOS.MAC"
        .EVEN
DEXT:   .WORD   0,0,0,0          ;NO DEFAULT EXTENSIONS
LIST:   .BLKW   5                ;LIST FOR EMT CALLS
SYNERR: .ASCIZ  "CSI ERROR"
ENMSG:  .ASCIZ  "ENTER FAILED"
        .EVEN
INPUT:  RTS     R5
OUTSPC=.                         ;CSI LIST GOES HERE
        .END    START
```

9.4.8.1 Passing Switch Information

In both general and special modes of the CSI, switches and their associated values are returned on the stack. A CSI switch is a slash (/) followed by any character. The CSI does not restrict the switch to printing characters, although it is suggested that printing characters be used wherever possible. The switch can be followed by an optional value, which is indicated by a : or ! separator. The : separator is followed by either an octal number or by one to three alphanumeric characters, the first of which must be alphabetic, which are converted to Radix-50. The ! separator is followed by a decimal value. Switches can be associated with files with the CSI. For example:

```
*DK:FOO/A,DT4:FILE.OBJ/A:100
```

In this case, there are two A switches. The first is associated with the input file DK:FOO. The second is associated with the input file DT4:FILE.OBJ, and has a value of 100(8). The stack output of the CSI is as follows:

Word #	Value	Meaning
1 (top of stack)	N	Number of switches found in command string. If N=0, no switches were found.
2	Switch value and file number	Even byte = 7-bit ASCII switch value. Bits 8-14 = Number (0-10) of the file with which the switch is associated. Bit 15 = 1 if the switch had a value. = 0 if the switch had no value.
3	Switch value or next switch	If word 2 was less than 0, word 3 = switch value. If word 2 was greater than 0, this word is the next switch value (if it exists).

For example, if the input to the CSI is:

```
*FILE/B:20,FIL2/E=DT3:INPUT/X:SY:20
```

on return, the stack is:

Stack Pointer→	
3	Three switches appeared.
101530	Last switch=X; with file 3, has a value.
20	Value of switch X=20
101530	Next switch =X; with file 3, has a value.
075250	Next value of switch X=RAD50 code for SY.
505	Next switch=E; associated with file 1, no value.
100102	Switch=B; associated with file 0 and has a value.
20	Value is 20.

As an extended example, assume the following string was input for the CSI in general mode:

```
*FILE[8],LP:,SY:FILE2[20]=PR:,DT1:IN1/B,DT2:IN2/M:7
```

Assume also that the default extension block is:

```
DEFEXT:  .RAD50   'MAC'    ;INPUT EXTENSION
         .RAD50   'OP1'    ;FIRST OUTPUT EXTENSION
         .RAD50   'OP2'    ;SECOND OUTPUT EXTENSION
         .RAD50   'OP3'    ;THIRD OUTPUT EXTENSION
```

The result of this CSI call would be:

1. A file named FILE.OP1 is entered on channel 0 on device DK; channel 1 is open for output to the device LP; a 20-block file named FILE2.OP3 is entered on the system device on channel 2.

2. Channel 3 is open for input from paper tape; channel 4 is open for input from a file IN1.MAC on device DT1; channel 5 is open for input from IN2.MAC on device DT2.

3. The stack contains switches and values as follows:

Stack	Explanation
2	2 switches found in string.
102515	Second switch is M, associated with Channel 5; has a numeric value.
7	Numeric value is 7.
2102	Switch is B, associated with Channel 4; has no numeric value.

If the CSI were called in special mode (Section 9.4.8), the stack would be the same as for the general mode call, and the descriptor table would contain:

```
.OUTSPC:  15270      ;.RAD50    'DK'
          23364      ;.RAD50    'FIL'
          17500      ;.RAD50    'E'
          60137      ;.RAD50    'OP1'
             10      ;LENGTH OF 8 BLOCKS
          46600      ;.RAD50    'LP'
              0      ;NO NAME OR LENGTH SPECIFIED
              0
              0
              0
          75250      ;.RAD50    'SY'
          23364      ;.RAD50    'FIL'
          22100      ;.RAD50    'E2'
          60141      ;.RAD50    'OP3'
             24      ;LENGTH OF 20 (DECIMAL)
          63320      ;.RAD50    'PR'
              0
              0
              0
          16077      ;.RAD50    'DT1'
          35217      ;.RAD50    'IN1'
              0      ;.RAD50    '   '
          50553      ;.RAD50    'MAC'
          16100      ;.RAD50    'DT2'
          35220      ;.RAD50    'IN2'
              0      ;.RAD50    '   '
          50553      ;.RAD50    'MAC'
              0
              .
              .
              .
              0      (twelve more zero words
                     are returned)
```

Keyboard error messages which may occur from incorrect use of the CSI when input is from the console keyboard include:

Message	Meaning
?ILL CMD?	Syntax error.
?FIL NOT FND?	Input file was not found.

?DEV FUL?	Output file will not fit.
?ILL DEV?	Device specified does not exist.

Notes:

1. In many cases, the user program does not need to process switches in CSI calls. However, the user at the console may inadvertently enter switches. In this case, it is wise for the program to save the value of the stack pointer before the call to the CSI, and restore it after the call. In this way, no extraneous values will be left on the stack.

2. In the F/B System, calls to the CSI which require console terminal input will always do an implicit .UNLOCK of the USR. This should be kept in mind when using .LOCK calls.

.CSTAT

9.4.9 .CSTAT (F/B only)

This request furnishes the user with information about a channel. It is supported only in the F/B environment; no information is returned in the Single-Job Monitor.

Macro Call: .CSTAT .area, .chan, .addr

where: .addr is the address of a 6-word block which is to contain the status

Request Format:

R0 → .area:

27	.chan
.addr	

The 6 words passed back to the user are:

1. Channel status word (see Section 9.4.34)
2. Starting block number of file (0 if sequential-access device or if channel was opened with a nonfile-structured .LOOKUP or .ENTER)
3. Length of file (no information if nonfile-structured device or if channel was opened with a nonfile-structured .LOOKUP or .ENTER)
4. Highest block written since file was opened (no information if nonfile-structured device)
5. Unit number of device with which this channel is associated
6. RAD50 of the device name with which the channel is associated (this is a physical device name, unaffected by any user name ASSIGNment in effect)

The fourth word (highest block) is maintained by the .WRITE requests. If data is being written on this channel, the highest relative block number is kept in this word.

Errors:

Code	Explanation
0	The channel is not open.

Example:

In this example, .CSTAT is used to determine the .RAD50 representation of the device with which the channel is associated.

```
                .MCALL   ..V2..,.REGDEF,.CSIGEN,.CSTAT
                ..V2..
                .REGDEF

                .MCALL   .PRINT,.EXIT


        ST:     .CSIGEN #DEVSDC,#DEFEXT ;OPEN FILES
                .CSTAT  #AREA,#0,#ADDR  ;GET THE STATUS
                BCS     NOCHAN          ;CHANNEL 0 NOT OPEN
                MOV     #ADDR+10,R5     ;POINT TO UNIT #
                MOV     (R5)+,R0        ;UNIT # TO R0
                ADD     (PC)+,R0        ;MAKE IT RAD50
                .RAD50  /  0/
                ADD     (R5),R0         ;GET DEVICE NAME
                MOV     R0,DEVNAM       ;DEVNAM HAS RAD50 DEVICE NAME
                .EXIT
        AREA:   .BLKW   5               ;EMT ARG LIST
        ADDR:   .BLKW   6               ;AREA FOR CHANNEL STATUS
        DEVNAM: .WORD   0               ;STORAGE FOR DEVICE NAME
        DEFEXT: .WORD   0,0,0,0
        NOCHAN: .PRINT  #MSG
                .EXIT
        MSG:    .ASCIZ  /NO OUTPUT FILE/
                .EVEN
        DEVSDC=.
                .END    ST
```

.DELETE

9.4.10 .DELETE

The .DELETE request deletes a named file from an indicated device.

Macro Call: .DELETE .area, .chan, .dblk, .count where: .count — is used by magtape/cassette only, (Refer to Appendix H for more information concerning the magtape and cassette handlers.)

Request Format:

R0 → .area:

0	.chan
.dblk	
.count	

Note:

The channel specified in the .DELETE request must not be in use when the request is made, or an error will occur. The file is deleted from the device, and an empty (UNUSED) entry of the same size is put in its place. A .DELETE issued to a nonfile-structured device is ignored. .DELETE requires that the handler to be used be in memory at the time the request is made. When the .DELETE is complete, the specified channel is left inactive.

Errors:

Code	Explanation
0	Channel is active
1	File was not found in the device directory

Example:

This example uses the special mode of CSI to delete files.

```
        .MCALL  ..V2..,.REGDEF
        ..V2..
        .REGDEF
        .MCALL  .SRESET,.CSISPC,.DELETE,.PRINT,.EXIT

START:  .SRESET                     ;MAKE SURE CHANNELS
                                    ;ARE FREE
        .CSISPC #OUTSPC,#DEFEXT     ;GET COMMAND LINE
                                    ;TERMINAL DIALOG WAS
                                    ;DT:FILE
        .DELETE #LIST,#0,#INSPC     ;USE CHANNEL 0 TO
                                    ;DELETE THE FILE
                                    ;WHICH IS AT THE
                                    ;FIRST INPUT SLOT.
        BCC     START               ;OK, LOOP AGAIN
        .PRINT  #NOFILE             ;NO SUCH FILE
        BR      START
NOFILE: .ASCIZ  /FILE NOT FOUND/
        .EVEN
DEFEXT: .RAD50  /MAC/               ;.MAC INPUT EXTENSION
        .WORD   0,0,0               ;NO OUTPUT DEFAULTS
LIST:   .BLKW   2                   ;EMT ARG LIST
OUTSPC=.
INSPC=.+36
        .BLKW   39.
        .END    START
```

INSPC is the address of the first input slot in the CSI input table.

.DEVICE

9.4.11 .DEVICE (F/B Only)

This request allows the user to set up a list of addresses to be loaded with specified values when a program is terminated. Upon an .EXIT or CTRL C, this list is picked up by the system and the appropriate addresses are set up with the corresponding values. This function is primarily designed to allow user programs to load device registers with necessary values. In particular, it is used to turn off a device's interrupt enable bit when the program servicing the device terminates.

Macro Call: .DEVICE .area, .addr

where: .addr is the address of the list of masks and words.

Request Format:

R0 ⇒ .area:

14	0
.addr	

The list is composed of address/value pairs and should be terminated by a 0 address. Only one list can be active at a given time. If multiple .DEVICE requests are given, the last list specified is the one used.

Note:

When the job is terminated for any reason, the list is scanned once. At that point, the monitor disables the feature until another .DEVICE call is executed. Thus, background programs which are re-enterable should include .DEVICE as a part of the reenter code.

Errors:

None.

Example:

The following example shows how .DEVICE is used to disable interrupts from the AFC11 (A-D converter sub-system).

```
        .MCALL  ..V2..,.REGDEF
        ..V2..
        .REGDEF
        .MCALL  .DEVICE,.EXIT

START:  .DEVICE #LIST
        .EXIT
LIST:   .BYTE   0,14            ;EMT ARG LIST
```

```
        .WORD   ATOD
ATOD:   172570                  ;ADDRESS IS 172570
        0                       ;JAM A 0 INTO IT
        0                       ;THIS 0 TERMINATES THE LIST.
        .END    START
```

.DSTATUS

9.4.12 .DSTATUS

This request is used to obtain information about a particular device.

Macro Call: .DSTATUS .cblk, .devnam

where: .cblk is the 4-word space used to store the status information.

.devnam is the pointer to the RAD50 device name.

.DSTATUS looks for the device specified by .devnam and, if found, returns four words of status starting at the address specified by .cblk. The four words returned are:

1. Status Word

Bits 7-0: contain a number which identifies the device in question. The values (octal) currently defined are:

0 = RK05 Disk
1 = TC11 DECtape
2 = Reserved
3 = Line Printer
4 = Console Terminal
5,6 = Reserved
7 = PC11 High-speed Reader
10 = PC11 High-speed Punch
11 = Magtape (TM11, TMA11)
12 = RF11 Disk
13 = TA11 Cassette
14 = Card Reader (CR11, CM11)
15 = Reserved
16 = RJS03/4 Fixed-head Disks
17 = Reserved
20 = TJU16 Magtape
21 = RP02, RP03 Disk
22 = RX01 Disk

Bit 15: 1= Random-access device (disk, DECtape)
0= Sequential-access device (line printer, paper tape, card reader, magtape, cassette, terminal)
Bit 14: 1= Read-only device (card reader, paper tape reader)
Bit 13: 1= Write-only device (line printer, paper tape punch)
Bit 12: 1= Non RT-11 directory-structured device (magtape, cassette)

Bit 11: 1= Enter handler abort entry every time a job is aborted
0= Handler abort entry taken only if there is an active queue element belonging to aborted job
Bit 10: 1= Handler accepts .SPFUN requests (e.g., MT, CT, DX)
0= .SPFUN requests are rejected as illegal

2. Handler size.

   The size of the device handler, in bytes.

3. Entry point.

   Non-zero implies the handler is now in memory; zero implies it must be .FETCHed before it can be used.

4. Device size.

   The size of the device (in 256-word blocks) for block-replaceable devices; zero for sequential-access devices.

The device name may be a user-assigned name.

Refer to the RT-11 Software Support Manual for greater detail.
Errors:

Code	Explanation
0	Device not found in tables.

Example:

This example shows how to determine if a particular device handler is in memory and, if it is not, how to .FETCH it there.

```
        .MCALL  ..V2..,.REGDEF
        ..V2..
        .REGDEF
        .MCALL  .DSTATUS,.PRINT,.EXIT,.FETCH

START:  .DSTATUS #CORE,#FPTR    ;GET STATUS OF DEVICE
        BCC     1$
        .PRINT  #ILLDEV         ;DEVICE NOT IN TABLES
        .EXIT
1$:     TST     CORE+4          ;IS DEVICE RESIDENT?
        BNE     2$
        .FETCH  #HNDLR,#FPTR    ;NO, GET IT
        BCC     2$
        .PRINT  #FEFAIL         ;FETCH FAILED
        .EXIT
2$:     .PRINT  #FECHOK
        .EXIT
CORE:   .BLKW   4       ;DSATUS GOES HERE
FPTR:   .RAD50  /DT0/   ;DEVICE NAME
        .RAD50  /FILE  MAC/ ;FILE NAME
FEFAIL: .ASCIZ  /FETCH FAILED/
ILLDEV: .ASCIZ  /ILLEGAL DEVICE/
        .EVEN
FECHOK: .ASCIZ  /FETCH O.K./
        .EVEN
HNDLR=.                 ;HANDLER WILL GO HERE
        .END    START
```

.ENTER

9.4.13 .ENTER

The .ENTER request allocates space on the specified device and creates a tentative entry for the named file. The channel number specified is associated with the file. (Note that if the program is overlaid, channel 15 is used by the overlay handler and should not be modified.)

Macro Call: .ENTER .area, .chan, .dblk, .length, .count

where: .length is the file size specification. The file length allocation is as follows:

- 0 - either 1/2 the largest empty entry or the entire second largest empty entry, whichever is largest. (A maximum size for non-specific .ENTERs may be patched in the monitor.)
- M - a file of M blocks. M may exceed the maximum mentioned above.
- -1 - the largest empty entry on the device.

.count file number for magtape/cassette (see Appendix H); if this argument is blank, a value of zero is assumed.

Request Format:

```
R0 → .area: |2|.chan  |
            |.dblk    |
            |.length  |
            |.count   |
```

The file created with an .ENTER is not a permanent file until the .CLOSE on that channel is given. Thus, the newly created file is not available to .LOOKUP and the channel may not be used by .SAVESTATUS requests. However, it is possible to go back and read data which has just been written into the file by referencing the appropriate block number. When the .CLOSE to the channel is given, any already existing permanent file of the same name on the same device is deleted and the new file becomes permanent. Although space is allocated to a file during the .ENTER operation, the actual length of the file is determined when .CLOSE is requested.

Each job may have up to 256 files open on the system at any time. If required, all 256 may be opened for output with the .ENTER function. .ENTER requires that the device handler be in memory when the request is made. Thus, a .FETCH should normally be executed before a .ENTER can be done. On return, R0 contains the size of the area actually allocated for use.

Notes:

When using the 0 length feature of .ENTER, it must be kept in mind that less than the largest empty space is allocated. This can have an important effect in transferring files between devices (particularly DECtape) which have a relatively small capacity. For example, to transfer a 200-block file to a DECtape on which the largest available empty space is 300 blocks, a 0 length transfer will not work. Since the .ENTER allocates half the largest space, only 150 blocks are really allocated and an output error will occur during the transfer. If a specific length of 200 is requested, however, the transfer will proceed without error.

Errors:

Code	Explanation
0	Channel is in use.
1	In a fixed length request, no space greater than or equal to M was found, or in a non-specific request the device or the directory was found to be full.

Example:

.ENTER may be used to open a file on a specified device, and then write data from memory into that file as follows:

```
        .MCALL  ..V2..,.REGDEF,.ENTER,.WRITW,.CLOSE,.PRINT
        .MCALL  .SRESET,.EXIT,.FETCH
        ..V2..
        .REGDEF

START:  .SRESET                         ;MAKE SURE ALL CHANNELS
                                        ;ARE CLOSED.
        .FETCH  #CORSPC,#FPRT           ;FETCH DEVICE HANDLER
        BCS     BADFET                  ;.FETCH ERROR, PROBABLY
                                        ;ILLEGAL DEVICE.
        .ENTER  #AREA,#0,#FPRT          ;OPEN A FILE ON THE DEVICE
                                        ;SPECIFIED. LENGTH 0 WILL
                                        ;GIVE 1/2 OF LARGEST EMPTY
                                        ;SPACE NOW AVAILABLE.
        BCS     BADENT                  ;FAILED. CHANNEL PROBABLY BUSY
        .WRITW  #AREA,#0,#BUFF,#END-BUFF/2,#0
                                        ;WRITE DATA FROM MEMORY. THE
                                        ;SIZE IS # OF WORDS BETWEEN
                                        ;BUFF AND END. START AT BLOCK 0.
        BCS     BADWRT                  ;WRITE FAILURE.
        .CLOSE  #0                      ;CLOSE THE FILE
        .EXIT                           ;AND GO TO KEYBOARD MONITOR.
FPRT:   .RAD50  /DK /                   ;FILE WILL BE ON DK
        .RAD50  /FILE  EXT/             ;NAMED FILE.EXT
AREA:   .BLKW   10                      ;EMT ARGUMENT LIST
BADFET: .PRINT  #FMSG
```

```
        .EXIT
BADENT: .PRINT  #EMSG
        .EXIT
BADWRT: .PRINT  #WMSG
        .EXIT
FMSG:   .ASCIZ  /BAD FETCH/
EMSG:   .ASCIZ  /BAD ENTER/
WMSG:   .ASCIZ  /WRITE ERROR/
        .EVEN
CORSPC: .BLKW   400             ;LEAVE 400(8) WORDS
                                ;FOR DEVICE HANDLER.
BUFF:
        .REPT   400             ;THIS IS BUFFER TO BE WRITTEN OUT
        .WORD   0,1
        .ENDR
END:
        .END    START
```

.EXIT

9.4.14 .EXIT

The .EXIT request causes the user program to terminate. When used from a background job under the F/B Monitor and when used under the Single-Job Monitor, .EXIT causes KMON to run in the background area. All outstanding mark time requests are cancelled. Any I/O requests and completion routines pending for that job are allowed to complete. If part of the background job resides where KMON and USR are to be read, the user job is written onto system device scratch blocks. KMON and USR are then loaded and control goes to KMON in the background area. If R0=0 when the .EXIT is done, an implicit INIT command is executed when KMON is entered, disabling the subsequent use of REENTER, START, or CLOSE.

.EXIT also resets any .CDFN and .QSET calls that were done and executes an .UNLOCK if a .LOCK has been done. Thus, the .CLOSE command from the Keyboard Monitor does not operate for programs which perform .CDFN requests.

In a F/B system, an .EXIT from a completion routine acts as if a double CTRL C has been typed, aborting all I/O in progress before exiting. In general, .EXIT from a completion routine should be avoided.

Macro Call: .EXIT

Errors:

None.

.FETCH

9.4.15 .FETCH

The .FETCH request loads device handlers into memory from the system device.

Macro Call: .FETCH .coradd, .devnam

where:	.coradd	is the address where the device handler is to be loaded.
	.devnam	is the pointer to the RAD50 device name.

The storage address for the device handler is passed on the stack. When the .FETCH is complete, R0 points to the first available location above the handler. If the handler is already in memory, R0 keeps the same value as was initially pushed onto the stack. If the argument on the stack is less than 400(8), it is assumed that a handler .RELEAS is being done. (.RELEAS does not dismiss a handler which was LOADED from the KMON; an UNLOAD must be done.) After a .RELEAS, a .FETCH must be issued in order to use the device again.

Several requests require a device handler to be in memory for successful operation. These include:

.CLOSE	.READC	.READ
.LOOKUP	.WRITC	.WRITE
.ENTER	.READW	.SPFUN
.RENAME	.WRITW	.DELETE

Since foreground jobs must have handlers resident, a .FETCH from the foreground will give a fatal error if the handler has not been previously LOADed.

Errors:

Code	Explanation
0	The device name specified does not exist, or there is no handler for that device in the system.

Example:

In the following example, the PR and PP handlers are fetched into memory in preparation for their use by a program. The program sets aside handler space from its free memory area.

```
        .MCALL  ..V2..,.REGDEF,.FETCH,.PRINT,.EXIT
        ..V2..
        .REGDEF

START:
        .FETCH  FREE,#PRNAME    ;FETCH PR HANDLER
        BCS     FERR            ;FETCH ERROR
        MOV     R0,R2
        .FETCH  R2,#PPNAME      ;FETCH PP HANDLER
                                ;IMMEDIATELY FOLLOWING
                                ;PR HANDLER. R0 POINTS
                                ;TO THE TOP OF PR
                                ;HANDLER ON RETURN
                                ;FROM THAT CALL.
        BCS     FERR            ;NO PP HANDLER
        MOV     R0,FREE         ;UPDATE FREE MEMORY
                                ;POINTER TO POINT TO
                                ;NEW BOTTOM OF FREE
                                ;AREA(TOP OF HANDLERS).

        .PRINT  #OK
        .EXIT
OK:     .ASCIZ  /FETCH O.K./
        .EVEN
FERR:   .PRINT  #MSG            ;PRINT ERROR MESSAGE
        .EXIT                   ;AND EXIT
        HALT
PRNAME: .RAD50  "PR "           ;DEVICE NAMES
PPNAME: .RAD50  "PP "
MSG:    .ASCIZ  "DEVICE NOT FOUND" ;ERROR MESSAGE
        .EVEN
FREE:   .+2                     ;POINTER TO FREE MEMORY
        .END    START
```

.GTIM

9.4.16 .GTIM

.GTIM allows user programs to access the current time of day. The time is returned in two words, and is given in terms of clock ticks past midnight.

Macro Call: .GTIM .area, .addr

where: .addr is a pointer to the two words of time to be returned.

Request Format:

R0 ⇒ .area:

21	0
.addr	

The high-order time is returned in the first word, the low-order time in the second word. User programs must make the conversion from clock ticks to hours-minutes-seconds. The basic clock frequency (50 or 60 Hz) may be determined from the configuration word in the monitor (see Section 9.2.6). Under a F/B Monitor, the time of day is automatically reset after 24:00 when a .GTIM is done; under the Single-Job Monitor, it is not. The month is not automatically 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.

Errors:

None.

Example:

```
                .MCALL  ..V2..,.REGDEF,.GTIM,.EXIT
                ..V2..
                .REGDEF

        START:
                .GTIM   #LIST,#TIME

                .EXIT
        TIME:   .WORD   0,0               ;LOW AND HI ORDER TIME
                                          ;RETURNED HERE.

        LIST:   .BLKW   2                 ;ARGUMENTS FOR THE EMT

                .END    START
```

.GTJB

9.4.17 .GTJB

The .GTJB request passes certain job parameters back to the user program.

Macro Call: .GTJB .area, .addr

where: .addr is the address of an eight-word block into which the parameters are passed. The values returned are:

Word 1	-	Job Number. 0=Background, 2=Foreground
2	-	High memory limit
3	-	Low memory limit
4	-	Beginning of I/O channel space
5-8	-	Reserved for future use

Request Format:

R0 → .area:	20	0
	.addr	

In the Single-Job Monitor, the job number is always 0 and the low limit 0.

In the F/B Monitor, the job number can either be 0 or 2. If the job number equals 0 (background job), word 2 equals 0 and word 4 describes where the I/O channel words begin. This is normally an address within the Resident Monitor. When a .CDFN is executed, however, the start of the I/O channel area changes to the user specified area.

Errors:

None.

Example:

Use .GTJB to determine if this program is executing as a foreground or background job.

```
        .MCALL  ..V2..,.REGDEF,.GTJB,.PRINT,.EXIT
        ..V2..
        .REGDEF

START:
        .GTJB   #LIST,#JOBARG       ;R0 POINTS TO 1ST WORD ON
                                    ;RETURN FROM CALL.
        MOV     #FMSG,R1
        TST     JOBARG              ;BACKGROUND?
        BNE     1$                  ;NO, PRINT FMSG
        MOV     #BMSG,R1
1$:     .PRINT  R1

        .EXIT

FMSG:   .ASCIZ  /PROGRAM IN FOREGROUND/
BMSG:   .ASCIZ  /PROGRAM IN BACKGROUND/
        .EVEN

LIST:   .BLKW   2                   ;ARGUMENTS FOR THE EMT
JOBARG: .BLKW   8.                  ;JOB PARAMETERS PASSED BACK HERE.

        .END    START
```

.HERR/.SERR

9.4.18 .HERR/.SERR

.HERR and .SERR are complementary requests used to govern monitor behavior for serious error conditions. During program execution, certain error conditions may arise which cause the executing program to be aborted (for example, trying to pass I/O to a device which has no handler in memory, or trying to load a device handler over the USR). Normally, these errors cause program termination with one of the

?M- error messages. However, in certain cases it is not feasible to abort the program because of these errors; for example, a multi-user program must be able to retain control and merely abort the user who has generated the error. .SERR accomplishes this by inhibiting the monitor from aborting the job. Instead, it causes an error return to the offending EMT to be taken. On return from that request, the C bit is set and byte 52 contains a negative value indicating the error condition which occurred.

.HERR turns off user error interception and allows the system to abort the job on fatal errors and generate an error message. (.HERR is the default case.)

Macro Calls: .HERR

.SERR

Errors:

Following is a list of the errors which are returned if soft error recovery is in effect:

Code	Explanation
-1	Called USR from completion routine.
-2	No device handler; this operation needs one.
-3	Error doing directory I/O.
-4	FETCH error. Either an I/O error occurred while reading the handler, or tried to load it over USR or RMON.
-5	Error reading an overlay.
-6	No more room for files in the directory.
-7	Illegal address (F/B only); tried to perform a monitor operation outside the job partition.
-10	Illegal channel number; number is greater than actual number of channels which exist.
-11	Illegal EMT; an illegal function code has been decoded.

Traps to 4 and 10, and floating point exception traps are not inhibited. These errors have their own recovery mechanism. (See Section 9.4.42.)

Example:

This example causes a normally fatal error to generate errors back to the user program. The error returned is used to print an appropriate message.

```
        .MCALL  ..V2..,.REGDEF,.FETCH,.ENTER,.HERR,.SERR
        .MCALL  .EXIT,.PRINT
        ..V2..
        .REGDEF

ST:     .SERR                   ;TURN ON SOFTWARE ERROR
                                ;RETURNS
        .FETCH  #HDLR,#PTR      ;GET A DEVICE HANDLER
        BCS     FCHERR
        .ENTER  #AREA,#1,#PTR   ;OPEN A FILE ON CHANNEL 1
        BCS     ENERR
```

```
        .HERR                           ;NOW PERMIT ?M-ERRORS.
        .EXIT

FCHERR: MOVB    @#52,R0                 ;WAS IT FATAL
        BMI     FTLERR                  ;YES
        .PRINT  #FMSG                   ;NO... NO DEVICE BY THAT NAME
        .EXIT
ENERR:  MOVB    @#52,R0
        BMI     FTLERR
        .PRINT  #EMSG
        .EXIT
FTLERR: NEG     R0                      ;THIS WILL TURN POSITIVE
        DEC     R0                      ;ADJUST BY ONE
        ASL     R0                      ;MAKE IT AN INDEX
        MOV     TBL(R0),R0              ;PUT MESSAGE ADDRESS INTO R0
        .PRINT                          ;AND PRINT IT.
        .EXIT

TBL:    M1                              ;CAN'T OCCUR IN THIS PROGRAM
        M2                              ;NO DEVICE HANDLER IN MEMORY
        M3                              ;DIRECTORY I/O ERROR
        M4                              ;FETCH ERROR
        M5                              ;IMPOSSIBLE FOR THIS PROGRAM
        M6                              ;NO ROOM IN DIRECTORY
        M7                              ;ILLEGAL ADDRESS (F/B)
        M10                             ;ILLEGAL CHANNEL
        M11                             ;ILLEGAL EMT

M1:                                     ;CAN'T OCCUR IN THIS PROGRAM
M2:     .ASCIZ  /NO DEVICE HANDLER/
M3:     .ASCIZ  "DIRECTORY I/O ERROR"
M4:     .ASCIZ  /ERROR DOING FETCH/
M5:                                     ;NOT APPLICABLE TO THIS PROGRAM
M6:     .ASCIZ  /NO ROOM IN DIRECTORY/
M7:     .ASCIZ  /ADDRESS CHECK ERROR/
M10:    .ASCIZ  /ILLEGAL CHANNEL/
M11:    .ASCIZ  /ILLEGAL EMT/
FMSG:   .ASCIZ  /FETCH FAILED/
EMSG:   .ASCIZ  /ENTER FAILED/
        .EVEN
HDLR:   .BLKW   300                     ;LEAVE 300 (OCTAL) FOR HANDLER
PTR:    .RAD50  /DT4/                   ;DEVICE AND FILE NAME.
        .RAD50  /EXAMPL/
        .RAD50  /MAC/
AREA:   .BLKW   4                       ;EMT AREA
        .END    ST
```

.HRESET

9.4.19 .HRESET

This request performs the same function as .SRESET, after stopping all I/O transfers in progress for that job. (.HRESET is not used to clear a hard-error condition.) Note that in the single-job environment, a hardware RESET instruction is used to terminate I/O, while in a F/B environment, only the I/O associated with the job which issued the .HRESET is affected. All other transfers continue.

Macro call: .HRESET

Errors:

None.

Example:

See the example for .SRESET (Section 9.4.40) for format.

.LOCK/.UNLOCK

9.4.20 .LOCK/.UNLOCK

.LOCK

The .LOCK request is used to "lock" the USR in memory for a series of operations. If all the conditions which cause swapping are satisfied, the user program is written into scratch blocks and the USR is loaded. Otherwise, the USR which is in memory is used, and no swapping occurs. The USR is not released until an .UNLOCK request is given. (Note that in a F/B System, calling the CSI may also perform an implicit .UNLOCK.) A program which has many USR requests to make can .LOCK the USR in memory, make all the requests, and then .UNLOCK the USR; no time is spent doing unnecessary swapping.

In a F/B environment, a .LOCK inhibits the other job from using the USR. Thus, the USR should be locked only as long as necessary.

Macro Call: .LOCK

Note that the .LOCK request reduces time spent in file handling by eliminating the swapping of the USR in and out of memory. If the USR is currently resident, .LOCK is ignored. After a .LOCK has been executed, an .UNLOCK request must be executed to release the USR from memory. The .LOCK/.UNLOCK requests are complimentary and must be matched. That is, if three .LOCK requests are issued, at least three .UNLOCKs must be done, otherwise the USR will not be released. More .UNLOCKs than .LOCKs may occur without error.

Notes:

1. It is vital that the .LOCK call not come from within the area into which the USR will be swapped. If this should occur, the return from the USR request would not be to the user program, but to the USR itself, since the LOCK function inhibits the user program from being re-read.

2. Once a .LOCK has been performed, it is not advisable for the program to destroy the area the USR is in, even though no further use of the USR is required. This causes unpredictable results when an .UNLOCK is done.

3. If a foreground job performs a .LOCK request while the background job owns the USR, foreground execution is suspended until the USR is available. Thus, in this case, it is possible for the background to lock out the foreground (see the .TWAIT request).

Errors:

None.

Example:

See the example following .UNLOCK.

.UNLOCK

The .UNLOCK request releases the User Service Routine from memory if it was placed there with a .LOCK request. If the .LOCK required a swap, the .UNLOCK loads the user program back into memory. If the USR does not require swapping, the .UNLOCK acts as a no-op.

Macro Call: .UNLOCK

Notes:

1. It is important that at least as many .UNLOCKS are given as .LOCKs. If more .LOCK requests were done, the USR remains locked in memory. It is not harmful to give more UNLOCKs than are required; those that are extra are ignored.

2. The .LOCK/.UNLOCK pairs should be used only when absolutely necessary when running two jobs in the F/B system. When a job .LOCKs the USR, the other job cannot get at it until it is .UNLOCKed. Thus, the USR should not be .LOCKed unnecessarily, as this may degrade performance in some cases.

3. In a F/B System, calling the CSI with input coming from the console terminal performs an implicit .UNLOCK.

Errors:

None.

Example:

This example shows the usage of .LOCK, .UNLOCK, and their interaction with the system.

```
        .MCALL  ..V2..,.REGDEF,.LOCK,.UNLOCK,.LOOKUP
        .MCALL  .SETTOP,.PRINT,.EXIT
        ..V2..
        .REGDEF

START:

SYSPTR=54

        .SETTOP @#SYSPTR        ;TRY FOR ALL OF MEMORY
        MOV     R0,TOP          ;R0 HAS THE TOP
```

```
        .LOCK                           ;BRING USR INTO MEMORY
        .LOOKUP #LIST,#0,#FILE1         ;LOOKUP A FILE ON CHANNEL 0
        BCC     1$                      ;ON ERROR, PRINT A
2$:     .PRINT  #LMSG                   ;MESSAGE AND EXIT
        .EXIT
1$:     MOV     #LIST,R0
        INC     (R0)                    ;DO LOOKUP ON CHANNEL 1
        MOV     #FILE2,2(R0)            ;NEW POINTER
        .LOOKUP         ;ALL ARGS ARE FILLED IN
        BCS     2$
        .UNLOCK                         ;NOW RELEASE USR
        .EXIT


LIST:   .BLKW   3                       ;SPACE FOR ARGUMENTS
FILE1:  .RAD50  /DK /
        .RAD50  /FILE1 MAC/
FILE2:  .RAD50  /DK /
        .RAD50  /FILE2 MAC/
TOP:    .WORD   0
LMSG:   .ASCIZ  /LOOKUP ERROR/
        .EVEN

        .END    START
```

In the above example, .SETTOP tries to obtain as much memory as it can. Most likely this will, in a background job, make the USR non-resident (i.e., unless a SET USR NOSWAP command is done at the keyboard). Thus, if the USR were non-resident, swapping must take place for each .LOOKUP given. Using the .LOCK, the USR is brought into memory and remains there until the .UNLOCK is given.

The second .LOOKUP makes use of the fact that the arguments have already been set up at LIST. Thus, it is possible to increment the channel number, put in a new file pointer and then give a simple .LOOKUP, which does not cause any arguments to be moved into LIST.

.LOOKUP

9.4.21 .LOOKUP

The .LOOKUP request associates a specified channel with a device and/or file, for the purpose of performing I/O operations. The channel used is then "busy" until one of the following requests is executed:

.CLOSE
.SAVESTATUS
.SRESET
.HRESET
.PURGE
.CSIGEN (if channel is in range 0-10 octal)

Note that if the program is overlaid, channel 15 is used by the overlay handler and should not be modified.

Macro Call: .LOOKUP .area, .chan, .dblk, .count

where: .count — is an argument which can optionally be used for the cassette and magtape handlers. Refer to Appendix H for details of this parameter. If .count is blank, a value of zero is assumed.

Request Format:

R0 ⇒ .area:

1	.chan
.dblk	
.count	

If the first word of the file name in .dblk is zero and the device is a file-structured device, absolute block 0 of the device is designated as the beginning of the "file". This technique allows I/O to any physical block on the device. If a file name is specified for a device which is not file-structured (i.e. PR:FILE.EXT), the name is ignored.

The handler for the selected device must be in memory for a .LOOKUP. On return from the .LOOKUP, R0 contains the length (number of blocks) of the file just looked up. If the length returned is 0, a nonfile-structured .LOOKUP was done to the device.

Errors:

Code	Explanation
0	Channel already open.
1	File indicated was not found on the device.

Example:

In the following example, the file "DATA.001" on device DT3 is opened for input on channel 7.

```
        .MCALL  ..V2..,.REGDEF,.FETCH,.LOOKUP,.PRINT,.EXIT
        ..V2..
        .REGDEF

START:
ERRWD=52
        .FETCH  #HSPACE,#DT3N   ;GET DEVICE HANDLER
        BCS     FERR            ;DT3 IS NOT AVAILABLE
        .LOOKUP #LIST,#7,#DT3N  ;LOOKUP THE FILE
                                ;ON CHANNEL 7
        BCC     LDONE           ;FILE WAS FOUND
        TSTB    @#ERRWD         ;ERROR, WHAT'S WRONG?
        BNE     NFD             ;FILE NOT FOUND
        .PRINT  #CAMSG          ;PRINT 'CHANNEL ACTIVE'
        .EXIT
NFD:    .PRINT  #NFMSG          ;FILE NOT FOUND
        .EXIT
CAMSG:  .ASCIZ  /CHANNEL ACTIVE/
```

```
NFMSG:  .ASCIZ  /FILE NOT FOUND/ ;ERROR MESSAGES
DTMSG:  .ASCIZ  /DT3 NOT AVAILABLE/
        .EVEN
FERR:   .PRINT  #DTMSG
        .EXIT
LDONE:                          ;PROGRAM CAN NOW
                                ;ISSUE READS  AND
                                ;WRITES TO FILE
                                ;DATA.001 VIA
                                ;CHANNEL 7

        .EXIT

LIST:   .BLKW   5
DT3N:   .RAD50  "DT3"           ;DEVICE
        .RAD50  "DAT"           ;FILENAME
        .RAD50  "A  "           ;FILENAME
        .RAD50  "001"           ;EXTENSION

HSPACE:                 ;RESERVED SPACE FOR DT
        .=.+400                 ;HANDLER

        .END    START
```

.MRKT

9.4.22 .MRKT

The .MRKT request schedules a completion routine to be entered after a specified time interval (clock ticks past midnight) has elapsed.

Macro Call: .MRKT .area, .time, .crtn, .id

where:

.time — is the pointer to the two words containing the time interval (high-order first; low-order second).

.id — is a number assigned by the user to identify the particular request to the completion routine and to any cancel mark time requests. The number must not be within the range of codes from 177400-177777; these are reserved for system use. The number need not be unique (i.e., several .MRKT requests may specify the same .id.) On entry to the completion routine, the .id number is in R0.

Request Format:

R0 ⇒ .area:

22	0
.time	
.crtn	
.id	

.MRKT requests require a queue element taken from the same list as the I/O queue elements. The element is in use until either the completion routine is entered or a cancel mark time request is issued. The user should allocate enough queue elements to handle at least as many mark time requests as he expects to have pending simultaneously.

Errors:

Code	Explanation
0	No queue element was available.

Example:

In this example, a mark time is set up to time out an I/O transfer. If the mark time expires before the transfer is done, a message is printed. If the I/O transfer completes before the mark time, the mark time is cancelled. (Note that the example assumes the I/O channel is already open.)

```
                .MCALL  ..V2..,.REGDEF,.READ,.WAIT,.MRKT,.CMKT
                .MCALL  .QSET,.PRINT,.EXIT,.LOOKUP
                ..V2..
                .REGDEF

        ST:     .LOOKUP #AREA,#0,#FILE    ;OPEN A FILE
                BCS     LKERR             ;FILE NOT FOUND
                MOV     #AREA,-(SP)       ;EMT LIST TO STACK
                .QSET   #QUEUE,#5         ;ALLOCATE 5 MORE ELEMENTS
                .MRKT   (SP),#INTRVL,#MRTN,#1 ;SET TIMER GOING
                BCS     NOMRKT            ;FAILED.
                .READ   #RDLST            ;START  I/O TRANSFER
                BCS     RDERR
                .WAIT   #0                ;AND WAIT A WHILE.
                .CMKT   (SP),#1           ;SEE IF MARK TIME IS
                                          ;DONE.
                BCS     NOTDUN            ;FAILED. THAT MEANS THAT
                                          ;THE MARK TIME ALREADY
                                          ;EXPIRED.

                .EXIT

        MRTN:   .CMKT   (SP),#1           ;OK, KILL THE TIMER.
                .PRINT  #FAIL             ;DON'T WORRY ABOUT AN
                                          ;ERROR HERE.
                RTS     PC
        LKERR:  .PRINT  #LM
                .EXIT
        RDERR:  .PRINT  #RDMSG
                .EXIT
        NOTDUN: .PRINT  #FAIL
                .EXIT
        NOMRKT: .PRINT  #NOQ
                .EXIT
        NOQ:    .ASCIZ  /NO QUEUE ELEMENTS AVAILABLE/
        FAIL:   .ASCIZ  /MARK TIME COMPLETED BEFORE TRANSFER/
        LM:     .ASCIZ  /LOOKUP ERROR/
        RDMSG:  .ASCIZ  /READ ERROR/
                .EVEN
        INTRVL: .WORD   0,13.             ;ALLOW 13 CLOCK
                                          ;TICKS FOR TRANSFER.
```

```
QUEUE:  .BLKW   5*7             ;AREA FOR QUEUE ELEMENTS
AREA:   .BLKW   5               ;A FEW WORDS FOR EMT LIST
FILE:   .RAD50  /DK FILE  TST/
RDLST:  .BYTE   0               ;CHANNEL 0
        .BYTE   10              ;A READ
BLOCK:  .WORD   0               ;BLOCK #
        .WORD   BUFF            ;BUFFER
        .WORD   256.            ;1 BLOCK
        .WORD   1
BUFF:   .BLKW   256.

        .END    ST
```

.MWAIT

9.4.23 .MWAIT

This request is similar to the .WAIT request. .MWAIT, however, suspends execution until all messages sent by the other job have been transmitted or received. It provides a means for ensuring that a required message has been processed. It should be used primarily in conjunction with the .RCVD or .SDAT modes of message handling, where no action is taken when a message is completed.

Macro Call: .MWAIT

Errors:

None.

Example:

This program requests a message, does some intermediate processing, and then waits until the message is actually sent.

```
        .MCALL  ..V2..,.REGDEF,.MWAIT,.RCVD,.EXIT,.PRINT
        ..V2..
        .REGDEF

WORDS=255.
START:
        .RCVD   #AREA,#RBUFF,#WORDS ;GET MESSAGE.

                                ;INTERMEDIATE PROCESS

        MOV     #RBUFF+2,R5
        .MWAIT                  ;MAKE SURE WE HAVE IT.
        CMPB    (R5)+,#'A       ;FIRST CHARACTER AN A?
        BNE     BADMSG          ;NO, INVALID MESSAGE

        .EXIT
BADMSG: .PRINT  #MSG
        .EXIT
```

```
MSG:    .ASCIZ  /BAD MESSAGE/
AREA:   .BLKW   10
RBUFF:  .BLKW   256.
        .EVEN
        .END    START
```

.PRINT

9.4.24 .PRINT

The .PRINT request causes output to be printed at the console terminal. When a foreground job is running and a change occurs in the job producing output, a B> or F> appears. Any text following the message has been printed by the job indicated (foreground or background) until another B> or F> is printed. The string to be printed may be terminated with either a null (0) byte or a 200 byte. If the null (ASCIZ) format is used, the output is automatically followed by a <CR><LF>. If a 200 byte terminates the string, no <CR><LF> is generated.

Macro Call: .PRINT .addr

where: .addr is the address of the string to be printed.

Control returns to the user program after all characters have been placed in the output buffer.

The foreground job issues a message immediately using .PRINT no matter what the state of the background job. Thus, for urgent messages, .PRINT should be used (rather than .TTYIN or .TTYOUT).

Errors:

None.

Example:

```
        .MCALL  ..V2..,.REGDEF,.PRINT,.EXIT
        ..V2..
        .REGDEF

START:
        .PRINT  #S2
        .PRINT  #S1

        .EXIT

S1:     .ASCIZ  /THIS WILL HAVE CR-LF FOLLOWING/
S2:     .ASCII  /THIS WILL NOT HAVE CR-LF/
        .BYTE   200
        .EVEN

        .END    START
```

.PROTECT

9.4.25 .PROTECT

The .PROTECT request is used by a job to obtain exclusive control of a vector (two words) in the region 0-476. If it is successful, it indicates that the locations are not currently in use by another job or by the monitor, in which case the job may place an interrupt address and priority into the protected locations and begin using the associated device.

Macro Call: .PROTECT .area, .addr

where: .addr is the address of the word pair to be protected. .addr must be a multiple of four, and must be less than 476 (octal). The two words at .addr and .addr+2 will be protected.

Request Format:

R0 ⇒ .area:

31	0
.addr	

Errors:

Code	Explanation
0	Protect failure; locations already in use.
1	Address greater than 476 or not a multiple of 4.

Example:

This example shows the use of .PROTECT to gain control of the UDC11 vectors.

```
        .MCALL  ..V2..,.REGDEF,.PROTECT,.PRINT,.EXIT
        ..V2..
        .REGDEF
ST:     MOV     #AREA,-(SP)
        MOV     #234,R5             ;UDC VECTOR ADDRESS
        .PROTECT (SP),R5            ;PROTECT 234,236
        BCS     ERR                 ;YOU CAN'T
        MOV     #UDCINT,(R5)+       ;INITIALIZE THE VECTORS,
        MOV     #340,(R5)           ;AT LEVEL 7

        .EXIT
ERR:    .PRINT  #NOVEC
        .EXIT
AREA:   .BLKW   5
NOVEC:  .ASCIZ  /VECTORS ALREADY IN USE/
```

```
                .EVEN
        UDCINT:
                .
                .
                .
```

.PURGE

9.4.26 .PURGE

The .PURGE request is used to de-activate a channel without performing a .HRESET, .SRESET, .SAVESTATUS, or .CLOSE request. It merely frees a channel without taking any other action. If a tentative file has been .ENTERed on the channel, it will be discarded. Purging an inactive channel acts as a no-op.

Macro Call: .PURGE .chan

Errors:

None.

Example:

The following code is used to make certain that channels 0-7 are free:

```
        .MCALL  ..V2..,.REGDEF,.PURGE,.EXIT
        ..V2..
        .REGDEF

START:
        CLR     R1                      ;START WITH CHANNEL 0
1$:     .PURGE  R1                      ;PURGE A CHANNEL
        INC     R1                      ;BUMP TO NEXT CHANNEL
        CMP     R1,#8.                  ;IS IT AT CHANNEL 8 YET?
        BLO     1$                      ;NO, KEEP GOING

        .EXIT
        .END    START
```

.QSET

9.4.27 .QSET

All RT-11 I/O transfers are done through a centralized queue management system. If I/O traffic is very heavy and not enough queue elements are available, the program issuing the I/O requests may be suspended until a queue element becomes available. In a F/B system, the other job runs while the first program waits for the element.

The .QSET request is used to make the RT-11 I/O queue larger (i.e., add available entries to the queue). A general rule to follow is that each program should contain one more queue element than the total number of I/O requests which will be active simultaneously. Timing requests such as .TWAIT and .MRKT also cause elements to be used and must be considered when allocating queue elements for a program. Note that if synchronous I/O is done (i.e. .READW/.WRITW, etc.) and no timing requests are done, no additional queue elements need be allocated.

Macro Call: .QSET .addr, .qleng

where:

.addr — is the address at which the new elements are to start.

.qleng — is the number of entries to be added. Each queue entry is seven words long; hence the space set aside for the queue should be .qleng * 7 words.

Each time .QSET is called, a contiguous area of memory is divided into seven-word segments and is added to the queue for that job. .QSET may be called as many times as required. The queue set up by multiple .QSET requests is a linked list. Thus, .QSET need not be called with strictly contiguous arguments. The space used for the new elements is allocated from the user's program space. Thus, care must be taken so that the program in no way alters the elements once they are set up. The .SRESET and .HRESET requests discard all user-defined queue elements; therefore any .QSETs must be reissued.

Care should also be taken to allocate enough memory for the queue. The elements in the queue are altered by the monitor; if enough space is not allocated, destructive references will occur in an unexpected area of memory.

Errors:

None.

Example:

```
        .MCALL  ..V2..,.REGDEF,.QSET,.EXIT
        ..V2..
        .REGDEF

START:
        .QSET   #Q1,#5          ;ADD 5 ELEMENTS TO THE QUEUE
                                ;STARTING AT Q1
        .QSET   #Q3,#3          ;AND 3 MORE AT Q3.

        .EXIT

Q1:     .BLKW   7*5.            ;FIRST QUEUE AREA (35 DECIMAL WORDS)
Q3:     .BLKW   7*3.            ;SECOND QUEUE AREA (21 DECIMAL WORDS)

        .END    START
```

Note that Q1 and Q3 need not have been contiguous.

9.4.28 .RCTRLO

The .RCTRLO request ensures that the console terminal is able to print. Since CTRL O (↑O) struck while output is directed to the console terminal inhibits the output from printing until either another ↑O is struck or until the program resets the ↑O switch, a program that has a message which must appear at the console can override ↑O struck at the keyboard.

Macro Call: .RCTRLO

Errors:

None.

Example:

In this example, the user program first calls the CSI in general mode, then processes the command. When finished, it returns to the CSI for another command line. To make certain that the prompting "*" typed by the CSI is not inhibited by a CTRL O in effect from the last operation, terminal output is re-enabled via a .RCTRLO command prior to the CSI call.

```
        .MCALL  ..V2..,.REGDEF,.RCTRLO,.CSIGEN,.EXIT
        ..V2..
        .REGDEF

START:  .RCTRLO                     ;MAKE SURE TT OUTPUT IS
                                    ;ENABLED
        .CSIGEN #DSPACE,#DEXT,#0    ;CALL CSI-IT WILL TYPE
                                    ;"*"


                                    ;PROCESS COMMAND


        JMP     START               ;GET NEXT COMMAND

DEXT:   0                           ;NO DEFAULT EXTENSIONS
        0
        0
        0
DSPACE: .=.+400             ;HANDLER SPACE

        .END    START
```

.RCVD/.RCVDC/.RCVDW

9.4.29 .RCVD/.RCVDC/.RCVDW (F/B Only)

There are three forms of the receive data request; these are used in conjunction with the .SDAT (Send Data) requests to allow a general data/message transfer system. .RCVD requests can be thought of as .READ requests, where data transfer is not from a peripheral device but from the other job in the system. Additional queue elements should be allocated for buffered I/O operations in .RCVD and .RCVDC requests (see .QSET).

.RCVD

This request is used to receive data and continue execution. The request is posted and the issuing job continues execution. At some point when the job needs to have the transmitted message, an .MWAIT should be executed. This causes the job to be suspended until the message has been received.

Macro Call: .RCVD .area, .buff, .wcnt

where: .buff is the address of the buffer to which the message is to be sent.

.wcnt is the number of words to be transferred.

Request Format:

R0 → .area:

26	0
(unused)	
.buff	
.wcnt	
1	

Word 0 (the first word) of the message buffer will contain the number of words transmitted whenever the .RCVD is complete. Thus, the space allocated for the message should always be at least one word larger than the actual message size expected.

The word count is a variable number, and as such, the .SDAT/.RCVD combination can be used to transmit a few words or entire buffers. The .RCVD operation is only complete when a .SDAT is issued from the other job.

Programs using .RCVD/.SDAT must be carefully designed to either always transmit/receive data in a fixed format or have the capability of handling variable formats. The messages are all processed in FIFO (first in-first out) order. Thus, the receiver must be certain it is receiving the message it actually wants.

Errors:

Code	Explanation
0	No other job exists in the system.

Example:

An example follows the .RCVDW section.

.RCVDC

The .RCVDC request receives data and enters a completion routine when the message is received. The .RCVDC request is posted and program execution stays with the issuing job. When the other job sends a message, the completion routine specified will be entered.

Macro Call: .RCVDC .area, .buff, .wcnt, .crtn

where:

.buff	is the address of the buffer to which the message is to be sent.
.wcnt	is the number of words to be transmitted.
.crtn	is the completion routine to be entered (see Section 9.2.8).

As in the others, word 0 of the buffer contains the number of words transmitted when the transfer is complete.

Request Format:

R0 → area:

26	0
(unused)	
.buff	
.wcnt	
.crtn	

Errors:

Code	Explanation
0	No other job exists in the system.

Example:

An example follows the .RCVDW section.

.RCVDW

.RCVDW is used to receive data and wait. A message request is posted and the job issuing the request is suspended until the other job sends a message to the issuing job. When the issuing job runs again, the message has been received, and word 0 of the buffer indicates the number of words which were transmitted.

Macro Call: .RCVDW .area, .buff, .wcnt

where: .buff is the address of the buffer to which the message is to be sent.

.wcnt is the number of words to be transmitted.

Request Format:

R0 ⇒ .area:

26	0
(unused)	
.buff	
.wcnt	
0	

Errors:

Code	Explanation
0	No other job exists in the system.

Example:

In this example, the running job receives a message from the second job and interprets it as the device and filename of a file to be opened and used. In this case, the message was in RAD50 format, and the receiving program did not use the transmitted length for any purpose.

```
        .MCALL  ..V2..,.REGDEF,.RCVDW,.PURGE,.LOOKUP,.EXIT,.PRINT
        ..V2..
        .REGDEF

START:
        MOV     #AREA,R5        ;R5=EMT ARG. AREA
        .RCVDW  R5,#FILE,#4     ;REQUEST MESSAGE AND WAIT
        BCS     MERR            ;AN ERROR?
        .PURGE  #0              ;CLEAR CHANNEL 0
        .LOOKUP R5,#0,#FILE+2   ;LOOKUP INDICATED FILE
        BCS     LKERR           ;ERROR

        .EXIT

AREA:   .BLKW   10              ;LEAVE SPACE FOR SAFETY
FILE:   .BLKW   1               ;ACTUAL WORD COUNT IS HERE
        .BLKW   4               ;DEV:FILE.EXT ARE HERE

MERR:   .PRINT  #MMSG
        .EXIT
LKERR:  .PRINT  #LKMSG
        .EXIT
MMSG:   .ASCIZ  /MESSAGE ERROR/
LKMSG:  .ASCIZ  /LOOKUP ERROR/
        .EVEN
        .END    START
```

The issuing job is suspended until the indicated data is transmitted. Either of the other modes could have also been used to receive the message.

.READ/.READC/.READW

9.4.30 .READ/.READC/.READW

RT-11 provides three modes of I/O: .READ/.WRITE, .READC/.WRITC, and .READW/.WRITW. Section 9.4.47 explains the output operations. The input operations are described next.

Note that in the case of .READ and .READC, additional queue elements should be allocated for buffered I/O operations (see .QSET).

.READ

The .READ request transfers a specified number of words from the specified channel to memory. Control returns to the user program immediately after the .READ is initiated. No special action is taken when the transfer is completed.

Macro Call: .READ .area, .chan, .buff, .wcnt, .blk

where: .buff is the address of the buffer to receive the data read.

.wcnt is the number of words to be read.

.blk is the block number to be read relative to the start of the file, not block 0 of the device. The monitor translates the block supplied into an absolute device block number. The user program normally updates .blk before it is used again. If .blk=0, TT: gives ^ prompt and LP: gives form feed. (This is true for all .READ and .WRITE requests.)

Request Format:

R0 ⇒ .area:

10	.chan
.blk	
.buff	
.wcnt	
1	

When the user program needs to access the data read on the specified channel, a .WAIT request should be issued. This ensures that the data has been read completely. If an error occurred during the transfer, the .WAIT request indicates the error.

Errors:

Code	Explanation
0	Attempt to read past end-of-file
1	Hard error occurred on channel
2	Channel is not open

Example:

Refer to the .WRITE/.WRITC/.WRITW examples.

.READC

The .READC request transfers a specified number of words from the indicated channel to memory. Control returns to the user program immediately after the .READC is initiated. Execution of the user program continues until the .READC is complete, then control passes to the routine specified in the request. When an RTS PC is executed in the completion routine, control returns to the user program.

Macro Call: .READC .area, .chan, .buff, .wcnt, .crtn, .blk

where: .buff is the address of the buffer to receive the data read.

.wcnt is the number of words to be read.

.crtn is the address of the user's completion routine (refer to Section 9.2.8).

.blk is the block number relative to the start of the file, not block 0 of the device. The monitor translates the block supplied into an absolute device block number. The user program normally updates .blk before it is used again.

Request Format:

R0 ⇒ .area:

10	.chan
.blk	
.buff	
.wcnt	
address of completion routine	

When entering a .READC completion function the following are true:

1. R0 contains the channel status word for the operation. If bit 0 of R0 is set, a hardware error occurred during the transfer. The data may not be reliable.

2. R1 contains the octal channel number of the operation. This is useful when the same completion function is to be used for several different transfers.

Errors:

Code	Explanation
0	Attempt to read past end-of-file
1	Hard error occurred on channel
2	Channel is not open

Example:

Refer to the .WRITE/.WRITC/.WRITW examples.

.READW

The .READW request transfers a specified number of words from the indicated channel to memory. Control returns to the user program when the .READW is complete or if an error is detected.

Macro Call: .READW .area, .chan, .buff, .wcnt, .blk

where:

- .buff — is the address of the buffer to receive the data read.
- .wcnt — is the number of words to be read. The number must be positive.
- .blk — is the block number relative to the start of the file, not block 0 of the device. The monitor translates the block supplied into an absolute device block number. The user program normally updates .blk before it is used again.

Request Format:

R0 → .area:

10	.chan
.blk	
.buff	
.wcnt	
0	

On return from this call, the C bit set indicates an error has occurred. If no error occurred, the data is in memory at the specified address. In an F/B system, the other job can be run while the issuing job is waiting for the I/O to complete.

Note:
Upon return from any READ programmed request, R0 will contain no information if the read is from a sequential-access device (for example, magtape). If the read is from a random-access device (disk, DECtape) R0 will contain the actual number of words that will be read (.READ or .READC) or have been read (.READW). This will be less than the requested word count if an attempt is made to read past the end-of-file, but a partial transfer is possible. In the case of a partial transfer, the C bit is set and error code 0 is returned. Therefore, a

program should always use the returned would count as the number of words available. For example, suppose a file is 5 blocks long (i.e., it has block numbers 0 to 4) and a request is issued to read 512 words, starting at block 4. The request is shortened to 256 words; no error is indicated. Also note that since the request will be shortened to an exact number of blocks, a request for 256 words will either succeed or fail, but cannot be shortened.

Errors:

Code	Explanation
0	Attempt to read past end-of-file
1	Hard error occurred on channel
2	Channel is not open

Example:

Refer to the .WRITE/.WRITC/.WRITW examples.

.RELEAS

9.4.31 .RELEAS

The .RELEAS request removes the handler for the specified device from memory. The .RELEAS is ignored if the handler is:

1. Part of RMON (i.e., the system device),

2. Not currently resident, or

3. Resident because of a LOAD command to the Keyboard Monitor,

.RELEAS from the foreground is always ignored, since the foreground can only use handlers which have been LOADed.

Macro Call: .RELEAS .devname

where: .devname is the pointer to the .RAD50 device name.

Errors:

Code	Explanation
0	Handler name was illegal.

Example:

In the following example, the DECtape handler (DT) is loaded into memory, used, then released. If the system device is DECtape, the handler is already resident, and .FETCH will return HSPACE in R0.

```
                .MCALL  ..V2..,.REGDEF,.FETCH,.RELEAS,.EXIT
                ..V2..
                .REGDEF

        START:  .FETCH  #HSPACE,#DTNAME ;LOAD DT HANDLER
                BCS     FERR            ;NOT AVAILABLE

        ; USE HANDLER

                .RELEAS #DTNAME         ;MARK DT NO LONGER IN
                                        ;MEMORY.
                BR      START
        FERR:   HALT                    ;DT NOT AVAILABLE
        DTNAME: .RAD50  /DT /           ;NAME FOR DT HANDLER
        HSPACE:                         ;BEGINNING OF HANDLER
                                        ;AREA

                .END    START
```

.RENAME

9.4.32 .RENAME

The .RENAME request causes an immediate change of name of the file specified. An error occurs if the channel specified is already open.

Macro Call: .RENAME .area, .chan, .dblk

Request Format:

R0 ⇒ .area:

4	.chan
.dblk	

The .dblk argument consists of two consecutive .RAD50 device and file specifications. For example:

```
                .RENAME   #AREA,#7,#DBLK       ;USE CHANNEL 7
                BCS       RNMERR               ;NOT FOUND
                .
                .
                .
        DBLK:   .RAD50    /DT3/
                .RAD50    /OLDFIL/
                .RAD50    /MAC/
                .RAD50    /DT3/
                .RAD50    /NEWFIL/
                .RAD50    /MAC/
```

The first string represents the file to be renamed and the device it is found on. The second represents the new file name. If a file with the same name as the new file name specified already exists on the indicated device, it is deleted. The second occurrence of the device name DT3 is necessary for proper operation, and should not be omitted. The specified channel is left inactive when the .RENAME is complete. .RENAME requires that the handler to be used be resident at the time the .RENAME request is made. If it is not, a monitor error occurs. Note that .RENAME is legal only on files which are on disk or DECtape. (.RENAMEs to other devices are ignored.)

Errors:

Code	Explanation
0	Channel open
1	File not found

Example:

In the following example, the file DATA.TMP on DT0 is renamed to DATA.001:

```
        .MCALL  ..V2..,.REGDEF,.FETCH,.PRINT
        .MCALL  .EXIT,.RENAME
        ..V2..
        .REGDEF

START:  .FETCH  #HSPACE,#NAMBLK ;GET HANDLER
        BCS     FERR            ;SOME ERROR
        .RENAME #AREA,#0,#NAMBLK ;DO THE RENAME
        BCS     RNMERR          ;ERROR
        .EXIT
FERR:   .PRINT  #FMSG
        .EXIT
RNMERR: .PRINT  #RNMSG
        .EXIT
AREA:   .BLKW   5               ;ROOM FOR ARGS.
NAMBLK: .RAD50  /DT0DATA  TMP/  ;OLD NAME
        .RAD50  /DT0DATA  001/  ;NEW NAME
FMSG:   .ASCIZ  /FETCH?/        ;ERROR MESSAGES
RNMSG:  .ASCIZ  /RENAME?/
        .EVEN
HSPACE=.

        .END    START
```

.REOPEN

9.4.33 .REOPEN

The .REOPEN request reassociates the specified channel with a file on which a .SAVESTATUS was performed. The .SAVESTATUS/.REOPEN combination is useful when a large number of files must be operated on at one time. As many files as are needed can be opened with .LOOKUP, and their status preserved with .SAVESTATUS. When data is required from a file, a .REOPEN enables the program to read from the file. The .REOPEN need not be done on the same channel as the original .LOOKUP and .SAVESTATUS.

Macro Call: .REOPEN .area, .chan, .cblk

where: .cblk is the address of the five-word block where the channel status information was stored.

Request Format:

```
R0 => .area: | 6 | .chan |
             |   .cblk   |
```

Errors:

Code	Explanation
0	The specified channel is in use. The .REOPEN has not been done.

Example:

Refer to the example following the description of .SAVESTATUS.

.SAVESTATUS

9.4.34 .SAVESTATUS

The .SAVESTATUS request stores five words of channel status information into a user-specified area of memory. These words contain all the information RT-11 requires to completely define a file. When a .SAVESTATUS is done, the data words are placed in memory, and the specified channel is again available for use. When the saved channel data is required, the .REOPEN request is used.

.SAVESTATUS can only be used if a file has been opened with .LOOKUP. If .ENTER was used, .SAVESTATUS is illegal and returns an error. Note that .SAVESTATUS is legal only on files which are not on magtape or cassette.

Macro Call: .SAVESTATUS .area, .chan, .cblk

where: .cblk — is the address of the user memory block (5 words) where the channel status information is to be stored.

Request Format:

R0 → .area:

5	.chan
.cblk	

The five words stored are the five words normally contained in the channel area, as follows:

Word #	Contents
1	Channel status word. The contents of the bits of this word are:
2	Starting block number of the file. Zero for sequential-access devices.
3	Length of file (in 256-word blocks).
4	Data length of file; currently unused.
5	Even Byte: I/O count. Count of how many I/O requests have been made on this channel. Odd Byte:

Bits of word 1:

Bit #	Contents
0	1 - a hardware error occurred on this channel.
1-5	Index into monitor tables. This describes the physical device with which the channel is associated.
6	1 - a .RENAME operation is in progress on the channel.
7	1 - a .CLOSE operation must rewrite the directory (i.e., set when a .ENTER is done).
8-12	Contains the directory segment number (1-37(8)) in which the current open file can be found.
13	1 - An end-of-file was found on the channel.
14	Unused.
15	1 - This channel is currently in use (i.e., a file is open on this channel).

unit number of the device associated with the channel (between 0 - 7).

While the .SAVESTATUS/.REOPEN combination is very useful, care must be observed when using it. In particular, the following cases should be avoided:

1. If a .SAVESTATUS is performed and the same file is then deleted before it is reopened, it becomes available as an empty space which could be used by the .ENTER command. If this sequence occurs, the contents of the file supposedly saved will change.

2. Although the device handler for the required peripheral need not be in memory for execution of a .REOPEN, if the handler is not in memory when a .READ or .WRITE is executed, a fatal error is generated.

Errors:

Code	Explanation
1	The file was opened via .ENTER, or is a magtape or cassette file, and a .SAVESTATUS is illegal.

Example:

One of the more common uses of .SAVESTATUS and .REOPEN is to consolidate all directory access motion and code at one place in the program. All files necessary are opened and their status saved, then they are re-opened one at a time as needed. USR swapping can be minimized by locking in the USR, doing .LOOKUPs as needed, using .SAVESTATUS to save the file data, and then .UNLOCKing the USR.

In the program segment below, three input files are specified in the command string; these are then processed one at a time.

```
        .MCALL  ..V2..,.REGDEF,.CSIGEN,.SAVESTATUS,.REOPEN
        .MCALL  .READ,.EXIT
        ..V2..
        .REGDEF

START:  MOV     #AREA,R5
        .CSIGEN #DSPACE,#DEXT      ;GET INPUT FILES

        MOV     R0,BUFF            ;SAVE POINTER TO FREE CORE

        .SAVESTATUS R5,#3,#BLOCK1 ;SAVE FIRST INPUT FILE
        .SAVESTATUS R5,#4,#BLOCK2 ;SAVE SECOND FILE
        .SAVESTATUS R5,#5,#BLOCK3 ;SAVE THIRD FILE


        MOV     #BLOCK1,R4
PROCESS: .REOPEN R5,#0,R4         ;REOPEN FILE ON
                                   ;CHANNEL 0

        .READ   R5,#0,BUFF,COUNT,BLOCK ;PROCESS FILE ON CHANNEL 0

DONE:   ADD     #12,R4             ;POINT TO NEXT SAVESTATUS BLOCK
        CMP     R4,#BLOCK3         ;LAST FILE PROCESSED?
```

```
        BLOS    PROCESS         ;NO = DO NEXT
        .EXIT

BLOCK1: .WORD   0,0,0,0,0       ;MEMORY BLOCKS FOR
BLOCK2: .WORD   0,0,0,0,0       ;SAVESTATUS INFORMATION
BLOCK3: .WORD   0,0,0,0,0
AREA:   .BLKW   10

BUFF:   .WORD   0
BLOCK:  .WORD   0
COUNT:  .WORD   256.

DEXT:   .WORD   0,0,0,0
DSPACE=.
        .END    START
```

.SDAT/.SDATC/.SDATW

9.4.35 .SDAT/.SDATC/.SDATW

These requests are used in conjunction with the .RCVD/.RCVDW/.RCVDC calls to allow message transfers with RT-11. .SDAT transfers can be considered similar to .WRITE requests in which data transfer is not from a peripheral, but from one job to another. Additional I/O queue elements should be allocated for buffered I/O operations in .SDAT and .SDATC requests (see .QSET).

.SDAT

Macro Call: .SDAT .area, .buff, .wcnt

where: .buff is the buffer address of the beginning of the message to be transferred.

.wcnt is the number of words to transfer.

Request Format:

R0 → .area:

25	0
unused	
.buff	
.wcnt	
1	

Errors:

Code	Explanation
0	No other job exists.

Example:

See the example following .SDATW.

.SDATC

Macro Call: .SDATC .area, .buff, .wcnt, .crtn

where:

.buff	is the buffer address of the beginning of the message to be transferred.
.wcnt	is the number of words to transfer.
.crtn	is the address of the completion routine to be entered when the message has been transmitted (refer to Section 9.2.8).

Request Format:

R0 → .area:

25	0
unused	
.buff	
.wcnt	
.crtn	

Errors:

Code	Explanation
0	No other job exists.

Example:

See the example following .SDATW.

.SDATW

Macro Call .SDATW .area, .buff, .wcnt

where:

.buff	is the buffer address of the beginning of the message to be transferred
.wcnt	is the number of words to transfer

Request Format:

R0 → .area:

25	0
unused	
.buff	
.wcnt	
0	

Errors:

Code	Explanation
0	No other job exists.

Example:

In this example, the job first sends a message interrogating the other job about the status of an operation, and then looks for an acknowledgement from the job.

```
                .MCALL  ..V2..,.REGDEF,.SDAT,.RCVD,.MWAIT,.PRINT,.EXIT
                ..V2..
                .REGDEF

        START:
                MOV     #AREA,R5          ;SET UP EMT BLOCK
                .SDAT   R5,#SBUFF,#MLGTH  ;ASK HIM A QUESTION
                BCS     NOJOB             ;NO OTHER JOB AROUND!

                                          ;MISCELLANEOUS PROCESSING

                .RCVD   R5,#BUFF2,#20.    ;RECEIVE 20 DECIMAL WORDS
                .MWAIT                    ;WAIT FOR ACKNOWLEDGE.
                MOV     #BUFF2+2,R1       ;POINT TO ACTUAL ANSWER.
                CMPB    (R1)+,#'Y         ;IS FIRST WORD Y FOR YES?
                BNE     PRNEG             ;NEGATIVE ACKNOWLEDGE
                .PRINT  #POSACK

                .EXIT

        PRNEG:  .PRINT  #NEGACK           ;NEGATIVE ON OUR INQUIRY
                .EXIT
        SBUFF:  .ASCII  /IS THE REQUIRED PROCESS GOING?/
        MLGTH=.-SBUFF
        BUFF2:  .WORD   0                 ;ACTUAL LENGTH IS HERE
                .BLKW   20.               ;SPACE FOR 20. WORDS

        NOJOB:  .PRINT  #NJMSG
                .EXIT
        NEGACK: .ASCIZ  /NEGATIVE ACKNOWLEDGE/
        POSACK: .ASCIZ  /POSITIVE ACKNOWLEDGE/
        NJMSG:  .ASCIZ  /NO JOB/
                .EVEN
        AREA:   .BLKW   10.

                .END    START
```

.SETTOP

9.4.36 .SETTOP

The .SETTOP request allows the user program to request that a new address be specified as a program's upper limit. The monitor determines whether this address is legal and whether or not a memory swap is necessary when the USR is required. For instance, if the program specified an upper limit below the start address of USR, no swapping is necessary, as the USR is not overlaid. If .SETTOP from the background specifies a high limit greater than the address of the USR and a SET USR NOSWAP command has not been given, a memory swap is required. Section 9.2.5 gives details on determining where the USR is in memory and how to optimize the .SETTOP.

On return from .SETTOP, both R0 and the word at location 50 (octal) contain the highest memory address allocated for use. If the job requested an address higher than the highest address which is legal for the requesting job, it is adjusted down to that address.

Macro Call: .SETTOP .addr

where: .addr is the address of the word immediately following the free area desired.

Notes:

1. A program should never do a .SETTOP and assume that its new upper limit is the address it requested. It must always examine the returned contents of R0 or location 50 to determine its actual high address.

2. In Version 1 of RT-11, .SETTOP did not return the high address in R0, but only in word 50.

3. It is imperative that the value returned in R0 or location 50 be used as the absolute upper limit. If this value is ever exceeded, vital parts of the monitor may be destroyed, and the system integrity will be violated.

Errors:

None.

Example:

Following is an example in two parts. The first indicates how a small background job (i.e., one with free space between itself and the USR) can be assured of reserving space up to but not including the USR. This in effect gives the job all the space it can without causing the USR to become non-resident.

The second part indicates how to always reserve the maximum amount of space by making the USR non-resident.

I)

```
        .MCALL  ..V2..,.REGDEF,.SETTOP,.EXIT
        ..V2..
        .REGDEF

START:
RMON=54                         ;POINTER TO START OF RESIDENT
USR=266                         ;OFFSET FROM RESIDENT TO POINTER
                                ;WHERE USR WILL START.
```

```
        MOV     @#RMON,R1       ;START OF RMON TO R1
        MOV     USR(R1),R0      ;POINT TO LOWEST USR WORD
        TST     -(R0)           ;POINT TO HIGHEST WORD NOT IN USR
        .SETTOP                 ;AND ASK FOR IT
        MOV     R0,HICORE       ;R0 CONTAINS THE HIGH ADDRESS
                                ;THAT WAS RETURNED.
II)
        .SETTOP #-2             ;IF WE ASK FOR A VALUE GREATER
                                ;THAN START OF RESIDENT, WE
                                ;WILL GET BACK THE ABSOLUTELY
                                ;HIGHEST USABLE ADDRESS.
        MOV     R0,HICORE       ;THAT IS OUR LIMIT NOW

        .EXIT
HICORE: .WORD   0
        .END    START
```

If a SET USR NOSWAP command is executed, the USR cannot be made non-resident. In this case, in both I & II above, R0 would return a value just below the USR.

Caution should be used concerning technique I, above. If the background program is so large that the USR is normally positioned over part of it, the high limit value returned by the .SETTOP may actually be lower than the original limit. The USR is then resident, with a portion of the user program destroyed. The example in Section 9.2.5 shows how to include checks that will avoid this situation.

.SFPA

9.4.37 .SFPA

.SFPA allows users with floating point hardware (FPP on 11/45 and FIS on 11/40) to set trap addresses to be entered when a floating point exception occurs. If no user trap address is specified and a floating point (FP) exception occurs, a ?M-FP TRAP occurs, and the job is aborted.

Macro Call: .SFPA .area, .addr

where: .addr is the address of the routine to be entered when an exception occurs.

Request Format:

R0 → .area:

30	0
.addr	

Notes:

1. If the address argument is 0, user floating point routines are disabled and the fatal ?M-FP TRAP error is produced.

2. In the F/B environment, an address value of 1 indicates that the FP registers should be switched when a context switch occurs, but no user traps are enabled. This allows both jobs to use the FP unit. An address of 1 to the Single-Job Monitor is equivalent to an address of 0.

3. When the user routine is activated, it is necessary to re-execute an .SFPA request, as the monitor inhibits user traps when any one is serviced. It does this to inhibit any possible infinite loop being set up by repeated FP exceptions.

4. If the 11/45 FPP is being used, the instruction STST -(SP) is executed by the monitor before entering the user's trap routine. Thus, the trap routine must pop the two status words off the stack before doing an RTI. The program can tell if FPP hardware is available by examining the configuration word in the monitor (see Section 9.2.6).

Errors:

None.

Example:

This example sets up a user FP trap address.

```
          .MCALL  ..V2..,.REGDEF,.SFPA,.EXIT
          ..V2..
          .REGDEF
START:

          .SFPA   #AREA,#FPTRAP

          .EXIT

FPTRAP:   .
          .
          MOV     R0,-(SP)          ;R0 USED BY .SFPA
          .SFPA   #AREA,#FPTRAP
          MOV     (SP)+,R0          ;RESTORE R0
          RTI
          .
AREA:     .BLKW   10
          .END    START
```

.SPFUN

9.4.38 .SPFUN

This request is used principally with cassettes and magtape handlers to do device-dependent functions, such as rewind and backspace, to those devices. (It may also be used with diskette to allow reading and writing of absolute sectors; specific information is in Appendix H.)

Macro Call: .SPFUN .area, .chan, .code, .buff, .wcnt, .blk, .crtn

where: .code is the numerical code of the function to be performed

.buff	is the buffer address; this parameter must be set to zero if no buffer is required.
.crtn	is the entry point of a completion routine. If left blank, 0 is automatically inserted.

Request Format:

```
R0 → .area: | 32  | .chan |
            | .blk        |
            | .buff       |
            | .wcnt       |
            | .code | 377 |
            | .crtn       |
```

All other arguments are the same as those defined for .READ/.WRITE requests (Sections 9.4.30 and 9.4.47). They are only required when doing a .WRITE with extended record gap to MT. If the .crtn argument is left blank, the requested operation will complete before control returns to the user program. .crtn=1 is equivalent to executing a .READ or .WRITE in that the function is initiated and returns immediately to the user program. A .WAIT on the channel ensures that the operation is completed. If .crtn=N, it is taken as a completion routine address to be entered when the operation is complete.

The available functions and their codes are:

Function	MT	CT
Forward to last file		377
Forward to last block		376
Forward to next file		375
Forward to next block		374
Rewind to load point	373	373
Write file gap		372
Write EOF	377	
Forward 1 record	376	
Backspace 1 record	375	
Write with extended file gap	374	
Offline	372	

To use the .SPFUN request, the handler must be in memory and a channel associated with a file via a .LOOKUP request.

Refer to Appendix H for details of MT and CT handlers.

Errors:

Errors are detected in the same way as for the .READ/.READC/.READW requests. Refer to Section 9.4.30 for details.

Example:

The following example rewinds a cassette and writes out a 256-word buffer and then a file gap.

```
        .MCALL  ..V2..,.REGDEF,.FETCH,.LOOKUP,.SPFUN,.WRITW
        .MCALL  .EXIT,.PRINT,.WAIT,.CLOSE
        ..V2..
        .REGDEF

START:
        .FETCH  #HSPC,#CT           ;GET A HANDLER
```

```
        BCS     FERR                ;FETCH ERROR
        .LOOKUP #AREA,#4,#CT        ;LOOK IT UP ON CHANNEL 4
        BCS     LKERR               ;LOOKUP ERROR
        .SPFUN  #AREA,#4,#373,#0    ;REWIND SYNCHRONOUSLY
        BCS     SPERR               ;AN ERROR OCCURRED.
        MOV     #3,R5               ;COUNT
                                    ;BLOCK 0.
        .WRITW  #AREA,#4,#BUFF,#256.,BLK
        BCS     WTERR
        .SPFUN  #AREA,#4,#372,#0,,#1 ;ASYNCHRONOUS FILE GAP
        .PRINT  #DONE
        .WAIT   #4                  ;WAIT FOR DONE
        .CLOSE  #4                  ;CLOSE THE FILE
        .EXIT

AREA:   .BLKW   10
FERR:   .PRINT  #FMSG
        .EXIT
LKERR:  .PRINT  #LKMSG
        .EXIT
SPERR:  .PRINT  #SPMSG
        .EXIT
WTERR:  .PRINT  #WTMSG
        .EXIT
DONE:   .ASCIZ  /ALL DONE/
FMSG:   .ASCIZ  /FETCH?/
LKMSG:  .ASCIZ  /FILE?/
SPMSG:  .ASCIZ  /SPECIAL FUNCTION ERROR/
WTMSG:  .ASCIZ  /WRITE ERROR/
        .EVEN
CT:     .RAD50  /CT /
        .WORD   0,0,0
BUFF:   .BLKW   256.
BLK:    .WORD   0
HSPC=.
        .END    START
```

.SPND/.RSUM

9.4.39 .SPND/.RSUM (F/B only)

The .SPND/.RSUM requests allow a job to control execution of its mainstream code (that code which is not executing as a result of a completion routine). .SPND suspends the mainstream and allows only completion routines (for I/O and mark time requests) to run. .RSUM from one of the completion routines resumes the mainstream code. These functions enable a program to wait for a particular I/O or mark time request by suspending the mainstream and having the selected event's completion routine issue a .RSUM. This differs from the .WAIT request, which suspends the mainstream until all I/O operations on a specific channel have completed.

.SPND

Macro Call: .SPND

where: R0 ⇒ | 1 | 0 |

.RSUM

Macro Call: .RSUM

where: R0 → | 2 | 0 |

Notes:

1. The monitor maintains a suspension counter for each job. This counter is decremented by .SPND and incremented by .RSUM. A job will actually be suspended only if this counter is negative. Thus, if a .RSUM is issued before a .SPND, the latter request will return immediately.

2. A program must issue an equal number of .SPNDs and .RSUMs.

3. A .RSUM request from the mainstream code increments the suspension counter.

4. A .SPND request from a completion routine decrements the suspension counter, but does not suspend the mainstream. If a completion routine does a .SPND, the mainstream continues until it also issues a .SPND, at which time it is suspended and will require two .RSUMs to proceed.

5. Since a .TWAIT is simulated in the monitor using suspend and resume, a .RSUM issued from a completion routine without a matching .SPND may cause the mainstream to continue past a timed wait before the entire time interval has elapsed.

6. A .SPND or .RSUM, like most other programmed requests, may be issued from a user-written interrupt routine if the .INTEN/ .SYNCH sequence is followed. All notes referring to .SPND/ .RSUM from a completion routine also apply to this case.

Errors:

None.

Example:

In this example, the program starts a number of read operations and suspends itself until at least two of them are complete.

```
        .MCALL  ..V2..,.REGDEF,.SPND,.RSUM,.READC,.EXIT,.LOOKUP
        .MCALL  .PRINT,.WAIT
        ..V2..
        .REGDEF
START:
        .LOOKUP #AREA,#2,#FILE2
        BCS     1$
        .LOOKUP #AREA,#3,#FILE3
        BCS     1$
        .LOOKUP #AREA,#4,#FILE4
        BCC     3$
```

```
1$:     .PRINT  #2$
        .EXIT
2$:     .ASCIZ  /LOOKUP ERROR/
        .EVEN
3$:


        MOV     #2,RSVCTR           ;WAIT FOR 2 COMPLETIONS
        MOV     #AREA,R5
        .READC  R5,#2,#BUF1,COUNT1,#CROUTN,BLOK1
        BCS     ERROR
        .READC  R5,#3,#BUF2,COUNT2,#CROUTN,BLOK2
        BCS     ERROR
        .READC  R5,#4,#BUF3,COUNT3,#CROUTN,BLOK3
        BCS     ERROR
        .SPND

        .WAIT   #2
        .WAIT   #3
        .WAIT   #4
        .EXIT

CROUTN: ASL     R1                  ;DOUBLE CHANNEL # FOR INDEXING

        INC     DONEFL(R1)          ;R1=CHANNEL THAT IS DONE
                                    ;SET A FLAG SAYING SO.
        ROR     R0                  ;ANY ERRORS?
        ADC     ERRFLG(R1)          ;IF CARRY SET, SET ERROR FLAG FOR CHANNEL
        DEC     RSVCTR              ;ARE WE THE SECOND TO FINISH?
        BNE     1$                  ;NO
        .RSUM                       ;YES, START UP
1$:     RTS     PC

ERROR:  .PRINT  #RDMSG
        .EXIT
RDMSG:  .ASCIZ  /READ ERROR/
        .EVEN
AREA:   .BLKW   10
RSVCTR: .WORD   0
COUNT1: .WORD   256.
COUNT2: .WORD   256.
COUNT3: .WORD   256.
BLOK1:  .WORD   0
BLOK2:  .WORD   0
BLOK3:  .WORD   0
FILE2:  .RAD50  /DK TEST2 TMP/
FILE3:  .RAD50  /DK TEST3 TMP/
FILE4:  .RAD50  /DK TEST4 TMP/
DONEFL: .WORD   0,0,0
ERRFLG: .WORD   0,0,0
BUF1:   .BLKW   256.
BUF2:   .BLKW   256.
BUF3:   .BLKW   256.

        .END    START
```

.SRESET

9.4.40 .SRESET

The .SRESET (software reset) request performs the following functions:

1. Dismisses any device handlers which were brought into memory via a .FETCH call. Handlers which were loaded via the Keyboard Monitor LOAD command remain resident, as does the system device handler.

2. Purges any currently open files. Files opened for output with .ENTER will never be made permanent.

3. Reverts to using only 16 (decimal) I/O channels. Any channels defined with .CDFN are discarded. A .CDFN must be reissued to open more than 16 (decimal) channels after a .SRESET is performed.

4. Resets the I/O queue to one element. A .QSET must be reissued to allocate extra queue elements.

5. Clears completion queue of any completion routines.

Macro Call .SRESET

Errors:

None.

Example:

In the example below, .SRESET is used prior to calling the CSI to ensure that all handlers are removed from memory and the CSI is started with a free handler area.

```
        .MCALL  ..V2..,.REGDEF,.CSIGEN,.SRESET,.EXIT
        ..V2..
        .REGDEF


START:  .CSIGEN #DSPACE,#DEXT,#0 ;GET COMMAND STRING
        MOV     R0,BUFFER        ;R0 POINTS TO FREE MEMORY


DONE:   .SRESET                  ;RELEASE HANDLERS, DELETE
                                 ;TENTATIVE FILES
        BR      START            ;AND REPEAT PROGRAM.
```

```
DEXT:   .WORD   0,0,0,0           ;NO DEFAULT EXTENSIONS
BUFFER: 0
DSPACE=.                          ;START OF HANDLER AREA.

        .END    START
```

If the .SRESET had not been performed prior to the second call of .CSIGEN, it is possible that the second command string would load a handler over one that the monitor thought was resident from the first command line.

.TLOCK

9.4.41 .TLOCK

.TLOCK is used in an F/B system to attempt to gain ownership of the USR. It is similar to .LOCK in that if successful, the user job returns with the USR in memory. However, if a job attempts to .LOCK the USR while the other job is using it, the requesting job is suspended until the USR is free. With .TLOCK, if the USR is not available, control returns immediately with the C bit set to indicate the .LOCK request failed.

Macro Call: .TLOCK

The .TLOCK request allows the job to continue running, with only one sub-job affected. With a .LOCK request, all sub-jobs would be automatically suspended, and the other job in the system would run.

Request Format:

R0: → area: | 7 | 0 |

Errors:

Code	Explanation
0	USR is already in use by another job.

Example:

In the following example, the user program needs the USR for a sub-job it is executing. If it fails to get the USR it suspends that sub-job and runs another sub-job. This type of procedure is useful to schedule several sub-jobs within a background or foreground program.

```
        .MCALL  ..V2..,.REGDEF,.TLOCK,.LOOKUP,.UNLOCK,.EXIT,.PRINT
        ..V2..
        .REGDEF

START:
```

```
        .TLOCK                          ;GET THE USR
        BCS     SUSPND                  ;FAILED, SUSPEND SUB-JOB
        .LOOKUP #AREA,#4,#J1NAM ;LOOKUP A FILE
        BCS     LKERR
        .UNLOCK                         ;RELEASE USR


        .EXIT

SUSPND: JSR     PC,SPSJOB               ;SUSPEND SUB-JOB
        JSR     PC,SCHED                ;AND SCHEDULE NEXT USER


AREA:   .BLKW   10
J1NAM:  .RAD50  /DK TEST1 TMP/
LKERR:  .PRINT  #LKMSG
        .EXIT
LKMSG:  .ASCIZ  /LOOKUP ERROR/
        .EVEN
SPSJOB: RTS     PC
SCHED:  RTS     PC

        .END    START
```

.TRPSET

9.4.42 .TRPSET

.TRPSET allows the user job to intercept traps to 4 and 10 instead of having the job aborted with a ?M-TRAP TO 4 or ?M-TRAP TO 10 message. If .TRPSET is in effect when a trap occurs, the user-specified routine is entered. The sense of the C bit on entry to the routine determines which trap occurred: C bit clear indicates a trap to 4; set indicates a trap to 10. The user routine should exit via an RTI instruction.

Macro Call: .TRPSET .area, .addr

where: .addr — is the address of the user's trap routine. If an address of 0 is specified, the user's trap interception is disabled.

Request Format:

R0 → .area:

3	0
.addr	

Notes:

It is necessary to reissue a .TRPSET request whenever a trap occurs and the user routine is entered. The monitor inhibits servicing user traps prior to entering the first user trap routine. Thus, if a trap should occur from within the user's trap routine, a ?M-TRAP message is generated. The last operation the user routine should perform before an RTI is to reissue the .TRPSET request.

Errors:

None.

Example:

The following example sets up a user trap routine and, when the trap occurs, prints an appropriate error message.

```
        .MCALL  ..V2..,.REGDEF,.TRPSET,.EXIT,.PRINT
        ..V2..
        .REGDEF
START:

        .TRPSET #AREA,#TRPLOC
        MOV     #101,R0             ;SET TO PRODUCE A TRAP
        TST     (R0)+               ;THIS WILL TRAP TO 4.
        .WORD   67                  ;THIS WILL TRAP TO 10.
        .EXIT


TRPLOC: MOV     R0,-(SP)            ;R0 USED BY EMTS
        BCS     1$                  ;C SET = TRAP TO 10
        .PRINT  #TRP4               ;TRAP TO 4
        BR      2$
1$:     .PRINT  #TRP10              ;TRAP TO 10
2$:     .TRPSET #AREA,#TRPLOC       ;RESET TRAP ADDRESS
        MOV     (SP)+,R0            ;RESTORE R0
        RTI


AREA:   .BLKW   10
TRP4:   .ASCIZ  /TRAP TO 4/
TRP10:  .ASCIZ  /TRAP TO 10/
        .EVEN

        .END    START
```

.TTYIN/.TTINR

9.4.43 .TTYIN/TTINR

These requests are used to transfer characters from the console terminal to the user program. The character thus obtained appears right-justified (even byte) in R0.

The expansion of .TTYIN is:

```
EMT 340
BCS .-2
```

while that for .TTINR is:

```
EMT 340
```

If no characters or lines are available when an EMT 340 is executed, return is made with the C bit set. The implication of these calls is that .TTYIN causes a tight loop waiting for a character/line to appear, while the user can either wait or continue processing using .TTINR.

Macro Calls: .TTYIN .char

.TTINR

where: .char is the location where the character in R0 is stored. If not specified, the character is left in R0.

If the carry bit is set when execution of the .TTINR request is completed, it indicates that no character was available; the user has not yet typed a valid line. Under the F/B Monitor, .TTINR does not return the carry bit set unless bit 6 of the Job Status Word was on when the request was issued (see below).

There are two modes of doing console terminal input. This is governed by bit 12 of the Job Status Word. If bit 12 = 0, normal I/O is performed. In this mode, the following conditions apply:

1. The monitor echoes all characters typed; lower case characters are converted to upper case.

2. CTRL U (↑U) and RUBOUT perform line deletion and character deletion, respectively.

3. A carriage return, line feed, CTRL Z, or CTRL C must be struck before characters on the current line are available to the program. When carriage return is typed, characters on the line typed are passed one-by-one to the user program; both carriage return and line feed are passed to the program.

4. ALTMODEs (octal codes 175 and 176) are converted to ESCAPEs (octal 33).

If bit 12 = 1, the console is in special mode. The effects are:

1. The monitor does not echo characters typed except for CTRL C and CTRL O.

2. CTRL U and RUBOUT do not perform special functions.

3. Characters are immediately available to the program.

4. No ALTMODE conversion is done.

In special mode, the user program must echo the characters received. However, CTRL C and CTRL O are acted on by the monitor in the usual way. Bit 12 in the JSW must be set by the user program. This bit is cleared when control returns to RT-11.

CTRL F and CTRL B are not affected by the setting of bit 12. The monitor always acts on these characters.

CTRL S and CTRL Q are intercepted by the monitor (unless, under the F/B monitor, the SET TTY NOPAGE command is issued).

Under the F/B Monitor, if a terminal input request is made and no character is available, job execution is blocked until a character is ready. This is true for both .TTYIN and .TTINR, and for both normal and special modes. If a program really requires execution to continue and the carry bit to be returned, it must turn on bit 6 of the JSW (location 44) before the .TTINR request. Bit 6 is cleared when a program terminates.

Errors:

Code	Explanation
0	No characters available in ring buffer.

Example:

Refer to the example following the description of .TTYOUT/.TTOUTR.

.TTYOUT/.TTOUTR

9.4.44 .TTYOUT/.TTOUTR

These requests cause a character to be transmitted from R0 to the console terminal. The difference, as in the .TTYIN/.TTINR requests, is that if there is no room for the character in the monitor's buffer, the .TTYOUT request waits for room before proceeding, while the .TTOUTR does not wait for room and the character in R0 is not output.

Macro Calls: .TTYOUT .char

.TTOUTR

where: .char is the location containing the character to be loaded in R0 and printed. If not specified, the character in R0 is printed. Upon return from the request, R0 still contains the character.

If the carry bit is set when execution of the .TTOUTR request is completed, it indicates that there is no room in the buffer and that no character was output. Under the F/B Monitor, .TTOUTR normally does not return the carry bit set. Instead, the job is blocked until room is available in the output buffer. If a job really requires execution to continue and the carry bit to be returned, it must turn on bit 6 of the Job Status Word (location 44) before issuing the request.

The .TTINR and .TTOUTR requests have been supplied as a help to those users who do not wish to suspend program execution until a console operation is complete. With these modes of I/O, if a no-character or no-room condition occurs, the user program can continue processing and try the operation again at a later time.

Note:

If a foreground job leaves bit 6 on in the JSW, any further foreground .TTYIN or .TTYOUT requests will cause the system to lock out the background. Note also that each partition has its own JSW, and therefore can be in different terminal modes independently.

Errors:

Code	Explanation
0	Output ring buffer full.

Example:

As an example of the various terminal requests, the following program is coded in two ways. The program itself accepts a line from the keyboard, then repeats it on the terminal.

The first example uses .TTYIN and .TTYOUT, which are synchronous. The monitor retains control until both requests are satisfied, hence there is no time available for any other processing while waiting.

```
          .MCALL  ..V2..,.REGDEF,.TTYIN,.TTYOUT
          ..V2..
          .REGDEF

START:    MOV     #BUFFER,R1        ;POINT R1 TO BUFFER
          CLR     R2                ;CLEAR CHARACTER COUNT
INLOOP:   .TTYIN  (R1)+             ;READ CHAR INTO BUFFER
          INC     R2                ;BUMP COUNT
          CMPB    #12,R0            ;WAS LAST CHAR=LF?
          BNE     INLOOP            ;NO-GET NEXT
          MOV     #BUFFER,R1        ;YES-POINT R1 TO BUFFER
OUTLOOP:  .TTYOUT (R1)+             ;PRINT CHAR
          DEC     R2                ;DECREASE COUNT
          BEQ     START             ;DONE IF COUNT = 0
          BR      OUTLOOP

BUFFER=.

          .END    START
```

Rather than wait for the user to type something at INLOOP or wait for the output buffer to have available space at OUTLOOP, the routine can be recoded using .TTINR and .TTOUTR as follows:

```
                .MCALL  ..V2..,.REGDEF,.TTYIN,.TTYOUT
                ..V2..
                .REGDEF
                .MCALL  .TTINR,.TTOUTR,.EXIT

        START:  MOV     #BUFFER,R1      ;POINT R1 TO BUFFER
                CLR     R2              ;CLEAR CHARACTER COUNT
                BIS     #100,@#44       ;WE REALLY WANT CARRY SET
        INLOOP: .TTINR                  ;GET CHAR FROM TERMINAL
                BCS     NOCHAR          ;NONE AVAILABLE
        CHRIN:  MOVB    R0,(R1)+        ;PUT CHAR IN BUFFER
                INC     R2              ;INCREASE COUNT
                CMPB    R0,#12          ;WAS LAST CHAR = LF?
                BNE     INLOOP          ;NO-GET NEXT
                MOV     #BUFFER,R1      ;YES-POINT R1 TO BUFFER
        OUTLOOP: MOVB   (R1),R0         ;PUT CHAR IN R0
                .TTOUTR                 ;TYPE IT
                BCS     NOROOM          ;NO ROOM IN OUTPUT BUFFER
        CHROUT: DEC     R2              ;DECREASE COUNT
                BEQ     START           ;DONE IF COUNT=0
                INC     R1              ;BUMP BUFFER POINTER
                BR      OUTLOOP         ;AND TYPE NEXT
        NOCHAR:


                .TTINR                  ;PERIODIC CHECK FOR
                                        ;CHARACTER AVAILABILITY
                BCC     CHRIN           ;GOT ONE
                  .
                  .
                (code to be executed
                while waiting)
                  .
                  .
                  .
                BR      NOCHAR

        NOROOM:

                MOVB    (R1),R0         ;PERIODIC ATTEMPT TO TYPE
                                        ;CHARACTER
                .TTOUTR
                BCC     CHROUT          ;SUCCESSFUL
                  .
                  .
                  (code to be executed
                  while waiting)
                  .
                  .
        TYPEIT: BIC     #100,@#44       ;MUST CLEAR THIS BIT
                                        ;SO HANG WHILE
                                        ;WAITING FOR ROOM.
                .TTYOUT (R1)            ;PUT CHAR
                BIS     #100,@#44       ;RESTORE NO-WAIT
                BR      CHROUT


        BUFFER: .BLKW   100.
                .END    START
```

.TWAIT

9.4.45 .TWAIT

The .TWAIT request suspends the user's job for an indicated length of time. .TWAIT requires a queue element, and thus, should be considered when the .QSET request is executed.

Macro call: .TWAIT .area, .time

where: .time is a pointer to two words of time (high-order first, low-order second).

Request Format:

R0 → .area:	24	0
	.time	

Note:

Since a .TWAIT is simulated in the monitor using suspend and resume, a .RSUM issued from a completion routine without a matching .SPND may cause the mainstream to continue past a timed wait before the entire time interval has elapsed.

Errors:

Code	Explanation
0	No queue element was available.

Example:

.TWAIT can be used in applications where a program must be only activated periodically. This example will 'wake up' every 10 seconds to perform a task, and then 'sleep' again.

```
        .MCALL  ..V2..,.REGDEF,.TWAIT,.QSET,.EXIT,.PRINT
        ..V2..
        .REGDEF
GO:     .QSET   #QAREA,#7       ;SET UP 7 EXTRA ELEMENTS
START:  MOV     #EMTLST,R0      ;SET R0 TO THE ARG. BLOCK
        .TWAIT                  ;GO TO SLEEP FOR 10 SECONDS
        BCS     NOQ             ;NO QUEUE ELEMENT?
        JSR     PC,TASK         ;DO SOMETHING HERE
        BR      START           ;AND THEN CYCLE
QAREA:  .BLKW   7*7             ;SPACE FOR 7 ELEMENTS
EMTLST: .BYTE   0,24
        .WORD   TIME
TIME:   .WORD   0,10.*60.       ;10 SECOND INTERVALS
TASK:                           ;
                                ;
```

```
        INC     MPTR
        BIT     #1,MPTR
        BEQ     1$
        .PRINT  #MSG
        RTS     PC
1$:     .PRINT  #MSG1
        RTS     PC
MPTR:   .WORD   0
MSG:    .ASCIZ  /TICK/
MSG1:   .ASCIZ  /TOCK/
        .EVEN
NOQ:    .EXIT
        .END    GO
```

.WAIT

9.4.46 .WAIT

The .WAIT request suspends program execution until all input/output requests on the specified channel are completed. The .WAIT request combined with the .READ/.WRITE requests makes double-buffering a simple process.

.WAIT also conveys information back through its error returns. An error is returned if either the channel is not currently open or if the last I/O operation resulted in a hardware error.

Macro Call: .WAIT .chan

Note:

In a F/B system, executing a .WAIT when I/O is pending causes that job to be suspended and the other job to run, if possible.

Request Format:

R0 → | 0 | .chan |

Errors:

Code	Explanation
0	Channel specified is not open.
1	Hardware error occurred on the previous I/O operation on this channel.

These error codes make the .WAIT request useful in checking channel status.

Example:

For an example of .WAIT used for I/O synchronization, see the examples in the next section.

An example of the use of .WAIT for error detection is its use in conjunction with .CSIGEN to determine which file fields in the command string have been specified. For example, a program such as MACRO might use the following code to determine if a listing file is desired.

```
        .MCALL  ..V2..,.REGDEF,.WAIT,.CSIGEN,.EXIT
        ..V2..
        .REGDEF

START:

        .CSIGEN #DSPACE,#DEXT,#0 ;PROCESS COMMAND STRING
        .WAIT   #0              ;CHECK FOR FILE IN FIRST FIELD
        BCS     NOBINARY        ;NO BINARY DESIRED

NOBINARY:

        .WAIT   #1              ;CHECK FOR LISTING SPECIFICATION
        BCS     NOLISTING       ;NO LISTING DESIRED

NOLISTING:

        .WAIT   #3              ;CHECK FOR INPUT FILE OPEN
        BCS     ERROR           ;NO INPUT FILE

ERROR:  .EXIT

DEXT:   .RAD50  /MAC/
        .RAD50  /OBJ/
        .RAD50  /LST/
        .WORD   0
DSPACE=.
        .END    START
```

.WRITE/.WRITC/.WRITW

9.4.47 .WRITE/.WRITC/.WRITW

Note that in the case of .WRITE and .WRITC, additional queue elements should be allocated for buffered I/O operations (see .QSET).

.WRITE

The .WRITE request transfers a specified number of words from memory to the specified channel. Control returns to the user program immediately after the request is queued.

Macro Call: .WRITE .area, .chan, .buff, .wcnt, .blk

where:	.buff	is the address of the memory buffer to be used for output.
	.wcnt	is the number of words to be written.
	.blk	is the number of the block to be written.

Request Format:

R0 ⇛ .area:

11	.chan
.blk	
.buff	
.wcnt	
1	

Notes:

See the note following .WRITW.

Errors:

Code	Explanation
0	Attempted to write past end-of-file.
1	Hardware error.
2	Channel was not opened.

Example:

Refer to the examples following .WRITW.

.WRITC

The .WRITC request transfers a specified number of words from memory to a specified channel. Control returns to the user program immediately after the request is queued. Execution of the user program continues until the .WRITC is complete, then control passes to the routine specified in the request. When an RTS PC is encountered in the routine, control returns to the user program.

Macro Call: .WRITC .area, .chan, .buff, .wcnt, .crtn, .blk

where:	.buff	is the address of the memory buffer to be used for output.
	.wcnt	is the number of words to be written.
	.crtn	is the address of the completion routine to be entered (see Section 9.2.8).
	.blk	is the number relative to the start of the file, not block 0 of the device. The monitor translates the block supplied into an absolute device block number. The user program normally updates .blk before it is used again.

Request Format:

R0 → .area:

11	.chan
.blk	
.buff	
.wcnt	
.crtn	

When entering a .WRITC completion function the following are true:

1. R0 contains the channel status word for the operation. If bit 0 of R0 is set, a hardware error occurred during the transfer. The data may not be reliable.

2. R1 contains the octal channel number of the operation. This is useful when the same completion function is to be used for several different transfers.

Notes:

See the note following .WRITW.

Errors:

Code	Explanation
0	End-of-file on output. Tried to write outside limits of file.
1	Hardware error occurred.
2	Specified channel is not open.

Example:

Refer to the examples following .WRITW.

.WRITW

The .WRITW request transfers a specified number of words from memory to the specified channel. Control returns to the user program when the .WRITW is complete.

Macro Call: .WRITW .area, chan, .buff, .wcnt, .blk

where: .buff is the address of the buffer to be used for output.

.wcnt is the number of words to be written. The number must be positive.

.blk is the number of the block to be written.

Request Format:

R0 → .area:	11	.chan
	.blk	
	.buff	
	.wcnt	
	0	

Note:
Upon return from any WRITE programmed request, R0 will contain no information if the write is to a sequential-access device (for example, magtape). If the write is to a random-access device (disk, DECtape), R0 contains the number of words that will be written (.WRITE or .WRITC) or have been written (.WRITW). If a request is made to write past the end-of-file on a random-access device, the word count is shortened and an error is returned. Note that the write will be done and a completion routine, if specified, will be entered, unless the request cannot be partially filled (shortened word count = 0).

Errors:

Code	Explanation
0	Attempted to write past EOF.
1	Hardware error.
2	Channel was not opened.

Examples:

The following routine illustrates the differences between the three types of .READ/.WRITE requests and is coded in three ways, each using a different mode of monitor I/O. The routine itself is a simple program to duplicate a paper tape.

In the first example, .READW and .WRITW are used. The I/O is completely synchronous, with each request retaining control until the buffer is filled (or emptied).

```
        .MCALL  ..V2..,.REGDEF,.FETCH,.READW,.WRITW
        .MCALL  .ENTER,.LOOKUP,.PRINT,.EXIT,.CLOSE,.WAIT
        ..V2..
        .REGDEF

ERRWD=52

START:  .FETCH  #HSPACE,#PRNAME ;GET PR HANDLER
        BCS     FERR            ;PR NOT AVAILABLE
        MOV     R0,R2           ;R0 HAS NEXT FREE LOCATION
        .FETCH  R2,#PPNAME      ;GET PP HANDLER
        BCS     FERR            ;NOT AVAILABLE
        MOV     #AREA,R5        ;EMT ARGUMENT AREA
        CLR     R4              ;R4 IS OUTPUT CHANNEL; 0
        MOV     #1,R3           ;R3 IS INPUT CHANNEL ;1
        .ENTER  R5,R4,#PPNAME   ;ENTER THE FILE
        BCS     ENERR           ;SOME ERROR IN ENTER
        .LOOKUP R5,R3,#PRNAME   ;LOOKUP FILE ON CHANNEL 1
        BCS     LKERR           ;ERROR IN LOOKUP
        CLR     R1              ;USE R1 AS BLOCK NUMBER
LOOP:   .READW  R5,R3,#BUFF,#256.,R1 ;READ ONE BLOCK
        BCS     RDERR
        .WRITW  R5,R4,#BUFF,#256.,R1 ;WRITE THAT BLOCK
```

```
        BCS     WTERR
        INC     R1              ;BUMP BLOCK, NOTE: THIS IS
                                ;NOT NECESSARY FOR NON-FILE
                                ;DEVICES IN GENERAL, IT IS
                                ;USED HERE AS AN EXAMPLE OF
                                ;A GENERAL TECHNIQUE.
        BR      LOOP            ;KEEP GOING
RDERR:  TSTB    ERRWD           ;ERROR, IS IT EOF?
        BEQ     1$              ;YES
        .PRINT  #RDMSG          ;NO, HARD READ ERROR
        .EXIT
1$:     .CLOSE  R3              ;CLOSE INPUT AND OUTPUT
        .CLOSE  R4
        .EXIT                   ;AND EXIT.
WTERR:  .PRINT  #WTMSG
        .EXIT
PRNAME: .RAD50  /PR /           ;NOTE THAT PR NEEDS NO FILE NAME
        .WORD   0               ;FILE NAME NEED ONLY BE 0.
PPNAME: .RAD50  /PP /
        .WORD   0
FERR:   .PRINT  #FMSG           ;ERROR ACTIONS GO HERE. IT IS
        .EXIT                   ;GENERALLY UNDESIRABLE TO
ENERR:  .PRINT  #EMSG           ;EXECUTE A HALT OR RESET
        .EXIT                   ;INSTRUCTION ON ERROR.
LKERR:  .PRINT  #LMSG
        .EXIT
FMSG:   .ASCIZ  /NO DEVICE?/
EMSG:   .ASCIZ  /ENTRY ERROR?/
LMSG:   .ASCIZ  /LOOKUP ERROR?/
RDMSG:  .ASCIZ  /READ ERROR?/
WTMSG:  .ASCIZ  /WRITE ERROR?/
        .EVEN
AREA:   .BLKW   10
BUFF:   .BLKW   256.
HSPACE=.
        .END    START
```

The same routine can be coded using .READ and .WRITE as follows. The .WAIT request is used to determine if the buffer is full or empty prior to its use.

```
        .MCALL  ..V2..,.REGDEF,.FETCH,.READ,.WRITE
        .MCALL  .ENTER,.LOOKUP,.PRINT,.EXIT,.CLOSE,.WAIT
        ..V2..
        .REGDEF

ERRWD=52

START:  .FETCH  #HSPACE,#PRNAME ;GET PR HANDLER
        BCS     FERR            ;PR NOT AVAILABLE
        MOV     R0,R2           ;R0 HAS NEXT FREE LOCATION
        .FETCH  R2,#PPNAME      ;GET PP HANDLER
        BCS     FERR            ;NOT AVAILABLE
        MOV     #AREA,R5        ;EMT ARGUMENT AREA
        CLR     R4              ;R4 IS OUTPUT CHANNEL; 0
        MOV     #1,R3           ;R3 IS INPUT CHANNEL #1
        .ENTER  R5,R4,#PPNAME   ;ENTER THE FILE
```

```
        BCS     ENERR               ;SOME ERROR IN ENTER
        .LOOKUP R5,R3,#PRNAME       ;LOOKUP FILE ON CHANNEL 1
        BCS     LKERR               ;ERROR IN LOOKUP
        CLR     R1                  ;USE R1 AS BLOCK NUMBER
LOOP:   .READ   R5,R3,#BUFF,#256,,R1 ;READ A BUFFER
        BCS     RDERR
        .WAIT   R3                  ;WAIT FOR BUFFER
        BCS     IOERR               ;ERROR HERE IS HARD ERROR
        .WRITE  R5,R4,#BUFF,#256,,R1 ;WRITE TTHE BUFFER
        BCS     IOERR               ;I/O ERROR
        INC     R1
        BR      LOOP                ;KEEP GOING
RDERR:  TSTB    ERRWD               ;ERROR, IS IT EOF?
        BNE     IOERR               ;NO, HARD ERROR
        .CLOSE  R3                  ;CLOSE INPUT AND OUTPUT
        .CLOSE  R4
        .EXIT                       ;AND EXIT.
IOERR:  .PRINT  #IOMSG              ;NO, HARD READ ERROR
        .EXIT
PRNAME: .RAD50  /PR /               ;NOTE THAT PR NEEDS NO FILE NAME
        .WORD   0                   ;FILE NAME NEED ONLY BE 0.
PPNAME: .RAD50  /PP /
        .WORD   0
FERR:   .PRINT  #FMSG               ;ERROR ACTIONS GO HERE. IT IS
        .EXIT                       ;GENERALLY UNDESIRABLE TO
ENERR:  .PRINT  #EMSG               ;EXECUTE A HALT OR RESET
        .EXIT                       ;INSTRUCTION ON ERROR.
LKERR:  .PRINT  #LMSG
        .EXIT
FMSG:   .ASCIZ  /NO DEVICE?/
EMSG:   .ASCIZ  /ENTRY ERROR?/
LMSG:   .ASCIZ  /LOOKUP ERROR?/
IOMSG:  .ASCIZ  "I/O ERROR?"
WTMSG:  .ASCIZ  /WRITE ERROR?/
        .EVEN
AREA:   .BLKW   10
BUFF:   .BLKW   256.
HSPACE=.
        .END    START
```

.READ and .WRITE are also often used for double-buffered I/O. The basic double-buffering algorithm for input is:

```
                                Explanation

        READ    BUFFER 1        Fill BUFFER 1
LOOP:   WAIT    BUFFER 1        Wait for BUFFER 1 to fill
        READ    BUFFER 2        Start filling BUFFER 2
        USE     BUFFER 1        Process BUFFER 1 while BUFFER 2 fills
        WAIT    BUFFER 2        Wait for BUFFER 2 to fill
        READ    BUFFER 1        Start filling BUFFER 1
        USE     BUFFER 2        Process BUFFER 2 while BUFFER 1 fills
        BR      LOOP
```

Correspondingly, the basic double-buffering algorithm for output is:

		Explanation
	FILL BUFFER 1	Prepare BUFFER 1 for output
LOOP:	WRITE BUFFER 1	Start emptying BUFFER 1
	FILL BUFFER 2	Fill BUFFER 2 while BUFFER 1 empties
	WAIT BUFFER 1	Wait for BUFFER 1 to empty
	WRITE BUFFER 2	Start emptying BUFFER 2
	FILL BUFFER 1	Fill BUFFER 1 while BUFFER 2 empties
	WAIT BUFFER 2	Wait for BUFFER 2 to empty
	BR LOOP	

The previous example program can be coded using completion routines via .READC and .WRITC as follows. Once the initial read is performed, the remainder of the I/O is performed by the completion routines.

```
        .MCALL  ..V2..,.REGDEF,.FETCH,.READC,.WRITC
        .MCALL  .ENTER,.LOOKUP,.PRINT,.EXIT,.CLOSE,.WAIT
        ..V2..
        .REGDEF

ERRBYT=52

START:  .FETCH  #HSPACE,#PRNAME ;GET PR HANDLER
        BCS     FLNK            ;PR NOT AVAILABLE
        MOV     R0,R2           ;R0 HAS NEXT FREE LOCATION
        .FETCH  R2,#PPNAME      ;GET PP HANDLER
FLNK:   BCS     FERR            ;NOT AVAILABLE
        MOV     #AREA,R5        ;EMT ARGUMENT AREA
        CLR     R4              ;R4 IS OUTPUT CHANNEL; 0
        MOV     #1,R3           ;R3 IS INPUT CHANNEL ;1
        .ENTER  R5,R4,#PPNAME   ;ENTER THE FILE
        BCS     ENERR           ;SOME ERROR IN ENTER
        .LOOKUP R5,R3,#PRNAME   ;LOOKUP FILE ON CHANNEL 1
        BCS     LKERR           ;ERROR IN LOOKUP
        CLR     R1              ;USE R1 AS BLOCK NUMBER
LOOP:   CLR     DFLG            ;CLEAR DONE/ERROR FLAG
        .READC  R5,R3,#BUFF,#256.,#RDCOMP,R1 ;READ ONE BLOCK
        BCS     EOF             ;NO ERROR WILL HAPPEN HERE
1$:     TST     DFLG            ;DONE FLAG SET?
        BEQ     1$              ;NO, WAIT FOR IT TO BE SET.
        BMI     IOERR           ;YES, BUT HARD ERROR OCCURRED
EOF:    .CLOSE  R3              ;CLOSE INPUT AND OUTPUT CHANNELS
        .CLOSE  R4
        .EXIT                   ;ALL DONE
.ENABL  LSB
RDCOMP: ROR     R0              ;IF BIT 0 SET
        BCS     RWERR           ;AN ERROR OCCURRED.
        .WRITC  #AREA,#0,#BUFF,#256.,#WRCOMP,BLKN ;WRITE THAT BLOCK
        BCC     2$              ;ERROR HERE IS HARDWARE
RWERR:  MOV     #-1,DFLG        ;FLAG THE ERROR
2$:     RTS     PC
WRCOMP: ROR     R0
        BCS     RWERR           ;HARDWARE ERROR
        INC     BLKN            ;BUMP BLOCK NUMBER.
        .READC  #AREA,#1,#BUFF,#256.,#RDCOMP,BLKN
        BCC     3$              ;NO ERROR
        TSTB    ERRBYT          ;EOF?
```

```
        BNE     RWFRR               ;NO, HARD  ERROR
        INC     DFLG                ;SAY WE'RE DONE
3$:     RTS     PC
.DSABL  LSB
FERR:   MOV     #FMSG,R0            ;ERROR ACTIONS GO HERE. IT IS
        BR      TYPIT               ;GENERALLY UNDESIRABLE TO
ENFRR:  MOV     #EMSG,R0            ;EXECUTE A HALT OR RESET
        BR      TYPIT               ;INSTRUCTION ON ERROR.
IOFRR:  MOV     #IOMSG,R0
        BR      TYPIT
LKERR:  MOV     #LMSG,R0
TYPIT:  .PRINT
        .EXIT
.NLIST BEX
FMSG:   .ASCIZ  /NO DEVICE?/
EMSG:   .ASCIZ  /ENTRY ERROR?/
LMSG:   .ASCIZ  /LOOKUP ERROR?/
IOMSG:  .ASCIZ  "I/O ERROR?"
.LIST BEX
.EVEN
DFLG:   .WORD   0
PRNAME: .RAD50  /PR /                ;NOTE THAT PR NEEDS NO FILE NAME
        .WORD   0                    ;FILE NAME NEED ONLY BE 0.
PPNAME: .RAD50  /PP /
        .WORD   0
BLKN:   .WORD   0                    ;BLOCK NUMBER
AREA:   .BLKW   10
BUFF:   .BLKW   256.
HSPACE=.
        .END    START
```

The following example incorporates the .LOOKUP, .READW, and .CLOSE requests. The program opens the file RT11.MAC which is on the system device, SY:, for input on channel 0. The first block is read and the file is then closed.

```
        .MCALL  ..V2..,.REGDEF,.CLOSE,.LOOKUP
        .MCALL  .PRINT,.EXIT,.READW,.FETCH

        ..V2..
        .REGDEF

START:  MOV     #LIST,R5            ;EMT ARGUMENT LIST POINTER
        CLR     R4                  ;BLOCK NUMBER
        CLR     R3                  ;CHANNEL #
        .FETCH  #CORADD,#FPTR       ;FETCH DEVICE HANDLER
        BCC     2$
        MOV     #FETMSG,R0          ;FETCH ERROR
1$:     .PRINT                      ;PRINT ERROR MESSAGE
        .EXIT
2$:     .LOOKUP R5,R3,#FPTR         ;LOOKUP FILE ON CHANNEL 0
        BCC     3$
        MOV     #LKMSG,R0           ;PRINT FAILURE MESSAGE
        BR      1$
3$:     .READW  R5,R3,#BUFF,#256,,R4 ;REA ONE BLOCK
        BCC     4$
        MOV     #HDMSG,R0           ;READ ERROR
        BR      1$
4$:     .CLOSE  R3                  ;CLOSE THE CHANNEL
        .EXIT

LIST:   .BLKW   5                   ;LIST FOR EMT CALLS
FPTR:   .RAD50  /SY RT11  MAC/      ;RAD50 OF FIEL NAME,DEVICE
FETMSG: .ASCIZ  /FETCH FAILED/      ;ASCII MESSAGES
LKMSG:  .ASCIZ  /LOOKUP FAILED/
```

```
RDMSG:  .ASCIZ  /READ FAILED/
        .EVEN
CORADD: .BLKW   2000            ;SPACE FOR LARGEST HANDLERS
BUFF=.
        .END    START
```

9.5 CONVERTING VERSION 1 MACRO CALLS TO VERSION 2

As mentioned in the introduction of this chapter, RT-11 Version 2 supports a slightly modified format for system macro calls than Version 1. This section details the conversion process from the Version 1 format to Version 2.

9.5.1 Macro Calls Requiring No Conversion

Version 1 macro calls which need no conversion are:

.CSIGN	.RCTLO
.CSISPC	.RELEAS
.DATE	.SETTOP*
.DSTATUS	.SRESET
.EXIT	.TTINR**
.FETCH	.TTOUTR
.HRESET	.TTYIN
.LOCK	.TTYOUT
.PRINT	.UNLOCK
.QSET	

*Provided location 50 is examined for the maximum value.

**Except in F/B System.

9.5.2 Macro Calls Which May Be Converted

The following Version 1 macro calls may be converted:

.CLOSE	.RENAME
.DELETE	.REOPEN
.ENTER	.SAVESTATUS
.LOOKUP	.WAIT
.READ	.WRITE

The general format of the V1 macro is:

```
.PRGREQ .chan, .arg1 .arg2,...argn
```

In this form, .chan is an integer between 0 and 17 (inclusive), and is not a general assembler argument. The channel number is assembled into the EMT instruction itself. The arguments arg1-argn are always pushed either into R0 or on the stack.

The V2 equivalent of the above call is:

```
.PRGREQ .area, .chan, .arg1,....argn
```

In the V2 call, the .chan argument can be any legal assembler argument; it need not be in the range 0 to 17 (octal), but should be in the range 0-377 (octal). .area points to a memory list where the arguments arg1...argn will go.

As an example, consider a .READ request in both forms:

```
     V1:        .READ 5,#BUFF,#256.,BLOCK

     V2:        .READ #AREA,#5,#BUFF,#256.,BLOCK
                     .
                     .
                     .

     AREA:      .WORD 0    ;CHANNEL/FUNCTION CODE HERE
                .WORD 0    ;BLOCK NUMBER HERE
                .WORD 0    ;BUFFER ADDRESS HERE
                .WORD 0    ;WORD COUNT HERE
                .WORD 0    ;A 1 GOES HERE.
```

Thus, the difference in the calls is that in Version 2 the channel number becomes a legal assembler argument and the .area argument has been added.

Following is a complete list of the conversions necessary for each of the EMT calls. Both the Version 1 and Version 2 formats are given. Note that parameters inside [] are optional parameters. Refer to the appropriate section in this chapter for more details of each request.

Version	Programmed Request
V1:	.DELETE .chan,.dblk
V2:	.DELETE .area,.chan,.dblk,[.count]
V1:	.LOOKUP .chan,.dblk
V2:	.LOOKUP .area,.chan,.dblk,[.count]
V1:	.ENTER .chan,.dblk,[.length]
V2:	.ENTER .area,.chan,.dblk,[.length],[.count]
V1:	.RENAME .chan,.dblk
V2:	.RENAME .area,.chan,.dblk
V1:	.SAVESTAT .chan,.cblk
V2:	.SAVESTAT .area,.chan,.cblk
V1:	.REOPEN .chan,.cblk
V2:	.REOPEN .area,.chan,.cblk
V1:	.CLOSE .chan
V2:	.CLOSE .chan
V1:	.READ/.READW .chan,.buff,.wcnt,.blk
V2:	.READ/.READW .area,.chan,.buff,.wcnt,.blk
V1:	.READC .chan,.buff,.wcnt,.crtn,.blk
V2:	.READC .area,.chan,.buff,.wcnt,.crtn,.blk
V1:	.WRITE/.WRITW .chan,.buff,.wcnt,.blk
V2:	.WRITE/.WRITW .area,.chan,.buff,.wcnt,.blk

```
V1:     .WRITC .chan,.buff,.wcnt,.crtn,.blk
V2:     .WRITC .area,.chan,.buff,.wcnt,.crtn,.blk

V1:     .WAIT .chan
V2:     .WAIT .chan
```

Important features to keep in mind for Version 2 calls are:

1. Version 2 calls require the .area argument, which points to the area where the other arguments will be.

2. Enough memory space must be allocated to hold all the required arguments.

3. .chan must be any legal assembler argument, not just an integer between 0-17 (octal).

4. Blank fields are permitted in the Version 2 calls. Any field not specified (left blank) is left alone in the argument block.

CHAPTER 10

EXPAND UTILITY PROGRAM

EXPAND is an RT-11 system program which processes the macro references in a macro assembly language source file. EXPAND accepts a subset of the complete macro language and, using the system library file SYSMAC.8K, produces an output file in which all legal macro references are expanded into macro-free source code. EXPAND is normally used with ASEMBL, the RT-11 assembler designed for minimum memory configurations (refer to Chapter 11).

10.1 LANGUAGE

EXPAND simply copies its input files to its output file unless it encounters any of the following directives (see Chapter 5 for more information about these directives):

1.	.MCALL	Directs EXPAND to search the file SY:SYSMAC.SML to find the macro names listed in the .MCALL directive. If the macro names are found, EXPAND stores their definitions in its internal tables.
2.	.MACRO	Directs EXPAND to copy a macro definition from the user's input file into the internal tables.
3.	.name	If .name is the name of a macro defined in either a .MCALL or .MACRO directive, then .name is expanded according to the definition stored for it in the EXPAND internal tables.
4.	.ENDM	If encountered while storing a macro definition, the .ENDM directive terminates the definition. It is not recognized outside macro definitions.

10.2 RESTRICTIONS

Unlike the full macro assembler (MACRO), EXPAND only expands macros that observe the following restrictions:

1. The following directives may not be used:

.ERROR	.NARG
.IF DIF	.NCHR
.IF IDN	.NTYPE
.IRP	.PRINT
.IRPC	.REPT
.MEXIT	

2. Macros cannot be nested. Recursive macros that call themselves directly or indirectly are illegal and cause an error message.

3. Macros cannot be redefined. Once a name has been used for a macro name, it cannot be used again in the program for a macro or symbol name.

4. Macro names must begin with a dot (.). If the dot is missing, an error message is printed.

5. Dummy argument names must begin with a dot (.). Such names cannot be used as dummy argument names in the macro but can be used for other purposes outside of the macro.

6. The backslash operator is not available.

7. Automatically created symbols are not available.

8. No more than 30 arguments may be used in any MACRO directive.

10.3 CALLING AND USING EXPAND

To run EXPAND, type:

```
R EXPAND
```

in response to the dot printed by the Keyboard Monitor. EXPAND responds with an asterisk indicating that it is ready to accept a command string. A command string must be of the following form:

```
*ofile=ifile1,ifile2,...,ifile6
```

ifile2 through ifile6 are optional. Each file specification follows the general RT-11 command string syntax (dev:filnam.ext). The default value for each file specification is noted below:

I/O File	Dev	Ext
ofile	DK	PAL
ifile1,..., ifile6	device used for last source file specified or DK	MAC

Type CTRL C to halt EXPAND and return control to the monitor. To restart EXPAND, type R EXPAND or the REENTER command in response to the monitor's dot.

EXPAND Utility Program

EXPAND copies sequentially the specified input files to the specified output file until a macro directive is encountered. EXPAND then changes the macro directive to a comment by inserting a semicolon so that it will not be seen later by the assembler (usually ASEMBL).

If the directive is .MCALL, EXPAND searches the system library file (SYSMAC.8K) for the requested macro definitions. The requested definitions are then included in the user's program in the order in which they are found in the library.

For the .MACRO directive, EXPAND reads each line following the directive up to the next .ENDM directive. Each line is stored in the internal definition table and then changed to a comment in the output file so that it is not processed later by the assembler. Also, any occurrence of a macro argument name within the definition is flagged internally so that it can be replaced by the real argument value whenever the macro is later referenced.

For macro references, EXPAND locates the stored macro definition in its internal tables, binds the actual argument values to the argument names, and changes the macro reference to a comment line. EXPAND then begins copying the stored definition to the output file. Whenever a macro argument name is encountered in the definition, it is replaced by the corresponding actual argument value.

Examples:

The following are examples of input and corresponding EXPAND output.

INPUT

```
R1=%1
SP=%6
PC=%7
.MACRO .CALL .SUBR
JSR PC,.SUBR
.ENDM
.MCALL .LOOKUP,.READ,..V2..
```

OUTPUT

```
; RT-11 MACRO EXPAND V02-02

        R1=%1
        SP=%6
        PC=%7
;       .MACRO .CALL .SUBR
;       JSR PC,.SUBR
;       .ENDM
;       .MCALL .LOOKUP,.READ,..V2..
;.MACRO ..V2..
;...V2=1
;.ENDM
;.MACRO .LOOKUP .AREA,.CHAN,.DEVBLK,.SPF
;.IF DF ...V1
;.IF NB .CHAN
;               MOV     .CHAN,%0
;.ENDC
;               EMT     ^O<20+.AREA>
;.IFF
;.IF NB .AREA
;               MOV     .AREA,%0
;               MOVB    #1,1(0)
;.ENDC
;.IF NB .CHAN
;               MOVB    .CHAN,(0)
;.ENDC
```

```
                                ;.IF NB .DEVBLK
                                ;                   MOV      .DEVBLK,2.(0)
                                ;.ENDC
                                ;.IF NB .SPF
                                ;                   MOV      .SPF,4.(0)
                                ;.IFF
                                ;                   CLR      4.(0)
                                ;.ENDC
                                ;                   EMT      ^O375
                                ;.ENDC
                                ;.ENDM
                                ;.MACRO .READ       .AREA,.CHAN,.BUFF,.WCNT,.BLK
                                ;.IF DF ...V1
                                ;.IF NB .WCNT
                                ;                   MOV      .WCNT,%0
                                ;.ENDC
                                ;                   MOV      #1,-(6.)
                                ;                   MOV      .BUFF,-(6.)
                                ;                   MOV      .CHAN,-(6.)
                                ;                   EMT      ^O<200+.AREA>
                                ;.IFF
                                ;.IF NB .AREA
                                ;                   MOV      .AREA,%0
                                ;                   MOVB     #8.,1(0)
                                ;.ENDC
                                ;.IF NB .CHAN
                                ;                   MOVB     .CHAN,(0)
                                ;.ENDC
                                ;.IF NB .BLK
                                ;                   MOV      .BLK,2.(0)
                                ;.ENDC
                                ;.IF NB .BUFF
                                ;                   MOV      .BUFF,4.(0)
                                ;.ENDC
                                ;.IF NB .WCNT
                                ;                   MOV      .WCNT,6.(0)
                                ;.ENDC
                                ;                   MOV      #1,8.(0)
                                ;                   EMT      ^O375
                                ;.ENDC
                                ;.ENDM
          ..V2..                ;         ..V2..
                                ...V2=1
          .CSECT MAIN                     .CSECT MAIN
          .GLOBL SORT                     .GLOBL SORT
STACK:    .BLKW 100             STACK:    .BLKW 100
AREA:     .BLKW 10              AREA:     .BLKW 10
BUFR:     .BLKW 100             BUFR:     .BLKW 100
INBLK:    .BLKW 5               INBLK:    .BLKW 5
START:    MOV #STACK,SP         START:    MOV #STACK,SP
A:        MOV R1,-(SP)          A:        MOV R1,-(SP)
B:        .CALL SORT            B:;       .CALL SORT
                                          JSR PC,SORT
          .LOOKUP #INBLK,0      ;         .LOOKUP #INBLK,0
                                .IF DF ...V1
                                .IF NB 0
                                                    MOV      0,%0
                                .ENDC
                                                    EMT      ^O<20+#INBLK>
                                .IFF
                                .IF NB #INBLK
```

```
                                        MOV     #INBLK,%0
                                        MOVB    #1,1(0)
                         .ENDC
                         .IF NB 0
                                        MOVB    0,(0)
                         .ENDC
                         .IF NB
                                        MOV     ,2.(0)
                         .ENDC
                         .IF NB
                                        MOV     ,4.(0)
                         .IFF
                                        CLR     4.(0)
                         .ENDC
                                        EMT     ^O375
                         .ENDC
CLR R1          ;BLOCK NUMBER                CLR R1          ;BLOCK NUMBER
.READ #AREA,#0,#BUFR,#256.,R1        ;       .READ #AREA,#0,#BUFR,#256.,R1
                         .IF DF ...V1
                         .IF NB #256.
                                        MOV     #256.,%0
                         .ENDC
                                        MOV     #1,-(6.)
                                        MOV     #BUFR,-(6.)
                                        MOV     #0,-(6.)
                                        EMT     ^O<200+#AREA>
                         .IFF
                         .IF NB #AREA
                                        MOV     #AREA,%0
                                        MOVB    #8.,1(0)
                         .ENDC
                         .IF NB #0
                                        MOVB    #0,(0)
                         .ENDC
                         .IF NB R1
                                        MOV     R1,2.(0)
                         .ENDC
                         .IF NB #BUFR
                                        MOV     #BUFR,4.(0)
                         .ENDC
                         .IF NB #256.
                                        MOV     #256.,6.(0)
                         .ENDC
                                        MOV     #1,8.(0)
                                        EMT     ^O375
                         .ENDC
HALT                                         HALT
.END START                                   .END START
```

10.4 EXPAND ERROR MESSAGES

The following messages are caused by fatal errors detected by EXPAND. They print on the console terminal and cause EXPAND to restart:

Message	Explanation
?BAD SWITCH?	An unrecognized command string switch was specified.
?INPUT ERROR?	Hardware error in reading an input file.
?INSUFFICIENT CORE?	Not enough memory to store macro definitions.
?MISSING END IN MACRO?	End of input was encountered while storing a macro definition; probably missing an .ENDM.
?NO INPUT FILE?	There must be at least one input file.
?OUTPUT DEVICE FULL?	No room to continue writing output; try to compress the device with PIP.
?WRONG NUMBER OF OUTPUT FILES?	There must be exactly one output file.

The following errors are non-fatal but indicate that something is wrong in the input file(s). These errors appear in the output file as a line in the following form:

?*** ERROR *** message

After each run of EXPAND, the total number of non-fatal errors is printed on the console terminal.

Message	Explanation
BAD MACRO ARG	The macro argument is not formatted correctly.
LINE TOO LONG	A line has become longer than 132 characters.
MACRO ALREADY DEFINED	A macro was defined more than once.
MACRO(S) NOT FOUND	Macros listed in an .MCALL statement were not found in SYSMAC.8K (make sure SYSMAC.8K is present on system).
MISSING COMMA IN MACRO ARG	Found spaces or tabs within a macro argument when a comma was expected; try using brackets around the arguments, e.g., <arg with spaces>.

MISSING DOT	A macro name or argument name does not begin with a dot.
NAME DOESN'T MATCH	Optional name given in .ENDM directive does not match name given in corresponding .MACRO directive.
NESTED MACROS	A macro is being defined or invoked within another macro.
NO NAME	A macro definition has no name.
SYNTAX	A macro directive is not constructed correctly.
TOO MANY ARGS	A macro directive has more than 30 arguments.

CHAPTER 11

ASEMBL, THE 8K ASSEMBLER

ASEMBL is designed for use on an RT-11 system with minimum memory space (or larger systems where system table space is critical) and is a subset of the RT-11 MACRO assembler described in Chapter 5. ASEMBL has the same features as MACRO with the following exceptions:

1. MACRO directives (.MACRO, .MCALL, .ENDM, .IRP, etc.) are not recognized
2. DATE is not printed in listings
3. Wide line-printer output is not available
4. There is no lower-case mode
5. There is no enable/disable punch directive
6. There are no floating point directives
7. There are no local symbols or local symbol blocks
8. CREF is not available

Many of the macro features are supported by the EXPAND program (as described in Chapter 10).

11.1 CALLING AND USING ASEMBL

ASEMBL is loaded in response to the dot printed by the Keyboard Monitor with the RT-11 monitor R Command as follows:

```
R ASEMBL
```

followed by the RETURN key. ASEMBL responds with an asterisk (*) and waits for specification of the output and input files in the standard RT-11 format as follows:

```
*object,listing=source1,...,source6
```

where:

object	is a binary object file output by ASEMBL.
listing	is the assembly listing file containing the assembly listing and symbol table.
source1,...,source6	are the ASCII source files containing the ASEMBL source program(s). A maximum of six source files is allowed.

A null specification in any of the file fields signifies that the associated input or output file is not desired. ASEMBL file specifications follow the standard RT-11 convention (dev:filnam.ext). The default value for each file specification is noted below:

I/O File	Dev	Ext
object	DK	.OBJ
listing	device used for object output	.LST
source1,...,sourcen	device used for last source file specified or DK	.PAL

Type CTRL C to halt ASEMBL and return control to the monitor. To restart ASEMBL, type R ASEMBL or the REENTER command in response to the monitor's dot.

Table 11-1 lists the RT-11 macro directives which are not available in ASEMBL.

Table 11-1
Directives not Available in ASEMBL

Directive	Explanation
.MACRO .ENDM .MEXIT .MCALL	Macros cannot be defined in ASEMBL.
.NCHR	The number of characters in an argument cannot be obtained with a macro.
.NARG	The number of arguments in a macro cannot be obtained with a macro.
\ (backslash)	Symbols used as macro arguments cannot be passed as a numeric string.

(continued on next page)

Table 11-1 (Cont.)
Directives not Available in ASEMBL

Directive	Explanation
.ERROR	Messages cannot be flagged with a P error code output as part of the assembly listing. Comment lines can be used to replace .ERROR.
.IF IDN .IF DIF	Strings cannot be compared.
.IRP .IRPC	Indefinite repeat blocks cannot be created.
.NTYPE	A macro cannot be modified based on the addressing mode of an argument.
.PRINT	Messages cannot be output as part of the assembly listing. Comment lines can be used to replace .PRINT.
.REPT	A block of code cannot be duplicated a number of times in-line with other source code using a directive.
.LIST ME .NLIST ME .LIST MEB .NLIST MEB .LIST MD .NLIST MD .LIST MC .NLIST MC	These directives have no effect.
.LIST TTM .NLIST TTM	Terminal mode is standard and cannot be changed.
.ENABL LC .DSABL LC	All lower case ASCII input is converted to upper case.
.ENABL LSB .DSABL LSB	Local symbols and local symbol blocks are not available in ASEMBL.
.ENABLE PNC .DSABL PNC	Binary output is always enabled.
.ENABL FPT .DSABL FPT .FLT2 .FLT4	Floating point directives are not available.

Example:

This example uses the output produced by the EXPAND program as input to ASEMBL. The assembly listing follows.

```
.MAIN.  RT-11 MACRO VS02-10   PAGE 1

                        ; RT-11 MACRO EXPAND V02-02

        000001                  R1=%1
        000006                  SP=%6
        000007                  PC=%7
                        ;       .MACRO .CALL .SUBR
                        ;       JSR PC,.SUBR
                        ;       .ENDM
                        ;       .MCALL .LOOKUP,.READ,..V2..
                        ;.MACRO ..V2..
                        ;...V2=1
                        ;.ENDM
                        ;.MACRO .LOOKUP .AREA,.CHAN,.DEVBLK,.SPF
                        ;.IF DF ...V1
                        ;.IF NB .CHAN
                        ;               MOV     .CHAN,%0
                        ;.ENDC
                        ;               EMT     ^O<20+.AREA>
                        ;.IFF
                        ;.IF NB .AREA
                        ;               MOV     .AREA,%0
                        ;               MOVB    #1,1(0)
                        ;.ENDC
                        ;.IF NB .CHAN
                        ;               MOVB    .CHAN,(0)
                        ;.ENDC
                        ;.IF NB .DEVBLK
                        ;               MOV     .DEVBLK,2.(0)
                        ;.ENDC
                        ;.IF NB .SPF
                        ;               MOV     .SPF,4.(0)
                        ;.IFF
                        ;               CLR     4.(0)
                        ;.ENDC
                        ;               EMT     ^O375
                        ;.ENDC
                        ;.ENDM
                        ;.MACRO .READ   .AREA,.CHAN,.BUFF,.WCNT,.BLK
                        ;.IF DF ...V1
                        ;.IF NB .WCNT
                        ;               MOV     .WCNT,%0
                        ;.ENDC
                        ;               MOV     #1,-(6.)
                        ;               MOV     .BUFF,-(6.)
                        ;               MOV     .CHAN,-(6.)
                        ;               EMT     ^O<200+.AREA>
                        ;.IFF
                        ;.IF NB .AREA
                        ;               MOV     .AREA,%0
                        ;               MOVB    #8.,1(0)
                        ;.ENDC
                        ;.IF NB .CHAN
                        ;               MOVB    .CHAN,(0)
                        ;.ENDC
                        ;.IF NB .BLK
                        ;               MOV     .BLK,2.(0)
                        ;.ENDC
```

```
.MAIN.  RT-11 MACRO V802-10   PAGE 1+


58                     ;.IF NB .BUFF
59                     ;                   MOV      .BUFF,4.(0)
60                     ;.ENDC
61                     ;.IF NB .WCNT
62                     ;                   MOV      .WCNT,6.(0)
63                     ;.ENDC
64                     ;                   MOV      #1,8.(0)
65                     ;                   EMT      ^O375
66                     ;.ENDC
67                     ;.ENDM
68                     ;         ..V2..
69           000001    ...V2=1
70           000000'             .CSECT MAIN
71                               .GLOBL SORT
72 00000               STACK:    .BLKW 100
73 00200               AREA:     .BLKW 10
74 00220               BUFR:     .BLKW 100
75 00420               INBLK:    .BLKW 5
76 00432   012706      START:    MOV #STACK,SP
           000000'
77 00436   010146      A:        MOV R1,-(SP)
78 00440               B::       .CALL SORT
79 00440   004767                JSR PC,SORT
           000000G
80                     ;         .LOOKUP #INBLK,0
81                     .IF DF ...V1
82                     .IF NB 0
83                                         MOV      0,%0
84                     .ENDC
85                                         EMT      ^O<20+#INBLK>
86                     .IFF
87                     .IF NB #INBLK
88 00444   012700                          MOV      #INBLK,%0
           000420'
89 00450   112760                          MOVB     #1,1(0)
           000001
           000001
90                     .ENDC
91                     .IF NB 0
92 00456   116710                          MOVB     0,(0)
           000000'
93                     .ENDC
94                     .IF NB
95                                         MOV      ,2.(0)
96                     .ENDC
97                     .IF NB
98                                         MOV      ,4.(0)
99                     .IFF
100 0462   005060                          CLR      4.(0)
           000004
101                    .ENDC
102 0466   104375                          EMT      ^O375
103                    .ENDC
104 0470   005001                CLR R1              ;BLOCK NUMBER
105                    ;         .READ #AREA,#0,#BUFR,#256.,,R1
106                    .IF DF ...V1
107                    .IF NB #256.
```

```
.MAIN.  RT-11 MACRO V802-10   PAGE 1+


108                                     MOV     #256.,%0
109                     .ENDC
110                                     MOV     #1,-(6.)
111                                     MOV     #BUFR,-(6.)
112                                     MOV     #0,-(6.)
113                                     EMT     ^O<200+#AREA>
114                     .IFF
115                     .IF NB #AREA
116 0472 012700                         MOV     #AREA,%0
         000200'
117 0476 112760                         MOVB    #8.,1(0)
         000010
         000001
118                     .ENDC
119                     .IF NB #0
120 0504 112710                         MOVB    #0,(0)
         000000
121                     .ENDC
122                     .IF NB R1
123 0510 010160                         MOV     R1,2.(0)
         000002
124                     .ENDC
125                     .IF NB #BUFR
126 0514 012760                         MOV     #BUFR,4.(0)
         000220'
         000004
127                     .ENDC
128                     .IF NB #256.
129 0522 012760                         MOV     #256.,6.(0)
         000400
         000006
130                     .ENDC
131 0530 012760                         MOV     #1,8.(0)
         000001
         000010
132 0536 104375                          EMT     ^O375
133                     .ENDC
134 0540 000000                 HALT
135      000432'                .END START


.MAIN.  RT-11 MACRO V802-10   PAGE 1+
SYMBOL TABLE

A        000436R    002  AREA    000200R    002  B        000440R    002
BUFR     000220R    002  INBLK   000420R    002  PC      =%000007
R1      =%000001         SP     =%000006         SQRT    = ****** G
STACK    000000R    002  START   000432R    002  ...V2   = 000001

. ABS.   000000     000
         000000     001
MAIN     000542     002
ERRORS DETECTED: 0
FREE CORE: 19088. WORDS

.LP:=TEST.PAL
```

11.2 ASEMBL ERROR MESSAGES

The system error messages output for ASEMBL are abbreviated as follows:

Abbreviation	Explanation
?BSW?	The switch specified was not recognized by the program.
?CORE?	There are too many symbols in the program being assembled. Try dividing program into separately assembled subprograms.
?NIF?	No input file was specified and there must be at least one input file.
?ODF?	No room to continue writing output; try to compress device with PIP.
?TMO?	Too many output files were specified.

CHAPTER 12

BATCH

12.1 INTRODUCTION TO RT-11 BATCH

RT-11 BATCH is a complete job control language that allows RT-11 to operate unattended. RT-11 BATCH processing is ideally suited to frequently run production jobs, large and long running programs, and programs that require little or no interaction with the user. Using BATCH, the user can prepare his job on any RT-11 input device and leave it for the operator to start and run.

RT-11 BATCH provides the ability to:

- Execute an RT-11 BATCH stream from any legal RT-11 input device.
- Output a log file to any legal RT-11 output device (except magtape or cassette).
- Execute the BATCH stream either with the Single-Job Monitor or in the background with a Foreground/Background Monitor.
- Generate and support system independent BATCH language jobs.
- Execute RT-11 monitor commands from the BATCH stream.

RT-11 BATCH consists of two main sections: the BATCH compiler and the BATCH run-time handler. The BATCH compiler reads the batch input stream created by the user, translates it into a format suitable for the RT-11 BATCH run-time handler, and stores it in a file. The BATCH run-time handler takes the file generated by the compiler and executes it in conjunction with the RT-11 monitor. As each command in the batch stream is executed, the command, along with any terminal output generated by executing the command, is output to the BATCH log device.

12.1.1 Hardware Requirements to Run BATCH

RT-11 BATCH can be run on any Single-Job system configured with at least 12K words of memory. A minimum system of 16K words of memory is required to run BATCH in a Foreground/Background environment. A line printer, although optional, is highly desirable as the log device.

12.1.2 Software Requirements to Run BATCH

BATCH uses certain RT-11 system programs to perform its operations. For example, the $BASIC command runs BASIC.SAV. The user must ensure that the following RT-11 programs are on the system device, with exactly the following names, before running BATCH.

```
BASIC.SAV     (BASIC users only)
BA.SYS
BATCH.SAV
CREF.SAV      (MACRO users only)
FORLIB.OBJ    (FORTRAN users only)
FORTRA.SAV    (FORTRAN users only)
LINK.SAV
MACRO.SAV     (MACRO users only)
PIP.SAV
```

12.2 BATCH CONTROL STATEMENT FORMAT

Input to RT-11 BATCH is either a file generated using the RT-11 Editor and input from any legal RT-11 input device, or punched cards input from the card reader. In both cases, the input consists of BATCH control statements. A BATCH control statement consists of three fields, separated from one another with spaces: command fields, specification fields, and comment fields. The control statement has the form:

```
$command/switch    specification/switch    !comment
```

Each control statement requires a specific combination of command and specification fields and switches (see Section 12.4). Control statements may not be longer than 80 characters, excluding multiple spaces, tabs, and comments. A line continuation character (-) may be used to indicate that the control statement is continued on the next line (see Table 12-4). Even if the line continuation character is used, the maximum control statement length is still 80 characters.

The following example of a $FORTRAN command illustrates the various fields in a control statement.

```
$FORTRAN/LIST/RUN    PROGA/LIBRARY  PROGB/EXE    !RUN FORTRAN
        |                      |                      |
command/switches     spec fields/switches       comment field
```

12.2.1 Command Fields

The command field in a BATCH control statement indicates the operation to be performed. It consists of a command name and certain command field switches. The command field is indicated by a $ in the first character position and is terminated by a space, tab, blank, or carriage return.

12.2.1.1 Command Names

12.2.1.1 Command Names -- The command name must appear first in a BATCH control statement. All BATCH command names have a dollar sign ($) in the first position of the command, e.g., $JOB. No intervening spaces are allowed in the command name. BATCH recognizes only two forms of a command name: the full name and an abbreviation consisting of $ and the first three characters of the command name. For example, the $FORTRAN command may be entered as:

    $FORTRAN

      or

    $FOR

but cannot be entered as:

    $FORT

      or

    $FORTR

12.2.1.2 Command Field Switches -- Switches that appear in a command field are command qualifiers and their functions apply to the entire control statement. All switch names must begin with a slash (/) that immediately follows the command name. Table 12-1 describes the command field switches that are legal in BATCH and indicates the commands on which they can be used. Those switch characters that appear in braces are optional. These switches will be mentioned again in the sections pertaining to the commands with which they can be used.

All /NO switches are the defaults, except the /WAIT switch in the $MOUNT and $DISMOUNT commands and the /OBJECT switch in the $LINK command.

Table 12-1
Command Field Switches

Switch	Function
/BAN{NER}	Print header of job on the log file. This switch is allowed only on the $JOB command.
/NOBAN{NER}	Do not print a job header.
/CRE{F}	Produce a cross reference listing during compilation. This switch is allowed only on the $MACRO command.
/NOCRE{F}	Do not create a cross reference listing.
/DEL{ETE}	Delete input files after the operation is complete. This switch is allowed on the $COPY and $PRINT commands.
/NODEL{ETE}	Do not delete input files after operation is complete.

(continued on next page)

Table 12-1 (cont.)
Command Field Switches

Switch	Function
/DOL{LARS}	The data following this command may have a $ in the first character position of a line. This switch is allowed on the $CREATE, $DATA, $FORTRAN, and $MACRO commands. Reading of the data is terminated by one of the following BATCH commands: $JOB $SEQUENCE $EOD $EOJ or by a physical end-of-file on the BATCH input stream.
/NODOL{LARS}	Following data may not have a $ in the first character position; a $ in the first character position signifies a BATCH control command.
/LIB{RARY}	Include the default library in the link operation. This switch is allowed on the $LINK and $MACRO commands.
/NOLIB{RARY}	Do not include the default library in the link operation.
/LIS{T}	Produce a temporary listing file (see Section 12.2.5) on the listing device (LST:) or write data images on the log device (LOG:). This switch is allowed on the $BASIC, $CREATE, $DATA, $FORTRAN, $JOB, and $MACRO commands. When used on the $JOB command, /LIST sends data lines in the job stream to the log device (LOG:).
/NOLIS{T}	Do not produce a temporary listing file.
/MAP	Produce a temporary linkage map on the listing device (LST:). This switch is allowed on the $FORTRAN, $LINK, and $MACRO commands.
/NOMAP	Do not create a MAP file.
/OBJ{ECT}	Produce a temporary object file as output of compilation or assembly (see Section 12.2.5). This switch is allowed on the $FORTRAN, $LINK, and $MACRO commands. When used on $LINK, this switch means that temporary files are to be included in the link operation.
/NOOBJ{ECT}	Do not produce object file as output of compilation, or, on $LINK, do not include temporary files in the link operation.
/RT11	Set BATCH to operate in RT-11 mode (see Section 12.5). This switch is allowed only on the $JOB command.
/NORT11	Do not set BATCH to operate in RT-11 mode.

(continued on next page)

Table 12-1 (cont.)
Command Field Switches

Switch	Function
/RUN	Link (if necessary) and execute programs compiled since the last "link-and-go" operation or start of job. This switch is allowed on the $BASIC, $FORTRAN, $LINK, and $MACRO commands.
/NORUN	Do not execute or link and execute the program after performing the specified command.
/TIM{E}	Write the time of day to the log file when commands are executed. This switch is allowed only on the $JOB command.
/NOTIM{E}	Do not write time of day to log file.
/UNI{QUE}	Check for unique spelling of switches and keynames (see Section 12.4.13). This switch is allowed only on the $JOB command.
/NOUNI{QUE}	Do not check for unique spelling.
/WAI{T}	Pause to wait for operator action. This switch is allowed on the $DISMOUNT, $MESSAGE, and $MOUNT commands.
/NOWAI{T}	Do not pause for operator action.
/WRI{TE}	Indicate that the operator is to WRITE-ENABLE a specified device or volume. This switch is allowed only on the $MOUNT command.
/NOWRI{TE}	Indicate that no writes are allowed or that the specified volume is read-only; the operator is informed and must WRITE-LOCK the appropriate device.

12.2.2 Specification Fields

Specification fields immediately follow command fields in a BATCH control statement and are used to name the devices and files involved in the command. These fields are separated from the command field, and from each other, by blanks or spaces.

If a specification field contains more than one file to be used in the same operation, the + operator is used between these files. For example, to assemble files F1 and F2 to produce an object file F3 and a temporary listing file, type:

```
$MACRO/LIST F1+F2/SOURCE F3/OBJECT
```

If a command is to be repeated for more than one field specification, the "," operator is used between these files. For example, the following command assembles F1 to produce F2, a temporary listing file, and a map file F3. It then assembles F4 and F5 to produce F6 and a temporary listing file.

```
$MACRO/LIST F1/SOURCE F2/OBJECT F3/MAP,F4+F5/SOURCE-
 F6/OBJECT
```

Note that the command field switches apply to the entire line, but the specification field switches apply only to the field they follow.

Depending on the command being used, specification fields may contain a device specification, file specification, or an arbitrary ASCII string. Any of these three may be followed by an appropriate specification field switch (see Table 12-3).

12.2.2.1 Physical Device Names -- Each device in an RT-11 BATCH specification field is referenced by means of a standard 2- or 3-character device name. Table 2-2 in Chapter 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.

In addition to the permanent names shown in Table 2-2, devices can be assigned logical device names. A logical device 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. For example, DK: is normally assigned to the system device, but that name can be assigned to DECtape unit 0 with an RT-11 monitor ASSIGN command.

Certain logical names must be assigned prior to running any BATCH job since BATCH uses them as default devices. These devices are:

LOG:	BATCH log device (cannot be magtape or cassette)
LST:	default for listing files generated by BATCH stream

The following are not legal device names in RT-11; if used, the operator must assign them as logical names with the ASSIGN command. These names are included here because they can be used in BATCH streams written for other DIGITAL systems.

DF:	Fixed-head disk (RF).
LL:	Line printer with upper and lower case characters.
M7:	7-track magtape.
M9:	9-track magtape.
PS:	Public storage (DK: as assigned by RT-11).

Refer to Sections 2.7.2.4 and 12.7.1 for instructions on assigning logical names to devices.

12.2.2.2 File Specifications -- Files are referenced symbolically in a BATCH control statement by a name of up to six alphanumeric characters followed, optionally, by a period and an extension of three alphanumeric characters. The extension to a file name generally indicates the format of a file. In most cases, it is a good practice to conform to the standard file name extensions for RT-11 BATCH. If an extension is not specified for an output file, BATCH and most other RT-11 system programs assign appropriate default extensions. If an extension for an input file is not specified, the system searches for that file name with a default extension. Table 12-2 lists the standard extensions used in RT-11 BATCH.

Table 12-2
File Name Extensions

Extension	Meaning
.BAS	BASIC source file (BASIC input).
.BAT	BATCH command file.
.CTL	BATCH control file generated by the BATCH compiler.
.CTT	BATCH temporary file generated by the BATCH compiler.
.DAT	BASIC or FORTRAN data file.
.DIR	Directory listing file.
.FOR	FORTRAN IV source file (FORTRAN input).
.LST	Listing file.
.LOG	BATCH log file.
.MAC	MACRO or EXPAND source file (MACRO, EXPAND, SRCCOM input).
.MAP	Linkage map output from $LINK operation.
.OBJ	Object file, output from compilation or assembly.
.SOU	Temporary source file.
.SAV	$RUNable file or program image output from $LINK.

12.2.2.3 Wild Card Construction -- The wild card construction means that the file name or extension in certain BATCH control statements (i.e., $COPY, $CREATE, $DELETE, $DIRECTORY, $PRINT) may be replaced totally with an asterisk (*). The asterisk is used as a wild card to designate the entire file name or extension. See Chapter 4, Section 4.1.1, for a complete description of the wild card construction.

12.2.2.4 Specification Field Switches -- Specification field switches follow file specifications in a BATCH control statement and designate how the file will be used. These switches apply only to the field in which they appear. Switch names begin with a slash. The specification field switches legal in RT-11 BATCH are listed in Table 12-3. Optional characters in the switch names are shown in braces.

Table 12-3
Specification Field Switches

Switch	Function
/BAS{IC}	BASIC source file.
/EXE{CUTABLE}	Indicates the runnable program image file to be created as the result of a link operation.
/FOR{TRAN}	FORTRAN source file.
/INP{UT}	Input file; default if no switches are specified.
/LIB{RARY}	Library file to be included in link operation (prior to default library).
/LIS{T}	Listing file.
/LOG{ICAL}	Indicates that the device is a logical device name; used in $DISMOUNT and $MOUNT commands.
/MAC{RO}	MACRO or EXPAND source file.
/MAP	Linker map file.
/OBJ{ECT}	Object file (output of assembly or compilation).
/OUT{PUT}	Output file.
/PHY{SICAL}	Indicates physical device name.
/SOU{RCE}	Indicates source file.
/VID	Volume identification.

12.2.3 Comment Fields

Comment fields, used to document a BATCH stream, are identified by an exclamation point (!) appearing anywhere except the first character position in the control statement. Any character following the ! and preceding the carriage return/line feed combination is treated as a comment and is ignored by the BATCH processor. For example, the following command:

```
$RUN PIP     !DELETE FILES ON DK:
```

runs the RT-11 system program PIP; the comment is ignored.

Comments can also be included as separate comment lines by typing a $ in character position 1, followed immediately by the ! operator and the comment, e.g.,

```
$!DELETE FILES ON DK:
```

12.2.4 BATCH Character Set

The RT-11 BATCH character set is limited to the 64 upper-case characters (i.e., ASCII 40 through 137). The current ASCII set is assumed (i.e., character 137 is underscore and not left arrow, and character 136 is circumflex, not up-arrow). No control characters other than tab, carriage return, and line feed are supported by the BATCH job control language.

Table 12-4 details the way in which BATCH normally interprets certain characters. Character interpretations are different when RT-11 mode is used (see Section 12.5).

Table 12-4
Character Interpretation

Character	Interpretation
blank/space	Specification field delimiter. Separates arguments in control statements. Any string of consecutive spaces and tabs (except in quoted strings) is considered a blank and is equivalent to a single space.
!	Comment delimiter. All characters after the exclamation point are ignored by the input routine, up to the carriage return/line feed.
"	Used to pass a text string containing delimiting characters where the normal precedence rules would create the wrong action, e.g., to include a space in a volume identification (/VID).
$	BATCH control statement recognition character. A dollar sign ($) in the first character position of a BATCH input stream line indicates that the line is a control statement.
.	Delimiter for file extension (type).
-	Indicates line continuation if the character after the hyphen is one of the following: • a carriage return/line feed • any number of spaces followed by a carriage return/line feed • a comment delimiter(!) • spaces followed by a comment delimiter (!) If any other character follows the hyphen, the hyphen is assumed to be a minus sign indicating a negative value in a switch.
/	Start of a switch name. Must be followed immediately by an alphanumeric string.
0-9	Numeric string components.
:	Immediately follows a device name. Also can be used to separate a switch name from its value or to separate a switch value from its subvalue (: can be used interchangeably with = for this).
A-Z	Alphabetic string components.

(continued on next page)

Table 12-4 (cont.)
Character Interpretation

Character	Interpretation
=	Separates switch name from value.
\	Illegal character except when preceding a directive to the BATCH run-time handler from the operator. (To include \ in an RT-11 mode command, use \\ .)
+	Delimiter separating multiple files in a single specification field. Also used to indicate a positive value in switches.
,	The comma (,) is used to separate sets of arguments for which the command is to be repeated.
*	The * is used as a wild card in utility command file specifications.
CR/LF	Carriage return/line feed. Indicates end-of-line (or end of logical record) for records in the BATCH input stream.

12.2.5 Temporary Files

When field specifications are not included in a BATCH command line, BATCH sometimes generates temporary files. For example, a $FORTRAN command which is followed in the BATCH stream by the FORTRAN source program could be entered as:

```
$FORTRAN/RUN/OBJECT/LIST
    FORTRAN source program
$EOD
```

This command generates a temporary source file from the source statements that follow, a temporary object file, a temporary listing file, and a temporary save image file.

BATCH sends temporary files to the default device (DK:) or the listing device (LST:) according to their nature. If the device is file-structured, BATCH assigns file names and extensions as follows:

nnnmmm.LST	for temporary listing files (sent to LST:)
nnnmmm.MAP	for temporary map files (sent to LST:)
nnnppp.OBJ	for temporary object files (sent to DK:)
nnnppp.SAV	for temporary save image files (sent to DK:)
nnnppp.SOU	for temporary source files (sent to DK:)

where:

nnn is the last three digits of the sequence number assigned to the job by the $SEQUENCE command (see Section 12.4.22). Thus, a sequence number of 12345 produces a file name beginning 345. If no $SEQUENCE command is used, nnn is set to 000.

mmm — is the mth listing (or map) file since the BATCH run-time handler (BA.SYS) was loaded. The first such file, listing or map, is 000. Each time a new temporary file is generated, the file name is incremented by 1. Thus, the second listing file produced under job sequence number 12345 is 345001.LST, and the first map file produced is 345000.MAP.

ppp — is the pth object, save image, or source file in the current BATCH run. The first such file (object, save image, or source) is 000. Each time a new temporary file is generated, the file name is incremented by 1. These file names are reset to 000 every time that BATCH is run and after every $LINK, $MACRO, or $FORTRAN command that uses the temporaries.

12.3 General Rules and Conventions

The following general rules and conventions are associated with RT-11 BATCH processing.

1. A dollar sign ($) is always in the first character position of a command line.
2. Each job must have a $JOB and $EOJ command (or card).
3. Command and switch names can be spelled out entirely or the first three characters of the command and required characters of the switch can be specified.
4. Wild card construction (*) can be specified only for the utility commands: $COPY, $CREATE, $DELETE, $DIRECTORY, and $PRINT.
5. Comments can be included at the end of command lines or in a separate comment line. When comments are included in a command line, they must follow the command and be preceded by an exclamation mark.
6. Only 80 characters per control statement (card record) are allowed, excluding multiple spaces, tabs, and comments.
7. When file specifications are omitted from BATCH commands and data is supplied in the BATCH stream, the system creates a temporary file with a default name (see Section 12.2.5).
8. The RT-11 monitor type-ahead feature is restricted to BATCH handler directives (see Section 12.7.3) to be inserted into a BATCH program. No other terminal input (except to the foreground) can be entered while a BATCH stream is executing.

12.4 BATCH COMMANDS

BATCH commands are placed in the input stream to indicate to the system which functions to perform in the job. All BATCH commands have a dollar sign ($) in the first character position of the command, e.g., $JOB. Intervening spaces are not allowed in command names. The command name must always start in the first character position of the line (card column 1).

BATCH commands are presented in alphabetical order in this chapter for ease of reference. However, a user who is unfamiliar with BATCH may prefer to read the commands in a functional order as detailed in Table 12-5. The characters shown in braces are optional.

Table 12-5
BATCH Commands

Command	Section	Function
$SEQ{UENCE}	12.4.22	Assigns an arbitrary identification number to a job.
$JOB	12.4.13	Indicates the start of a job.
$EOJ	12.4.11	Indicates the end of a job.
$MOU{NT}	12.4.18	Signals the operator to mount a volume on a device and optionally assigns a logical device name.
$DIS{MOUNT}	12.4.9	Signals the operator to dismount a volume from a device and deassigns a logical device name.
$FOR{TRAN}	12.4.12	Compiles a FORTRAN source program.
$BAS{IC}	12.4.1	Compiles a BASIC source program.
$MAC{RO}	12.4.16	Assembles a MACRO source program.
$LIB{RARY}	12.4.14	Specifies libraries that are to be used in linkage operations.
$LIN{K}	12.4.15	Links modules for execution.
$RUN	12.4.21	Causes a program to execute.
$CAL{L}	12.4.2	Transfers control to another BATCH file, executes that BATCH file, and returns to the calling BATCH stream.
$CHA{IN}	12.4.3	Passes control to another BATCH file.
$DAT{A}	12.4.6	Indicates the start of data.
$EOD	12.4.10	Indicates the end of data.
$MES{SAGE}	12.4.17	Issues a message to the operator.

(continued on next page)

Table 12-5 (cont.)
BATCH Commands

Command	Section	Function
$COP{Y}	12.4.4	Copies files.
$CRE{ATE}	12.4.5	Creates new files from data included in BATCH stream.
$DEL{ETE}	12.4.7	Deletes files.
$DIR{ECTORY}	12.4.8	Provides a directory of the specified device.
$PRI{NT}	12.4.19	Prints files.
$RT1{1}	12.4.20	Specifies that the following lines are RT-11 mode commands.

$BASIC

12.4.1 $BASIC Command

The $BASIC command calls RT-11 Single-User BASIC to execute a BASIC source program. The $BASIC command has the following format:

```
$BASIC{/switch} {dev:filnam.ext/sw} {!comments}
```

where:

/switch — indicates the switches that can be appended to the $BASIC command. The switches are as follows:

- /RUN — indicates that the source program is to be executed.
- /NORUN — indicates that the program is to be compiled only; error messages are sent to the log file.
- /LIST — writes data images that are contained in the job stream to the log file (LOG:).
- /NOLIST — writes data images to the log file only if $JOB/LIST was specified.

dev:filnam.ext — indicates the name of the source file and the device on which it resides. If dev: is omitted, DK: is assumed. If ext is omitted, the extension .BAS is assumed. If this specification is omitted, the source statements must immediately follow the $BASIC command in the input stream.

The source program included after a $BASIC statement can be terminated either by a $EOD command or by any other BATCH command that starts with a $ in the first position.

/sw indicates the switches that may follow the source file name. Any file name with no switch appended is assumed to be the name of a source file. This switch can have one of the following values or can be omitted.

/BASIC	indicates that the file name specified is a BASIC source program.
/SOURCE	performs the same function as /BASIC.
/INPUT	performs the same function as /BASIC.

The $BASIC command can be followed by the source program, by legal BASIC commands such as RUN, and by data. The following two BATCH streams, for example, produce the same results.

```
$BASIC              $BASIC/RUN
10 INPUT A          10 INPUT A
20 PRINT A          20 PRINT A
30 END              30 END
RUN                 $DATA
123                 123
$EOD                $EOD
```

$CALL

12.4.2 $CALL Command

The $CALL command transfers control to another BATCH control file, temporarily suspending execution of the current control file. The CALLed file is executed until $EOJ is reached or until the job aborts; control then returns to the statement following the $CALL in the originating BATCH control file. Calls can be nested up to 31 levels. The log file for the CALLed BATCH file is included in the log file for the originating BATCH program.

The format of the $CALL command is :

```
$CALL dev:filnam.ext {!comments}
```

No switches are allowed in the $CALL command. $JOB command switches are saved across a $CALL but do not apply to the called BATCH file.

If .CTL is specified as the file extension, a precompiled BATCH control file is assumed. If no .ext is specified, .BAT is assumed, and the called BATCH stream is compiled before execution.

Please note that if the program called generates temporary files, those files may supersede currently existing temporary files if the two jobs have the same sequence number. For example, consider the following two BATCH streams:

```
$FOR/OBJ A                  $FOR/OBJ A
$FOR/OBJ B                  $CALL C
$LINK/RUN                   $FOR/OBJ B
                            $LINK/RUN
```

The called BATCH file (C.BAT) contains the following:

```
$JOB
$FOR/OBJ A1
$FOR/OBJ B1
$LINK/RUN
$EOJ
```

The temporary object files generated by C.BAT would change the behavior of the above two BATCH statement sequences since the first temporary file created by C.BAT (000000.OBJ) would supercede the temporary file produced by the first $FORTRAN command (000000.OBJ). This could be prevented by giving the BATCH job C.BAT a unique sequence number (see Section 12.4.22).

$CHAIN

12.4.3 $CHAIN Command

The $CHAIN command transfers control to a named BATCH control file but does not return to the input stream which executed the $CHAIN command. The format of the $CHAIN command is:

```
$CHAIN dev:filnam.ext {!comments}
```

No switches are allowed in the $CHAIN command. If .CTL is specified as the file extension, a precompiled BATCH control file is assumed. If no .ext is specified, .BAT is assumed, and the chained BATCH stream is compiled before execution.

An $EOJ command should always follow the $CHAIN command in the BATCH stream.

NOTE

The values of BATCH run-time variables remain constant across a $CALL, $CHAIN, or return from call. See Section 12.5.2.2 for a description of these variables.

The $CHAIN command is useful for transferring control to programs that need only be run once at the end of a BATCH stream. For example, the following BATCH program (PRINT.BAT) could be used to print and then delete all temporary listing files generated during the current BATCH job.

```
$JOB     !PRINT ALL LIST FILES
$PRINT/DELETE *.LST
$EOJ
```

PRINT.BAT can then be run with the $CHAIN command, e.g.,

```
$JOB
$MACRO/RUN/LIST FILE1,FILE2,FILE3
$CHAIN PRINT
$EOJ
```

$COPY

12.4.4 $COPY Command

The $COPY command copies files in image mode from one device to another. The wild card construction (see Section 12.2.2.3) can be used in the input and output file specifications. More than one input file can be concatenated to form one output file so long as the output specification does not contain a wild card. The $COPY command has the following format.

```
$COPY{/switch} dev:filnam.ext/OUTPUT dev:filnaml.ext{/INPUT}-
 {!comments}
```

where:

/switch — indicates switches that can be appended to the $COPY command.

- /DELETE — indicates that input files are to be deleted after the copy operation.
- /NODELETE — indicates that input files are not to be deleted after the copy operation.

dev: indicates the device containing the files to be copied for the input portion of the command or the device to which the files are to be copied for the output portion of the command. If dev: is not specified, DK: is assumed.

filnam indicates the name to be assigned to the output file; a wild card may be used instead of an explicit file name. Additional output files can be specified if they are separated from each other with commas.

ext indicates the file extension and must be specified. For input files, a wild card can be used instead of an explicit extension. For output files, wild cards can be used so long as no concatenation is specified.

/OUTPUT is appended to a file specification to indicate that it is for the output file.

filnaml specifies the name of the input file. Wild cards can be used instead of an explicit file name. Additional input files can be specified if they are separated from each other with commas. Files are copied to the output file in the order specified.

/INPUT is appended to the input file specifier(s). The system assumes input if no switch is used.

The following are examples of the $COPY command:

```
$COPY *.BAS/OUTPUT DT1:*.BAS
```

The above command copies all files with the extension .BAS from the DECtape on unit 1 to the default storage device DK.

```
$COPY FILE2.FOR/OUTPUT FILE0.FOR+FILE1.FOR
```

The above command merges the input files FILE0.FOR and FILE1.FOR to form one file called FILE2.FOR and stores FILE2.FOR on device DK.

```
$COPY FILE2.FOR+FILE3.FOR/OUTPUT FILE0.FOR+FILE1.FOR
```

The above command copies FILE0.FOR to DK: as FILE2.FOR and FILE1.FOR as FILE3.FOR.

```
$COPY *.*/OUT DT0:*.FOR,DT1:*.*/OUT DT0:*.*
```

The above command copies all files with the extension .FOR from DT0: to DK: and all files on DT0: to DT1:.

$CREATE

12.4.5 $CREATE Command

The $CREATE command generates a file consisting of data records from data that follows the $CREATE command in the input stream. An error occurs if the data does not immediately follow the $CREATE command. A $DATA command must not precede the data records.

The data associated with $CREATE can be followed by a $EOD command to signify the end of data or any other BATCH control statement can be used to indicate end of data and initiate a new function. The $CREATE command has the following format:

```
$CREATE{/switch} dev:filnam.ext {!comments}
```

where:

/switch — indicates switches that can be appended to the $CREATE command.

/DOLLARS	indicates that the data following this command may have a $ in the first character position of a line.
/NODOLLARS	indicates that a $ may not be in the first character position of a line.
/LIST	writes data image lines to log file.
/NOLIST	does not write data image lines to log file. If $JOB/LIST was specified, this switch is ignored.

dev: — device on which the file is to reside; DK: if not specified.

filnam — indicates the name to be assigned to the file. The file name must be specified.

ext — indicates the file extension. If extension is omitted, filnam must be followed by a period.

The following is an example of the $CREATE command:

```
$CREATE/LIST PROG.FOR
.    FORTRAN source file
.
.
$EOD
```

The data records following the $CREATE command become a new file (PROG.FOR) on the default device (DK:) and a listing is generated on logical device LOG:.

$DATA

12.4.6 $DATA Command

The $DATA command is used to include data records in the input stream. No file name is associated with the data; the data is transferred to the appropriate program as though input from the console terminal. For example, the $RUN command for a particular program can be followed by a $DATA command and the data records to be processed by the program. The data records must be valid data for the program that is to use them.

The $DATA command has the following format:

```
$DATA{/switch} {!comments}
```

Four switches can be used with the $DATA command.

/DOLLARS	indicates that the data following this command may have a $ in the first character position of a line.
/NODOLLARS	indicates that a $ may not be in the first character position of a line.
/LIST	writes data image lines to the log file.
/NOLIST	does not write data images to the log file. If $JOB/LIST was specified, this switch is ignored.

An $EOD command normally follows the last data record. However, any other BATCH command may also signal the end of the data so long as $DATA/DOLLARS is not specified (see Table 12-1).

The following example shows data being entered into a BASIC program (TEST1.BAS).

```
$BASIC/RUN TEST1.BAS
$DATA
25,75,125,146
180,210,520,874
$EOD
```

$DELETE

12.4.7 $DELETE Command

The $DELETE command is used to delete files from the specified device. This command has the form:

```
$DELETE dev:filnaml.ext{,dev:filnam2.ext,...,dev:filnamn.ext}-
 {!comments}
```

where filnaml through filnamn are the names of the files to be deleted. The file name and extension are required. Wild cards can be used in the file name and extension positions.

The following example deletes all files named TEST1 on the default device (DK:).

```
$DELETE TEST1.*
```

The following example deletes all files with .FOR extensions on DT1: then deletes all files with .MAC extensions on DK:.

```
$DELETE DT1:*.FOR,*.MAC
```

$DIRECTORY

12.4.8 $DIRECTORY Command

The $DIRECTORY command outputs a directory of the specified device to a listing file. If no listing file is specified, the listing goes to the BATCH log file. Wild cards can be used in the specification fields. This command has the form:

```
$DIRECTORY {dev:filnam.ext/LIST} {dev:filnam.ext}{/INPUT}-
 {!comments}
```

where:

/LIST indicates name of directory listing file.

/INPUT indicates input files to be included in directory (default).

The following are examples of the $DIRECTORY command:

```
$DIRECTORY
```

The above command outputs a directory of the device DK: to the BATCH log file.

```
$DIRECTORY FOR.DIR/LIST *.FOR
```

The above command creates a directory file (FOR.DIR) on the device DK:. The directory contains the names, lengths, and dates of creation of all FORTRAN source files on the device DK:.

$DISMOUNT

12.4.9 $DISMOUNT Command

The $DISMOUNT command removes the logical device name assigned by a $MOUNT command. When $DISMOUNT is encountered during the execution of a job, the entire $DISMOUNT command line is printed on the console terminal to inform the operator of the specific device to unload. This command has the form:

```
$DISMOUNT{/switch} ldn:{/LOGICAL} {!comments}
```

where:

/switch indicates the switches that can be appended to the $DISMOUNT command.

- /WAIT indicates that the job is to pause until the operator enters a response. If neither /WAIT nor /NOWAIT is specified, /WAIT is assumed. BATCH rings a bell at the terminal, prints the physical device name to be dismounted and a ?, and waits for a response. (Input to the BATCH handler can be entered, see Section 12.7.3.)
- /NOWAIT does not pause for operator response. BATCH prints the physical device name to be dismounted.

ldn: is the logical device name to be deassigned from the physical device.

/LOGICAL identifies the device specification as a logical device name.

The following example instructs the operator to dismount the physical device with the logical device name OUT and removes the logical assignment of device OUT. In this case, OUT is DT0. The operator dismounts DT0 and types a carriage return.

```
$DISMOUNT/WAIT OUT:/LOGICAL
DT0?
```

$EOD

12.4.10 $EOD Command

The $EOD command indicates the end of data record or the end of a source program in the job stream. The format of this command is:

```
$EOD {!comments}
```

The $EOD command can signal the end of data associated with any of the following commands:

```
$BASIC
$CREATE
$DATA
$FORTRAN
$MACRO
```

The $EOD command in the following example indicates the end of a source program that is to be compiled, linked, and executed.

```
$FORTRAN/RUN
.    source program
.
.
$EOD
```

$EOJ

12.4.11 $EOJ Command

The $EOJ command indicates the end of a job. This command must be the last statement in every BATCH job. The command has the following format:

```
$EOJ {!comments}
```

If a $JOB command, a $SEQUENCE command, or a physical end-of-file is encountered in the input stream before $EOJ, the error message NO $EOJ appears in the log file.

$FORTRAN

12.4.12 $FORTRAN Command

The $FORTRAN command calls the FORTRAN compiler to compile a source program. Optionally, this command can provide printed listings or list files and may produce a linkage map in the listing. The $FORTRAN command has the following format:

```
$FORTRAN{/switch} {dev:filnaml.ext/sw} {dev:filnam2.ext/OBJECT}-
 {dev:filnam3.ext/LIST} {dev:filnam4.ext/EXECUTE}-
 {dev:filnam5.ext/MAP} {dev:filnam6.ext/LIBRARY} {!comments}
```

where:

/switch — indicates the switches that can be appended to the $FORTRAN command. The switches are as follows:

- /RUN — indicates that the source program is to be compiled, linked with the default library (initially FORLIB.OBJ, may be reset with the $LIBRARY command), and executed.

/NORUN	indicates that the program is to be compiled only.
/OBJECT	indicates that a temporary object file is to be produced.
/NOOBJECT	indicates that a temporary object file is not to be produced.
/LIST	indicates that a list file is to be produced on the listing device (LST:).
/NOLIST	indicates that a list file is not to be produced.
/MAP	produces a linkage map on the listing device (LST:).
/NOMAP	does not create MAP file.
/DOLLARS	indicates that the data following this command may have a $ in the first character position of a line.
/NODOLLARS	indicates that a $ may not be in the first character position of a line.

dev:filnaml.ext

indicates the device, file name, and extension of the FORTRAN source file. If filnaml is not specified, the $FORTRAN source statements must immediately follow the $FORTRAN command in the input stream; BATCH generates a temporary source file that is deleted after it is compiled (see Section 12.2.5).

The source program included after a $FORTRAN statement can be terminated either by a $EOD command or by any other BATCH command so long as $FORTRAN/DOLLARS is not specified (see Table 12-1). A BATCH command is one that starts with a $ in the first position.

/sw can have one of the following values or can be omitted:

/FORTRAN	indicates that the file name specified is a FORTRAN source program. Any file name with no switch appended is assumed to be the name of a source file.
/SOURCE	performs the same function as /FORTRAN.
/INPUT	performs the same function as /FORTRAN.

dev:filnam2.ext/OBJECT

indicates the device, file name, and extension of the object file produced by compilation. The object file remains on the specified device after the job finishes. The object file specification, if included, must be followed by the /OBJECT switch.

If the object file specification is omitted but $FORTRAN/OBJECT is specified, a temporary object file is created, included in any $LINK operations that follow it in the job, and deleted after the link operation.

dev:filnam3.ext/LIST

indicates the name to be assigned to the list file created by the compiler. The list file is not printed automatically if LST: is assigned to a file-structured device, but it can be listed using the $PRINT command. The list file specification must be followed by the /LIST switch.

dev:filnam4.ext/EXECUTE
: indicates the name to be assigned to a save image file. The save image file specification must be followed by the /EXECUTE switch. If this field is not included, BATCH generates a temporary save image file (see Section 12.2.5) and then deletes the temporary file.

dev:filnam5.ext/MAP
: indicates the name to be assigned to the linkage map file created by the Linker. The map specification must be followed by the /MAP switch.

dev:filnam6.ext/LIBRARY
: indicates that the specified file is to be included in the link procedure as a library before FORLIB.OBJ. The file must be a library file (produced by RT-11 LIBR). The library specification must be followed by the /LIBRARY switch.

The following are examples of $FORTRAN commands:

```
$FORTRAN/RUN PROGA.FOR
```

The above command calls FORTRAN to compile a source program named PROGA.FOR. The program is compiled and executed.

```
$FORTRAN/NOOBJ/LIST
.   source program
.
.
$EOD
```

The above command sequence compiles the FORTRAN program but does not produce an object file. A temporary listing file is created on LST:.

$JOB

12.4.13 $JOB Command

The $JOB command indicates the beginning of a job. Each job must have its own $JOB command. This command has the following format:

```
$JOB{/switch}{/switch2}{/switchn} {!comments}
```

The switches allowed in the $JOB command are:

/BANNER	print header (a repetition of the $JOB command) on the log file.
/NOBANNER	do not print job header.
/LIST	write data image lines that are contained in the job stream to the log file.
/NOLIST	write data image lines to the log file only when a /LIST switch exists on a $BASIC, $CREATE, or $DATA command that has data lines following it.
/RT11	if no $ appears in column 1 when one is expected, assume that the line or card is an RT-11 mode command (see Section 12.5).
/NORT11	do not process RT-11 mode commands.
/TIME	write the time of day to the log file when command lines are executed (see NOTE on following page).
/NOTIME	do not write time of day.
/UNIQUE	check for unique spelling of switches and keynames. When this switch is used, commands and switches may be abbreviated to the least number of characters that still make their names unique. For example, the /DOLLARS switch can be abbreviated to /DO since no other switches begin with the characters DO.
/NOUNIQUE	check only for normal switch and keyname spellings.

Each job must be ended with a $EOJ command if it is to be run. If an input stream consists of more than one job, BATCH automatically terminates one job when the $JOB command for the next job is encountered. A job terminated with another $JOB command will never be run; an error message (NO $EOJ) will appear in the log.

The following $JOB command specifies that the time of day be written to the log file before each BATCH command beginning with a $ is executed and that unique abbreviations of BATCH commands and switches be accepted.

```
$JOB/TIME/UNIQUE
```

NOTE

If the /TIME switch is used on the $JOB command, the $DATA command cannot be used. For example, this job will not run properly:

```
$JOB/TIME
$RUN PROG
$DATA
123
$EOD
$EOJ
```

The /TIME switch uses the KMON TIME command to print the current time on the log for each BATCH command, including $DATA, causing an abort of the program that was to use the data. To avoid the problem, use RT-11 mode:

```
$JOB/TIME
$RT11
.R PROG
*123
$EOJ
```

This page intentionally blank.

$LIBRARY

12.4.14 $LIBRARY Command

The $LIBRARY command allows the user to specify a list of library files that will be included in FORTRAN links or with other linkage operations that specify the /LIBRARY switch. By default, the list of libraries contains only FORLIB.OBJ, the RT-11 FORTRAN library. This command has the form:

```
$LIBRARY mylib {!comments}

or

$LIBRARY mylib+FORLIB {!comments}
```

where:

FORLIB is the RT-11 FORTRAN library and mylib is a user library. Libraries are linked in order of their appearance in the $LIBRARY command.

The following example shows two user libraries (LIB1.OBJ and LIB2.OBJ) to be included in FORTRAN links before FORLIB.OBJ.

```
$LIBRARY LIB1.OBJ+LIB2.OBJ+FORLIB.OBJ
```

$LINK

12.4.15 $LINK Command

The $LINK command is used to produce save image files from object files. This command links the specified files (if any) with all temporary object files created since the last link or "link-and-go" operation (if any).

Temporary object files are those created as a result of a $FORTRAN or $MACRO command in which object files were neither specifically named by using the /OBJECT switch nor suppressed by using the /NOOBJECT switch. Permanent object files are created by using the /OBJECT switch on a $FORTRAN or $MACRO file descriptor.

Files are linked in the following order:

1. First, temporary files are linked in the order in which they were compiled.

2. Then, permanent files are linked in the order in which they are specified in the $LINK command.

3. If a library is specified in the $LINK command, it is linked next, providing that unresolved references remain.

4. If $LINK/LIBRARY is specified, the default library list is searched and linked.

The format for this command is:

```
$LINK{/switch} {filnaml.ext/OBJECT} {filnam2.ext/LIBRARY}-
 {filnam3.ext/MAP} {filnam4.ext/EXECUTE} {!comments}
```

where:

/switch — indicates the switches that can be appended to the $LINK command. The switches are as follows:

/LIBRARY	indicates that the FORTRAN library (FORLIB.OBJ) and any default libraries specified in the $LIBRARY command are to be included in this $LINK operation. This switch is normally used when the files being linked do not include any temporary FORTRAN object files or when $FORTRAN was specified without the /RUN or /MAP switch but the default library list is to be searched for unresolved references.
/NOLIBRARY	Indicates that the default libraries are not to be included.
/MAP	produces a temporary load map on the listing device (LST:).
/NOMAP	indicates that a map file is not to be produced.
/OBJECT	indicates that temporary object files are to be included in the link. If neither /OBJECT nor /NOOBJECT is specified, $LINK/OBJECT is assumed.
/NOOBJECT	indicates that temporary files are not to be included in the link.
/RUN	indicates that the save image files associated with this $LINK command are to be executed when the link is complete.
/NORUN	indicates that program linking only is to occur.

filnam1.ext/OBJECT
: indicates the name of the object file to be linked. If /OBJECT is not specified, it is assumed as the default.

filnam2.ext/LIBRARY
: indicates that the specified file is to be included in the link procedure as a library. The file specified must be a library file (produced by RT-11 LIBR).

filnam3.ext/MAP
: indicates the load map file to be created as a result of the $LINK command.

filnam4.ext/EXECUTE
: indicates the save image file to be created as a result of the $LINK command.

The following are examples of the $LINK command:

```
$LINK/RUN
```

The above command links all temporary object files created since the last $LINK command or the last $FORTRAN/OBJ or $MACRO/OBJ command.

```
$LINK/MAP PROG1.OBJ+PROG2.OBJ PROGA.SAV/EXE
```

The above command links the temporary files and the object files PROG1.OBJ and PROG2.OBJ to form a save image file named PROGA.SAV. It also creates and outputs a temporary map file.

$MACRO

12.4.16 $MACRO Command

The $MACRO command calls the MACRO assembler to assemble a source program and, optionally, to provide printed listings or list files. MACRO listing directives, if any, must be specified in the source program to enable their use, as they cannot be entered at BATCH command level.

The $MACRO command has the following format:

```
$MACRO{/switch} {filnam1.ext/sw} {filnam2.ext/OBJECT}-
 {filnam3.ext/LIST} {filnam4.ext/MAP} {filnam5.ext/LIBRARY}-
 {filnam6.ext/EXECUTE} {!comments}
```

where:

/switch — indicates the switches that can be appended to the $MACRO command. The switches are as follows:

/RUN	indicates that the source program is to be assembled, linked, and run.
/NORUN	indicates that the source program is to be assembled only.
/OBJECT	indicates that a temporary object file is to be produced.
/NOOBJECT	indicates that a temporary object file is not to be produced.
/LIST	indicates that a listing file is to be produced on the listing device (LST:).
/NOLIST	indicates that a list file is not to be produced.
/CREF	specifies that a cross reference listing is to be produced during assembly.
/NOCREF	indicates that a cross reference listing is not to be produced during assembly.
/MAP	produces a linkage map as part of the listing file on LST:.
/NOMAP	does not create MAP file.
/DOLLARS	indicates that the data following this command may have a $ in the first character position of a line.
/NODOLLARS	indicates that a $ may not be in the first character position of a line.
/LIBRARY	indicates that the default library is to be included in the link operation.
/NOLIBRARY	indicates that the default library is not to be included in the link operation.

filnaml.ext

indicates the name of the source file in the format dev:filnam.ext. If filnaml is not specified, the $MACRO source statements must immediately follow the $MACRO command in the input stream.

The source program included after a $MACRO statement can be terminated either by a $EOD command or by any other BATCH command so long as $MACRO/DOLLARS is not specified. A BATCH command is one that starts with a $ in the first position.

/sw — can have one of the following values or can be omitted:

/MACRO	indicates that the file name specified is a MACRO source program. Any file name with no switch appended is assumed to be the name of a source file.
/SOURCE	performs the same function as /MACRO.
/INPUT	performs the same function as /MACRO.

filnam2.ext/OBJECT

indicates the name (in the format dev:filnam.ext) to be assigned to the object file produced by compilation.
The object file remains on the specified device after the job finishes. The object file specification, if included, must be followed by the /OBJECT switch.

If the object file specification is omitted but $MACRO/OBJECT is specified, a temporary object file is created, included in any $LINK operations that follow the $MACRO command in the job, and deleted after the link operation (see Section 12.2.5).

filnam3.ext/LIST
: indicates the name to be assigned to the list file created by the assembler. The list file is not printed automatically if LST: is assigned to a file-structured device, but can be listed using the $PRINT command. The list file specification must be followed by the /LIST switch.

filnam4.ext/MAP
: indicates the file to which the storage map is to be output.

filnam5.ext/LIBRARY
: indicates that the specified file is to be included in the link procedure as a library. The library file specification must be followed by the /LIBRARY switch.

filenam6.ext/EXECUTE
: indicates the name to be assigned to a save image file. The save image file specification must be followed by the /EXECUTE switch. If this field is not included, BATCH generates a temporary save image file (see Section 12.2.5), runs it, and then deletes the temporary file.

The following $MACRO command assembles a program named PROG0.MAC and creates a temporary object file and a temporary listing file.

```
$MACRO/LIST/OBJECT PROG0.MAC
```

$MESSAGE

12.4.17 $MESSAGE Command

The $MESSAGE command is used to issue a message to the operator at the console terminal. It provides a means for the job to communicate with the operator. The $MESSAGE command has the form:

```
$MESSAGE{/switch} message {!comments}
```

where:

/switch — indicates the switches that can be appended to the $MESSAGE command. These switches are:

- /WAIT — indicates that the job is to pause until the operator types a carriage return to continue or enters commands to the BATCH handler followed by a carriage return (see Section 12.7.3).
- /NOWAIT — do not pause for operator response.

message — is a string of characters that must fit on one console line. The message is printed on the console.

For example, if the following message is included in the input stream:

```
$MESSAGE/WAIT MOUNT SCRATCH TAPE ON MT0
```

The message:

```
MOUNT SCRATCH TAPE ON MT0
?
```

appears on the console terminal and a bell sounds. The operator mounts the tape and types carriage return to allow further processing of the job. (See Section 12.7.3 for operator interaction with BATCH.)

$MOUNT

12.4.18 $MOUNT Command

The $MOUNT command assigns a logical device name and other characteristics to a physical device. When $MOUNT is encountered during the execution of a job, the entire $MOUNT command line is printed on the console terminal to notify the operator of the volume to be used.

The $MOUNT command has the form:

$MOUNT{/switch} dev:{/PHYSICAL}{/VID=x} {ldn:/LOGICAL} {!comments}

where:

/switch — indicates the switches that can be appended to the $MOUNT command. The switches are:

- /WAIT — indicates that the job is to pause until the operator enters a response. If neither /WAIT nor /NOWAIT is specified,

/WAIT is assumed. BATCH rings a bell, prints a ?, and waits for a response. (The response can be input to the BATCH handler; see Section 12.7.3.

/NOWAIT does not pause for operator response.

/WRITE tells the operator to WRITE-ENABLE the volume.

/NOWRITE tells the operator to WRITE-PROTECT the volume.

dev is required and specifies the physical device name and an optional unit number followed by a colon, e.g., DT1:. If dev is specified without a unit number, the operator can enter one in response to the ? printed by the $MOUNT command. If the operator is to supply a unit number, do not use the /NOWAIT switch because it will assume unit 0.

/PHYSICAL identifies the device specification as a physical unit specification. If neither /PHYSICAL nor /LOGICAL is specified, /PHYSICAL is assumed.

/VID=x
/VID="x" provides volume identification. The volume identification is the name physically attached to the volume. It is included to help the operator locate the volume. This switch may appear only on the physical device file specification. If x contains spaces, it must be input as "x".

ldn:/LOGICAL is required to identify the logical device name, if any, to be assigned to the device. The logical device name specification must be followed by the /LOGICAL switch.

The following are examples of the $MOUNT command:

```
$MOUNT/WAIT/WRITE DT:/VID=BAT01 2:/LOGICAL
```

This command instructs the operator to select a DECtape unit and mount DECtape volume BAT01 on that unit, WRITE-ENABLED. It informs the operator by printing:

```
$MOUNT/WAIT/WRITE DT:/VID=BAT01 2:/LOGICAL
?1
```

The operator selects a unit, mounts DECtape volume BAT01, WRITE-ENABLED, and responds to the ? by typing the unit number (e.g., 1) followed by a carriage return. BATCH assigns logical device name 2 to the physical device (e.g., DT1:) and proceeds.

If no unit number response is necessary, e.g.,

```
$MOUNT/WAIT/WRITE DT1: 2:/LOGICAL
?
```

the operator responds with a carriage return after mounting the DECtape and WRITE-ENABLING the device.

$PRINT

12.4.19 $PRINT Command

The $PRINT command is used to print the contents of the specified files on the listing device (LST:). This command has the form:

```
$PRINT{/switch} dev:filnam.ext{/INPUT}{,dev:filnam2.ext,...,dev:-
 filnamn.ext} {!comments}
```

where:

/switch — indicates the switches that can be appended to the $PRINT command. The switches are:

- /DELETE — indicates that input files are to be deleted after printing.
- /NODELETE — indicates that input files are not to be deleted after printing.

dev: — is the device containing the files to be printed; if dev: is not specified, DK: is assumed.

filnaml.ext-
filnamn.ext — indicate names of the files to be printed. Wild cards can be used for the file name or extension.

/INPUT — indicates that the file is an input file; /INPUT is assumed if it is not entered.

The following command prints a listing of files with extension .MAC that are stored on default device DK:.

```
$PRINT *.MAC
```

The following example creates listing files for the programs A and B, prints the listing files, and then deletes them.

```
$MACRO A.MAC A/LIST
$MACRO B.MAC B/LIST
$PRINT/DELETE A.LST,B.LST
```

$RT11

12.4.20 $RT11 Command

The $RT11 command allows the BATCH job to communicate directly with the RT-11 system. This command puts BATCH in RT-11 mode, i.e., until a line beginning with $ is encountered, all data images are interpreted as commands to the RT-11 monitor, RT-11 system programs, or to the BATCH run-time system. The $RT11 command has the form:

```
$RT11 {!comments}
```

See Section 12.5 for a complete description of the RT-11 mode.

$RUN

12.4.21 $RUN Command

The $RUN command requests execution of a program for which a save image file (.SAV) was previously created. It can also be used to run RT-11 system programs.

The $RUN command has the form:

```
$RUN dev:filnam.ext {!comments}
```

where filnam.ext is the name of the program (system or user) to be executed. If no extension is specified, .SAV is assumed.

For example, the user can run PIP to print a directory listing.

```
$RUN PIP
$DATA
LP:=DK:/L
$EOD
```

$SEQUENCE

12.4.22 $SEQUENCE Command

The $SEQUENCE command is an optional command. If used, it must immediately precede a $JOB command. The $SEQUENCE command assigns a job an arbitrary identification number. The last three characters of a sequence number are assigned by BATCH as the first three characters of a temporary listing or object file (see Section 12.2.5). If a sequence number is less than three characters long, it is padded with zeroes on the left.

The form of this command is:

$SEQUENCE id {!comments}

where id is an unsigned decimal number indicating the identification number of a job.

The following are examples of the $SEQUENCE command:

```
$SEQUENCE 3      !SEQUENCE NUMBER IS 003
$JOB

$SEQUENCE 100    !SEQUENCE NUMBER IS 100
$JOB
```

12.4.23 Example BATCH Stream

The following example BATCH stream creates a MACRO program, assembles and links that program, and runs the save image file. It then deletes the object, save image, and source files created and prints a directory of DK: showing the files created by the BATCH stream.

```
$JOB
$MESSAGE          THIS IS AN EXAMPLE BATCH STREAM
$MESSAGE          NOW CREATE A MACRO PROGRAM
$CREATE/LIST      EXAMPL.MAC
.TITLE  EXAMPL FOR BATCH
        .MCALL  .REGDEF,..V2..,.PRINT,.EXIT
        .REGDEF
        ..V2..
START:  .PRINT  #MESSAG
        .EXIT
MESSAGE: .ASCIZ          /EXAMPLE MACRO PROGRAM FOR BATCH/
        .END    START
```

```
$EOD
$MACRO  EXAMPL EXAMPL/OBJECT EXAMPL/LIST          !ASSEMBLE
$LINK   EXAMPL EXAMPL/EXECUTE   !AND LINK
$PRINT/DELETE EXAMPL.LST
$MESSAGE        RUN THE MACRO PROGRAM
$RUN    EXAMPL  !AND EXECUTE
$DELETE EXAMPL.OBJ+EXAMPL.SAV+EXAMPL.MAC
$MESSAGE        PRINT A DIRECTORY
$DIRECTORY      DK:EXAMPL.*
$MESSAGE        END OF THE EXAMPLE BATCH STREAM
$EOJ
```

To run this batch stream, the user types the following at the console. Messages are printed by BATCH.

```
.LOAD BA,LP
.ASSIGN LP:LOG
.ASSIGN LP:LST
.R BATCH
*EXAMPL
 THIS IS AN EXAMPLE BATCH STREAM
 NOW CREATE A MACRO PROGRAM
 RUN THE MACRO PROGRAM
 PRINT A DIRECTORY
 END OF THE EXAMPLE BATCH STREAM

END BATCH
```

The above example BATCH stream produces the following log file on the line printer.

```
$JOB

$MESSAGE        THIS IS AN EXAMPLE BATCH STREAM

$MESSAGE        NOW CREATE A MACRO PROGRAM

$CREATE/LIST    EXAMPL.MAC

.TITLE  EXAMPL FOR BATCH
        .MCALL  .REGDEF,..V2..,.PRINT,.EXIT
        .REGDEF
        ..V2..
START:  .PRINT  #MESSAG
        .EXIT
MESSAGE: .ASCIZ         /EXAMPLE MACRO PROGRAM FOR BATCH/
        .END    START
Ø

$EOD

$MACRO  EXAMPL EXAMPL/OBJECT EXAMPL/LIST          !ASSEMBLE

*ERRORS DETECTED: Ø
FREE CORE: 14764. WORDS

*
$LINK   EXAMPL EXAMPL/EXECUTE   !AND LINK

$PRINT/DELETE EXAMPL.LST

$MESSAGE        RUN THE MACRO PROGRAM

$RUN    EXAMPL  !AND EXECUTE
```

```
EXAMPLE MACRO PROGRAM FOR BATCH

$DELETE EXAMPL.OBJ+EXAMPL.SAV+EXAMPL.MAC


$MESSAGE        PRINT A DIRECTORY

$DIRECTORY      DK:EXAMPL.*

 9-MAY-75
EXAMPL.BAT    2 28-APR-75
EXAMPL.CTL    3  9-MAY-75
2 FILES, 5 BLOCKS
1436 FREE BLOCKS

$MESSAGE        END OF THE EXAMPLE BATCH STREAM

$EOJ
```

12.5 RT-11 MODE

RT-11 mode provides the capability to enter commands to the RT-11 monitor or to system programs and to create BATCH programs. RT-11 mode may be entered with either the $JOB/RT11 command or the $RT11 command. If entered with the $JOB/RT11 command, RT11 mode remains in effect until the next $JOB command is encountered in the BATCH stream. If entered with the $RT11 command, RT-11 mode is in effect until a $ is encountered in the first position of the command line.

The characters ., $, *, and tab or space appearing in the first position of a line (or card column 1) are interpreted as control characters and indicate the following:

. command to the RT-11 monitor, e.g.,

```
.R PIP
```

* data line; any line not intended to go to the RT-11 monitor or to the BATCH run-time handler, e.g., a command to the RT-11 PIP program:

```
*FILE1.DAT/D
```

NOTE

The * is not passed as data to the program. Comment lines (!) cannot appear on data lines as they would be considered as data.

$ BATCH command. Causes exit from RT-11 mode if RT-11 mode was entered with the $RT11 command. For example:

```
$RT11    !ENTER RT-11 MODE
.R PIP
*DT0:FILE1.DAT/D
$DIRECTORY DT0: !LEAVE RT-11 MODE
```

space/tab separator to indicate line directed to BATCH run-time handler. This separator is indicated by a → in the following descriptions.

12.5.1 Running RT-11 System Programs

The most common use of RT-11 mode is to send commands to the RT-11 monitor and to system programs. For example, the following commands can be inserted in the BATCH stream to run PIP and save backup copies of files on DECtape.

```
$RT11
.R PIP
*DT1:*.*=*.FOR/X
```

The user must anticipate and include in the BATCH input stream responses that are required by the called program, e.g., the Y response to PIP's ARE YOU SURE? query.

BATCH standard commands cannot be mixed with RT-11 mode data lines (i.e., lines beginning with an asterisk). For example, the proper way to do a $MOUNT within a sequence of RT-11 mode data commands is:

```
$JOB/RT11
.R MACRO
*A1=A1
*A2=A2
$MOUNT DT0:/PHYSICAL
.R MACRO
*B1=DT:B1
*B2=DT:B2
```

12.5.2 Creating RT-11 Mode BATCH Programs

RT-11 mode may be used by advanced system programmers to create BATCH programs. These BATCH programs consist of standard RT-11 mode commands (monitor commands, data lines for input to system programs, etc.) plus special RT-11 mode commands. These special commands are interpreted by the BATCH run-time handler to allow dynamic calculations and conditional execution of the RT-11 mode standard commands. The following facilities allow the user to create BATCH programs and to dynamically control the execution of these programs at run-time.

A. Labels
B. Variable modification
   1) equating a variable to a constant or character (LET statement)
   2) incrementing the value of a variable by 1
   3) reading a value into a variable
   4) conditional transfers on comparison of variable values with numeric or character values (IF and GOTO statements)
C. Commands to control terminal I/O
D. Other Control Characters
E. Comments

12.5.2.1 Labels -- Labels in RT-11 mode are user-defined symbols that provide a symbolic means of referring to a specific location within a BATCH program. If present, a label must begin in the first character position, must be unique within the first six characters, and must terminate with a colon (:) and a carriage return/line feed combination.

12.5.2.2 Variables -- A variable in RT-11 mode is a symbol representing a value that can change during program execution. The 26 variables permitted in a BATCH program have the names A-Z; each variable requires one byte of physical storage. The user may assign values to variables in a LET statement. These values may then be tested by an IF statement to control the direction of program execution.

Variables may be assigned values with a LET statement of the following form:

```
→|LET x="c
```

where x is a variable name A-Z and "c indicates the ASCII value of a character. For example:

```
→|LET A="0
```

indicates that the value of variable A is equal to the 7-bit ASCII value of the character 0 (60).

The LET statement can also specify an octal value in the form:

```
→|LET A=n
```

where n is an 8-bit signed octal value in the range 0 to 377; positive numbers range from 0 to 177 and negative numbers from 200 to 377 (-200 to -1).

Variables may be used to introduce control characters, such as ALTMODE, into a BATCH stream. For example, wherever 'A' appears in the following BATCH stream, BATCH substitutes the contents of variable A (the code for an ALTMODE):

```
$JOB/RT11
        LET A=33
        !A IS AN ALTMODE
.R EDIT
*EBFILE.MAC'A''A'
        !EDIT FILE TO CHANGE THE VERSION NUMBER TO 2
*GVERSION='A'DI2'A''A'
*EX'A''A'
$EOJ
```

The value of a variable can be incremented by 1 by placing a percent sign (%) before the variable. For example:

```
→|%A
```

indicates that the unsigned contents of variable A are to be increased by 1.

Conditional transfers of control according to the value of a variable are indicated with an IF statement. The IF statement has the form:

```
→|IF(x-"c) label1, label2, label3

    or

→|IF(x-n) label1, label2, label3
```

where x is the variable to be tested, "c is the ASCII value to be compared with the contents of the variable, or n is an octal value in the range 0 to 377, and label1, label2, and label3 are the names of labels included in the BATCH stream.

When BATCH evaluates the expression (x-"c) or (x-n), the BATCH run-time handler transfers control to:

- label1 if the value of the expression is less than zero.
- label2 if the value of the expression is equal to zero.
- label3 if the value of the expression is greater than zero.

If one of the labels is omitted, and the condition is met for the omitted label, control transfers to the line following the IF statement.

NOTE

Since this comparison is a signed byte comparison, 377 is considered to be -1.

The characters + and - allow the user to control where BATCH begins searching for label1, label2, and label3. If the label is preceded by a minus sign (-), the label search starts just after the $JOB command. If a plus sign (+) or no sign precedes the label, the label search starts after the IF statement. For example, the following statement:

```
→|IF(B-"9) -LOOP, LOOP1,
```

transfers program control to the label LOOP following the $JOB command if the contents of variable B are less than the ASCII value of 9 or to the label LOOP1 following the IF statement if B is equal to ASCII 9. If the contents of variable B are greater than the ASCII value of 9, program control goes to the next BATCH statement in sequence.

The GOTO statement unconditionally transfers program control to a label specified as the argument of the statement. This statement may be one of the following three forms:

→|GOTO label	transfers control to the first occurrence of label that appears after this GOTO statement in the BATCH stream.
→|GOTO +label	same as GOTO label.
→|GOTO -label	transfers control to the first occurrence of label that appears after the $JOB command.

If the label is not found, transfer goes to $EOJ.

The following GOTO statement transfers control unconditionally to the next label LOOP if such a label appears in the BATCH stream following the GOTO statement.

```
→|GOTO LOOP
```

NOTE

> If a label cannot be found, e.g., a minus sign was intended but omitted, the BATCH handler searches until the end of the CTL file is reached and ends the job.

12.5.2.3 Terminal I/O Control -- Commands may be issued directly to the BATCH run-time handler to control console terminal input/output to the log file. If none of the following commands is entered, TTYOUT is assumed.

→|NOTTY	do not write terminal input/output to the log file. Comments to the log will still be logged.
→|TTYIN	write only terminal input to the log file.
→|TTYIO	write terminal input and output to the log file.
→|TTYOUT	write only terminal output to the log file (default).

12.5.2.4 Other Control Characters -- Other control characters allowed in an RT-11 mode command that begins with a period (.) or an asterisk (*) indicate the following:

'text' command to BATCH run-time handler, where text can be one of the following:

CTY	accept input from the console terminal; notify the operator that action is required by ringing a bell and printing a ?.
FF	output current log buffer.
NL	insert a new line (line feed) in the BATCH stream.
x	insert contents of variable where x is an alphanumeric variable A-Z, indicates that the contents of the variable are to be inserted as an ASCII character at this place in the command string.
"message"	direct the message to the console terminal.

Example 1:

The following commands allow the operator to enter the name of a MACRO program to be assembled. The BATCH stream contains:

```
$JOB/RT11
.R MACRO
*'"ENTER MACRO COMMAND STRING"''CTY'
$EOJ
```

The operator receives the following message at the terminal; he types a response, followed by carriage return, and BATCH processing continues.

```
ENTER MACRO COMMAND STRING?FILE,FILE=FILE
```

Example 2:

The user may want to run the same BATCH file on several systems with different configurations and would want to assign a device dynamically. The following RT-11 mode command allows the user to request that the listing device name be entered by the operator.

```
.ASSIGN '"PLEASE TYPE LST DEVICE NAME"''CTY'LST
```

The operator receives the message and responds with the device to be used as the listing device (DT2:).

```
PLEASE TYPE LST DEVICE NAME?DT2:
```

12.5.2.5 Comments -- Comments can be included in RT-11 mode as separate comment statements. This is accomplished by typing a separator followed by a ! and the comment, e.g.,

```
→|!OPERATOR ACTION IS REQUESTED IN THIS JOB. BE PREPARED.
```

12.5.3 RT-11 Mode Examples

The following are examples of BATCH programs using the RT-11 mode.

Example 1:

This BATCH program assembles, lists, and maps 10 programs with only 12 BATCH commands.

```
$JOB/RT11         !ASSEMBLE, LIST, MAP PROG0 TO PROG9
          TTYIO
          !WRITE TERMINAL I/O TO THE LOG FILE
          LET N="0
          !START AT PROG0
LOOP:
.R MACRO
*PROG'N',LOG:/C=PROG'N'/N:TTM
.R LINK
*,LOG:=PROG'N'
          %N
          !INCREMENT VARIABLE N
          IF(N-"9)-LOOP,-LOOP,END
          !TEST FOR END
END:
$EOJ
```

Example 2:

The following program allows the user to set up a master control stream to run several BATCH jobs with one call to BATCH. First the user sets up a BATCH job (INIT.BAT) that will $CHAIN to the master control stream:

```
$JOB/RT11
        LET I="0
        !INITIALIZE INDEX
$CHAIN MASTER   !GO TO MASTER
$EOJ
```

The following is the master control stream (MASTER.BAT) to which INIT chains.

```
$JOB/RT11       !MASTER CONTROL STREAM
        %I
        !BUMP INDEX BY 1
        IF(I-"7),,END
.R BATCH
        !THIS IS A $CHAIN
*JOB'I'
        !RUNS JOB1-JOB7
END:
$MESSAGE END OF BATCH RUN
$EOJ
```

Each job to be run by MASTER.BAT must contain the following:

```
$JOB
   !BATCH COMMANDS
$CHAIN MASTER
$EOJ
```

The master control stream is activated by calling BATCH as follows:

```
.R BATCH
*INIT
```

12.6 CREATING BATCH PROGRAMS ON PUNCHED CARDS

To create a BATCH program on punched cards, the user punches into the cards the commands as described in Section 12.4. Each command line occupies a single punched card. Only one card, the EOF card, is different from the standard BATCH commands. The EOF (end-of-file) card terminates the list of jobs from the card reader.

To create the EOF card, hold the MULT PCH key on the keypunch keyboard while typing the following characters (DEC029 codes, see Appendix H for DEC026 codes):.

```
- & 0 1 6 7 8 9
```

This procedure produces an EOF card with holes punched in the first column (see Figure 12-1).

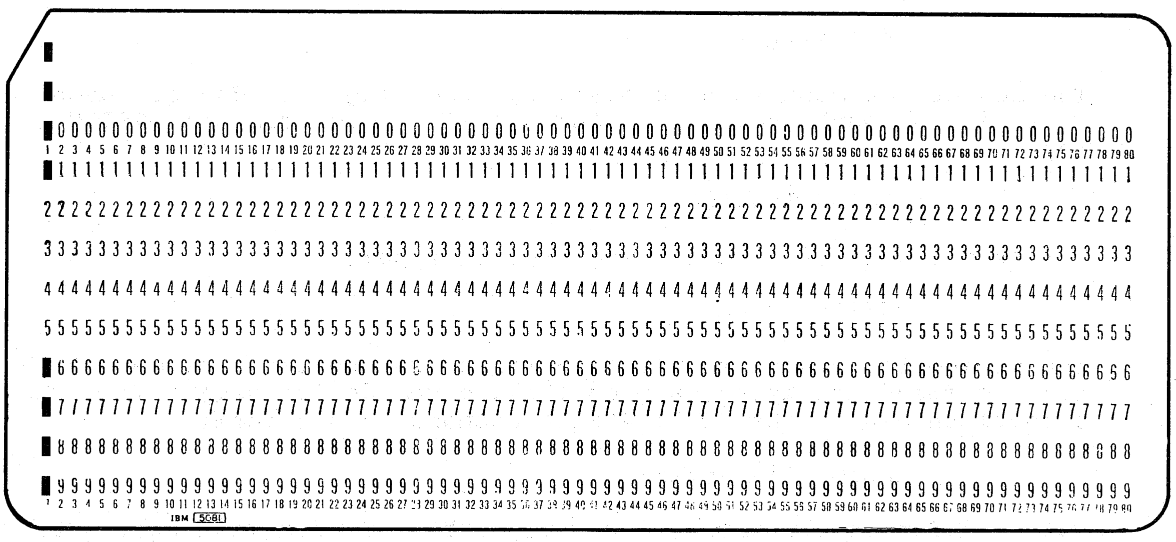


Figure 12-1 EOF Card

12.6.1 Terminating BATCH Jobs on Cards

To terminate BATCH, type \F, followed by a carriage return, put two EOF cards and a blank card in the reader, and ensure that the card reader is ready. Note that a small card deck (less than 512 characters) may require more than two EOF cards to terminate the deck.

12.7 OPERATING PROCEDURES

12.7.1 Loading BATCH

After the RT-11 system has been bootstrapped and the date and time entered, the BATCH run-time handler must be made resident by typing the RT-11 LOAD command as follows:

```
.LOAD BA
```

The BATCH run-time handler must be detached with the /U switch to the BATCH compiler command line (see Section 12.7.2) before it is removed with the RT-11 command UNLOAD.

Unless the log device is SY: or any device for which the handler is already resident, the BATCH log device must be made resident. The log device is loaded by typing:

```
.LOAD log
```

where log is the device to which the BATCH log is to be written, e.g.,

```
.LOAD LP
```

The loading of device handlers can, of course, be done with a single LOAD command, e.g.,

```
.LOAD BA,LP
```

The log device must then be assigned the logical device name LOG. This is accomplished with the RT-11 monitor command ASSIGN in the form:

```
.ASSIGN log:LOG
```

For example, if LP: is the log device, type:

```
.ASSIGN LP:LOG
```

The device to be used for listing files must be made resident with the LOAD command and then assigned the logical device name LST:. This is accomplished with the RT-11 ASSIGN command in the form:

```
.ASSIGN lisdev:LST
```

where lisdev is the physical device to be used for listings. If, for example, listings are to be produced on the line printer, type:

```
.ASSIGN LP:LST
```

NOTE

> The ASSIGN command with no arguments should not be used in a BATCH program since it would then deassign the log and list devices, possibly causing the BATCH job to terminate.

The BATCH Run-Time Handler input device (Compiler output device) must also be made resident. If this device is already resident or is SY:, it need not be loaded. For example, to load the DECtape handler as the input device, type:

```
.LOAD DT
```

If the input file to the BATCH Compiler is on cards, the card reader handler must be loaded by typing:

```
.LOAD CR
```

NOTE

> If input is on cards, the RT-11 monitor command SET must be used (before the handler is loaded) to specify CRLF and NOIMAGE modes, i.e.,
>
> ```
> .SET CR CRLF
> ```
>
> to append a carriage return/line feed combination to each card image, and
>
> ```
> .SET CR NOIMAGE
> ```
>
> to translate the card by packing card code into ASCII data, one column per byte.

12.7.2 Running BATCH

When all necessary handlers have been loaded, the BATCH Compiler is run as follows:

```
.R BATCH
```

BATCH responds by printing an asterisk (*) to indicate readiness to accept command string input. In response to the *, type the output file specifications for the control file followed by an equal sign or left angle bracket, followed by the input file specifications for the BATCH file as follows:

```
{{spc0}{,spc1}{/switch}=}spc3{,...}{,spc8/switch}
```

where:

spc0 — is the BATCH compiler output device and file to be used by the BATCH run-time handler. The device specified must be random-access. This file should not be deleted or moved by the job that is within it; nor should the system ever be compressed from within a batch job. If spc0 is omitted, a file is generated on the default device (DK:) with the same name as the first input file but with a .CTL extension. If no extension is specified, .CTL is assumed.

spc1 — is the log file created by the BATCH run-time handler. If no log device is specified, LOG: is assumed. The device name specified for spc1 must be that assigned to LOG:.

The size of a log file on a file-structured device may be changed from the default size of 64 (decimal) by enclosing the requested size in square brackets. For example:

```
*,FILE.LOG[10]=FILE
```

The default extension for spc1 is .LOG.

spc3-spc8 — are input file specifications. If no input extensions are specified, the default extension is .BAT. If a .CTL file is specified, a precompiled file is assumed and it must be the only file in the input list.

/switch — is one or more of the following:

- /N — compiles but does not execute; creates BATCH control file (.CTL), generates an ABORT JOB message at the beginning of the log file, and returns to RT-11 monitor.
- /T:n — if n=0, sets the /NOTIME switch as default on the $JOB command; if n=1, the default switch on the $JOB command is /TIME.

/U indicates that BATCH compiler is to detach the BATCH run-time handler from the RT-11 monitor for subsequent removal. The prompting message !UNLOAD BA! is printed.

NOTE

The RT-11 monitor command UNLOAD BA must be specified to actually remove the handler. The handler must be reloaded before BATCH is run again.

/V prints version number of BATCH compiler

/X indicates that input is a precompiled BATCH program (used when .CTL is not the extension or when the extension is omitted).

Example 1:

In the following example, the user calls BATCH to compile and execute the three input files (PROG1.BAT, PROG2.BAT, PROG3.BAT), to generate on DK: the compiler output files and to generate on LOG: a log file.

```
.R BATCH
*PROG1.BAT,PROG2.BAT,PROG3.BAT
```

Example 2:

The following commands compile and run SYBILD.BAT, printing the version number of BATCH.

```
.R BATCH
*SYBILD/V
BATCH V01-02
```

Example 3:

The following commands compile PROTO.BAT to create PROTO.CTL but do not run the compiled program.

```
.R BATCH
*PROTO/N
```

Example 4:

Type the following commands to unlink BA.SYS from the monitor and to unload it.

```
.R BATCH
*/U
!UNLOAD BA!

.UNLOAD BA
```

Example 5:

The following commands compile FILE.BAT from magtape to create FILE.CTL on DK1:, execute the compiled file, and create a log file named FILE.LOG of size 20 on LOG:.

```
.R BATCH
*DK1:FILE,FILE[20]=MT:FILE
```

Example 6:

The following commands execute a precompiled job called FILE.TST.

```
.R BATCH
*FILE.TST/X
```

Example 7:

The following commands execute a precompiled job called FILE.CTL.

```
.R BATCH
*FILE/X
```

Example 8:

The following commands accept input from the card reader to create on DK: a file called TEMP.CTL and execute that file.

```
.R BATCH
*CR:
```

Example 9:

The following commands accept input from the card reader to create on DK: a file called JOB.CTL and execute that file.

```
.R BATCH
*JOB=CR:
```

12.7.3 Communicating with BATCH Jobs

During the execution of a BATCH stream, the operator may be requested to supply action or provide information or to insert a command line into the BATCH stream. The operator can accomplish this by typing directives to the BATCH handler via the console terminal.

NOTE

These directives are equivalent to the compiler output generated by BATCH in the .CTL file. The .CTL file is an ASCII file that can be listed using PIP.

These directives have the form:

```
\dir
```

where dir is one of the directives listed in Table 12-6.

To use these directives, the operator must get control of the BATCH run-time handler by typing a carriage return on the console terminal. When BATCH executes a command, it notices the carriage return and prints a carriage return/line feed combination at the terminal. The directives in Table 12-6 may then be entered by the operator; those most useful to the operator are marked with an asterisk (*).

Table 12-6
Operator Directives to BATCH Run-Time Handler

Directive	Explanation
\@	Send the characters that follow to the console terminal.
*\A	Change the input source to be the console terminal.
*\B	Change the input source to be the BATCH stream.
*\C	Send the following characters to the log device.
*\D	Consider the following characters as user data.
*\E	Send the following characters to the RT-11 monitor.
*\F	Force the output of the current log block. If this directive is followed by any characters other than another BATCH backslash (\) directive, the BATCH job prints an error message and terminates; control returns to the RT-11 monitor.
\G	Get characters from the console terminal until a carriage return is encountered.
\Hn	Help function to change the logging mode where n specifies the following: 0 Log only .TTYOUT and .PRINT. 1 Log .TTYOUT, .PRINT, and .TTYIN 2 Do not log .TTYOUT, .PRINT, and .TTYIN 3 Log only .TTYIN
\Ivxlabel1?label2?label3?	IF statement which causes conditional transfer, where v is a variable name in the range A-Z; x is a value for the signed 8-bit comparison (v-x); and label1, label2, label3 are 6-character labels to which control is transferred under certain conditions. (All labels must be six characters in length; if too short, pad with spaces.) If v-x is less than 0, control transfers to label1; if v-x is equal to 0, control goes to label2; if v-x is greater than 0, control goes to label3. The direction for the label search is indicated by ?; if ? is 0, the search begins at the beginning of this job; if ? is 1, the label search begins after the IF statement.

(continued on next page)

Table 12-6 (cont.)
Operator Directives to BATCH Run-Time Handler

Directive	Explanation
\Jlabel?	Jump, unconditional transfer; where label is a 6-character label and ? is 0 or 1. (All labels must be six characters in length; if too short, pad with spaces.) If ?=0, label is a backward reference; if ?=1, label is a forward reference.
\Kv0	Increment variable v where v is a variable name in the range A-Z.
\Kv1n	Store the 8-bit number n in variable v.
\Kv2	Take the value in variable v and return it to the program (via .TTYIN).
\Llabel	Insert label as a 6-character alphanumeric string in the BATCH stream. (All labels must be six characters in length; if too short, pad with spaces.) Labels must not include backslash characters. Characters beyond six are ignored.

Example 1:

The operator may wish to interrupt the BATCH handler to enter information from the console as a result of a /WAIT or 'CTY' in the BATCH stream. For example, the following message appears at the terminal.

```
        $MESSAGE/WAIT WRITE NECESSARY FILES TO DISK
```

To divert BATCH stream input from the current file to the console terminal, the operator types a \E and can then enter commands to the RT-11 monitor until he types a \B. Control then returns to the BATCH stream. For example, to respond to the preceding message, the operator types:

```
WRITE NECESSARY FILES TO DISK
?\A\ER PIP                   changes input source to TT:
PIP                          and calls the RT-11 monitor to
\ER PIP                      run PIP
                             writes FILE.MAC from DT2: to
\DRK:*.*=DT2:FILE.MAC/X      RK:
\E\F\B                       outputs log block and returns
                             control to the BATCH stream
```

Example 2:

The following BATCH program allows the user to make frequent edits to a file and list only the edits. First the user creates a BATCH program which will assemble with a listing and link the file. This BATCH program, called COMPIL.BAT, contains:

```
$JOB/RT11
        TTYIO
        !WRITE TERMINAL I/O TO THE LOG FILE
.R MACRO
```

```
        !CALL THE MACRO ASSEMBLER
*FILE,FILE/C=FILE
$MESSAGE/WAIT OK TO TYPE EDIT COMMANDS
.R LINK
        !CALL THE RT-11 LINKER
*FILE,LOG:=FILE
$EOJ
```

At run-time, the user can insert commands into the BATCH stream via the console terminal. These commands will search for the section of the listing file that has been edited and then list this section to the log. The command must be inserted after the R MACRO command but before the R LINK command. The operator types the following to use COMPIL.BAT.

```
.LOAD BA,LP
.ASSIGN LP:LOG
.R BATCH
*COMPIL
 OK TO TYPE EDIT COMMANDS
?\A\E
\ER EDIT
  :
*ERFILE.LST$$      find FILE.LST on system device
*EWFILE.LST$$
*PBELL:$=J$$       find BELL, move pointer back
*/L$$              print contents of buffer to log and console
*EX$$              print remainder of input file to log
\E\C\B             return to BATCH

 END BATCH
```

12.7.4 Terminating BATCH

When BATCH terminates normally, it prints:

```
END BATCH
```

and returns control to the RT-11 monitor.

To abort BATCH while it is executing a BATCH stream, the best method is to interrupt the BATCH handler by typing a carriage return. When BATCH executes the next command after the carriage return was typed, it prints a carriage return/line feed combination at the console terminal and the operator has control. The operator then types \F followed by a carriage return. The BATCH handler responds with the FE (forced exit) error message and writes the remainder of the log buffer. Control returns to the RT-11 monitor.

CTRL C can be typed to terminate BATCH but is a more drastic method than that described above. CTRL C should be used when BATCH is in a loop or when a long assembly is running.

12.8 DIFFERENCES BETWEEN RT-11 BATCH AND RSX-11D BATCH

For those users who may wish to run their RT-11 BATCH programs under RSX-11D, or vice versa, Table 12-7 lists the differences between the two programs. A user who plans to run his BATCH programs under both systems should ensure that the programs are compatible with both RT-11 and RSX-11D BATCH.

Table 12-7
Differences Between RT-11 and RSX-11D BATCH

Differing Aspects	RT-11	RSX-11D
File descriptors	dev:filnam.ext/sw	SY:filnam.ext/sw
Default listing extension	.LST(or .LIS)	.LIS
Executable file extension	.SAV	.EXE
Incompatible commands	$BASIC $CALL $CHAIN $LIBRARY $RT11 $SEQUENCE	$MCR
Incompatible switches	$COPY/DELETE $CREATE/DOLLARS $CREATE/LIST $DATA/DOLLARS $DATA/LIST $DIR file/LIST $DISMOUNT/WAIT $DISMOUNT lun:/LOGICAL $FORTRAN/DOLLARS $FORTRAN/MAP $JOB/BANNER $JOB/LIST $JOB/RT11 $JOB/TIME $JOB/UNIQUE $LINK/LIBRARY $LINK/OBJECT $MACRO/CREF $MACRO/DOLLARS $MACRO/LIBRARY $MACRO/MAP $MESSAGE/WAIT $MESSAGE/WRITE $PRINT/DELETE	 $DIR file/DIRECTORY $JOB/NAME $JOB/LIMIT $JOB/MCR $LINK/MCR
$DATA input	appears as if from input	appears as if from a file named FOR001.DAT
Logical device names	in $MOUNT and $DISMOUNT	logical unit numbers only
$RUN	file name must be specified	RSX11DBAT.EXE is default

12.9 ERROR MESSAGES

Two types of errors are reported by the BATCH system: user BATCH stream errors found by the compiler and noted in the BATCH log file, and system or operator errors noted at the console at run-time. Note that when BATCH is called from within a BATCH stream, both types of error messages are sent to the log.

The following messages detail errors found by the compiler and noted in the log file. An error is indicated by the printing of the erroneous line, an uparrow (or circumflex) under the error, and one of the following messages.

Message	Meaning
ABORT JOB	An error has occurred in compiling a BATCH program; the compiler forces the job to abort. All error messages are printed in the log file.
BAD CONSTRUCTION	In RT-11 mode, an IF statement is not in the correct form or there was an illegal 'text' directive.
BAD SEQUENCE ARGUMENT	The identification number specified in a $SEQUENCE command is not numeric.
BAD VARIABLE	The variable specified is not one of the characters A-Z.
BAD VID	The volume identification specified in a $MOUNT command is not in the correct form, e.g., the equal sign (=) or the text is missing.
BATCH HANDLER NOT RESIDENT	The BATCH run-time handler was not loaded with the RT-11 LOAD command (see Section 12,7,1).
BATCH STACK OVERFLOW	There are too many nested $CALL commands in the BATCH stream.
COMMAND NOT UNIQUE	The spelling of a command is not unique (appears only when $JOB/UNIQUE was specified).
DISMOUNT ERROR	The logical device name specified does not exist.
'$' MISSING	A $ is not present in the first position of the command line (or card column 1).
ILLEGAL '+'	The + construction was used when not allowed (e.g., in a $RUN or $BASIC input file descriptor), there is a + in an output file descriptor, or a + terminates a control statement.
ILLEGAL CHARACTER	The character specified is not used in proper context.

(continued on next page)

Message	Meaning
ILLEGAL SWITCH	The switch name specified is not a legal RT-11 BATCH switch or is not legal for this field.
ILLEGAL SWITCH COMBINATION	More than one switch of the same type, e.g., /INPUT and /SOURCE exists on same command line.
LINE TOO LONG	The input line entered is greater than 80 characters.
MULTIPLE SWITCH	The same switch is specified more than once in a single command line.
NO $EOJ	A $JOB or $SEQUENCE command appears without a preceding $EOJ to end the previous job.
NO FILE	No file descriptor was found in the BATCH stream where one was expected, or no file name was entered in the $CREATE command.
NO FILE NAME BEFORE "."	An extension was specified but no file name preceded it.
NO LOGICAL DEVICE	No logical device is specified in a $MOUNT command.
NO PHYSICAL DEVICE	No physical device is specified in a $MOUNT command.
PLEASE ASSIGN LOG,LST	The log device (LOG:) and/or the listing device (LST:) were not assigned (see Section 12.7.1).
PLEASE LOAD LOG HANDLER	The log device handler is not resident (see Section 12.7.1).
SEPARATOR MISSING	A file descriptor was not terminated by a space, a +, a , or a carriage return.
SWITCH NOT UNIQUE	The spelling of a switch is not unique (appears only when $JOB/UNIQUE is specified).
TOO MANY FILE DESCRIPTORS	More than six file descriptors are specified in a $command line.
UNKNOWN COMMAND	The command specified with a $ in character position 1 is not a legal BATCH command.

The following messages, generated by system or operator errors, are noted on the console at run-time.

Message	Meaning
BAD COPY OF HANDLER	The copy of BA.SYS in memory is bad. BATCH cannot be run until BA.SYS is unloaded and reloaded (see Section 12.7.1).
BAD SWITCH	The command line to the BATCH Compiler contains an illegal switch.
BATCH FATAL ERROR	A nonrecoverable error has occurred (e.g., the system device is WRITE-LOCKed); BATCH must be rerun. The system may have to be rebootstrapped.
BATCH STACK OVERFLOW	There are too many nested $CALL commands in the BATCH stream.
BC	Fatal error. A Bad Code was found in the control file by the BATCH handler. This can happen when a .CTL file has been garbled or if an editing mistake was made by the programmer when altering or creating the file with EDIT.
END BATCH	A BATCH job has been terminated.
EOF WITH NO EOJ	A file was not terminated with a $EOJ command.
FE	Fatal error. A Forced End occurred due to the appearance in the .CTL file of an illegal \F followed by a carriage return or BATCH was terminated from the console with a \F followed by a carriage return. The \F must be followed by another BATCH control directive.
ILLEGAL COMMAND LINE	Command line to the BATCH compiler is incorrect.
ILLEGAL DEVICE	A $DISMOUNT command was attempted on device that was not assigned.
ILLEGAL LOG DEVICE	Magtape, cassette, or a read-only device (e.g., PTR:) was specified as the log device.
INPUT ERROR	An error occurred while BATCH was attempting to read the compiler input file (.BAT).

(continued on next page)

Meaning	Message
INPUT FILE?	An input file descriptor was not specified to the BATCH compiler command line.
IO	Fatal error. An Input or Output error occurred when the BATCH handler was attempting to read the .CTL file or write to the log file. The probable cause of this error is a log file overflow.
LOG DEVICE ERROR	An error occurred on the log device.
LU	Fatal error. A Lock Up occurred in the BATCH handler because it could not find a free channel or a channel it could use. This can happen if all channels are opened for magtape or cassette operations within the BATCH stream.
NO CONTROL FILE	The .CTL file was probably sent to a nonfile-structured device (e.g., LP:) without using the /N switch to inhibit execution.
OUTPUT DEVICE FULL	The temporary file (.CTL) created by BATCH is too large for the specified device. Try compressing the device with PIP, or use another device.
OUTPUT ERROR	An error occurred while BATCH was attempting to write the compiler output file (.CTL) or magtape or cassette was specified as the .CTL output device.
OUTPUT FILE NOT OPEN	Fatal error. BATCH attempted to write on a .CTL file without opening it.
RETURN FROM CALL ERROR	Fatal error. BATCH cannot read control file which called a subprogram. BATCH cannot resume execution on return from call.
TOO MANY OUTPUT FILES	A third output file was specified to the BATCH compiler; only two output files may be specified.
!UNLOAD BA!	Prompting message printed when /U switch is given to the compiler.

APPENDIX A

ASSEMBLY, LINK, AND BUILD INSTRUCTIONS

The information that formerly appeared in this appendix has now been incorporated into the RT-11 System Generation Manual, DEC-11-ORGMA-A-D.

APPENDIX B

COMMAND AND SWITCH SUMMARIES

Command and switch summaries of the various RT-11 system and utility programs are grouped here for the user's convenience. Refer to the appropriate chapter for details.

B.1 KEYBOARD MONITOR (Chapter 2)

B.1.1 Command Summary

Only those command characters underlined need be entered; all command lines are terminated by typing a carriage return.

Command Format	Used Under	Explanation
ASSIGN dev:udev	F/B,S/J	Assigns a user-defined name (udev) as an alternate name for a device (dev). Deassigns synonyms when used without any arguments.
B location	F/B,S/J	Sets a relocation base (location), which is an octal address to be used as a base address for subsequent Examine and Deposit commands.
CLOSE	F/B,S/J (B only)	Causes all currently open files to become permanent.
DATE dd-mmm-yy	F/B,S/J	Enters the indicated day-month-year (dd-mmm-yy); this date is then assigned to newly created files, new device directory entries, and listing output. When used without an argument, the current date (as entered) is printed.
D location = value1,value2,...,valuen	F/B,S/J	Deposits the specified values starting at the given location (location represents an octal address which is added to the base address to obtain the actual address at which values will be deposited).

Command Format	Used Under	Explanation
E location m-location n	F/B,S/J	Prints the contents of the specified locations in octal on the console terminal (location represents an octal address which is added to the base address to obtain the actual address examined).
FRUN dev:filnam.ext/N:n/S:n/P	F/B (F only)	Initiates a foreground job which exists on the device indicated (dev) under the specified filename and extension. /N:n is optionally used to allocate n (decimal) words over and above actual program size; /S:n is optionally used to allocate n words for stack space; /P is used for debugging purposes (the load address is printed but the program must be started using RSUME).
GET dev:filnam.ext	F/B,S/J	Loads the specified memory image file (filnam.ext) into memory from the indicated device (dev:).
GT OFF	F/B,S/J	Used (after GT ON) to clear the display processor and resume printout on the console terminal.
GT ON/L:n/T:n	F/B,S/J	Enables the display processor so that the display screen replaces the console as the terminal output device. /L:n may be optionally used to designate the number of lines to display (12" screen - 1<=n<=37 octal; 17" screen - 1<=n<=50 octal). /T:n may be optionally used to indicate the top position of the scroll display (12" screen - 1<=n<=1350 octal; 17" screen - 1<=n<=1750 octal).
INITIALIZE	F/B,S/J (B only)	Resets background system tables; makes nonresident all handlers not loaded and purges background's I/O channels.
LOAD dev,...	F/B,S/J	Makes a device handler resident for use.
LOAD dev=B,dev=F,...	F/B	Makes a device handler resident for use with background and foreground jobs.
R filnam.ext	F/B,S/J (B only)	Loads the specific memory image file (filnam.ext) into memory from the system device and starts execution.
REENTER	F/B,S/J	Starts a program at its reentry address (i.e., its start address -2).
RSUME	F/B (F only)	Resumes execution of a foreground program where it was suspended.
RUN dev:filnam.ext	F/B,S/J (B only)	Loads the specified memory image file (filnam.ext) into memory from the indicated device (dev:) and starts execution.

Command Format	Used Under	Explanation
SAVE dev:filnam.ext areal,area2-arean	F/B,S/J	Writes the area(s) of user memory specified into the named file (filnam.ext) in save image format. Memory is transferred in 256-word blocks.
SET dev:{NO}option=value	F/B,S/J	Used to change device (dev:) handler characteristics and certain system configuration parameters. See Table 2-5 (Section 2.7.2.8) for a list of options.
START address	F/B,S/J	Begins execution of the program currently in memory at the specified address. If an address is not indicated, the starting address in location 40 is used.
SUSPEND	F/B (F only)	Suspends execution of the foreground job currently running.
TIME hh:mm:ss	F/B,S/J	Enters time of day in hours, minutes, seconds past midnight (hh:mm:ss). If all three arguments are omitted, the current time of day is output.
UNLOAD dev,dev,...	F/B,S/J	Makes previously loaded handlers (dev) nonresident and frees the memory space they were using.

B.1.2 Special Function Keys

Key	Used Under	Function
CTRL A	F/B,S/J	Valid when the monitor GT ON command has been typed and the display is in use. Does not echo on the terminal. Used after CTRL S has been typed to effectively page output.
CTRL B	F/B	Echoes B> on the terminal 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). B> does not echo if output is already coming from the background job. (Control can be redirected to the foreground job via CTRL F.)

Key	Used Under	Function
CTRL C	F/B,S/J	Echoes ↑C on the terminal, interrupts current program execution, and returns control to the Keyboard Monitor. If a program is waiting for terminal input or is using the device handler TT: for input, typing a single CTRL C interrupts execution and returns control to the monitor command level. Otherwise, two CTRL C's must be typed in order to interrupt execution.
CTRL E	F/B,S/J	Valid when the monitor GT ON command has been typed and the display is in use. Does not echo on the terminal, but causes all I/O to appear on both the display screen and the console terminal simultaneously. A second CTRL E disables console terminal output.
CTRL F	F/B	Echoes F> on the terminal and directs all keyboard input to the foreground job and all output to be taken from the foreground job. Control remains with the foreground job until redirected to the background job (via CTRL B) or until the foreground job terminates.
CTRL O	F/B,S/J	Echoes ↑O on the terminal and causes suppression of teleprinter output while continuing program execution. Teleprinter output is reenabled when one of the following occurs: 1. A second CTRL O is typed 2. A return to the monitor is indicated via CTRL C 3. The running program issues a .RCTRLO directive (see Chapter 9)
CTRL Q	F/B,S/J	Does not echo. Resumes printing characters on the terminal from the point at which printing was previously stopped (via CTRL S).
CTRL S	F/B,S/J	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.
CTRL U	F/B,S/J	Echoes ↑U followed by a carriage return on the terminal and deletes the current input line.
CTRL Z	F/B,S/J	Echoes ↑Z on the terminal and terminates input when used with the terminal device handler (TT:).

Key	Used Under	Function
RUBOUT	F/B,S/J	Deletes the last character from the current line. 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.

B.2 EDITOR (Chapter 3)

B.2.1 Command Arguments

Format	Meaning
n	A decimal integer (in the range -16383 to +16383) which may, except where noted, be preceded by a + or -. Whenever an argument is acceptable in a command, its absence implies an argument of 1.
0	Refers to the beginning of the current line.
/	Refers to the end of the 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 argument used.

B.2.2 Input and Output Commands

Command	Form	Meaning
EDIT BACKUP	EB dev:filnam.ext[n]$	Opens a file for editing, creating a backup copy (.BAK).
EDIT READ	ER dev:filnam.ext$	Opens a file for input.
EDIT WRITE	EW dev:filnam.ext[n]$	Creates a new file for output.
END FILE	EF	Closes the current output file without performing any further input/output operations.
EXIT	EX	Outputs the remainder of the input file to the output file and returns control to the monitor.
LIST	(-)nL 0L /L	Prints a specified number of lines on the console terminal.

Command	Form	Meaning
NEXT	nN	Outputs the contents of the Text Buffer to the output file, clears the buffer, and reads in the next page of the input file.
READ	R	Reads a page of text from the input file and appends it to the contents of the buffer.
VERIFY	V	Prints the current text line (the line containing the pointer) on the console terminal.
WRITE	(-)nW 0W /W	Outputs a specified number of lines of text from the Text Buffer to the output file.

B.2.3 Pointer Relocation Commands

Command	Form	Meaning
ADVANCE	(-)nA 0A /A	Moves the pointer over a specified number of lines in the Text Buffer. The pointer is positioned at the beginning of the line.
BEGINNING	B	Moves the current location pointer to the beginning of the Text Buffer.
JUMP	(-)nJ 0J /J =J	Moves the pointer over a specified number of characters in the Text Buffer.

B.2.4 Search Commands

Command	Form	Meaning
FIND	nFtext$	Beginning at the current location pointer, searches the entire text file for the nth occurrence of the specified character string. Pages of text are read into the Text Buffer, searched, and then written to the output file until the text string is found.
GET	nGtext$	Searches the contents of the Text Buffer, beginning at the current location pointer, for the next occurrence of the text string.
POSITION	nPtext$	Searches the input file for the nth occurrence of the text string; if the text string is not found, the buffer is cleared and a new page is read from the input file.

B.2.5 Text Modification Commands

Command	Form	Meaning
CHANGE	(-)nCtext$ 0Ctext$ /Ctext$	Replaces n characters, beginning at the pointer, with the indicated text string.
DELETE	(-)nD 0D /D =D	Removes a specified number of characters from the Text Buffer, beginning at the current location pointer.
EXCHANGE	(-)nXtext$ 0Xtext$ /Xtext$	Replaces n lines, beginning at the pointer, with the indicated text string.
INSERT	Itext$	Inserts text immediately following the current location pointer; an ALTMODE terminates the text.
KILL	(-)nK 0K /K	Removes n lines from the Text Buffer beginning at the current location pointer.

B.2.6 Utility Commands

Command	Form	Meaning
EDIT CONSOLE	EC	If scroller is in use, EC returns scroller to its normal mode (using full screen for display of text and commands); if scroller is not in use, EC clears screen and returns control to console terminal (following ED).
EDIT DISPLAY	ED	If scroller is in use, ED recalls the text window (following EC) and arranges text and commands on screen; if scroller is not in use, ED displays text window only.
EXECUTE MACRO	nE	Executes the command string specified in the last macro command.
MACRO	M/command string/	Inserts a command string into the Macro Buffer.
	0M M//	Clears the Macro Buffer and reclaims the area for text.
SAVE	nS	Copies the specified number of lines, beginning at the pointer, into the Save Buffer.
UNSAVE	U	Inserts the entire contents of the Save Buffer into the Text Buffer at the position of the current location pointer.

Command	Form	Meaning
EDIT VERSION	EV	Displays the version number of the Editor on the console terminal.
EDIT LOWER	EL	Enables editing in upper- and lower-case.
EDIT UPPER	EU	Returns editing to upper-case only (after EL).

B.2.7 Immediate Mode Commands

Command	Meaning
CTRL N	Advances the pointer (cursor) to the beginning of the next line.
CTRL G	Moves the pointer (cursor) to the beginning of the previous line.
CTRL D	Moves the pointer (cursor) forward by one character.
CTRL V	Moves the pointer (cursor) back by one character.
RUBOUT	Deletes the character immediately preceding the pointer (cursor).
ALTMODE (two) (one only)	Enters Immediate Mode. Returns control to Editor Command Mode.
Any character other than the above (with the exception of CTRL C)	Inserts the character as text positioned immediately before the pointer (cursor).

B.2.8 Key Commands

Command	Meaning
ALTMODE	Echoes $. A single ALTMODE terminates a text string. A double ALTMODE executes the command string. (When used alone on a line, two ALTMODES cause control to enter Immediate Mode, while a single ALTMODE returns control to Editor Command Mode.)
CTRL C	Echoes at the terminal as ↑C and a carriage return. Terminates execution of EDIT commands, closes any open files, and returns to monitor command mode.
CTRL O	Echoes ↑O and a carriage return. Inhibits printing on the terminal until completion of the current command string. Typing a second CTRL O resumes output.
CTRL U	Echoes ↑U and a carriage return. Deletes all the characters on the current terminal input line.

Command	Meaning
RUBOUT	Deletes character from the current line.
TAB	Spaces to the next tab stop. Tab stops are positioned every eight spaces on the terminal.
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.

B.3 PIP (Chapter 4)

B.3.1 Switch Summary

Switch	Explanation
/A	Copies file(s) in ASCII mode; ignores nulls and rubouts; converts to 7-bit ASCII.
/B	Copies files in formatted binary mode.
/C	Used in conjunction with another switch; causes only files with current date to be included in the specified operation.
/D	Deletes file(s) from specified device.
/E	Lists the device directory including unused spaces and their sizes. Sequence numbers are listed for cassettes.
/F	Prints a short directory (filenames only) of the specified device.
/G	Ignores any input errors which occur during a file transfer and continues copying.
/I or no switch	Copies file(s) in image mode (byte by byte).
/K	Scans the specified device and types the absolute block numbers (in octal) of any bad blocks on the device.
/L	Lists the directory of the specified device. Sequence numbers are listed for cassettes.
/M:n	Used when I/O transfers involve either cassette or magtape, n represents the numeric position of the file to be accessed in relation to the physical position of the cassette or magtape on the drive.

Switch	Explanation
/N:n	Used with /Z to specify the number of directory blocks (n) to allocate to the directory.
/O	Bootstraps the specified device (DT0, RKn, RF, DPn, DSn, DXn).
/Q	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.
/R	Renames the specified file.
/S	Compresses the files on the specified directory device so that free blocks are combined into one area.
/T	Extends number of blocks allocated for a file.
/U	Copies the bootstrap from the specified file into absolute blocks 0 and 2 of the specified device.
/V	Types the version number of the PIP program being used.
/W	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	Copies files individually (without concatenation).
/Y	Causes system flies and .BAD files to be operated on by the command specified.
/Z:n	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.

B.4 MACRO/CREF (Chapter 5)

Refer to Appendix C for a complete summary of MACRO features. CREF switches are also included in that appendix.

B.5 LINKER (Chapter 6)

B.5.1 Switch Summary

The Linker switches (and the command line on which each must appear) are:

Switch Name	Command Line	Meaning
/A	1st	Alphabetizes the entries in the load map.
/B:n	1st	Bottom address of program is indicated as n (illegal for foreground links).
/C	any	Continues input files on another command line (must be used with /O).
/F	1st	Indicates that the Linker will use the default FORTRAN library, FORLIB.OBJ.
/I	1st	Includes the global symbols to be searched from the library.
/L	1st	Produces an output file in LDA format (illegal for foreground links).
/M:n	1st	Allows terminal keyboard specification of the user's stack address. n represents an optional 6-digit unsigned octal number.
/O:n	any but the 1st	Indicates that the program will be an overlay structure; n specifies the overlay region to which the module is assigned.
/R	1st	Produces output in REL format; only files in REL format will run in the foreground (REL format files may not be run under a Single-Job system).
/S	1st	Allows the maximum amount of space in memory to be available for the Linker's symbol table. (This switch should only be used when a particular link stream causes a symbol table overflow.)
/T or /T:n	1st	Transfer address is to be specified at terminal keyboard via n.

B.6 LIBRARIAN (Chapter 7)

B.6.1 Switch Summary

The Librarian (LIBR) switches (and the command line on which each must appear) are:

Switch	Command Line	Meaning
/C	Any	The command is too long for the current line and is continued on the next line.
/D	1st	Deletes modules from a library file.
/G	1st	Global deletion; deletes entry points from the library directory.
/R	1st	Replaces modules in a library file.
/U	1st	Update; inserts and replaces modules in a library file.

B.7 ODT (Chapter 8)

B.7.1 Command Summary

In the command format shown below, r represents a relocatable expression and n represents an octal number.

Command	Format	Explanation
RETURN		Closes open location and accepts the next command.
LINE FEED		Closes current location and opens next sequential location.
↑ or ^	↑ or ^	Opens previous location.
← or _	← or _	Indexes the contents of the opened location by the contents of the PC and opens the resulting location.
>	>	Uses the contents of the opened location as a relative branch instruction and opens the referenced location.
<	<	Returns to sequence prior to last @, >, or ← command and opens the succeeding location.
@	@	Uses the contents of the opened location as an absolute address and opens that location.

Command and Switch Summaries

Command	Format	Explanation
/	/	Reopens the last opened location.
	r/	Opens the word at location r.
\	\	Reopens the last opened byte (SHIFT L).
	r\	Opens the byte at location r.
!	! n!	After a word or byte has been opened, prints the address of the opened location relative to relocation register n. If n is omitted, ODT selects the relocation register whose contents are closest to but less than or equal to the address of the opened location.
$	$n/	Opens general register n (0-7).
	$B/	Opens the first word of the breakpoint table.
	$C/	Opens Constant Register.
	$F/	Opens Format Register.
	$P/	Opens Priority Register.
	$R/	Opens first Relocation Register (register 0).
	$S/	Opens Status Register.
A	r;nA	Starting at location r, prints n bytes in their ASCII format; then inputs n bytes from the terminal starting at location r.
B	;B	Removes all Breakpoints.
	r;B	Sets Breakpoint at location r.
	r;nB	Sets Breakpoint n at location r.
	;nB	Removes the nth Breakpoint.
C	r;C	Prints the value of r and stores it in the Constant Register.
E	r;E	Searches for instructions that reference effective address r.
F	;F	Fills memory words with contents of the Constant Register.
G	r;G	Goes to location r and starts program.
I	;I	Fills memory bytes with the low-order 8 bits of the Constant Register.
O	r;O	Calculates offset from currently open location to r.
P	;P	Proceeds with program execution from breakpoint. In single instruction mode only, executes next instruction.

Command	Format	Explanation
	k;P	Proceeds with program execution from breakpoint; stops after encountering the breakpoint k times. In single instruction mode only, executes next k instructions.
R	;R	Sets all Relocation Registers to -1 (highest address value).
	;nR	Sets Relocation Register n to -1.
	r;nR	Sets Relocation Register n to the value of r. If n is omitted, it is assumed to be 0.
	R	Selects the Relocation Register whose contents are closest to but less than or equal to contents of the opened location. Subtracts the contents of the register from the contents of the opened word and prints the result.
	nR	Subtracts the contents of the Relocation Register n from the contents of the opened word and prints the result.
S	;S	Disables single instruction mode; reenables breakpoints.
	;nS	Enables single instruction mode (n can have any value and is not significant); disables breakpoints.
W	r;W	Searches for words with bit patterns which match r.
X	X	Performs a Radix 50 unpack of the binary contents of the current opened word; then permits the storage of a new Radix 50 binary number in the same location.

B.8 PROGRAMMED REQUESTS (Chapter 9)

Appendix E summaries the programmed requests available under RT-11, Version 2C.

B.9 BATCH (Chapter 12)

Only those characters underlined need be entered.

B.9.1 Switch Summary

(Command Field)

Switch	Meaning
/BANNER	Prints header of job on the log file.
/CREF	Specifies that a cross reference listing is to be produced during compilation.
/DELETE	Indicates that input is to be deleted after the operation is complete.

Switch	Meaning
/DOLLARS	Indicates that the data following this command may have an $ in the first character position of a line. Reading of the data is terminated by one of the following BATCH commands: $JOB $SEQUENCE $EOD $EOJ or by a physical end-of-file on the BATCH input stream.
/LIBRARY	Includes the default library in the link operation.
/LIST	Produces a temporary listing on the listing device (LST:) or writes data images on the log device (LOG:).
/MAP	Produces a temporary linkage map on the listing device (LST:).
/OBJECT	Produces a temporary object file as output of compilation or assembly.
/RT11	Sets BATCH to operate in RT-11 mode.
/RUN	Causes linking (if necessary) and execution of programs compiled since the last link and go operation or start of job.
/TIME	Writes the time of day to the log file when commands are executed.
/UNIQUE	Checks for unique spelling of switches and keynames.
/WAIT	Pauses to wait for operator action.
/WRITE	Indicates that the operator is to WRITE-ENABLE a specified device or volume.
/NOBANNER	Does not print a job header.
/NOCREF	Does not create a cross reference listing.
/NODELETE	Does not delete input after operation is complete.
/NODOLLARS	Following data may not have a $ in the first character position; a $ in the first character position signifies a BATCH control command.
/NOLIBRARY	Does not include the default library in the link operation.

Switch	Meaning
/NOLIST	Does not produce a listing.
/NOMAP	Does not create MAP file.
/NOOBJECT	Does not produce object file as output of compilation.
/NORT11	Does not set BATCH to operate in RT-11 mode.
/NORUN	Does not execute or link and execute the program after performing the specified command.
/NOTIME	Does not write time of day to log file.
/NOUNIQUE	Does not check for unique spelling.
/NOWAIT	Does not pause for operator action.
/NOWRITE	Indicates that no writes are allowed or that the volume specification is read-only; the operator is informed and must WRITE-LOCK the appropriate device.

(Specification Field)

/BASIC	BASIC source file.
/EXECUTABLE	Indicates that runnable program image file is to be created as the result of a link operation.
/FORTRAN	FORTRAN source file.
/INPUT	Input file; default if no switches are specified.
/LIBRARY	Library file to be included in link operation (prior to default library).
/LIST	Listing file.
/LOGICAL	Indicates that the device is a logical device name.
/MACRO	MACRO or EXPAND source file.
/MAP	Linker map file.
/OBJECT	Object file (output of assembly or compilation).
/OUTPUT	Output file.
/PHYSICAL	Indicates physical device name.

Switch	Meaning
/SOURCE	Indicates source file.
/VID	Volume identification.

B.9.2 Command Summary

Command	Meaning
$BASIC	Compiles a BASIC source program.
$CALL	Transfers control to another BATCH file, executes that BATCH file, and returns to the calling BATCH stream.
$CHAIN	Passes execution to another BATCH file.
$COPY	Copies files.
$CREATE	Creates new files from data included in BATCH stream.
$DATA	Indicates the start of data.
$DELETE	Deletes files.
$DIRECTORY	Provides a directory of the specified device.
$DISMOUNT	Signals operator to dismount a volume from a device and deassigns a logical device name.
$EOD	Indicates the end of data.
$EOJ	Indicates the end of a job.
$FORTRAN	Compiles a FORTRAN source program.
$JOB	Indicates the start of a job.
$LIBRARY	Specifies library that is to be used in linkage operations.
$LINK	Links modules for execution.
$MACRO	Assembles a MACRO source program.
$MESSAGE	Issues message to the operator.
$MOUNT	Signals operator to mount a volume on a device and optionally assigns a logical device name.
$PRINT	Prints files.
$RT11	Specifies that the following lines are RT-11 mode commands.

Command	Meaning
$RUN	Causes a program to execute.
$SEQUENCE	Assigns an arbitrary identification number to a job.

B.10 DUMP (Appendix I)

B.10.1 Switch Summary

Switch	Meaning
/B	Outputs octal bytes.
/E:n	Ends output at block n.
/G	Ignores input errors.
/N	Suppresses ASCII output.
/O:n	Outputs only block number n.
/S:n	Starts output with block n.
/W	Outputs octal words.
/X	Outputs RAD50 characters.

B.11 FILEX (Appendix J)

B.11.1 Switch Summary

Switch	Meaning
/A	Indicates a character-by-character ASCII transfer in which rubouts and nulls are deleted; when /T is also used, each PDP-10 word is assumed to contain five 7-bit ASCII bytes.
/D	Deletes the named file from the device; valid only for DOS/BATCH and RSTS DECtape.
/F	Causes a "fast" listing of the device directory by listing filenames only.

Switch	Meaning
/I	Performs an image mode transfer; in the input is either DOS/BATCH, RSTS or RT-11, this is a word-for-word transfer; if the input is from DECsystem-10, /I indicates that the file resembles a file created on DECsystem-10 by MACY11, MACX11, or LNKX11 with the /I switch: in this case, each DECsystem-10 36-bit word contains one PDP-11 8-bit byte in its low-order bits.
/L	Causes a complete listing of the device directory, including filenames, block lengths, and creation dates.
/P	Performs a packed image transfer; if the input is from either DOS/BATCH, RSTS-11 or RT-11, this is a word-for-word transfer; if the input is from DECsystem-10, /P indicates that the file resembles a file created on DECsystem-10 by MACY11, MACX11, or LNKX11 with the /P switch, in which case each DECsystem-10 36-bit word contains four PDP-11 8-bit bytes aligned on bits 0, 8, 18, and 26. This mode is assumed if no mode switch (/A, /I) is indicated in a command line.
/S	Indicates the device is a DOS/BATCH (or RSTS) file-structured device.
/T	Indicates the device is a DECsystem-10 file-structured device.
/V	Types out version number of FILEX.
/Z	Zeroes the directory of the specified device in the proper format (valid only for DOS/BATCH and RSTS DECtape).

B.12 SRCCOM (Appendix K)

B.12.1 Switch Summary

Switch	Meaning
/B	Compares blank lines. Without this switch, blank lines are ignored.
/C	Ignores comments (all text on a line preceded by a semicolon) and spacing (spaces and tabs).
/F	Includes form feeds in the output file (form feeds are still compared if /F is not used, but they are not included in the output of differences).

Command	Meaning
/H	Types list of switches available (help text).
/L:n	Specifies the number of lines that determine a match (where n is an octal number <=310). The default value for n is 3.
/S	Ignores spaces and tabs.

B.13 PATCH (Appendix L)

B.13.1 Command Summary

Command	Meaning
/O	Indicates overlay-structured file.
/M	Indicates monitor file.
Vr;nR	Sets relocation register n to value Vr.
b;B	Sets bottom address of overlay file to b.
[s:]r,o/	Opens word location Vr + o in overlay segment s.
[s:]r,o\	Opens byte location Vr + o in overlay segment s.
<CR>	Closes currently open word/byte.
<LF>	Closes currently open word/byte and opens the next one.
↑ or ^	Closes the currently open word/byte and opens the previous one.
@	Closes the currently open word and opens the word addressed by it.
F	Begins patching a new file.
E	Exits to RT-11 monitor.

B.14 PATCHO (Appendix M)

B.14.1 Command Summary

Command	Format	Meaning
BYTE	$\text{BYTE}\ \begin{matrix}\text{CSECT + OFFSET}\\ \text{NAME}\end{matrix} = \left\{\begin{matrix}\#\\ \%\end{matrix}\right\}\left\{\begin{matrix}\text{NAME OF}\\ \text{CSECT OR}\\ \text{GLOBAL}\end{matrix}\right\}\left\{\begin{matrix}+\\ -\end{matrix}\right\}\text{OFFSET}$	Modifies a given byte in an object module.
DEC	*DEC	Used when the proper checksum for the patch being made is unknown. PATCHO computes a checksum and prints out its value.
DUMP	*DUMP	Prints the contents of an object module in octal and causes an automatic POINT to the next module, if any, in the input file.
EXIT	*EXIT	Terminates the patch session by closing the file and returning control to the monitor.
HELP	*HELP	Prints an explanation of the PATCHO commands.
LIST	*LIST	Lists the names of all object modules in the input file in the order in which they appear in the file. (A POINT command should be used after a LIST.)
OPEN	*OPEN	Opens files for input and output.
POINT	*POINT modnam	Locates the specified object module (modnam) and prepares it for subsequent WORD, BYTE, or DUMP operations.
WORD	$\text{WORD}\ \begin{matrix}\text{CSECT + OFFSET}\\ \text{NAME}\end{matrix} = \left\{\begin{matrix}\#\\ \%\end{matrix}\right\}\left\{\begin{matrix}\text{NAME OF}\\ \text{CSECT OR}\\ \text{GLOBAL}\end{matrix}\right\}\left\{\begin{matrix}+\\ -\end{matrix}\right\}\text{OFFSET}$	Modifies a given word in an object module.

APPENDIX C

MACRO ASSEMBLER, INSTRUCTION, AND CHARACTER CODE SUMMARIES

C.1 ASCII CHARACTER SET

Even Parity Bit	7-Bit Octal Code	Character	Remarks
0	000	NUL	Null, Tape Feed, CTRL SHIFT P.
1	001	SOH	Start of Heading; also SOM (Start Of Message), CTRL A.
1	002	STX	Start of Text; also EOA (End Of Address), CTRL B.
0	003	ETX	End of Text; also EOM (End Of Message), CTRL C.
1	004	EOT	End of Transmission (END); Shuts off TWX machines, CTRL D.
0	005	ENQ	Enquiry (ENQRY); also WRU, CTRL E.
0	006	ACK	Acknowledge; also RU, CTRL F.
1	007	BEL	Rings the Bell. CTRL G.
1	010	BS	Backspace; also FEO, Format Effector. Backspaces some machines, CTRL H.
0	011	HT	Horizontal TAB. CTRL I.
0	012	LF	Line Feed or Line Space (New Line); Advances paper to next line; duplicated by CTRL J.
1	013	VT	Vertical TAB (VTAB). CTRL K.
0	014	FF	FORM FEED to top of next page (PAGE). CTRL L.
1	015	CR	Carriage Return to beginning of line. Duplicated by CTRL M.
1	016	SO	Shift Out; Changes ribbon color to red. CTRL N.
0	017	SI	Shift In; Changes ribbon color to black. CTRL O.
1	020	DLE	Data Link Escape. CTRL B (DC0).
0	021	DC1	Device Control 1, turns transmitter (reader) on, CTRL Q (X ON).
0	022	DC2	Device Control 2, turns punch or auxiliary on, CTRL R (TAPE, AUX ON).
1	023	DC3	Device Control 3, turns transmitter (reader) off, CTRL S (X OFF).
0	024	DC4	Device Control 4, turns punch or auxiliary off, CTRL T (AUX OFF).
1	025	NAK	Negative Acknowledge; also ERR, Error, CTRL U.

1	026	SYN	Synchronous File (SYNC), CTRL V.
0	027	ETB	End of Transmission Block; also LEM, Logical End of Medium, CTRL W.
0	030	CAN	Cancel (CANCL), CTRL X.
1	031	EM	End of Medium, CTRL Y.
1	032	SUB	Substitute, CTRL Z.
0	033	ESC	Escape, CTRL SHIFT K.
1	034	FS	File Separator, CTRL SHIFT L.
0	035	GS	Group Separator, CTRL SHIFT M.
0	036	RS	Record Separator, CTRL SHIFT N.
1	037	US	Unit Separator, CTRL SHIFT O.
1	040	SP	Space.
0	041	!	
0	042	"	
1	043	#	
0	044	$	
1	045	%	
0	046	&	
0	047	'	Apostrophe or Acute Accent.
0	050	(	
0	051	)	
1	052	*	
0	053	+	
1	054	,	
0	055	-	
0	056	.	
1	057	/	
0	060	0	
1	061	1	
1	062	2	
0	063	3	
1	064	4	
0	065	5	
0	066	6	
1	067	7	
1	070	8	
0	071	9	
0	072	:	
1	073	;	
0	074	<	
1	075	=	
1	076	>	
0	077	?	
1	100	@	
0	101	A	
0	102	B	
1	103	C	
0	104	D	
1	105	E	
1	106	F	
0	107	G	
0	110	H	
1	111	I	
1	112	J	
0	113	K	
1	114	L	
0	115	M	
0	116	N	
1	117	O	
0	120	P	
1	121	Q	

1	122	R	
0	123	S	
1	124	T	
0	125	U	
0	126	V	
1	127	W	
1	130	X	
0	131	Y	
0	132	Z	
1	133	[	SHIFT K.
0	134	\	SHIFT L.
1	135	]	SHIFT M.
1	136	↑	(Appears as ^ on some machines).
0	137	←	(Appears as ___ (Underscore) on some machines).
0	140	`	Accent Grave.
1	141	a	
1	142	b	
0	143	c	
1	144	d	
0	145	e	
0	146	f	
1	147	g	
1	150	h	
0	151	i	
0	152	j	
1	153	k	
0	154	l	
1	155	m	
1	156	n	
0	157	o	
1	160	p	
0	161	q	
0	162	r	
1	163	s	
0	164	t	
1	165	u	
1	166	v	
0	167	w	
0	170	x	
1	171	y	
1	172	z	
0	173	{	
1	174	|	
0	175	}	This Code Generated by ALTMODE.
0	176	~	This Code Generated by PREFIX key (if Present)
1	177	DEL	DELETE, RUBOUT.

C.2 RADIX-50 CHARACTER SET

Character	ASCII Octal Equivalent	Radix-50 Equivalent
space	40	0
A-Z	101-132	1-32

$	44	33
.	56	34
unused		35
0-9	60-71	36-47

The maximum Radix-50 value is, thus:

$$47*50^{2}+47*50+47=174777$$

The following table provides a convenient means of translating between the ASCII character set and its Radix-50 equivalents. For example, given the ASCII string X2B, the Radix-50 equivalent is (arithmetic is performed in octal):

```
 X=113000
 2=002400
 B=000002
---------
X2B=115402
```

Single Char. or First Char.		Second Character		Third Character	
A	003100	A	000050	A	000001
B	006200	B	000120	B	000002
C	011300	C	000170	C	000003
D	014400	D	000240	D	000004
E	017500	E	000310	E	000005
F	022600	F	000360	F	000006
G	025700	G	000430	G	000007
H	031000	H	000500	H	000010
I	034100	I	000550	I	000011
J	037200	J	000620	J	000012
K	042300	K	000670	K	000013
L	045400	L	000740	L	000014
M	050500	M	001010	M	000015
N	053600	N	001060	N	000016
O	056700	O	001130	O	000017
P	062000	P	001200	P	000020
Q	065100	Q	001250	Q	000021
R	070200	R	001320	R	000022
S	073300	S	001370	S	000023
T	076400	T	001440	T	000024
U	101500	U	001510	U	000025
V	104600	V	001560	V	000026
W	107700	W	001630	W	000027
X	113000	X	001700	X	000030
Y	116100	Y	001750	Y	000031
Z	121200	Z	002020	Z	000032
$	124300	$	002070	$	000033
.	127400	.	002140	.	000034
	132500		002210		000035
0	135600	0	002260	0	000036
1	140700	1	002330	1	000037
2	144000	2	002400	2	000040

3	147100	3	002450	3	000041
4	152200	4	002520	4	000042
5	155300	5	002570	5	000043
6	160400	6	002640	6	000044
7	163500	7	002710	7	000045
8	166600	8	002760	8	000046
9	171700	9	003030	9	000047

C.3 MACRO SPECIAL CHARACTERS

Character	Function
form feed	Source line terminator, forces a new listing page
line feed	Source line terminator
carriage return	Formatting character
vertical tab	Source line terminator
:	Label terminator
=	Direct assignment indicator
%	Register term indicator
tab	Item terminator, field terminator
space	Item terminator, field terminator
#	Immediate expression indicator
@	Deferred addressing indicator
(	Initial register indicator
)	Terminal register indicator
, (comma)	Operand field separator
;	Comment field indicator
+	Arithmetic addition operator or autoincrement indicator
-	Arithmetic subtraction operator or autodecrement indicator
*	Arithmetic multiplication operator
/	Arithmetic division operator
&	Logical AND operator
!	Logical OR operator
"	Double ASCII character indicator
' (apostrophe)	Single ASCII character indicator
.	Assembly location counter
<	Initial argument indicator
>	Terminal argument indicator
↑	Universal unary operator
↑	Argument indicator
\	MACRO numeric argument indicator

C.4 ADDRESS MODE SYNTAX

In the following syntax table, n represents an integer between 0 and 7; R is a register expression; E represents any expression; ER represents either a register expression or an absolute expression in the range 0 to 7.

0n	Register	R	Register R contains the operand. R is a register expression.
1n	Deferred Register	@R or (R)	Register R contains the operand address.

2n	Autoincrement	(ER)+	The contents of the register specified by ER are incremented after being used as the address of the operand.
3n	Deferred Autoincrement	@(ER)+	ER contains a pointer to the address of the operand. ER is incremented after use.
4n	Autodecrement	-(ER)	The contents of register ER are decremented before being used as the address of the operand.
5n	Deferred Autodecrement	@-(ER)	The contents of register ER are decremented before being used as a pointer to the address of the operand.
6n	Index by the Register Specified	E(ER)	The value obtained when E is combined with the contents of the register specified (ER) is the address of the operand.
7n	Deferred index by the Register Specified	@E(ER)	E added to ER produces a pointer to the address of the operand.
27	Immediate Operand	#E	E is the operand.
37	Absolute address	@#E	E is the operand address.
67	Relative address	E	E is the address of the operand.
77	Deferred relative address	@E	E is a pointer to the address of the operand.

C.5 INSTRUCTIONS

The tables of instructions which follow are grouped according to the operands they take and according to the bit patterns of their op-codes.

The following symbols are used to indicate the instruction type format:

OP	Instruction mnemonic
R	Register Expression
E	Expression
ER	Register expression or expression 0<=ER<=7
AC	Floating point register expression
A	General address specification

In the representation of op-codes, the following symbols are used:

SS	Source operand	Specified by a 6-bit address mode
DD	Destination operand	Specified by a 6-bit address mode
XX	8-bit offset to a location	Branch instructions
R	Integer between 0 and 7	Represents a general register

Symbols used in the description of instruction operations are:

SE   Source Effective Address
FSE   Floating Source Effective Address
DE   Destination Effective Address
FDE   Floating Destination Effective Address
| | Absolute Value of
( )   Contents of
→   Becomes

The condition codes in the processor status word (PS) are affected by the instructions; these condition codes are represented as follows:

N	Negative bit:	Set if the result is negative
Z	Zero bit:	Set if the result is zero
V	Overflow bit:	Set if the operation caused an overflow
C	Carry bit:	Set if the operation caused a carry

In the representation of the instruction's effect on the condition codes, the following symbols are used:

*	Conditionally set
-	Not affected
0	Cleared
1	Set

To set conditionally means to use the instruction's result to determine the state of the code.

Logical operators are represented by the following symbols:

!	Inclusive OR
!	Exclusive OR
&	AND
—	Used over a symbol to represent the 1's complement of the symbol

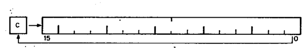
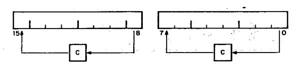
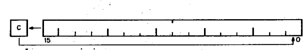
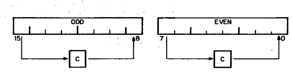
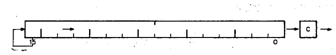
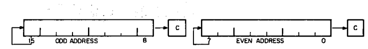
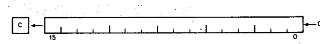
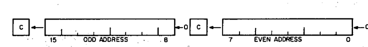
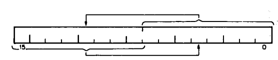
C.5.1 Double Operand Instructions (OP A,A)

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
01SSDD 11SSDD	MOV MOVB	MOVe MOVe Byte	(SE) → (DE)	*	*	0	-
02SSDD 12SSDD	CMP CMPB	CoMPare CoMPare Byte	(SE) - (DE)	*	*	*	*
03SSDD 13SSDD	BIT BITB	BIt Test BIt Test Byte	(SE) & (DE)	*	*	0	-
04SSDD 14SSDD	BIC BICB	BIt Clear BIt Clear Byte	$\overline{\text{(SE)}}$ & (DE) → (DE)	*	*	0	-
05SSDD 15SSDD	BIS BISB	BIt Set BIt Set Byte	(SE) ! (DE) → (DE)	*	*	0	-
06SSDD 16SSDD	ADD SUB	ADD SUBtract	(SE) + (DE) → (DE) (DE) - (SE) → (DE)	* *	* *	* *	* *

C.5.2 Single Operand Instructions (OP A)

Op-code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
0050DD 1050DD	CLR CLRB	CLeaR CLeaR Byte	0 → (DE)	0	1	0	0
0051DD 1051DD	COM COMB	COMplement COMplement Byte	$\overline{\text{(DE)}}$ → (DE)	*	*	0	1
0052DD 1052DD	INC INCB	INCrement INCrement Byte	(DE) + 1 → (DE)	*	*	*	1
0053DD 1053DD	DEC DECB	DECrement DECrement Byte	(DE) - 1 → (DE)	*	*	*	-
0054DD 1054DD	NEG NEGB	NEGate NEGate Byte	$\overline{\text{(DE)}}$ + 1 → (DE)	*	*	*	*
0055DD 1055DD	ADC ADCB	ADd Carry ADd Carry Byte	$\overline{\text{(DE)}}$ + (C) → (DE)	*	*	*	*
0056DD 1056DD	SBC SBCB	SuBtract Carry SuBtract Carry Byte	(DE) - (C) → (DE)	*	*	*	*
0057DD 1057DD	TST TSTB	TeST TeST Byte	(DE)	*	*	0	0

C.5.3 Rotate/Shift

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
0060DD	ROR	ROtate Right		*	*	*	*
1060DD	RORB	ROtate Right Byte		*	*	*	*
0061DD	ROL	ROtate Left		*	*	*	*
1061DD	ROLB	ROtate Left Byte		*	*	*	*
0062DD	ASR	Arithmetic Shift Right		*	*	*	*
1062DD	ASRB	Arithmetic Shift Right Byte		*	*	*	*
0063DD	ASL	Arithmetic Shift Left		*	*	*	*
1063DD	ASLB	Arithmetic Shift Left Byte		*	*	*	*
0001DD	JMP	JuMP	DE → (PC)	-	-	-	-
0003DD	SWAB	SWAp Bytes		*	*	0	0

The following instructions are available on the PDP-11/03 (LSI/11) only:

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
1067DD	MFPS	Move byte From Processor Status word	(DE)←PSW	*	*	0	-
1064SS	MTPS	Move byte To Processor Status word	(SE)→PSW	*	*	*	*

The following instructions are available on the PDP-11/35,40,45 as noted:

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
11/35, 11/40, 11/45 with KT11							
0065SS	MFPI	Move From Previous Instruction space	(SE) → (TEMP) (SP)-2 → (SP) (TEMP) → ((SP))	*	*	0	-

11/45 with KT11 only

1065SS	MFPD	Move From Previous Data space	(SE) → (TEMP) (SP)-2 → (SP) (TEMP) → ((SP))	*	*	0	-

11/35, 11/40, 11/45 with KT11

0066DD	MTPI	Move To Previous Instruction space	((SP)) → (TEMP) (SP+2) → (SP) (TEMP) → (DE)	*	*	0	-

11/45 with KT11 only

1066DD	MTPD	Move To Previous Data space	((SP)) → (TEMP) (SP+2) → (SP) (TEMP) → (DE)	*	*	0	-
1701DD	LDFPS	Load FPP program status	DE → FPS	-	-	-	-

11/35, 11/40, 11/45

0067DD	SXT	Sign eXTend	0 → DE if N bit is clear -1 → DE if N bit is set	-	*	-	-

11/45 with FP11-B

0707DD	NEGD	NEGate Double	-(FDE) → FDE				
0707DD	NEGF	NEGate Floating	-(FDE) → FDE	*	*	0	0
1702DD	STFPS	STore Floating Point processor program Status	See Chapter 7 in PDP-11/45 Processor Handbook	-	-	-	-
1703DD	STST	STore floating point processor STatus		-	-	-	-
1704DD	CLRD	CLeaR Double	0 → (FDE)	0	1	0	0
1704DD	CLRF	CLeaR Floating	0 → (FDE)	0	1	0	0
1705DD	TSTD	TeST Double	(FDE)	*	*	0	0
1705DD	TSTF	TeST Floating	(FDE)	*	*	0	0
1706DD	ABSD	make ABSolute Double	|(FDE)| → (FDE)	0	*	0	0
1706DD	ABSF	make ABSolute Floating	|(FDE)| → (FDE)	0	*	0	0

C.5.4 Operate Instructions (OP)

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
000000	HALT	HALT	The computer stops all functions.	-	-	-	-
000001	WAIT	WAIT	The computer stops and waits for an interrupt.	-	-	-	-
000002	RTI	ReTurn from Interrupt (Return from Trap)	The PC and ST are popped off the SP stack: ((SP)) → (PC) (SP)+2 → (SP) ((SP)) → (ST) (SP)+2 → (SP)	*	*	*	*
000005	RESET	RESET	Returns all I/O devices to power-on state.	-	-	-	-
000241	CLC	CLear Carry bit	0 → C	-	-	-	0
000261	SEC	SEt Carry bit	1 → C	-	-	-	1
000242	CLV	CLear oVerflow	0 → V	-	-	0	-
000262	SEV	SEt oVerflow bit	1 → V	-	-	1	-
000244	CLZ	CLear Zero bit	0 → Z	-	0	-	-
000264	SEZ	SEt Zero bit	1 → Z	-	1	-	-
000250	CLN	CLear Negative bit	0 → N	0	-	-	-
000270	SEN	SEt Negative bit	1 → N	1	-	-	-
000257	CCC	Clear all Condition Codes	0 → N 0 → Z 0 → V 0 → C	0	0	0	0
000277	SCC	Set all Condition Codes	1 → N 1 → Z 1 → V 1 → C	1	1	1	1
000240	NOP	No OPeration		-	-	-	-

The following instructions are available on the PDP-11/45 with FP11-B only:

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
170000	CFCC	Copy Floating Condition Code	Copy FPP condition codes into CPU condition codes.	*	*	*	*
170011	SETD	SET Double floating mode	FPP set to double precision	-	-	-	-
170001	SETF	SET Floating mode	FPP set to single precision mode	-	-	-	-
170002	SETI	SET Integer mode	FPP set for integer data (16 bits)	-	-	-	-
170012	SETL	SET Long integer mode	FPP set for long integer data (32 bits)	-	-	-	-

All 11/45's, with and without FP11-B

Op-Code	Mnemonic	Stands for	Operation	N	Z	V	C
000006	RTT	ReTurn from inTerrupt	Same as RTI instruction but inhibits trace trap	*	*	*	*

C.5.5 Trap Instructions (OP or OP e where 0<=E<=377(8)) *(OP (only))

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
*000003	BPT	BreakPoint Trap	Trap to location 14. This is used to call ODT.	*	*	*	*
*000004	IOT	Input Output Trap	Trap to location 20. This is used to call IOX.	*	*	*	*

104000- 104377	EMT	EMulator Trap	Trap to location 30. This is used to call system programs.	*	*	*	*
104400- 104777	TRAP	TRAP	Trap to location 34. This is used to call any routine desired by the programmer.	*	*	*	*

C.5.6 Branch Instructions OP E

(where -128(decimal)<(E-.-2)/2<127(decimal))

Op-Code	Mnemonic	Stands for	Condition to be met if branch is to occur
0004XX	BR	BRanch always	
0010XX	BNE	Branch if Not Equal (to zero)	Z=0
0014XX	BEQ	Branch if EQual (to zero)	Z=1
0020XX	BGE	Branch if Greater than or Equal (to zero)	N ! V=0
0024XX	BLT	Branch if Less Than (zero)	N ! V = 1
0030XX	BGT	Branch if Greater Than (zero)	Z!(N ! V)=0
0034XX	BLE	Branch if Less than or Equal (to zero)	Z!(N ! V)=1
1000XX	BPL	Branch if PLus	N=0
1004XX	BMI	Branch if MInus	N=1
1010XX	BHI	Branch if HIgher	C ! Z=0
1014XX	BLOS	Branch if LOwer or Same	C!Z=1
1020XX	BVC	Branch if oVerflow Clear	V=0
1024XX	BVS	Branch if oVerflow Set	V=1
1030XX	BCC (or BHIS)	Branch if Carry Clear (or Branch if HIgh or Same)	C=0
1034XX	BCS (or BLO)	Branch if Carry Set (or Branch if LOw)	C=1

C.5.7 Register Destination (OP ER,A)

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N Z V C
004RDD	JSR	Jump to SubRoutine	Push register on the SP stack, put the PC in the register: DE TEMP (TEMP= temporary storage register internal to processor.) (SP)-2 → SP (REG) → (SP) (PC) → REG (TEMP) → PC	- - - -

The following instruction is available only on the 11/35, 11/40, 11/45:

Op-Code	Mnemonic	Stands for	Operation	N Z V C
074RDD	XOR	eXclusive OR	(R) (!) (DE) → (DE)	* * 0 -

C.5.8 Register-Offset (OP R,E)

The following instruction is available only on the PDP-11/35, 11/40, 11/45:

Op-Code	Mnemonic	Stands for	Operation	N Z V C
077RDD	SOB	Subtract One and Branch	(R)-1→(R) (PC)-(2*DE) → (PC)	- - - -

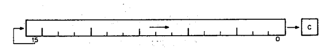
C.5.9 Subroutine Return (OP ER)

Op-Code	Mnemonic	Stands for	Operation	N Z V C
00020R	RTS	ReTurn from Subroutine	Put register in PC and pop old contents from SP stack into register.	- - - -

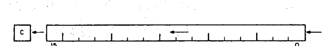
C.5.10 Source-Register (OP A,R)

The following instructions are available on the 11/35, 11/40, 11/45 only:

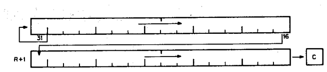
Op-Code	Mnemonic	Stands for	Operation	Status Word Floating Condition Codes N	Z	V	C
071RSS	DIV	DIVide	(R),(R!1)/(SRC) → (R),(R!1)	*	*	*	*
070RSS	MUL	MULtiply	(R)*(SRC) → (R),(R!1)	*	*	*	*
072RSS	ASH	Arithmetic SHift	R is shifted according to low-order 6-bits of source				



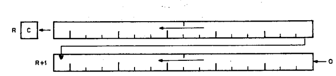
or



Op-Code	Mnemonic	Stands for	Operation	N	Z	V	C
073RSS	ASHC	Arithmetic SHift Combined	R,RVL are shifted according to low-order 6 bits of source	*	*	*	*



or



C.5.11 Floating-Point Source Double Register (OP A,AC)

The following instructions are available on the PDP-11/45 with FP11-B only:

Op-Code	Mnemonic	Stands for	Operation	Status Word Floating Condition Codes FN	FZ	FV	FC
172(AC)SS	ADDD	ADD Double	(FSE) +(AC) → (AC)	*	*	*	0

172(AC)SS	ADDF	ADD Floating	(FSE) +(AC) → (AC)	*	*	*	0
173(AC+4)SS	CMPD	CoMPare Double	(FSE) -(AC)	*	*	0	0
173(AC+4)SS	CMPF	CoMPare Floating	(FSE) -(AC)	*	*	0	0
174(AC+4)SS	DIVD	DIVide Double	(AC)/(FSE) → (AC)	*	*	*	0
174(AC+4)SS	DIVF	DIVide Floating	(AC)/(FSE) → (AC)	*	*	*	0
177(AC+4)SS	LDCDF	LoaD and Convert from Double to Floating	(FSE) → (AC)	*	*	*	0
177(AC+4)SS	LDCFD	LoaD and Convert from Floating to Double	(FSE) → (AC)	*	*	*	0
172(AC+4)SS	LDD	LoaD Double	(FSE) → (AC)	*	*	0	0
172(AC+4)SS	LDF	LoaD Floating	(FSE) → (AC)	*	*	0	0
171(AC+4)SS	MODD	Multiply and integerize double	(AC)*(FSE) → (AC),(AC1)	*	*	*	*
171(AC+4)SS	MODF	Multiply and integerize floating-point	(AC)*(FSE) → (AC)	*	*	*	0
171(AC)SS	MULD	MULtiply Double	(AC)*(FSE) → (AC)	*	*	*	0
171(AC)SS	MULF	MULtiply Floating	(AC)*(FSE) → (AC)	*	*	*	0
173(AC)SS	SUBD	SUBtract Double	(FSE) -(AC) → (AC)	*	*	*	0
173(AC)SS	SUBF	SUBtract Floating	(FSE) -(AC) → (AC)	*	*	*	0

C.5.12 Source-Double Register (OP A,AC)

The following instructions are available on the PDP-11/45 with FP11-B only:

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes FN	FZ	FV	FC
177(AC)SS	LDCID	LoaD and Convert Integer to Double	(SE) → (AC)	*	*	*	0
177(AC)SS	LDCIF	LoaD and Convert Integer to Floating	(SE) → (AC)	*	*	*	0
177(AC)SS	LDCLD	LoaD and Convert Long integer to Double	(SE) → (AC)	*	*	*	0
177(AC)SS	LDCLF	LoaD and Convert Long Integer to Floating	(SE) → (AC)	*	*	*	0
176(AC+4)SS	LDEXP	LoaD EXPonent	(SE)+200 → (AC EXP)	*	*	0	0

C.5.13 Double Register-Destination (OP AC,A)

The following instructions are available on the PDP-11/45 with FP11-B only:

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes FN	FZ	FV	FC
176(AC)DD	STCFD	STore, Convert from Floating to Double	(AC) → (FDE)	*	*	*	0
176(AC)DD	STCDF	STore, Convert from Double to Floating	(AC) → (FDE)	*	*	*	0
175(AC+4)DD	STCDI(1)	STore, Convert from Double to Integer	(AC) → (FDE)	*	*	0	*

(1) These instructions set both the floating-point and processor condition codes as indicated.

175(AC+4)DD STCDL(1)	STore, Convert from Double to Long integer	(AC) → (FDE)	*	*	0	*
175(AC+4)DD STCFI(1)	STore, Convert from Floating to Integer	(AC) → (FDE)	*	*	0	*
174(AC+4)DD STCFL(1)	STore, Convert from Floating to Long integer	(AC) → (FDE)	*	*	0	*
174(AC)DD STD	STore Double	(AC) → (FDE)	-	-	-	-
174(AC)DD STF	STore Floating	(AC) → (FDE)	-	-	-	-
175(AC)DD STEXP(1)	STore EXPonent	(AC EXP)-200 → (DE)	*	*	0	0

C.5.14 Number

The following instruction is available on the 11/35, 11/40, 11/45 only:

Op-Code	Mnemonic	Stands for	Operation	Status Word Condition Codes N	Z	V	C
0064NN	MARK	MARK	Stack cleanup on return from subroutine.	-	-	-	-

C.5.15 Priority

The following instruction is available on the PDP-11/45 only:

Op-Code	Mnemonic	Stands for	Operations	Status Word Condition Codes N	Z	V	C
00023N	SPL	Set Priority Level	(X) → (PS) (bits 7-5)	-	-	-	-

(1)These instructions set both the floating-point and processor condition codes as indicated.

C.6 ASSEMBLER DIRECTIVES

Form	Described in Manual Section	Operation
'	5.5.3.3	A single quote character (apostrophe) followed by one ASCII character generates a word containing the 7-bit ASCII representation of the character in the low-order byte and zero in the high-order byte.
"	5.5.3.3	A double quote character followed by two ASCII characters generates a word containing the 7-bit ASCII representation of the two characters.
\	5.6.3.3	A backslash preceding an argument causes the number to be treated in the current radix.
↑Bn	5.5.4.2	Temporary radix control; causes the number n to be treated as a binary number.
↑Cn	5.5.6.2	Creates a word containing the one's complement of n.
↑Dn	5.5.4.2	Temporary radix control; causes the number n to be treated as a decimal number.
↑Fn	5.5.6.2	Creates a one-word floating point quantity to represent n.
↑On	5.5.4.2	Temporary radix control; causes the number n to be treated as an octal number.
.ASCII string	5.5.3.4	Generates a block of data containing the ASCII equivalent of the character string (enclosed in delimiting characters) one character per byte.
.ASCIZ string	5.5.3.5	Generates a block of data containing the ASCII equivalent of the character string (enclosed in delimiting characters) one character per byte with a zero byte following the specified string.
.ASECT	5.5.9	Begin or resume absolute section.
.BLKB exp	5.5.5.3	Reserves a block of storage space exp bytes long.
.BLKW exp	5.5.5.3	Reserves a block of storage space exp words long.

.BYTE exp1, exp2,...	5.5.3.1	Generates successive bytes of data containing the octal equivalent of the expression(s) specified.
.CSECT symbol	5.5.9	Begins or resumes named or unnamed relocatable section.
.DSABL arg	5.5.2	Disables the assembler function specified by the argument.
.ENABL arg	5.5.2	Provides the assembler function specified by the argument.
.END .END exp	5.5.7.1	Indicates the physical end of the source program. An optional argument specifies the transfer address.
.ENDC	5.5.11	Indicates the end of a condition block.
.ENDM .ENDM symbol	5.6.1.2	Indicates the end of the current repeat block, indefinite repeat block, or macro. The optional symbol, if used, must be identical to the macro name.
.EOT	5.5.7.2	Ignored. Indicates End-of-Tape which is detected automatically by the hardware.
.ERROR exp,string	5.6.5	Causes a text string to be output to the listing containing the optional expression specified and the indicated text string. The line will be flagged with the "P" error code.
.EVEN	5.5.5.1	Ensures that the assembly location counter contains an even address by adding 1 if it is odd.
.FLT2 arg1, arg2,...	5.5.6.1	Generates successive two-word floating point equivalents for the floating-point numbers specified as arguments.
.FLT4 arg1, arg2,...	5.5.6.1	Generates successive four-word floating point equivalents for the floating-point numbers specified as arguments.
.GLOBL sym1, sym2,...	5.5.10	Defines the symbol(s) specified as global symbol(s).
.IDENT symbol	5.5.1.5	Provides a means of labeling the object module produced as a result of assembly. This directive is not supported by RT-11, but is included for compatibility with other systems.
.IF cond, arguments	5.5.11	Begins a conditional block of source code which is included in the assembly

		only if the stated condition is met with respect to the argument(s) specified.
.IFF	5.5.11.1	Appears only within a conditional block and indicates the beginning of a section of code to be assembled if the condition tested false.
.IFT	5.5.11.1	Appears only within a conditional block and indicates the beginning of a section of code to be assembled if the condition tested true.
.IFTF	5.5.11.1	Appears only within a conditional block and indicates the beginning of a section of code to be unconditionally assembled.
.IIF cond,arg, statement	5.5.11.2	Acts as a one-line conditional block where the condition is tested for the argument specified. The statement is assembled only if the condition tests true.
.IRP sym, <arg1,arg2,...>	5.6.6	Indicates the beginning of an indefinite repeat block in which the symbol specified is replaced with successive elements of the real argument list (which is enclosed in angle brackets).
.IRPC sym,string	5.6.6	Indicates the beginning of an indefinite repeat block in which the symbol specified takes on the value of successive characters in the character string.
.LIMIT	5.5.8	Reserves two words into which the Linker inserts the low and high addresses of the relocated code.
.LIST .LIST arg	5.5.1.1	Without an argument, .LIST increments the listing level count by 1. With an argument, .LIST does not alter the listing level count but formats the assembly listing according to the argument specified.
.MACRO sym,arg1, arg1,...	5.6.1.1	Indicates the start of a macro with the specified name containing the dummy arguments specified.
.MCALL	5.6.8	Used to specify the names of all system macro definitions not defined in the current program but required by the program.
.MEXIT	5.6.1.3	Causes an exit from the current macro or indefinite repeat block.

.NARG symbol	5.6.4	Appears only within a macro definition and equates the specified symbol to the number of arguments in the macro call currently being expanded.
.NCHR sym,<string>	5.6.4	Can appear anywhere in a source program; equates the symbol specified to the number of characters in the string (enclosed in delimiting characters).
.NLIST .NLIST arg	5.5.1.1	Without an argument, .NLIST decrements the listing level count by 1. With an argument, .NLIST deletes the portion of the listing indicated by the argument.
.NTYPE symbol,arg	5.6.4	Appears only in a macro definition and equates the low-order six bits of the symbol specified to the six-bit addressing mode of the argument.
.ODD	5.5.5.2	Ensures that the assembly location counter contains an odd address by adding 1 if it is even.
.PAGE	5.5.1.6	Causes the assembly listing to skip to the top of the next page.
.PRINT exp,string	5.6.5	Causes a text string to be output to the listing containing the optional expression specified and the indicated text string.
.RADIX n	5.5.4.1	Alters the current program radix to n, where n can be 2, 4, 8, or 10.
.RAD50 string	5.5.3.6	Generates a block of data containing the Radix-50 equivalent of the character string (enclosed in delimiting characters).
.REPT exp	5.6.7	Begins a repeat block. Causes the section of code up to the next .ENDM or .ENDR to be repeated exp times.
.SBTTL string	5.5.1.4	Causes the string to be printed as part of the assembly listing page header. The string part of each .SBTTL directive is collected into a table of contents at the beginning of the assembly listing.
.TITLE string	5.5.1.3	Assigns the first symbolic name in the string to the object module and causes the string to appear on each page of the assembly listing. One .TITLE directive should be issued per program.

.WORD exp1, exp2,...	5.5.3.2	Generates successive words of data containing the octal equivalent of the expression(s) specified.

C.7 MACRO/CREF SWITCHES

C.7.1 Listing Control Switches

Switch	Meaning
/L:arg /N:arg	These switches are used to control listing output. Arguments which are valid for either switch include:

Arg	Controls Listing of:
SEQ	Source line sequence numbers
LOC	Location counter
BIN	Generated binary code
BEX	Binary extensions
SRC	Source code
COM	Comments
MD	Macro definitions and repeat range definitions
MC	Macro calls and repeat range expansions
ME	Macro expansions
MEB	Macro expansion binary code
CND	Unsatisfied conditions and all .IF and .ENDC statements.
LD	Listing directives having no arguments
TOC	Table of contents
TTM	Listing output format
SYM	Symbol table
<no arg>	/N with no argument causes only table of contents, symbol table, and error listings to be produced. /L with no argument causes .LIST and .NLIST directives without arguments which appear in the source program to be ignored.

C.7.2 Function Control Switches

Switch	Meaning
/D:arg /E:arg	These switches are used to enable or disable certain functions in source input files. Valid arguments include:

Arg	Enables or Disables:
ABS	Absolute binary output
AMA	Assembly of all absolute addresses as relative addresses
CDR	Source columns 73 and greater to be treated as comments
FPT	Floating point truncation
LC	Accepts lower case ASCII input
LSB	Local symbol block
PNC	Binary output

C.7.3 CREF Switches

Switch	Produces Cross-Reference of:
/C:S	User-defined symbols
/C:R	Register symbols
/C:M	MACRO symbolic names
/C:P	Permanent symbols
/C:C	Control sections
/C:E	Error codes
/C:<no arg>	Equivalent to /C:S:M:E

C.8 OCTAL-DECIMAL CONVERSIONS

0000 to 0777 (Octal) | 0000 to 0511 (Decimal)

Octal	Decimal
10000	4096
20000	8192
30000	12288
40000	16384
50000	20480
60000	24576
70000	28672

	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255

	0	1	2	3	4	5	6	7
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

1000 to 1777 (Octal) | 0512 to 1023 (Decimal)

	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0649	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767

	0	1	2	3	4	5	6	7
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0852	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

2000 to 2777 (Octal)	1024 to 1535 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2110	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1509	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

3000 to 3777 (Octal)	1536 to 2047 (Decimal)

	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

4000 to 4777 (Octal) | 2048 to 2559 (Decimal)

Octal Decimal
10000 - 4096
20000 - 8192
30000 - 12288
40000 - 16384
50000 - 20480
60000 - 24576
70000 - 28672

	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	2537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2559

5000 to 5777 (Octal) | 2560 to 3071 (Decimal)

	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2568	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2663
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2691	2692	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2760	2761	2762	2763	2764	2765	2766	2767
5320	2768	2769	2770	2771	2772	2773	2774	2775
5330	2776	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815

	0	1	2	3	4	5	6	7
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2886	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	3058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

6000 to 6777 (Octal) | 3072 to 3583 (Decimal)

Octal	Decimal
10000	4096
20000	8192
30000	12288
40000	16384
50000	20480
60000	24576
70000	28672

	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

7000 to 7777 (Octal) | 3584 to 4095 (Decimal)

	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3596	3597	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	1022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

APPENDIX D

SYSTEM MACRO FILE

The following is a listing of the system macro library, SYSMAC.SML. This file is stored on the system device, and used by MACRO when it expands the programmed requests discussed in Chapter 9.

```
; SYSMAC.SML--SYSTEM MACRO LIBRARY
; FOR RT11 V2C.
;
; DEC-11-ORSYA-E
;
; COPYRIGHT (C) 1974,1975
;
; DIGITAL EQUIPMENT CORPORATION
; MAYNARD, MASSACHUSETTS 01754
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
; ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH
; THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE,
; OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE
; AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO
; ONE WHO AGREES TO THESE LICENSE TERMS. TITLE TO AND OWNERSHIP OF THE
; SOFTWARE SHALL AT ALL TIMES REMAIN IN DIGITAL.
;
; THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO
; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
;
; DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE
; OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
; WHICH IS NOT SUPPLIED BY DIGITAL.
;
; EF,JD
;
```

```
.MACRO ..V1..
...V1=1
.ENDM

.MACRO ..V2..
.MCALL  ...CM1,...CM2,...CM3,...CM4
...V2=1
.ENDM


.MACRO ...CM1 .AREA,.CODE,.CHAN
.IF NB .AREA
                        MOV     .AREA,%0
                        MOVB    #.CODE,1(0)
.ENDC
.IF NB .CHAN
        .IF IDN <.CHAN>,<#0>
                        CLRB    (0)
        .IFF
                        MOVB    .CHAN,(0)
        .ENDC
.ENDC
.ENDM

.MACRO ...CM2 .ARG,.OFFSET,.INS
.IIF NB <.ARG>,         MOV     .ARG,.OFFSET(0)
.IIF NB <.INS>,         EMT     ^O375
.ENDM

.MACRO ...CM3 .CHAN,.CODE
                        MOV     #.CODE*^O400,%0
.IIF NB <.CHAN>,        BISB    .CHAN,%0
                        EMT     ^O374
.ENDM

.MACRO ...CM4 .AREA,.CHAN,.BUFF,.WCNT,.BLK,.CRTN,.CODE
...CM1 <.AREA>,<.CODE>,<.CHAN>
...CM2 <.BLK>,2.
...CM2 <.BUFF>,4.
...CM2 <.WCNT>,6.
...CM2 <.CRTN>,8.,X
.ENDM

.MACRO .CDFN .AREA,.ADD,.NUM
...CM1 <.AREA>,13.,#0
...CM2 <.ADD>,2.
...CM2 <.NUM>,4.,X
.ENDM

.MACRO .CHAIN
...CM3 ,8.
.ENDM

.MACRO .CHCOPY .AREA,.CHAN,.OCHAN
...CM1 <.AREA>,11.,<.CHAN>
...CM2 <.OCHAN>,2.,X
.ENDM
```

```
.MACRO .CNTXSW .AREA,.ADD
...CM1 <.AREA>,27.,#0
...CM2 <.ADD>,2.,X
.ENDM

.MACRO .CMKT .AREA,.ID,.TIME
...CM1 <.AREA>,19.,#0
...CM2 <.ID>,2.
.IF B .TIME
                        CLR     4.(0)
.IFF
                        MOV     .TIME,4.(0)
.ENDC
                        EMT     ^O375
.ENDM

.MACRO .CLOSE .CHAN
.IF DF ...V1
                        EMT     ^O<160+.CHAN>
.IFF
...CM3 <.CHAN>,6.
.ENDC
.ENDM

.MACRO .CSIGEN .DEVSPC,.DEFEXT,.CSTRING
                        MOV     .DEVSPC,-(6.)
                        MOV     .DEFEXT,-(6.)
.IF B .CSTRING
                        CLR     -(6.)
.IFF
                        MOV     .CSTRING,-(6.)
.ENDC
                        EMT     ^O344
.ENDM


.MACRO .CSISPC .OUTSPC,.DEFEXT,.CSTRING
                        MOV     .OUTSPC,-(6.)
                        MOV     .DEFEXT,-(6.)
.IF B .CSTRING
                        CLR     -(6.)
.IFF
                        MOV     .CSTRING,-(6.)
.ENDC
                        EMT     ^O345
.ENDM

.MACRO .CSTAT .AREA,.CHAN,.ADD
...CM1 <.AREA>,23.,<.CHAN>
...CM2 <.ADD>,2.,X
.ENDM

.MACRO .DATE
                        MOV     @#54,%0
                        MOV     ^O262(0),%0
.ENDM
```

```
.MACRO .DELETE .AREA,.CHAN,.DEVBLK,.SPF
.IF DF ...V1
 .IIF NB <.CHAN>          MOV     .CHAN,%0
                          EMT     ^O<.AREA>
.IFF
...CM1 <.AREA>,0,<.CHAN>
...CM2 <.DEVBLK>,2.
 .IF B .SPF
                          CLR     4.(0)
 .IFF
                          MOV     .SPF,4.(0)
 .ENDC
                          EMT     ^O375
.ENDC
.ENDM

.MACRO .DEVICE .AREA,.ADD
...CM1 <.AREA>,12.,#0
...CM2 <.ADD>,2.,X
.ENDM

.MACRO .DSTATUS .RETSPC,.DNAME
.IIF NB <.DNAME>,         MOV     .DNAME,%0
                          MOV     .RETSPC,-(6.)
                          EMT     ^O342
.ENDM

.MACRO .ENTER .AREA,.CHAN,.DEVBLK,.LEN,.SPF
.IF DF ...V1
                          MOV     .CHAN,%0
 .IF B .DEVBLK
                          CLR     -(6.)
 .IFF
                          MOV     .DEVBLK,-(6.)
 .ENDC
                          EMT     ^O<40+.AREA>
.IFF
...CM1 <.AREA>,2.,<.CHAN>
...CM2 <.DEVBLK>,2.
 .IF NB .LEN
                          MOV     .LEN,4.(0)
 .IFF
                          CLR     4.(0)
 .ENDC
 .IF NB .SPF
                          MOV     .SPF,6.(0)
 .IFF
                          CLR     6.(0)
 .ENDC
                          EMT     ^O375
.ENDC
.ENDM

.MACRO .EXIT
                          EMT     ^O350
.ENDM
```

```
.MACRO .FETCH .ADD,.DNAME
.IIF NB <.DNAME>,                MOV     .DNAME,%Ø
                                 MOV     .ADD,-(6.)
                                 EMT     ^O343
.ENDM

.MACRO .GTIM .AREA,.ADD
...CM1 <.AREA>,17.,#Ø
...CM2 <.ADD>,2.,X
.ENDM

.MACRO .GTJB .AREA,.ADD
...CM1 <.AREA>,16.,#Ø
...CM2 <.ADD>,2.,X
.ENDM

.MACRO .HERR
...CM3 ,5.
.ENDM

.MACRO .HRESET
                        EMT     ^O357
.ENDM

.MACRO .INTEN .PRIO,.PIC
.IF NB .PIC
                MOV     @#^O54,-(6.)
                JSR     5.,@(6.)+
.IFF
                JSR     5.,@^O54
.ENDC
                .WORD   ^C<.PRIO*32.>&224.
.ENDM

.MACRO .LOCK
                        EMT     ^O346
.ENDM

.MACRO .LOOKUP .AREA,.CHAN,.DEVBLK,.SPF
.IF DF ...V1
 .IIF NB <.CHAN>,       MOV     .CHAN,%Ø
                        EMT     ^O<20+.AREA>
.IFF
...CM1 <.AREA>,1,<.CHAN>
...CM2 <.DEVBLK>,2.
 .IF B .SPF
                        CLR     4.(Ø)
 .IFF
                        MOV     .SPF,4.(Ø)
 .ENDC
                        EMT     ^O375
.ENDC
.ENDM

.MACRO  .MFPS           .ADD
                        MOV     @#^O54,-(6.)
                        ADD     #^O362,(6.)
                        JSR     7.,@(6.)+
.IIF NB <.ADD>,         MOVB    (6.)+,.ADD
.ENDM
```

```
.MACRO .MRKT .AREA,.TIME,.CRTN,.ID
...CM1 <.AREA>,18.,#0
...CM2 <.TIME>,2.
...CM2 <.CRTN>,4.
...CM2 <.ID>,6.,X
.ENDM

.MACRO  .MTPS               .ADD
.IIF NB <.ADD>,             CLR     -(6.)
.IIF NB <.ADD>,             MOVB    .ADD,(6.)
                            MOV     @#^O54,-(6.)
                            ADD     #^O360,(6.)
                            JSR     7.,@(6.)+
.ENDM


.MACRO .MWAIT
...CM3 ,9.
.ENDM

.MACRO .PRINT .ADD
.IIF NB <.ADD>,             MOV     .ADD,%0
                            EMT     ^O351
.ENDM

.MACRO .PROTECT .AREA,.ADD
...CM1 <.AREA>,25.,#0
...CM2 <.ADD>,2.,X
.ENDM

.MACRO .PURGE .CHAN
...CM3 <.CHAN>,3.
.ENDM

.MACRO .QSET .QADD,.QLEN
.IIF NB <.QLEN>,            MOVB    .QLEN,%0
                            MOV     .QADD,-(6.)
                            EMT     ^O353
.ENDM

.MACRO .RCTRLO
                            EMT     ^O355
.ENDM

.MACRO .RCVD .AREA,.BUFF,.WCNT
...CM4 <.AREA>,#0,<.BUFF>,<.WCNT>,,#1,22.
.ENDM

.MACRO .RCVDC .AREA,.BUFF,.WCNT,.CRTN
...CM4 <.AREA>,#0,<.BUFF>,<.WCNT>,,<.CRTN>,22.
.ENDM

.MACRO .RCVDW .AREA,.BUFF,.WCNT
...CM4 <.AREA>,#0,<.BUFF>,<.WCNT>,,#0,22.
.ENDM
```

```
.MACRO .READ .AREA,.CHAN,.BUFF,.WCNT,.BLK
.IF DF ...V1
 .IIF NB <.WCNT>,        MOV     .WCNT,%0
                         MOV     #1,-(6.)
                         MOV     .BUFF,-(6.)
                         MOV     .CHAN,-(6.)
                         EMT     ^O<200+.AREA>
.IFF
...CM4 <.AREA>,<.CHAN>,<.BUFF>,<.WCNT>,<.BLK>,#1,8.
.ENDC
.ENDM

.MACRO .READC .AREA,.CHAN,.BUFF,.WCNT,.CRTN,.BLK
.IF DF ...V1
 .IIF NB <.CRTN>,        MOV     .CRTN,%0
                         MOV     .WCNT,-(6.)
                         MOV     .BUFF,-(6.)
                         MOV     .CHAN,-(6.)
                         EMT     ^O<200+.AREA>
.IFF
...CM4 <.AREA>,<.CHAN>,<.BUFF>,<.WCNT>,<.BLK>,<.CRTN>,8.
.ENDC
.ENDM

.MACRO .READW .AREA,.CHAN,.BUFF,.WCNT,.BLK
.IF DF ...V1
 .IIF NB <.WCNT>,        MOV     .WCNT,%0
                         CLR     -(6.)
                         MOV     .BUFF,-(6.)
                         MOV     .CHAN,-(6.)
                         EMT     ^O<200+.AREA>
.IFF
...CM4 <.AREA>,<.CHAN>,<.BUFF>,<.WCNT>,<.BLK>,#0,8.
.ENDC
.ENDM

.MACRO .REGDEF
R0=%0
R1=%1
R2=%2
R3=%3
R4=%4
R5=%5
SP=%6
PC=%7
.ENDM

.MACRO .RELEASE .DEVBLK
.IIF NB <.DEVBLK>,       MOV     .DEVBLK,%0
                         CLR     -(6.)
                         EMT     ^O343
.ENDM
```

```
.MACRO .RENAME .AREA,.CHAN,.DEVBLK
.IF DF ...V1
.IIF NB <.CHAN>,          MOV      .CHAN,%0
                          EMT      ^O<100+.AREA>
.IFF
...CM1 <.AREA>,4.,<.CHAN>
...CM2 <.DEVBLK>,2.,X
.ENDC
.ENDM

.MACRO .REOPEN .AREA,.CHAN,.CBLK
.IF DF ...V1
 .IIF NB <.CHAN>,         MOV      .CHAN,%0
                          EMT      ^O<140+.AREA>
.IFF
...CM1 <.AREA>,6.,<.CHAN>
...CM2 <.CBLK>,2.,X
.ENDC
.ENDM

.MACRO .SAVESTAT .AREA,.CHAN,.CBLK
.IF DF ...V1
 .IIF NB <.CHAN>,         MOV      .CHAN,%0
                          EMT      ^O<120+.AREA>
.IFF
...CM1 <.AREA>,5.,<.CHAN>
...CM2 <.CBLK>,2.,X
.ENDC
.ENDM

.MACRO .RSUM
...CM3 ,2.
.ENDM

.MACRO .SDAT .AREA,.BUFF,.WCNT
...CM4 <.AREA>,#0,<.BUFF>,<.WCNT>,,#1,21.
.ENDM

.MACRO .SDATC .AREA,.BUFF,.WCNT,.CRTN
...CM4 <.AREA>,#0,<.BUFF>,<.WCNT>,,<.CRTN>,21.
.ENDM

.MACRO .SDATW .AREA,.BUFF,.WCNT
...CM4 <.AREA>,#0,<.BUFF>,<.WCNT>,,#0,21.
.ENDM

.MACRO .SERR
...CM3 ,4.
.ENDM

.MACRO .SETTOP .ADD
.IIF NB <.ADD>,           MOV      .ADD,%0
                          EMT      ^O354
.ENDM

.MACRO .SFPA .AREA,.ADD
...CM1 <.AREA>,24.,#0
...CM2 <.ADD>,2.,X
.ENDM
```

```
.MACRO .SPFUN .AREA,.CHAN,.CODE,.BUFF,.WCNT,.BLK,.CRTN
...CM1 <.AREA>,26.,<.CHAN>
...CM2 <.BLK>,2.
...CM2 <.BUFF>,4.
...CM2 <.WCNT>,6.
.IF NB .CODE
                        MOVB    #^O377,8.(0)
                        MOVB    .CODE,9.(0)
.ENDC
.IF B .CRTN
                        CLR     10.(0)
.IFF
                        MOV     .CRTN,10.(0)
.ENDC
                        EMT     ^O375
.ENDM

.MACRO .SRESET
                        EMT     ^O352
.ENDM

.MACRO .SPND
...CM3 ,1
.ENDM

.MACRO .SYNCH .AREA
.IIF NB <.AREA>,        MOV     .AREA,%4
                        MOV     @#^O54,%5
                        JSR     5.,@^O324(5.)
.ENDM

.MACRO .TLOCK
...CM3 ,7.
.ENDM

.MACRO .TRPSET .AREA,.ADD
...CM1 <.AREA>,3.,#0
...CM2 <.ADD>,2.,X
.ENDM

.MACRO .TTINR
                        EMT     ^O340
.ENDM

.MACRO .TTYIN .CHAR
                        EMT     ^O340
                        BCS     .-2
.IIF NB <.CHAR>,        MOVB    %0,.CHAR
.ENDM

.MACRO .TTOUTR
                        EMT     ^O341
.ENDM

.MACRO .TTYOUT .CHAR
.IIF NB <.CHAR>,        MOVB    .CHAR,%0
                        EMT     ^O341
                        BCS     .-2
.ENDM
```

```
.MACRO .TWAIT .AREA,.TIME
...CM1 <.AREA>,20.,#0
...CM2 <.TIME>,2.,X
.ENDM

.MACRO .UNLOCK
                            EMT     ^O347
.ENDM

.MACRO .WAIT .CHAN
.IF DF ...V1
                            EMT     ^O<240+.CHAN>
.IFF
...CM3 <.CHAN>,0
.ENDC
.ENDM

.MACRO .WRITE .AREA,.CHAN,.BUFF,.WCNT,.BLK
.IF DF ...V1
 .IIF NB <.WCNT>,           MOV     .WCNT,%0
                            MOV     #1,-(6.)
                            MOV     .BUFF,-(6.)
                            MOV     .CHAN,-(6.)
                            EMT     ^O<220+.AREA>
.IFF
...CM4 <.AREA>,<.CHAN>,<.BUFF>,<.WCNT>,<.BLK>,#1,9.
.ENDC
.ENDM

.MACRO .WRITW .AREA,.CHAN,.BUFF,.WCNT,.BLK
.IF DF ...V1
 .IIF NB <.WCNT>,           MOV     .WCNT,%0
                            CLR     -(6.)
                            MOV     .BUFF,-(6.)
                            MOV     .CHAN,-(6.)
                            EMT     ^O<220+.AREA>
.IFF
...CM4 <.AREA>,<.CHAN>,<.BUFF>,<.WCNT>,<.BLK>,#0,9.
.ENDC
.ENDM

.MACRO .WRITC .AREA,.CHAN,.BUFF,.WCNT,.CRTN,.BLK
.IF DF ...V1
 .IIF NB <.CRTN>,           MOV     .CRTN,%0
                            MOV     .WCNT,-(6.)
                            MOV     .BUFF,-(6.)
                            MOV     .CHAN,-(6.)
                            EMT     ^O<220+.AREA>
.IFF
...CM4 <.AREA>,<.CHAN>,<.BUFF>,<.WCNT>,<.BLK>,<.CRTN>,9.
.ENDC
.ENDM
```

APPENDIX E

PROGRAMMED REQUEST SUMMARY

E.1 PARAMETERS

The following parameters are used as arguments in various calls. (Any parameters used which are not mentioned here are particular to a request and the appropriate section in Chapter 9 should be consulted.)

Parameter	Description
.addr	an address, the meaning of which depends on the request being used
.area	a pointer to the EMT argument list
.blk	a block number specifying the relative block in a file where an I/O operation is to begin
.buff	a buffer address specifying a memory location into which or from which an I/O transfer is to be performed
.chan	a channel number in the range 0-377 (octal)
.crtn	the entry point of a completion routine
.count	file number for magtape/cassette operations
.dblk	the address of a four-word RAD50 descriptor of the file to be opened
.num	a number, the value of which depends on the request
.wcnt	a word count specifying the number of words to be transferred to or from the buffer during an I/O operation

E.2 REQUEST SUMMARY

Refer to Appendix D (SYSMAC.SML) to see how each macro call is expanded in assembly language code.

Mnemonic	Function	Macro Call	Error Codes (Byte 52=)
.CDFN	Increases number of I/O channels to as many as 255(decimal)	.CDFN .area,.addr,.num	0 - attempt to define fewer channels than already exist
.CHAIN	Allows one background program to transfer control to another without operator intervention	.CHAIN	Can produce any errors which the monitor RUN command can produce
.CHCOPY	Opens a channel for input, logically connecting it to another job open for either input or output (F/B only)	.CHCOPY .area,.chan,.ochan	0 - other job does not exist, or does not have enough channels defined, or does not have specified channel open 1 - channel already open
.CLOSE	Terminates activity on specified channel and frees it for use in another operation; makes tentative files permanent	.CLOSE .chan	Fatal monitor error if device handler is not in memory
.CMKT	Cancels one or more outstanding mark time requests (F/B only)	.CMKT .area,.id,.time	0 - id is not 0 and mark time with that identification number not found

Mnemonic	Function	Macro Call	Error Codes (Byte 52=)
.CNTXSW	Specifies locations to be included in the context switch (F/B only)	.CNTXSW .area,.addr	0 - list is in an area into which the the USR swaps; or list is modified while the job is running.
.CSIGEN	Calls the CSI in general mode Note: if input is taken from TT:, all errors are printed out	.CSIGEN .devspc,.defext,.cstring	0 - illegal command 1 - device not found 2 - unused 3 - full directory 4 - input file not found
.CSISPC	Calls the CSI in special mode Note: if input is taken from TT:, all errors are printed out	.CSISPC .outspc,.defext,.cstring	0 - illegal command line 1 - illegal device
.CSTAT	Furnishes information about a channel (F/B only)	.CSTAT .area,.chan,.addr	0 - channel not open

Mnemonic	Function	Macro Call	Error Codes (Byte 52=)
.DATE	Moves current date information into R0	.DATE	None
.DELETE	Deletes named file from indicated device	.DELETE .area,.chan,.dblk,.count	0 - active channel 1 - file not found
.DEVICE	Sets up list of addresses to be loaded with specified values upon program termination (F/B only)	.DEVICE .area,.addr	None
.DSTATUS	Provides information about a device	.DSTATUS .cblk,.devnam	0 - device not found
.ENTER	Allocates space on specified device and creates tentative entry for named file	.ENTER .area,.chan,.dblk,.length,.count	0 - channel in use 1 - no space greater than or equal to the specified length was found
.EXIT	Terminates user program and returns control to monitor	.EXIT	None
.FETCH	Loads device handlers into memory from system device	.FETCH .coradd,.devnam	0 - nonexistent device name or no handler for that device

Mnemonic	Function	Macro Call	Error Codes (Byte 52=)
.GTIM	Allows access to the current time of day	.GTIM .area,.addr	None
.GTJB	Passes certain job parameters back to user program	.GTJB .area,.addr	None
.HERR	Disables error interception and allows system to detect and act on normally fatal errors	.HERR	Monitor Error occurs if: 1. called USR from completion routine 2. no device handler 3. error doing directory I/O 4. FETCH error 5. Error reading overlay 6. no room in directory 7. illegal address 8. illegal channel number 9. illegal EMT
.HRESET	Resets channels, releases device handlers and stops all I/O transfers in progress	.HRESET	None

Mnemonic	Function	Macro Call	Error Codes (Byte 52=)
.INTEN	Notifies monitor that interrupt has occurred, and sets processor priority to correct state	.INTEN .priority, .pic	None
.LOCK	Locks the USR in memory	.LOCK	None
.LOOKUP	Associates a specified channel with a device and/or file on that device	.LOOKUP .area,.chan,.dblk,.count	0 - channel already open 1 - file not found
.MFPS	Reads priority bits from processor status word	.MFPS .addr	None
.MRKT	Schedules a completion routine to be entered after specified time interval (F/B only)	.MRKT .area,.time,.crtn,.id	0 - no queue element is available
.MFTS	Sets priority bits, condition codes, and T bit in processor status word	.MFTS .addr	None
.MWAIT	Suspends execution until all messages have been transmitted or received (F/B only)	.MWAIT	None
.PRINT	Outputs a string to the terminal	.PRINT .addr	None

Mnemonic	Function	Macro Call	Error Codes (Byte 52=)
.PROTECT	Used to obtain exclusive control of a vector in the range 0-476 (F/B only)	.PROTECT .area,.addr	0 - protect failure; locations already in use 1 - address > 476 or not a multiple of 4
.PURGE	Deactivates a channel without taking any other action	.PURGE .chan	None
.QSET	Enlarges I/O queue for monitor	.QSET .addr,.qleng	None
.RCTRLO	Enables terminal printing	.RCTRLO	None
.RCVD	Posts request to receive message and continues execution (F/B only)	.RCVD .area,.buff,.wcnt	0 - no other job exists in system
.RCVDC	Posts request to receive message and enters completion routine when message received (F/B only)	.RCVDC .area,.buff,.wcnt,.crtn	0 - no other job exists in system
.RCVDW	Posts request to receive message and waits (F/B only) until received	.RCVDW .area,.buff,.wcnt	0 - no other job exists in system

Mnemonic	Function	Macro Call	Error Codes (Byte 52=)
.READ	Initiates transfer from specified channel to memory; returns to program immediately	.READ .area,.chan,.buff,.wcnt,.blk	0 - attempt to read past end-of-file 1 - hard error on channel 2 - channel not open
.READC	Transfers words from specified channel to memory; returns control to specified routine when complete	.READC .area,.chan,.buff,.wcnt,.crtn,.blk	0 - attempt to read past end-of-file 1 - hard error on channel 2 - channel not open
.READW	Transfers words from specified channel to memory; returns control to user program when transfer complete	.READW .area,.chan,.buff,.wcnt,.blk	0 - attempt to read past end-of-file 1 - hard error on channel 2 - channel not open

Mnemonic	Function	Macro Call	Error Codes (Byte 52=)
.REGDEF	Defines general registers R0-R5 SP,PC	.REGDEF	None
.RELEAS	Removes device handler from memory	.RELEAS .devnam	0 - handler name is illegal
.RENAME	Changes file name	.RENAME .area,.chan,.dblk	0 - channel open 1 - file not found
.REOPEN	Reassociates channel with file on which a SAVESTATUS was performed	.REOPEN .area,.chan,.cblk	0 - channel is in use
.RSUM	Resumes job after it was suspended via .SPND (F/B only)	.RSUM	None
.SAVESTATUS	Stores five words (containing data concerning file definition) into memory	.SAVESTATUS .area,.chan,.cblk	1 - SAVESTATUS is illegal
.SDAT	Initiates message transfer; returns control to user program immediately (F/B only)	.SDAT .area,.buff,.wcnt	0 - no other job exists

Mnemonic	Function	Macro Call	Error Codes (Byte 52=)
.SDATC	Initiates message transfer; transfers control to specified routine when complete (F/B only)	.SDATC .area,.buff,.wcnt,.crtn	0 - no other job exists
.SDATW	Initiates message transfer; returns control to user program when complete (F/B only)	.SDATW .area,.buff,.wcnt	0 - no other job exists
.SERR	Inhibits fatal errors from aborting job	.SERR	-1 - called USR from completion routine -2 - no device handler -3 - error doing directory I/O -4 - FETCH error -5 - error reading overlay -6 - no more room for files in directory -7 - illegal address -10 - illegal channel number -11 - illegal EMT

Mnemonic	Function	Macro Call	Error Codes (Byte 52=)
.SETTOP	Requests additional memory for program	.SETTOP .addr	None
.SFPA	Sets user interrupt address for floating point processor exceptions	.SFPA .area,.addr	None
.SPFUN	Provides special device functions to magtape and cassette (and diskette).	.SPFUN .area,.chan,.code,.buff,.wcnt,.crtn,.blk	0 - attempt to read past end-of-file 1 - hardware error on channel 2 - channel not open
.SPND	Suspends running job (F/B only)	.SPND	None
.SRESET	Resets certain areas of memory, dismisses device handlers brought in by FETCH, purges currently open files, resets to 16 I/O channels, queue to one element	.SRESET	None

Mnemonic	Function	Macro Call	Error Codes (Byte 52=)
.SYNCH	Enables monitor programmed request from within an interrupt service routine; normal return is to 2nd location following .SYNCH	SYNCH .area	Monitor returns to location following .SYNCH if: 1. another .SYNCH specifying same 7-word block is pending 2. illegal job number was specified 3. job is not running
.TLOCK	Attempts to gain ownership of USR; if unsuccessful, control returns with C bit set (F/B only)	.TLOCK	0 - USR is in use by another job
.TRPSET	Allows user job to intercept traps to 4 and 10	.TRPSET .area,.addr	None
.TTYIN	Inputs character from terminal and waits until done	.TTYIN .char	None
.TTINR	Inputs character from terminal	.TTINR	0 - No characters available in ring buffer

Mnemonic	Function	Macro Call	Error Codes (Byte 52=)
.TTOUTR	Outputs character to terminal	.TTOUTR	0 - Output ring buffer full
.TTYOUT	Outputs character to terminal and waits until done	.TTYOUT .char	None
.TWAIT	Suspends the running job for a specified amount of time (F/B only)	.TWAIT .area,.time	0 - No queue element was available
.UNLOCK	Releases USR from memory	.UNLOCK	None
..V1..	Enables macro expansions to occur in Version 1 format	..V1..	None
..V2..	Enables macro expansions to occur in Version 2 format	..V2..	None
.WAIT	Suspends program execution until all channel I/O is complete	.WAIT .chan	0 - channel not open 1 - hardware error
.WRITC	Initiates transfer from memory to specified channel and returns to user program; when complete, passes control to specified routine	.WRITC .area,.chan,.buff,.wcnt,.crtn,.blk	0 - end-of-file reached 1 - hardware error 2 - channel not open

Programmed Request Summary

Mnemonic	Function	Macro Call	Error Codes (Byte 52=)
.WRITE	Initiates transfer from memory to channel; returns control to user program immediately	.WRITE .area,.chan,.buff,.wcnt,.blk	0 - end-of-file reached 1 - hardware error 2 - channel not open
.WRITW	Transfers words from memory to channel; when complete, returns control to user program	.WRITW .area,.chan,.buff,.wcnt,.blk	0 - end-of-file reached 1 - hardware error 2 - channel not open

APPENDIX F

BASIC/RT-11 LANGUAGE SUMMARY

BASIC/RT-11 is a single-user conversational BASIC for use under the RT-11 system. While the BASIC language is one of the simplest computer languages to learn, it contains advanced techniques which the experienced programmer can use to perform more intricate manipulations or to express a problem more efficiently.

BASIC/RT-11 interfaces with the RT-11 monitor to provide powerful sequential and random-access file capabilities and allows the user to save and retrieve programs from peripheral devices. BASIC/RT-11 has provision for alphanumeric character string I/O and string variables (12K or larger systems) and allows user-defined functions and assembly language subroutine calls from user BASIC programs.

For details on using the BASIC language, and for instructions on running BASIC/RT-11, refer to the BASIC/RT-11 Language Reference Manual (DEC-11-LBACA-D-D). A summary of the BASIC/RT-11 commands and error messages is included here for quick reference.

F.1 BASIC/RT-11 STATEMENTS

The following summary of BASIC statements defines the general format for the statement and gives a brief explanation of its use.

CALL "function name" [(argument list)]

Used to call assembly language user functions from a BASIC program.

CHAIN"file descriptor"[LINE number]

Terminates execution of user program, loads and executes the specified program starting at the line number, if included.

CLOSE[{VFn / #n}]

Closes the logical file specified. If no file is specified, closes all files which are open.

DATA data list

Used in conjunction with READ to input data into an executing program.

DEF FNletter (argument)=expression
: Defines a user function to be used in the program (letter is any alphabetic letter).

DIM variable(n), variable(n,m),variable$(n),variable$(n,m)
: Reserves space for lists and tables according to subscripts specified after variable name.

END
: Placed at the physical end of the program to terminate program execution.

FOR variable = expression1 TO expression2 STEP expression3
: Sets up a loop to be executed the specified number of times.

GOSUB line number
: Used to transfer control to a specified line of a subroutine.

GO TO line number
: Used to unconditionally transfer control to other than the next sequential line in the program.

IF expression rel.op. expression {THEN | GO TO} line number
: Used to conditionally transfer control to the specified line of the program.

IF END #n {THEN | GO TO} line number
: Used to test for end file on sequential input file #n.

INPUT list
: Used to input data from the terminal keyboard or papertape reader.

INPUT #expression: list
: Inputs from a sequential file or particular device.

[LET] variable=expression
: Used to assign a value to the specified variable.

[LET] VFn(i)=expression
: Used to set the value of a virtual memory file element.

NEXT variable
: Placed at the end of a FOR loop to return control to the FOR statement.

OPEN file [FOR {INPUT | OUTPUT}] [(b)] AS FILE #n [DOUBLE BUF]
: Opens a sequential file for input or output as specified. File may be of the form "dev:filnam.ext" or a scalar string variable. The number of blocks may be specified by b.

OPEN file [FOR {INPUT | OUTPUT}] [(b)] AS FILE VFnx (dimension)=string length
: Opens a virtual memory file for input or output. x represents the type of file: floating point (blank), integer (%), or character strings ($). File may be of the form "dev:fil.ext" or a scalar

	string variable. The number of blocks may be specified by b.
OVERLAY "file descriptor"	Used to overlay or merge the program currently in memory with a specified file, and continue execution.
PRINT list	Used to output data to the terminal. The list can contain expressions or text strings.
PRINT "text"	Used to print a message or a string of characters.
PRINT #expression: expression list	Outputs to a particular output device, as specified in an OPEN statement.
PRINT TAB(x);	Used to space to the specified column.
RANDOMIZE	Causes the random number generator to calculate different random numbers every time the program is run.
READ variable list	Used to assign the values listed in a DATA statement to the specified variables.
REM comment	Used to insert explanatory comments into a BASIC program.
RESTORE	Used to reset data block pointer so the same data can be used again.
RESTORE #n	Rewinds the input sequential file #n to the beginning.
RETURN	Used to return program control to the statement following the last GOSUB statement.
STOP	Used at the logical end of the program to terminate execution.

F.2 BASIC/RT-11 COMMANDS

The following key commands halt program execution, erase characters or delete lines.

Key	Explanation
ALTMODE	Deletes the entire current line. Echoes DELETED message (same as CTRL U). On some terminals the ESC key must be used.
CTRL C	Interrupts execution of a command or program and returns control to the RT-11 monitor. BASIC may be restarted without loss of the current program by using the monitor REENTER command.

CTRL O	Stops output to the terminal and returns BASIC to the READY message when program or command execution is completed.
CTRL U	Deletes the entire current line. Echoes DELETED message (same as ALTMODE).
←	(SHIFT O) Deletes the last character typed and echoes a backarrow (same as RUBOUT). On VT05 or LA30 use the underscore (_) key.
RUBOUT	Deletes the last character typed and echoes a backarrow (same as ←).

The following commands list, punch, erase, execute and save the program currently in memory.

Command	Explanation
CLEAR	Sets the array and string buffers to nulls and zeroes.
LIST	Prints the user program currently in memory on the terminal.
LIST line number LIST -line number LIST line number-[END] LIST line number-line number	Types out the specified program line(s) on the terminal.
LISTNH line number LISTNH -line number LISTNH line number-[END] LISTNH line number-line number	Lists the lines associated with the specified numbers but does not print a header line.
NEW "filnam"	Does a SCRatch and sets the current program name to the one specified.
OLD "file"	Does a SCRatch and inputs the program from the specified file.
RENAME "filnam"	Changes the current program name to the one specified.
REPLACE "dev:filnam.ext"	Replaces the specified file with the current program.
RUN	Executes the program in memory.
RUNNH	Executes the program in memory but does not print a header line.
SAVE "dev:filnam.ext"	Outputs the program in memory as the specified file.

SCRatch	Erases the entire storage area.

F.3 BASIC/RT-11 FUNCTIONS

The following functions perform standard mathematical operations in BASIC.

Name	Explanation
ABS(x)	Returns the absolute value of x.
ATN(x)	Returns the arctangent of x as an angle in radians in the range + or - pi/2.
BIN(x$)	Computes the integer value of a string of blanks (ignored), 1's and 0's.
COS(x)	Returns the cosine of x radians.
EXP(x)	Returns the value of e↑x where e=2.71828.
INT(x)	Returns the greatest integer less than or equal to x.
LOG(x)	Returns the natural logarithm of x.
OCT(x$)	Computes an integer value from a string of blanks (ignored) and digits from 0 to 7.
RND(x)	Returns a random number between 0 and 1.
SGN(x)	Returns a value indicating the sign of x.
SIN(x)	Returns the sine of x radians.
SQR(x)	Returns the square root of x.
TAB(x)	Causes the terminal type head to tab to column number x.

The string functions are:

ASC(x$)	Returns as a decimal number the seven-bit internal code for the one-character string (x$).
CHR$(x)	Generates a one-character string having the ASCII value of x.
DAT$	Returns the current date in the format 07-MAY-73.
LEN(x$)	Returns the number of characters in the string (x$).
POS(x$,y$,z)	Searches for and returns the position of the first occurrence of y$ in x$ starting with the zth position.

SEG$(x$,y,z)	Returns the string of characters in positions y through z in x$.
STR$(x)	Returns the string which represents the numeric value of x.
TRM$(x$)	Returns x$ without trailing blanks.
VAL(x$)	Returns the number represented by the string (x$).

F.4 BASIC/RT-11 ERROR MESSAGES

The information that formerly appeared in Section F.4 has now been incorporated into the RT-11 System Message Manual, DEC-11-ORMEA-A-D. BASIC/RT-11 error messages are also found in the BASIC/RT-11 Language Reference Manual, DEC-11-LBACA-D-D.

APPENDIX G

FORTRAN LANGUAGE SUMMARY

FORTRAN IV is a problem-solving language designed to allow scientists and engineers to express mathematical operations in a familiar format.

A FORTRAN source program is composed of a series of statements. Statements are usually made up of one- or two-word commands, which, when used in conjunction with data variables and constants, can be used to control program execution, assign values to variables and read and write data. The FORTRAN library contains routines to simplify mathematical functions such as computing sines, cosines, square roots, etc. The source program is translated by the FORTRAN compiler into a machine language program which, when linked with the FORTRAN object time routines, can be executed by the computer.

This appendix describes first how to run a FORTRAN program as a foreground job, and then summarizes briefly the RT-11 FORTRAN IV language, statements, and error messages. For details, refer to the PDP-11 FORTRAN Language Reference Manual and the RT-11/RSTS/E FORTRAN IV User's Guide.

The FORTRAN programmer may also want to read Appendix O for details concerning the FORTRAN System Subroutine Library. SYSLIB allows the programmer to easily call and use the programmed requests described in Chapter 9. A summary of the SYSLIB library calls is in Table O-1.

G.1 RUNNING A FORTRAN PROGRAM IN THE FOREGROUND

Since the FORTRAN Object Time System needs work areas to perform some of its functions, the FRUN/N!x monitor command must be used to allocate the needed space when running a FORTRAN program as a foreground job. The following formula can be used to calculate the decimal number (x) which must be designated to the /N option:

$$x = 1/2\ [378 + (29*N) + (R-136) + A*512)\ ]$$

where:

N = the value of the /N FORTRAN compiler switch option (the compiler defaults this to 6).

R = the value of the /R FORTRAN compiler switch option (the compiler defaults this to 210 octal, 136 decimal).

A = the number of I/O buffers in simultaneous use during execution. (Console terminal I/O does not require any buffers. Each open logical unit typically requires one buffer.)

For example, to calculate x for a program (FILENA.REL) which uses only terminal I/O and allows the compiler to assign the standard defaults, set the formula as follows (all values are decimal):

x = 1/2 [378 + (29*6) + (136-136) + (Ø*512)]

276 decimal words are required. The FORTRAN program is executed using the FRUN command as shown:

.FRUN FILENA.REL/N!276 <CR>

NOTE

An exception can occur if the size of the linked FORTRAN program is less than the size of the USR (2K) and the USR is to be swapped. In this case, additional space must be allocated to allow the USR to successfully swap over the linked FORTRAN program. The additional space needed may be calculated by subtracting the size of the linked FORTRAN program from 10000 octal bytes (the size of the USR) and adding this result to the value of x previously calculated.

G.2 FORTRAN CHARACTER SET

The characters that may appear in a FORTRAN statement are restricted to those listed below.

1. The letters A through Z

2. The numerals 0 through 9

3. The following "special" characters:

Character	Name
	Space or Blank
=	Equals
+	Plus
-	Minus
*	Asterisk
/	Slash
(	Left Parenthesis

)	Right Parenthesis
,	Comma
.	Decimal Point
'	Apostrophe
"	Double Quote
$	Dollar Sign

Other printable characters may appear in a FORTRAN statement only as part of a Hollerith constant or alphanumeric literal. Any printable character may appear in a comment.

G.3 EXPRESSION OPERATORS

Each group of operators is shown in order of descending precedence during execution.

Type	Operator	Operates Upon
Arithmetic		
**	exponentiation	numeric constants, variables and expressions
*,/	multiplication, division,	
+,-	addition, subtraction	
-	unary minus	
Relational		
.GT.	greater than	variables, constants, and arithmetic expressions (all relational operators have equal priority)
.GE.	greater than or equal to	
.LT.	less than	
.LE.	less than or equal to	
.EQ.	equal to	
.NE.	not equal to	
Logical		
.NOT.	.NOT.A is true if and only if A is false, and is false if A is true.	logical variables, logical constants, integer variables and constants, logical integers and expressions
.AND.	A.AND.B is true if and only if A and B are both true and is false if either A or B is false.	
.OR.	A.OR.B is true if and only if either A or	

	B or both A and B are true, and false if both A and B are false.
.EQV.	A.EQV.B is true if and only if A and B are both true or if A and B are both false.
.XOR.	A.XOR.B is true if and only if A is true and B is false or if B is true and A is false.

G.4 SUMMARY OF FORTRAN STATEMENTS

The following is a summary of statements available in the PDP-11 FORTRAN IV language, including the general format for the statement and its effect.

Statement Formats	Effect
Arithmetic/Logical Assignment	
variable=expression	The value of the arithmetic or logical expression is assigned to the variable.
Arithmetic Statement Functions	
name(var1,var2,...)=expression	Creates a user-defined function having the variables (var) as dummy arguments. When referenced, the expression is evaluated using the actual arguments in the function call.
ACCEPT	
ACCEPT f,list	Reads input from logical unit 5 (TT: is default); f is the format statement label and list is an optional data list.
ASSIGN	
ASSIGN n TO ivar	Assigns the statement number n to the integer variable ivar.

BACKSPACE

BACKSPACE u	The currently open file on logical unit number u is backspaced one record.

BLOCK DATA

BLOCK DATA	Specifies the subprogram which follows as a BLOCK DATA subprogram.

CALL

CALL name CALL name (arg1, arg2,...)	Calls the SUBROUTINE subprogram with the name specified, passing the actual arguments (arg) to replace the dummy arguments in the SUBROUTINE definition.

COMMON

COMMON/name1/var1, var2,.../name1/var(j),var(k)

Reserves one or more blocks of storage space under the name specified to contain the variables (var) associated with that block name.

CONTINUE

CONTINUE	Causes no processing.

DATA

DATA var1, var2,.../val1, val2,.../

Causes elements in the list of values (val) to be initially stored in the corresponding elements of the list of variable names (var).

DECODE

DECODE (c,f,v) list	Changes the elements in the list of variables from ASCII into the desired internal format; c is the number of characters, f is the format, and v is the name of an array.

DEFINE FILE

```
DEFINE FILE u(m,n,U,lvar),...
```

Defines the record structure of a disk or DECtape direct access file where u is the logical unit number, m is the number of fixed length records in the file, n is the length in words of a single record, U is a fixed argument, and lvar is the associated variable pointing to the next record.

DIMENSION

```
DIMENSION s1,s2,...,sk
```

Reserves storage space for the specified array(s). Each s consists of an array variable name followed by up to seven integer constants, separated by commas and enclosed in parentheses.

DO

```
DO n lvar = e1,e2,e3
```

1. Sets integer variable ivar = to expression (e1, e2, e3 must result in an integer quantity)
2. Executes statements through statement number n
3. Increments lvar = lvar+e3
4. If e3>0 and ivar <=e2, or e3<0 and ivar >=e2, goes back to 2 above

END

```
END
```

Delimits a program unit.

ENCODE

```
ENCODE (c,f,v)list
```

Changes the elements in the list of variables into ASCII format; c is the number of characters in the buffer, f is the format statement number, and v is the name of the array to be used as the resulting buffer.

END FILE

```
END FILE u
```

The file currently open on logical unit number u is closed.

EQUIVALENCE

EQUIVALENCE (var1,var2,...),(varj,vark,...)

Each of the variables within a set of parentheses is assigned the same storage location.

EXTERNAL

EXTERNAL name1,name2,...

Informs the system that the names specified are those of FUNCTION or SUBROUTINE subprograms.

FIND

FIND(u'r)

Positions the direct access file on logical unit number u to record r and sets the associated variable to record number r.

FORMAT

n FORMAT(field specification,...)

Describes the format in which one or more records are to be transmitted; the statement number n must be specified (the field specifications may include the field descriptors, I, O, F, E, D, G, L, A, H, X, T, Q, and $, delimited by field separators , and /).

FUNCTION

FUNCTION name
FUNCTION name (var1,var2,...)
type FUNCTION name (var1,var2,...)

Begins a FUNCTION subprogram, indicating the program name and any dummy variable names (var). An optional type specification can be included.

GOTO

<u>Unconditional</u>

GOTO n

Transfers control to statement number n.

Computed

GOTO (k1,k2,...,kn),e	Transfers control to the statement number ki where i = value of expression e. If e<1 or e>n no transfer takes place.

Assigned

GOTO ivar GOTO ivar,(k1,k2,...,kn)	Transfers control to the statement most recently associated with ivar by an ASSIGN statement.

IF

Arithmetic

IF (expression) n1,n2,n3	Transfers control to statement number n depending upon the value of the expression. If the value of the expression is less than zero, transfers to n1; if the value of the expression is equal to zero, transfers to n2, if the value of the expression is greater than zero, transfers to n3.

Logical

IF (expression) statement	Executes the statement if the logical expression tests true.

IMPLICIT

IMPLICIT type (a1,b1-b2,...),...	The elements a and b represent single (or a range of) letter(s) whose presence as the initial letter of a variable specifies the variable to be of that type, if that variable is not explicitly given a type.

PAUSE

PAUSE PAUSE display	Suspends program execution and prints the octal constant, Hollerith constant, or alphanumeric literal display, if one is specified. Program execution is resumed by typing a carriage return.

PRINT

PRINT f,list	Writes output on logical unit 6 (LP: is default); f is the format statement label and list is an optional data list.

READ

Formatted

READ(u,f) READ(u,f)list READ f,list	Reads at least one logical record from device u according to format specification f and assigns values to the variables in the list. Logical unit number 1 is used as default in the third form above.

Unformatted

READ(u) READ(u) list	Reads one logical record from device u, assigning values to the variables in the list.

Direct Access

READ(u'r)list	Reads from logical unit number u, record number r, and assigns values to the variables in the list.

Transfer of Control on Error

END=n ERR=m END=n,ERR=m	Optional elements in the READ statement list (e.g., READ(u,f, END=n, ERR=m) allowing control transfer on error conditions. If an end-of-file condition is detected and END=n is specified, execution continues at statement n. If a hardware I/O error occurs and ERR=m is specified, execution continues at statement m.

RETURN

RETURN	Returns control to the calling program from the current subprogram.

REWIND

REWIND u	Logical unit number u is repositioned to the beginning of the currently opened file.

STOP

STOP STOP display	Terminates program execution and prints the octal constant, Hollerith constant, or alphanumeric literal display, if one is specified.

SUBROUTINE

SUBROUTINE name SUBROUTINE name (var1,var2,...)	Begins a SUBROUTINE subprogram, indicating the program name and any dummy variable names (var).

TYPE

TYPE f,list	Writes output on logical unit 7 (TT: is default); f is the format statement label and list is an optional data list.

<u>Type Declarations</u>

type var1,var2,...,vark	The variable names, (var), are assigned the specified data type in the program unit. type is one of: INTEGER(*2,*4) REAL(*4,*8) DOUBLE PRECISION COMPLEX(*8) LOGICAL(*4,*1) The optional * indicates that an explicit length is to be set for the variable; the number that follows is the length in bytes. The length must be one of the permitted values above.

WRITE

Formatted

WRITE (u,f)
WRITE (u,f) list

Causes one or more logical records containing the values of the variables in the list to be written onto device u according to the format specification f.

Unformatted

WRITE (u)
WRITE (u)list

Causes one logical record containing the values of the variables in the list to be written onto device u.

Direct Access

WRITE (u'r) list

Causes one logical record containing the values of the variables in the list to be written onto record r of the logical unit number u.

Transfer of Control on Error

END=n
ERR=m
END=n,ERR=m

Optional elements in the WRITE statement list (e.g., WRITE u,f, END=n, ERR=m) allowing control transfer on error conditions. If an end-of-file condition is detected and END=n is specified, execution continues at statement n. If a hardware I/O error occurs and ERR=m is specified, execution continues at statement m.

G.5 COMPILER ERROR DIAGNOSTICS

The information that formerly appeared in Sections G.5 and G.6 has been incorporated into the RT-11 System Message Manual, DEC-11-ORMEA-A-D. RT-11 FORTRAN error messages are also found in the RT-11/RSTS/E FORTRAN IV User's Guide, DEC-11-LRRUA-A-D.

APPENDIX H

F/B PROGRAMMING AND DEVICE HANDLERS

H.1 F/B PROGRAMMING IN RT-11, VERSION 2

Certain programming conventions must be observed in RT-11, Version 2, which were not required in Version 1. These conventions are necessary to permit interrupt routines to function properly while running two jobs in the F/B environment. All Version 2 device handlers follow these conventions; the user is urged to consult the listings of the example handlers (Section H.3) as they illustrate some of the techniques discussed.

NOTE

> Device handlers distributed with RT-11, Version 1, will not work properly with Version 2. Also, any user-written device handlers should be re-written to comply with the Version 2 conditions. Instructions for interfacing new handlers to RT-11 are provided in the RT-11 Software Support Manual. (DEC-11-ORPGA-B-D). Section H.2 describes Version 2 device handlers.

The procedures described in this appendix are necessary and must be followed to prevent system failures when jobs are running under RT-11. If at any time a program which services its own interrupts (and does not follow the guidelines described here) is run with another job, the system may malfunction. Therefore, it is required that all programs follow the procedures that are indicated here.

H.1.1 Interrupt Priorities

The status word for each interrupt vector should be set such that when an interrupt occurs, the processor takes it at level 7. Thus, a device which has its vectors at 70 and 72 has location 70 set to its service routine; location 72 contains 340. The 340 causes the service routine to be entered with the processor set to inhibit any device interrupts.

H.1.2 Interrupt Service Routine

If the conventions outlined in Section H.1.1 are followed, when an interrupt occurs, the processor priority will be 7. The first task of the interrupt service routine is to lower the processor priority to the correct value. This can be done using the .INTEN macro call. The call is:

```
        .INTEN  .priority
or
        .INTEN  .priority,.pic
```

The .INTEN call is explained in Chapter 9, Programmed Requests. On return from the .INTEN call, the processor priority is set properly; registers 4 and 5 have been saved and can be used without the necessity of saving them again.

For example, a user device interrupts at processor priority 5:

```
                DEVPRI=5

        DEVINT: .INTEN DEVPRI           ;NOTE, NOT #DEVPRI
                .
                .
                .
                RTS PC
```

If the contents of the processor status word, loaded from the interrupt vector, is of significance to the interrupt service routine (e.g., the condition bits), the PS should be moved to a memory location (not the stack) before issuing the .INTEN. A device handler's interrupt service routine uses the monitor stack and should avoid excessive use of stack space.

H.1.3 Return from Interrupt Service

When an interrupt has been serviced, instead of issuing an RTI to return from the interrupt, the routine must exit with an RTS PC. This RTS PC returns control to the monitor, which then restores registers 4 and 5, and executes the RTI.

When a device handler has completed a transfer and is ready to return to the monitor via the internal monitor completion address, R4 must be pointing to the fifth word of the handler. It is no longer necessary to have R0 and R3 on the stack, as it was in Version 1. The user is urged to consult the example handlers at the end of this appendix, which illustrates the operation of device handlers.

H.1.4 Issuing Programmed Requests at the Interrupt Level

Programmed requests from interrupt routines must be preceded by a .SYNCH call. This ensures that the proper job is running when the programmed request is issued. The .SYNCH call assumes that nothing has been pushed onto the stack by the user program between the .INTEN call and the .SYNCH call. On successful completion of a .SYNCH, R0 and R1 have been saved and are free to be used. R4 and R5 are no longer free, and should be saved and restored if they are to be used.

Programmed requests that require USR action should not be called from within interrupt routines.

H.1.5 Setting Up Interrupt Vectors

Devices for which no RT-11 handler exists must be serviced by the user program. For example, no LPS device handler exists; to use an LPS, the user must write his own interrupt service routine. It is the responsibility of the program to set up the vector for devices such as this. The recommended procedure is not to simply move the service routine address and 340 into the desired vector; rather, this operation must be preceded by a .PROTECT macro call. The .PROTECT ensures that neither the other job nor the monitor already has control of that device. If the .PROTECT is successful, the vector can be initialized.

H.1.6 Using .ASECT Directives in Relocatable Image Files

With some exceptions, user programs in REL format should not contain .ASECT directives. The Linker does not allow .ASECTs above 1000 when the /R switch is used; in general all .ASECTs below 1000 are ignored. The exceptions are:

34,36	2 words	TRAP instruction vector
40	1 word	program start address
42	1 word	stack start address
44	1 word	job status word
46	1 word	USR swap address
50	1 word	program high limit

.ASECTs may be used to initialize any of the above locations providing they are not in a program to be linked for the foreground. To initialize any other locations between 0-777, the user should first .PROTECT the appropriate words, and then move the correct values into place.

A job which contains overlays cannot have any .ASECTs at all, including those listed above. If overlays are to be used, the program start address should be set up as the argument for the .END statement; the other values can be initialized when the program is initiated.

Since .ASECTs into device vectors are not permitted, the user program must initialize vectors at runtime. To do this, execute a .PROTECT for the specified vector. If the .PROTECT is successful, it is safe to move values into the vectors. If the .PROTECT request fails, it is an indication that the vector is already in use by another job and that it is not safe to initialize the vector.

H.1.7 Using .SETTOP

Proper use of .SETTOP is vital to preserve system integrity. Since RT-11 employs no hardware memory protection, the user must at all times observe the value returned in R0 from a .SETTOP request. It must never be assumed that the value requested by the program is the value returned by .SETTOP.

The monitor loads foreground programs into a memory partition just large enough for the program plus job header (impure area). No free space is available for a .SETTOP request, which may cause some programs to fail, in particular, any FORTRAN program. Free space may be provided within the program using, for example, the .BLKW directive, or it may be allocated at runtime using the FRUN /N switch. Appendix G, Section G.1, explains how to allocate more space for FORTRAN programs.

H.1.8 Making Device Handlers Resident

Device handlers used by a foreground job must be made permanently resident with a LOAD command. A .FETCH from a foreground job of a non-resident handler will cause a fatal error and abort the job. Therefore, a program written specifically for the foreground, or one that is to be run in either foreground or background partitions, should first execute a .DSTATUS on the device, to determine if the handler is resident, before issuing a .FETCH directive.

H.2 DEVICE HANDLERS

This section deals with the device handlers which are part of the RT-11 System, Version 2. Any device dependent information or general information required by the user is contained here. No mention of a handler implies that no special conditions must be met to use that device (all disks except diskette and DECtape are in this category, and therefore are not covered here).

This page intentionally blank.

Differences Between V1 and V2 Device Handlers

User-written device handlers must, in all cases, conform to the standard practices for Version 2. Since the last word of a device handler is used by the monitor, the user should be sure to include one extra word at the end of his program when indicating the size of the handler.

The differences between Version 1 and Version 2 handlers include the following:

A. Header Words

In Version 1 handlers, the third header word was taken to be the priority at which the device interrupt was taken. In Version 2, this word should be 340, indicating that the interrupt should be taken at priority level 7. (The priority level is set to 7 by the FETCH code regardless of what appears in the third word of the header.)

B. Entry Conditions

It is not necessary to save/restore registers when the handler is first entered, although to do so is not harmful. In Version 1, it was necessary to preserve R5 on first entering the handler.

C. Interrupt Handling

When an interrupt occurs, Version 2 handlers must execute an .INTEN request or its equivalent. (This was not necessary in Version 1.) On return from the .INTEN, R4 and R5 may be used as scratch registers. Device handlers may not do EMT requests without executing a .SYNCH request. (Refer to Chapter 9 for explanations of .INTEN and .SYNCH.)

The handler must return from an interrupt via an RTS PC rather than an RTI, as in Version 1. The .INTEN request essentially reenters the handler via a JSR PC, and thus the RTS PC returns control to the monitor, which executes the RTI when appropriate.

When the transfer is complete, the handler must exit to the monitor to terminate the transfer and/or enter a completion routine. When return is made to the monitor, R4 should point to the fifth word of the handler.

D. Abort Entry Point

All Version 2 device handlers must have an abort entry point to which control is transferred on CTRL C, hard .EXIT, or

.HRESET. This abort entry point is located one word before the interrupt service routine entry point and must contain a BR instruction to abort code. The abort code must stop the device operation and then exit through the jump into the completion code in RMON. Some devices, such as moving head disks, cannot be effectively stopped. In this case, it is permissible for the abort code to consist of just an RTS PC instruction. Examples are the RK05 and RPR02/RP03 device drivers. See Section H.3 for examples of device handlers with abort entry code.

The user is urged to refer to the example handlers enclosed. The same general techniques can be used to interface user device handlers.

H.2.1 PR (High-Speed Paper Tape Reader)

Unlike the PR handler for Version 1, the Version 2 PR handler does not print a ↑ on the terminal when it is entered for the first time. The tape must be in the reader when the command is issued, or an input error occurs. This prohibits any two-pass operations from being done using PR. For example, linking and assembling from PR will not work properly; an input error will occur when the second pass is initiated. The correct (and recommended) procedure is to use PIP to transfer the paper tape to disk or DECtape, and then perform the operation on the transferred file.

H.2.2 TT (Handler for Console Terminal)

The console terminal may be used as a peripheral device by using the TT handler. Note that:

1. A ↑ is typed when the handler is ready for input.

2. CTRL Z can be used to specify the end of input to TT. No carriage return is required after the CTRL Z. If CTRL Z is not typed, the TT handler accepts characters until the word count of the input request is satisfied.

3. CTRL O, struck while output is directed to TT, causes an entire output buffer (i.e., all characters currently queued) to be ignored. This is somewhat different than the normal action of CTRL O while at the console.

4. A single CTRL C struck while typing input to TT causes a return to the monitor. If output is directed to TT, a double CTRL C is required to return to the monitor if F/B is running. If the S/J monitor is running, only a single CTRL C is required to terminate output.

5. The TT handler can be in use for only one job (foreground or background) at a time, and for only one function (input or output) at a time. The terminal communication for the job not using TT is not affected at all.

6. The user may type ahead to TT; the input ring buffer is emptied before the keyboard is referenced. The terminating CTRL Z may also be typed ahead.

7. If the main line code of a job is using TT for input, and a completion routine does a .TTYIN, typed characters will be passed unpredictably to the .TTYIN and TT. Therefore, this should not be done.

8. If a job sends a buffer to TT for output and then does a .TTYOUT or a .PRINT, the output from the latter is delayed until the handler completes its transfer. If a TT output operation is started when the monitor's terminal output ring buffer is not empty (i.e., before the print-ahead is complete), the handler supersedes the ring buffer until its transfer is complete.

H.2.3 CR (Card Readers)

The card reader handler can transfer data either as ASCII characters in DEC 026 or DEC 029 card codes (Section H.4), or as column images controlled by the SET command (see Chapter 2). In ASCII mode (SET CR NOIMAGE), invalid punch combinations are decoded as the error character 134(octal)--backslash. In IMAGE mode, no punch combination is invalid; each column is read as 12 bits of data right-justified in one word of the input buffer. The handler continues reading until the transfer word count is satisfied or until a standard end-of-file card is encountered (12-11-0-1-6-7-8-9 punch in column 1; the rest of the card is arbitrary). On end-of-file, the buffer is filled with zeroes and the request terminates successfully; the next input request from the card reader gives an end-of-file error. Note that if the transfer count is satisfied at a point which is not a card boundary, the next request continues from the middle of the card with no loss of information. If the input hopper is emptied before the transfer request is complete, the handler will loop until the hopper is reloaded and the "START" button on the reader is pressed again. The transfer will then continue until completion or until another hopper empty condition exists. End-of-file is not reported on the hopper empty condition. The handler will loop if the hopper empties during the transfer regardless of the status of the SET CR HANG/NOHANG option. No special action is required to use the card reader handler with the CM-11 mark sense card reader. The program should be aware of the fact that mark sense cards may contain less than 80 characters. Note also that when the CR handler is set to CRLF or TRIM and is reading in IMAGE mode, unpredictable results may occur.

H.2.4 MT/CT (Magtape (TM11, TU16) and Cassette (TA11))

These devices have a file structure which differs from the common RT-11 structure. Each handler is capable of entirely supporting its own file structure and of allowing RT-11 users full access to the devices without being totally familiar with them.

H.2.4.1 General Characteristics - Both magtape and cassette can operate in two possible modes: "hardware" mode and "software" mode. These names refer to the type of operation which can be performed on the device at a given time. Software mode is the normal mode of operation and the mode used when accessing the device through any of the RT-11 system programs. In software mode, the handler automatically attends to file headers and uses a fixed record length to transfer data (256-word for magtape; 64-word for cassette).

Hardware mode allows the user to read or write any format desired, using any record size. In this mode, the word count is taken as the physical record size.

F/B Programming and Device Handlers

When the handlers are initially loaded by either the .FETCH programmed request or the LOAD command, only software functions are permitted. To switch from software to hardware mode, either a rewind or a nonfile-structured lookup must be performed. (A nonfile-structured lookup is a lookup in which the first word of the filename is null.) Note that if the TM11 handler has been modified (as described in the RT-11 System Generation Manual) to one of the following: 8ØØ BPI non-dump 7-track, 556 BPI 7-track, or 2ØØ BPI 7-track, the handler is used somewhat differently. Refer to the RT-11 System Generation Manual for details.

NOTE

> Due to the size of the TM11 and TJU16 magtape handlers, users who have systems with less than 20K, and who have loaded both handlers simultaneously, may experience user memory restrictions.

In software mode, the following functions are permitted:

ENTER	-	Open new file for output
LOOKUP	-	Open existing file for input and/or output
DELETE	-	Delete an existing file on the device
CLOSE	-	Close a file which was opened with ENTER or LOOKUP
READ/WRITE	-	Perform data transfer requests

In ENTER, LOOKUP, and DELETE, an optional file count parameter can be specified for use with magtape or cassette. Its meaning is as follows:

Count Argument	Meaning
=0	A rewind is done before the operation.
>0	No rewind is done. The value of the count is taken as a limit of how many files to skip before performing the operation (e.g., a count of 2 will skip over, at most, one file. A count of 1 will not skip at all).
<0	A rewind is done. The absolute value of the switch value is then used as the limit.

If the file indicated in the request is located before the limit is exhausted, the search succeeds at that point.

As an example, consider:

```
        .LOOKUP #AREA,#0,#PTR,#5
        BCS     A1
          .
          .
AREA:   .BLKW   10.
PTR:    .RAD50  /MT0/
        .RAD50  /EXAMPLMAC/
```

In this case, the file count argument is +5, indicating that no rewind is to be done and that MT0 is to be searched for the indicated file (EXAMPL.MAC). If the file is not found after four files have been skipped, or if an end-of-tape occurs in that space, the search is stopped, and the tape is positioned either at the end-of-tape (EOT) or at the start of the fifth file. If the named file is found within the five files, the tape is positioned at its start. If the EOT is encountered first, an error is generated.

As another example:

```
.LOOKUP #AREA,#0,#PTR,#-5
```

This performs a rewind, and then uses a file count of five in the same way the above example does.

H.2.4.2 Handler Functions

1. LOOKUP

   If the filename (or the first word of the filename) is null, the operation is considered to be a nonfile-structured LOOKUP. This operation puts the handler into hardware mode. A rewind is automatically done in this case.

   If the filename is not null, the handler tries to find the indicated file. LOOKUP uses the optional file count as illustrated above. Only software functions are allowed.

2. DELETE

   DELETE will delete a file of the indicated name from the device. DELETE also uses the file count argument, and can thus do a delete of a numbered file as well as a delete by name. When a file is deleted from MT or CT, an unused space is created there. However, it is not possible to reclaim that space, as it is when the device is random access. The unused spot will remain until the volume is re-initialized and rewritten.

   If a filename is not present, a nonfile-structured DELETE is performed and the tape is zeroed.

3. ENTER

   ENTER creates a new file of the indicated name on the device. ENTER uses the optional file count, and can thus ENTER a file by name or by number. If ENTER by name is done, the handler deletes any files of the same name it finds in passing. If ENTER by number is done, the indicated number of files is skipped, and the tape is positioned at the start of the next file.

   NOTE

   Care must be used in performing numbered ENTERs, as it is possible to ENTER a file in the middle of existing files and thus destroy any files from the next file to the end of the tape.

   It is also possible to create more than one file with the same name, since ENTER will only delete files of the same name it sees while passing down the tape. If an ENTER is done with a count greater than 0, no rewind is performed before the file is entered. If a file of the same name is present at an earlier spot on the tape, the handler cannot delete it. A nonfile-structured ENTER performs the same function as a nonfile-structured LOOKUP but does not rewind the tape. Since both functions allow writing to the tape without regard to the tape's file structure, they should be used with care on a file-structured tape.

4. CLOSE

CLOSE terminates operations to a file on magtape or cassette and resets the handler to allow more LOOKUPs, ENTERs, or DELETEs. If a CLOSE is not performed to an ENTERed file, the end-of-tape mark will be missing and no new files can be created on that volume. In this case, the last file on the tape must be rewritten and CLOSEd to create a valid volume.

5. READ/WRITE

READ and WRITE are unique in that they can be done either in hardware or software mode. In software mode (file opened with LOOKUP or ENTER), records are written in a fixed size (256 words for magtape, 64 words for cassette). The word count specified in the operation is translated to the correct number of records. On a READ, the user buffer is filled with zeroes if the word count exceeds the amount of data available.

Following is a discussion of how the various parameters for READ/WRITE are used for magtape and cassette.

a. Block Number

For READ operations to magtape, the block number is used for random access (i.e., blocks need not be read sequentially from magtape). WRITE operations, however, disregard the block number and merely write the next sequential block. Block 0 on either READ or WRITE causes the device to rewind to the start of the file. A subsequent WRITE will destroy all previous output to that file.

For cassette, only sequential operations are performed. If the block number is 0, the cassette is rewound to the start of the file. Any other block number is disregarded.

b. Word Count

On a magtape READ, if the word count is 0, the block number is ignored and the next sequential record is read, no matter how big it is.

NOTE

Care must be taken when performing a magtape READ, because the READ will be done even if the record to be read is larger than the buffer allocated.

On a magtape WRITE, if the word count is 0, one 256-word block is written.

If the word count for cassette is 0, the following conditions are possible:

If the block number is non-zero, the operation is actually a file-name seek. The block number is interpreted as the file count argument, as discussed in the above example of LOOKUP. The buffer address should point to the RAD50 of the device and file to be located. This feature essentially allows an asynchronous LOOKUP to be performed. The standard LOOKUP request does not return control to the user program until the tape has been positioned properly, whereas this asynchronous version will return control immediately and interrupt when the file is positioned.

The user may then do a synchronous, positively numbered LOOKUP to the file just positioned, thus avoiding a long synchronous search of the tape.

If the block number is 0, a CRC (cyclical redundancy check) error occurs.

Following is a description of the allowed hardware mode functions for the handlers, as well as examples of how to call them. In general, special functions are called by using the .SPFUN request; examples of usage follow each function. The special functions require a channel number as an argument. The channel must initially be opened with a non-file structured LOOKUP which places the handler in hardware mode.

The general form of the .SPFUN request is:

```
.SPFUN .area,.chan,.code,.buff,.wcnt,.blk,.crtn
```

where:

.code is the function code to be performed.

The request format is:

```
R0 → .area: | 32 | .chan |
            | .blk       |
            | .buff      |
            | .wcnt      |
            | .code  377 |
            | .crtn      |
```

1. Magtape Special Functions

a. Rewind (Code = 373) - Rewinds the tape to the load point. This puts the unit into hardware mode (as does a nonfile-structured LOOKUP) where any of the other functions may be done.

Sample Macro Call:

```
.SPFUN  #AREA,#0,#373,#0
```

The above performs a synchronous rewind on channel 0 (i.e., control will not return before the tape is positioned). An asynchronous rewind could be done with:

```
.SPFUN  #AREA,#0,#373,#0,,,#CRTN;REWIND MT, CHANNEL 0
```

where CRTN is a completion routine to be entered when the operation is finished. The other arguments are not required for this call.

b. Write End-of-File (Code = 377) - This request writes an end-of-file mark, thus terminating a file.

Sample Macro Call:

```
.SPFUN  #AREA,#0,#377,#0;THIS IS THE SYNCHRONOUS FORM
```

The asynchronous form is:

```
.SPFUN  #AREA,#0,#377,#0,,,#CRTN;WRITE EOF ON
                                ;MT, CHANNEL 0
```

c. Forward Space (Code = 376) - This function spaces forward the specified number of records and then stops. If the EOT or EOF is discovered before the count is exhausted, the tape stops there. Note that the number of records to space forward is contained in the word count argument, and the number passed should be the positive value of the desired number.

Sample Macro Call:

```
.SPFUN  #AREA,#0,#376,#0,#2     ;SPACE FORWARD 2
                                ;RECORDS, CHANNEL 0
```

This spaces forward two records.

d. Back Space (Code = 375) - This is similar to Forward Space except the tape is backed up by the indicated number of blocks. Again, the word count must be the positive value of the number to back up.

Sample Macro Call:

```
.SPFUN   #AREA,#0,#375,#0,#2
```

This backs the tape up by two records.

e. Write With Extended Gap (Code = 374) - This request is the same as any of the WRITE requests, except that a longer file gap is written between records. This can be used to get past bad spots on the tape.

Sample Macro Call:

```
.SPFUN   #AREA,#0,#374,#BUFF,#100.,#0
```

This performs a synchronous write, while:

```
.SPFUN   #AREA,#0,#374,#BUFF,#100.,#1,#CRTN
```

is asynchronous and goes to CRTN when the operation is complete.

f. Offline (Code = 372) - This request sets the drive to an off-line state. The unit can only be set on-line by manual control.

Sample Macro Call:

```
.SPFUN   #AREA,#0,#372,#0
```

Since this in an instantaneous function, no asynchronous forms are required.

2. Cassette Special Functions

With exceptions noted, these requests are identical to those involving magtape.

a. Rewind (Code = 373) - This request is identical to the rewind request for magtape, except that unless a completion routine is specified, control does not return to the user until the rewind completes.

b. Last File (Code = 377)- This request rewinds the cassette and positions it immediately before the sentinel file (logical end-of-tape). The macro call is the same as for magtape code 377.

c. Last Block (Code = 376)- This request rewinds one record. See the magtape code 376 for a sample macro call.

d. Next File (Code = 375)- This request spaces the cassette forward to the next file.

e. Next Block (Code = 374)- This request spaces the cassette forward by one record.

f. **Write File Gap (code = 372) - This request terminates a file written by the user program when in hardware mode.**

Sample Macro Call:

```
.SPFUN  #AREA,#0,#372,#0
```

This writes a file gap synchronously, while:

```
.SPFUN  #AREA,#0,#372,#0,,,#1
```

or

```
.SPFUN  #AREA,#0,#372,#0,,,#CRTN
```

performs asynchronous write file gap operations.

H.2.4.3 Magtape and Cassette End-of-File Detection - Since magtape and cassette are sequential devices, the handlers for these devices cannot know in advance the number of blocks in a particular file, and thus cannot determine if a particular read request is attempting to read past the end of the file. User programs can use the following procedures to determine if the handlers have encountered end-of-file in either software or hardware mode.

In software mode, if end-of-file is encountered, magtape and cassette handlers will set the end-of-file bit (bit 13) in the channel status word. The next read attempted on that channel wil return with the carry bit set and with the EMT error byte (absolute location 52) set to indicate an attempt to read past end-of-file. To avoid having magtape or cassette files appear one block longer than they really are, user software should check the end-of-file bit in the channel status word after a magtape or cassette read completes. If the bit is set, this indicates that the read just completed encountered end-of-file.

The following example demonstrates the procedure in software mode:

```
.MCALL ..V2..,.REGDEF,.GTJB,.LOOKUP,.READW
.REGDEF
..V2..
CSWEOF=20000            ;END OF FILE BIT IN CHANNEL STATUS WORD
ERRWRD=52               ;EMT ERROR WORD
CHNL=1                  ;I/O CHANNEL #
        .
        .
        .
.GTJB #LIST,#JOBARG     ;GET JOB PARAMETERS
        .
        .
.LOOKUP #AREA,#CHNL,#FILNAM     ;LOOKUP MAGTAPE OR CASSETTE FILE
        .
        .
        .
.READW #AREA,#CHNL,#BUFF,#400,BLKNUM    ;READ A BLOCK
BCS EMTERR              ;CHECK FOR ERROR
MOV JOBARG+6,R1         ;GET POINTER TO I/O CHANNEL SPACE
                        ;(CHANNEL SPACE IS 5 WORDS PER CHANNEL
                        ;BEGINNING WITH CHANNEL 0).
BIT #CSWEOF,CHNL*10.(R1)        ;IS THE EOF BIT SET FOR THIS CHANNNEL?
BNE EOF                 ;IF NE - YES - EOF ENCOUNTERED ON THIS READ
        .
        .
```

```
EOF:                                  ;END OF FILE CODE
                  .
                  .
                  .
FILNAM: .RAD50 'MT0FILNAMEXT'         ;OR .RAD50 'CT0FILNAMEXT'
AREA:   .BLKW 10
LIST:   .BLKW 2
JOBARG: .BLKW 10                      ;JOB PARAMETERS STORED HERE
```

In hardware mode, the cassette and magtape handlers do not report end-of-file, as they do in software mode. The best way that user programs may determine if a magtape read has encountered a tape mark or if a cassette read has encountered a file gap is to check the device status registers after each hardware mode read is complete.

Examples are:

For TM11/TMA11 Magtape

```
        MTS=172520                    ;TM11 STATUS REGISTER
        MTC=MTS+2                     ;TM11 COMMAND REGISTER
        MTSEOF=40000                  ;EOF BIT IN MTS
        MTSEOT=2000                   ;EOT BIT IN MTS
                  .
                  .
                  .
        .READW #AREA,#CHNL,#BUFF,#400,BLKNUM     ;READ A BLOCK FROM MT
        BCS EMTERR                    ;CHECK ERRORS
        TST @#MTC                     ;ERROR BIT SET IN COMMAND REGISTER?
        BPL NOERR                     ;IF PL - NO
        BIT #MTSEOF,@#MTS             ;YES - WAS IT EOF (TAPE MARK)?
        BNE EOF                       ;IF NE - YES - DO END OF FILE PROCESSING
                  .
                  .
                  .
EOF:                                  ;MT EOF ENCOUNTERED
                  .
                  .
                  .
```

For Cassette

```
        TACS=177500                   ;TA11 CONTROL AND STATUS REGISTER
        TAEOF=4000                    ;EOF BIT IN TACS
        TAEOT=20000                   ;EOT BIT IN TACS
                  .
                  .
                  .
        .READW #AREA,#CHNL,#BUFF,#400,BLKNUM     ;READ FROM CT
        BCS EMTERR                    ;TEST ERRORS
        TST @#TACS                    ;ERROR BIT SET IN TACS?
        BPL NOERR                     ;IF PL - NO
        BIT #TAEOF,@#TACS             ;YES - WAS IT END OF FILE?
        BNE EOF                       ;IF NE - YES
                  .
                  .
                  .
EOF:                                  ;CASSETTE END OF FILE ENCOUNTERED
                  .
                  .
                  .
```

If desired, both the EOF and EOT bits could be checked:

```
BIT #MTSEOF+MTSEOT,@#MTS        ;MT EOF OR EOT?
```

or

```
BIT #TAEOF+TAEOT,@#TACS         ;CT EOF OR EOT?
```

H.2.5 DX (RX01 Diskette)

The .SPFUN request is used to allow reading and writing of absolute sectors on the RX01 diskette. An RX01 disk contains 77 tracks, each having 26 sectors, for a total of 2002 sectors. Each sector is 64 words long. RT-11 normally reads and writes groups of 4 sectors. Sectors may be accessed individually using the .SPFUN request as follows:

```
.SPFUN  .area,.chan,.code,.buff,.wcnt,.blk,.crtn
```

where:

.code — is the function code to be performed (377 for READ physical, 376 for WRITE physical, 375 for WRITE physical with deleted data mark)

.buff — is the location of a 65-word buffer; word 1 is the flag word and is normally set to 0 (if set to 1, a READ on physical sector containing deleted data mark is indicated); words 2-65 are the 64-word area to read (or write) the absolute sector

.wcnt — is the absolute track number, 0 through 76, to be read (or written)

.blk — is the absolute sector number, 1 through 26, to be read (or written)

The diskette should be opened with a nonfile-structured LOOKUP. Note also that the .buff, .wcnt, and .blk arguments have different meanings when used with diskette.

Sample Macro Call:

```
.SPFUN  #RDLIST,#1,#377,#BUFF,#0,#7,#0
                        ;PERFORM A SYNCHRONOUS SECTOR
                        ;READ FROM TRACK 0, SECTOR 7
                        ;(THE VOLUME ID) INTO THE
                        ;65-WORD AREA BUFF
```

RT-11 currently provides only one diskette handler as part of the system. The user may optionally install a second diskette handler by following the procedures outlined in Chapter 4 of the RT-11 System Generation Manual.

```
PP  V02-04  11/12/75    RT-11 MACRO VM02-12    PAGE 1

                        .TITLE  PP  V02-04  11/12/75

                        ; RT-11 HIGH SPEED PAPER TAPE (PC11) PUNCH HANDLER
                        ;
                        ; DEC-11-ORTPA-E
                        ;
                        ; ABC/RGB/DV
                        ;
                        ; MAY 1974
                        ;
                        ; COPYRIGHT (C) 1974,1975
                        ;
                        ; DIGITAL EQUIPMENT CORPORATION
                        ; MAYNARD, MASSACHUSETTS 01754
                        ;
                        ; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
                        ; ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH
                        ; THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE,
                        ; OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE
                        ; AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO
                        ; ONE WHO AGREES TO THESE LICENSE TERMS. TITLE TO AND OWNERSHIP OF THE
                        ; SOFTWARE SHALL AT ALL TIMES REMAIN IN DIGITAL.
                        ;
                        ; THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO
                        ; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
                        ; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
                        ;
                        ; DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE
                        ; OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
                        ; WHICH IS NOT SUPPLIED BY DIGITAL.
                        ;
```

```
PP  V02-04  11/12/75    RT-11 MACRO VM02-12    PAGE 2

        000000'         .CSECT  PC11PP

        000000          R0=%0
        000001          R1=%1
        000002          R2=%2
        000003          R3=%3
        000004          R4=%4
        000005          R5=%5
        000006          SP=%6
        000007          PC=%7

                        ; PAPER TAPE PUNCH CONTROL REGISTERS
        177554          PPS     = 177554        ;PUNCH CONTROL REGISTER
        177556          PPB     = 177556        ;PUNCH DATA BUFFER
        000074          PPVEC   = 74            ;PUNCH VECTOR ADDR

                        ; CONSTANTS FOR MONITOR COMMUNICATION
        000001          HDERR   = 1             ;HARD ERROR BIT
        000054          MONLOW  = 54            ;MONITOR BASE ADDR
        000270          OFFSET  = 270           ;POINTER TO Q MANAGER COMPL ENTRY
        177776          PS      = 177776        ;PSW
        000340          PR7     = 340           ;PRIORITY 7
        000200          PR4     = 200           ;PRIORITY 4
```

```
PP   V02-04   11/12/75    RT-11 MACRO VM02-12    PAGE 3

                                        ; LOAD POINT
 000000  000074                         LOADPT: .WORD   PPVEC           ;ADDR OF INTERRUPT VECTOR
 000002  000034                                 .WORD   PPINT-.         ;OFFSET TO INTERRUPT SERVICE
 000004  000340                                 .WORD   PR7             ;PRIORITY 4
 000006  000000                         PPLQE:  .WORD   0               ;POINTER TO LAST Q ENTRY
 000010  000000                         PPCQE:  .WORD   0               ;POINTER TO CURRENT Q ENTRY

                                        ; ENTRY POINT

 000012  016704  177772                 PP:     MOV     PPCQE,R4        ;R4 POINTS TO CURRENT Q ENTRY
 000016  006364  000006                         ASL     6(R4)           ;WORD COUNT TO BYTE COUNT
 000022  103026                                 BCC     PPERR           ;A READ REQUEST IS ILLEGAL
 000024  052737  000100  177554                 BIS     #100,@#PPS      ;CAUSES INTERRUPT,STARTING TRANSFER
 000032  000207                                 RTS     PC

 000034  000423                                 BR      PPDONE          ;ABORT BY STOPPING

                                        ; INTERRUPT SERVICE

 000036  004577  000064                 PPINT:  JSR     R5,@$INPTR
 000042  000140                                 .WORD   ^C<PR4>&PR7
 000044  016704  177740                         MOV     PPCQE,R4        ;R4 POINTS TO CURRENT Q ENTRY
 000050  005737  177554                         TST     @#PPS           ;ERROR?
 000054  100411                                 BMI     PPERR           ;YES-PUNCH OUT OF PAPER
 000056  062704  000006                         ADD     #6,R4           ;POINT R4 TO BYTE COUNT
 000062  005714                                 TST     @R4             ;ANY MORE CHARS TO OUTPUT?
 000064  001407                                 BEQ     PPDONE          ;NO-TRANSFER DONE
 000066  005214                                 INC     @R4             ;DECREMENT BYTE COUNT (IT IS NEGATIVE)
 000070  115437  177556                         MOVB    @-(R4),@#PPB    ;PUNCH CHARACTER
 000074  005214                                 INC     @R4             ;BUMP POINTER
 000076  000207                                 RTS     PC

 000100  052754  000001                 PPERR:  BIS     #HDERR,@-(R4)   ;SET HARD ERROR BIT

 000104  005037  177554                 PPDONE: CLR     @#PPS           ;CLEAR INTERRUPT ENABLE
 000110  010704                                 MOV     PC,R4
 000112  062704  177676                         ADD     #PPCQE-.,R4     ;ADDR OF NEXT Q ENTRY POINTER
 000116  013705  000054                         MOV     @#MONLOW,R5
 000122  000175  000270                         JMP     @OFFSET(R5)     ;JUMP TO Q MANAGER

 000126  000000                         $INPTR: .WORD   0               ;POINTS TO COMMON ENTRY CODE

         000130                         PPSIZE  =  .-LOADPT

         000001'                        .END
```

```
PP  V02-04  11/12/75     RT-11 MACRO VM02-12     PAGE 3+
SYMBOL TABLE

HDERR = 000001            LOADPT  000000R     002  MONLOW= 000054            OFFSET= 000270            PC      =%000007
PP      000012R     002   PPB     = 177556          PPCQE    000010R     002  PPDONE   000104R     002  PPERR    000100R     002
PPINT   000036R     002   PPLQE    000006R     002  PPS      = 177554          PPSIZE= 000130            PPVEC  = 000074
PR4   = 000200            PR7     = 000340          PS       = 177776          R0      =%000000          R1      =%000001
R2    =%000002            R3      =%000003          R4      =%000004           R5      =%000005          SP      =%000006
SINPTR  000126R     002
. ABS.  000000      000
        000000      001
PC11PP  000130      002
ERRORS DETECTED: 0
FREE CORE: 18104. WORDS

,LP:=PP/C/N:TTM:CND



PP  V02-04  11/12/75     RT-11 MACRO VM02-12     PAGE S-1
CROSS REFERENCE TABLE (CREF V01-04 )


SINPTR   3-20     3-41#
.        3-3      3-37     3-43
HDERR    2-19#    3-33
LOADPT   3-2#     3-43
MONLOW   2-20#    3-38
OFFSET   2-21#    3-39
PP       3-10#
PPB      2-15#    3-29*
PPCQE    3-6#     3-10     3-22     3-37
PPDONE   3-16     3-27     3-35#
PPERR    3-12     3-24     3-33#
PPINT    3-3      3-20#
PPLQE    3-5#
PPS      2-14#    3-13*    3-23     3-35*
PPSIZE   3-43#
PPVEC    2-16#    3-2
PR4      2-24#    3-21
PR7      2-23#    3-4      3-21
PS       2-22#
```

```
PR      V02-03      RT-11 MACRO VM02-10      00:04:49 PAGE 1

                    .TITLE PR    V02-03

                    ; RT-11 HIGH SPEED PAPER TAPE READER (PC11) HANDLER
                    ;
                    ; DEC-11-ORPHA-D
                    ;
                    ; ABC/ECR/RGB/FF
                    ;
                    ; MAY 1974
                    ;
                    ; COPYRIGHT (C) 1974,1975
                    ;
                    ; DIGITAL EQUIPMENT CORPORATION
                    ; MAYNARD, MASSACHUSETTS 01754
                    ;
                    ; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
                    ; ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH
                    ; THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE,
                    ; OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE
                    ; AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO
                    ; ONE WHO AGREES TO THESE LICENSE TERMS. TITLE TO AND OWNERSHIP OF THE
                    ; SOFTWARE SHALL AT ALL TIMES REMAIN IN DIGITAL.
                    ;
                    ; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
                    ; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
                    ; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
                    ;
                    ; DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE
                    ; OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
                    ; WHICH IS NOT SUPPLIED BY DIGITAL.
                    ;
```

```
PR    V02-03      RT-11 MACRO VM02-10     00:04:49 PAGE 2


      000000'                   .CSECT PC11PR

      000000                    R0=%0
      000001                    R1=%1
      000002                    R2=%2
      000003                    R3=%3
      000004                    R4=%4
      000005                    R5=%5
      000006                    SP=%6
      000007                    PC=%7

                                ; PAPER TAPE READER CONTROL REGISTERS
      177550                    PRS     = 177550                ;CONTROL REGISTER
      177552                    PRB     = 177552                ;DATA REGISTER
      000070                    PRVEC   = 70                    ;READER VECTOR ADDR

      000001                    PRGO    = 1                     ;READER ENABLE BIT
      000101                    PINT    = 101                   ;INTERRUPT ENABLE BIT AND GO BIT
                                ; CONSTANTS FOR MONITOR COMMUNICATION
      000001                    HDERR   = 1                     ;HARD ERROR BIT
      000054                    MONLOW  = 54                    ;POINTER TO MONITOR BASE
      000270                    OFFSET  = 270                   ;POINTER TO Q MANAGER COMPLETION ENTRY

```

```
PR    V02-03      RT-11 MACRO VM02-10     00:04:49 PAGE 3

                                        ; LOAD POINT

   000000  000070                       PRSTRT: .WORD   PRVEC           ;ADDR OF INTERRUPT VECTOR
   000002  000044                               .WORD   PRINT-.         ;OFFSET TO INTERRUPT ENTRY
   000004  000340                               .WORD   340             ;PRIORITY 7
   000006  000000                       PRLQE:  .WORD   0               ;POINTER TO LAST Q ENTRY
   000010  000000                       PRCQE:  .WORD   0               ;POINTER TO CURRENT Q ENTRY

                                        ; ENTRY POINT

   000012  016704  177772               PR:     MOV     PRCQE,R4        ;POINTER TO Q ENTRY IN R4
   000016  006364  000006                       ASL     6(R4)           ;WORD COUNT TO BYTE COUNT
   000022  103461                               BCS     PRERR           ;A WRITE REQUEST IS ILLEGAL TO PR
   000024  001446                               BEQ     SEEK            ;A REQUEST FOR 0 BYTES IS A SEEK
   000026  005737  177550                       TST     @#PRS           ;IS READER READY?
   000032  100455                               BMI     PRERR           ;NO, NO TAPE, PROBABLY
   000034  012737  000101  177550               MOV     #PINT,@#PRS     ;ENABLE READER INTERRUPT, GET A CHAR
   000042  000207                               RTS     PC

   000044  000436                               BR      PRABRT          ;ABORT ENTRY FOR F/B


                                        ; INTERRUPT SERVICE

   000046  013746  000054               PRINT:  MOV     @#MONLOW,-(SP)  ;INTO SYSTEM STATE
   000052  004536                               JSR     R5,@(SP)+       ;RETURN AT LEVEL 4
   000054  000140                               .WORD   ^C<200>&340
   000056  016704  177726                       MOV     PRCQE,R4        ;R4 POINTS TO Q ENTRY
   000062  062704  000004                       ADD     #4,R4           ;POINT R4 TO BUFFER ADDRESS
   000066  005737  177550                       TST     @#PRS           ;ANY ERRORS?
   000072  100413                               BMI     PREOF           ;YES-TREAT AS EOF
   000074  113774  177552  000000               MOVB    @#PRB,@(R4)     ;PUT CHAR IN BUFFER
   000102  005224                               INC     (R4)+           ;BUMP BUFFER POINTER
   000104  005314                               DEC     @R4             ;DECREASE BYTE COUNT
   000106  001415                               BEQ     PRDONE          ;IF ZERO,WE ARE DONE
   000110  052737  000001  177550               BIS     #PRGO,@#PRS     ;RE-ENABLE READER
   000116  000207                               RTS     PC

   000120  005214                       PREO1:  INC     (R4)
   000122  105074  000000               PREOF:  CLRB    @(R4)           ;CLEAR REMAINDER OF BUFFER
   000126  005364  000002                       DEC     2(R4)
```

```
41 000132  001372                              BNE     PREO1             ;MORE
42 000134  052774  020000  177772              BIS     #20000,@-6(R4)    ;SET EOF BIT IN CHANNEL STATUS
43
44                                 ; OPERATION COMPLETE
45 000142                          SEEK:
46 000142                          PRABRT:
47 000142  042737  000101  177550  PRDONE: BIC     #PINT,@#PRS       ;TURN OFF THE READER INTERRUPT
48                                                                   ;IN CASE WE GET AN ERROR LATER
49 000150  010704                          MOV     PC,R4
50 000152  062704  177636                  ADD     #PRCQE-.,R4       ;GET ADDR OF CURRENT Q ENTRY
51 000156  013705  000054                  MOV     @#MONLOW,R5
52 000162  000175  000270                  JMP     @OFFSET(R5)       ;TO MONITOR COMPLETION
53
54 000166  052754  000001          PRERR:  BIS     #HDERR,@-(R4)     ;SET HARD ERROR BIT
55 000172  000763                          BR      PRDONE            ;AND COMPLETE OPERATION
56
57         000174                          PRSIZE=.-PRSTRT


PR    V02-03     RT-11 MACRO VM02-10    00:04:49 PAGE 3+


     58         000001'                    .END



PR    V02-03     RT-11 MACRO VM02-10    00:04:49 PAGE 3+
SYMBOL TABLE

HDERR = 000001          MONLOW= 000054          OFFSET= 000270          PC     =%000007          PINT   = 000101
PR      000012R    002  PRABRT  000142R    002  PRB    = 177552          PRCQE   000010R    002  PRDONE  000142R   002
PREOF   000122R    002  PREO1   000120R    002  PRERR   000166R    002  PRGO   = 000001          PRINT   000046R   002
PRLQE   000006R    002  PRS    = 177550          PRSIZE= 000174          PRSTRT  000000R    002  PRVEC  = 000070
R0     =%000000          R1     =%000001          R2     =%000002          R3     =%000003          R4     =%000004
R5     =%000005          SEEK    000142R    002  SP     =%000006
. ABS.  000000     000
        000000     001
PC11PR  000174     002
ERRORS DETECTED: 0
FREE CORE: 16308. WORDS

,LP:=PR/C/N:TTM:CND
```

```
PR     V02-03      RT-11 MACRO VM02-10     00:04:49 PAGE S-1
CROSS REFERENCE TABLE (CREF V01-03 )

.          3-4      3-50     3-57
HDERR      2-21#    3-54
MONLOW     2-22#    3-24     3-51
OFFSET     2-23#    3-52
PINT       2-19#    3-17     3-47
PR         3-11#
PRABRT     3-20     3-46#
PRB        2-15#    3-31
PRCQE      3-7#     3-11     3-27     3-50
PRDONE     3-34     3-47#    3-55
PREO1      3-38#    3-41
PREOF      3-30     3-39#
PRERR      3-13     3-16     3-54#
PRGO       2-18#    3-35
PRINT      3-4      3-24#
PRLQE      3-6#
PRS        2-14#    3-15     3-17*    3-29     3-35*    3-47*
PRSIZE     3-57#
PRSTRT     3-3#     3-57
PRVEC      2-16#    3-3
SEEK       3-14     3-45#
```

H.4 DEC 026/DEC 029 CARD CODE CONVERSION TABLE

Card codes in the following table are broken down by zone punch. Except where noted, all codes apply to both DEC 026 and DEC 029 punched cards.

Table H-1
Card Code Conversions

Zone	Digit	Octal	Character	Name
none				
	none	040		SPACE
	1	061	1	digit 1
	2	062	2	digit 2
	3	063	3	digit 3
	4	064	4	digit 4
	5	065	5	digit 5
	6	066	6	digit 6
	7	067	7	digit 7
12				
(DEC 029)	none	046	&	ampersand
(DEC 026)		053	+	plus sign
	1	101	A	upper case A
	2	102	B	upper case B
	3	103	C	upper case C
	4	104	D	upper case D
	5	105	E	upper case E
	6	106	F	upper case F
	7	107	G	upper case G
11				
	none	055	-	minus sign
	1	112	J	upper case J
	2	113	K	upper case K
	3	114	L	upper case L
	4	115	M	upper case M
	5	116	N	upper case N
	6	117	O	upper case O
	7	107	P	upper case P
0				
	none	060	0	digit 0
	1	057	/	slash
	2	123	S	upper case S
	3	124	T	upper case T
	4	125	U	upper case U
	5	126	V	upper case V
	6	127	W	upper case W
	7	130	X	upper case X
8				
	none	70	8	digit 8
	1	140	`	accent grave
(DEC 029)	2	072	:	colon
(DEC 026)		137	←	backarrow (underscore)
(DEC 029)	3	043	#	number sign

(continued on next page)

Table H-1 (Cont.)
Card Code Conversions

Zone	Digit	Octal	Character	Name
(DEC 026)		075	=	equal sign
	4	100	@	commercial "at"
(DEC 029)	5	047	'	single quote
(DEC 026)		136	↑	uparrow (circumflex)
(DEC 029)	6	075	=	equal sign
(DEC 026)		047	'	single quote
(DEC 029)	7	042	"	double quote
(DEC 026)		134	\	backslash
9				
	none	071	9	digit 9
	2	026	ctrl - V	SYN
	7	004	ctrl - D	EOT
12-11				
	none	174	|	vertical bar
	1	152	j	lower-case J
	2	153	k	lower-case K
	3	154	l	lower-case L
	4	155	m	lower-case M
	5	156	n	lower-case N
	6	157	o	lower-case O
	7	160	p	lower-case P
12-0				
	none	173	{	open brace
	1	141	a	lower-case A
	2	142	b	lower-case B
	3	143	c	lower-case C
	4	144	d	lower-case D
	5	145	e	lower-case E
	6	146	f	lower-case F
	7	147	g	lower-case G
12-8				
	none	110	H	upper-case H
(DEC 029)	2	133	[	open square bracket
(DEC 026)		077	?	question mark
	3	056	.	period
(DEC 029)	4	074	<	open angle bracket
(DEC 026)		051	)	close parenthesis
(DEC 029)	5	050	(	open parenthesis
(DEC 026)		135	]	close square bracket
(DEC 029)	6	053	+	plus sign
(DEC 026)		074	<	open angle bracket
	7	041	!	exclamation mark
12-9				
	none	111	I	upper-case I
	1	001	ctrl - A	SOH
	2	002	ctrl - B	STX
	3	003	ctrl - C	ETX
	5	011	ctrl - I	HT
	7	177		DEL

(continued on next page)

Table H-1 (Cont.)
Card Code Conversions

Zone	Digit	Octal	Character	Name
11-0				
	none	175	}	close brace
	1	176	~	tilde
	2	163	s	lower-case S
	3	164	t	lower-case T
	4	165	u	lower-case U
	5	166	v	lower-case V
	6	167	w	lower-case W
	7	170	x	lower-case X
11-8				
	none	121	Q	upper-case Q
(DEC 029)	2	135	]	close square bracket
(DEC 026)		072	:	colon
	3	044	$	currency symbol
	4	052	*	asterisk
(DEC 029)	5	051	)	close parenthesis
(DEC 026)		133	[	open square bracket
(DEC 029)	6	073	;	semi-colon
(DEC 026)		076	>	close angle bracket
(DEC 029)	7	136	↑	uparrow (circumflex)
(DEC 026)		046	&	ampersand
11-9				
	none	122	R	upper-case R
	1	021	ctrl - Q	DC1
	2	022	ctrl - R	DC2
	3	023	ctrl - S	DC3
	6	010	ctrl - H	BS
0-8				
	null	131	Y	upper-case Y
(DEC 029)	2	134	\	backslash
(DEC 026)		073	;	semi-colon
	3	054	,	comma
(DEC 029)	4	045	%	percent sign
(DEC 026)		050	(	open parenthesis
(DEC 029)	5	137	←	backarrow (underscore)
(DEC 026)		042	"	double quote
(DEC 029)	6	076	>	close angle bracket
(DEC 026)		043	#	number sign
(DEC 029)	7	077	?	question mark
(DEC 026)		045	%	percent sign
0-9				
	null	132	Z	upper-case Z
	5	012	ctrl - J	LF
	6	027	ctrl - W	ETB
	7	033		ESC
9-8				
	4	024	ctrl - T	DC4
	5	025	ctrl - U	NAK
	7	032	ctrl - Z	SUB

(continued on next page)

Table H-1 (Cont.)
Card Code Conversions

Zone	Digit	Octal	Character	Name
12-9-8				
	3	013	ctrl - K	VT
	4	014	ctrl - L	FF
	5	015	ctrl - M	CR
	6	016	ctrl - N	SO
	7	017	ctrl - O	SI
11-9-8				
	none	030	ctrl - X	CAN
	1	031	ctrl - Y	EM
	4	034	ctrl - \	FS
	5	035	ctrl -]	GS
	6	036	ctrl - ↑	RS
	7	037	ctrl - _	US
0-9-8				
	5	005	ctrl - E	ENQ
	6	006	ctrl - F	ACK
	7	007	ctrl - G	BEL
12-0-8	none	150	h	lower-case H
12-0-9	none	151	i	lower-case I
12-11-8	none	161	q	lower-case Q
12-11-9	none	162	r	lower-case R
11-0-8	none	171	y	lower-case Y
11-0-9	none	172	z	lower-case Z
12-11-9-8				
	1	020	ctrl - P	DLE
12-0-9-8				
	1	000		NUL

APPENDIX I

DUMP

RT-11 DUMP is a program which outputs to the console or lineprinter all or any part of a file in octal words, octal bytes, ASCII characters and/or RAD50 characters. DUMP is particularly useful for examining data such as directories or files.

I.1 CALLING AND USING DUMP

DUMP is called using the monitor command:

```
R DUMP
```

in response to the dot printed by the Keyboard Monitor. The name of the file which is to be output is entered as follows in response to the asterisk printed by the Command String Interpreter:

```
dev:output=dev:input/s
```

where:

dev: represents any valid device specification (line printer is default for output if no output file is designated. The default can be changed; refer to Chapter 4 of the RT-11 System Generation Manual.

output represents the filename and extension assigned to the output file. The default extension for file-structured output is .DMP.

input represents the input source filename and extension.

NOTE

Input files residing on magtape and cassette must have been written under the RT-11 system; magtape and cassette files written under other system environments cannot be dumped.

/s represents one or more of the switches listed in Table I-1.

Type CTRL C to halt DUMP at any time and return control to the monitor. To restart DUMP type R DUMP or the REENTER command in response to the monitor's dot.

I.1.1 DUMP Switches

The following switches can appear in the command string for DUMP:

Table I-1
DUMP Switches

Switch	Meaning
/B	Output octal bytes
/E:n	End output at block number n
/G	Ignore input errors
/N	Suppress ASCII output
/O:n	Output only block number n (same as /E:n, /S:n)
/S:n	Start output with block number n
/W	Output octal words
/X	Output RAD50 characters

If neither /W nor /B is given, /W is assumed. ASCII characters are always dumped unless /N is given. The number n is an octal block number.

If an input filename is given, block numbers are relative to the beginning of the file to which the block belongs. If not, block numbers are absolute block numbers on the device (i.e., the physical block numbers on the corresponding device).

I.1.2 Examples

The following are two examples of DUMP. /B is used in the first example to output octal bytes of the file SQRT.FTN into a file called DIF.DMP on device DT1.

```
R DUMP
*DT1:DIF=SQRT.FTN/B
```

If DIF.DMP is then listed on the line printer (using PIP), it appears as follows:

```
BLOCK NUMBER 0000
000/ 001 000 056 000 001 000 011 261 214 072 150 000 000 000 001 257
      .   .   .   .   .   .   .   1   .   !   H   .   .   .   .   /
020/ 224 017 110 001 000 000 262 252 304 037 110 004 210 000 374 252
      .   .   H   .   .   .   2   *   D   .   H   .   .   .   ◊   *
040/ 016 024 110 004 006 000 033 254 217 163 110 004 000 000 012 001
      .   .   H   .   .   .   .   ,   .   S   H   .   .   .   .   .
060/ 000 036 000 001 000 054 253 100 070 100 004 000 000 123 263 024
      .   .   .   .   .   ,   +   @   8   @   .   .   .   S   3   .
100/ 255 150 001 000 000 000 000 000 000 000 003 000 000 032 001 000
      -   H   .   .   .   .   .   .   .   .   .   .   .   .   .   .
```

DUMP

```
120/ 016 000 004 000 007 000 000 000 000 000 000 000 346 001 000 016
      .   .   .   .   .   .   .   .   .   .   .   .   F   .   .   .
140/ 000 003 000 000 000 067 011 076 371 000 000 167 001 000 014 000
      .   .   .   .   .   7   .   >   Y   .   .   W   .   .   .   .
160/ 004 000 004 006 054 253 100 070 226 000 000 000 000 000 000 000
      .   .   .   .   ,   +   @   8   .   .   .   .   .   .   .   .
200/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
220/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
240/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
260/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
300/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
320/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
340/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
360/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
400/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
420/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
440/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
460/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
500/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
520/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
540/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
560/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
600/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
620/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
640/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
660/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
700/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
720/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
740/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .


760/ 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
```

The second example illustrates the use of /X to output RAD50 and octal values for locations in the file. The numbers in the left column represent the byte displacement. ASCII characters are printed in the far right hand column (dot represents a non-printing character):

```
.R DUMP
*RK1:LINK0.FB/X

BLOCK NUMBER 0000
000/ 027011 044524 046124 004505 052122 044514 045516 051040 *..TITLE.RTLINK R*
     GNY    K/L    LHT    ASM    MSZ    K/D    LA8    MEX
020/ 047517 020124 047503 042504 020040 054040 031460 030455 *OOT CODE   X03-1*
     LSW    EF6    LSK    KCL    EEX    ND     HGX    G4/
040/ 006466 035412 051040 026524 030461 046040 047111 042513 *6..; RT-11 LINKE*
     BDV    IQ4    MEX    GJD    G43    LGH    LUA    KCS
060/ 006522 035412 042040 041505 030455 026461 051117 046114 *R..; DEC-11-ORLL*
     BEJ    IQ4    J6     J0U    G4/    GII    MF1    LHL
100/ 026501 026502 040514 005015 020073 040515 020131 032461 *A-B-LA..; MAY 15*
     GIY    GIZ    JQ6    AXM    EFK    JQ7    EGA    HTQ
120/ 020054 034461 032067 005015 020073 050105 020057 047105 *, 1974..; EP/ EN*
     EE6    IFA    HNG    AXM    EFK    L3/    EE9    LT7
140/ 040510 041516 042105 041040 020131 043512 005015 005015 *HANCED BY JG....*
     JQ2    J00    J67    JWH    EGA    KPJ    AXM    AXM
160/ 006473 035412 041440 050117 051131 043511 052110 024040 *;..; COPYRIGHT (*
     BDS    IQ4    J/X    L39    MGA    KPI    MSP    FP2
200/ 024503 024040 034461 032067 006451 035412 006440 035412 *C) (1974)..; ..;*
     FXC    FP2    IFA    HNG    BDI    IQ4    BD     IQ4
220/ 042040 043511 052111 046101 042440 052521 050111 042515 * DIGITAL EQUIPME*
     J6     KPI    MSQ    LHA    KBP    MZA    L33    KCU
240/ 052116 041440 051117 047520 040522 044524 047117 005015 *NT CORPORATION..*
     MSV    J/X    MF1    LSX    JRB    K/L    LUG    AXM
260/ 020073 040515 047131 051101 026104 046440 051501 040523 *; MAYNARD, MASSA*
     EFK    JQ7    LUQ    MFQ    GCL    LMX    ML3    JRC
300/ 044103 051525 052105 051524 030040 033461 032065 005015 *CHUSETTS 01754..*
     KVS    MMM    MSM    MML    G.     H3I    HNE    AXM
320/ 006473 035412 052040 042510 044440 043116 051117 040515 *;..; THE INFORMA*
     BDS    IQ4    MRP    KCP    K.     KI8    MF1    JQ7
340/ 044524 047117 044440 020116 044124 051511 042040 041517 *TION IN THIS DOC*
     K/L    LUG    K.     EF0    KV6    MMA    J6     J01
360/ 046525 047105 020124 051511 051440 041125 042512 052103 *UMENT IS SUBJECT*
     LN7    LT7    EF6    MMA    ML     JXU    KCR    MSK
400/ 052040 006517 035412 041440 040510 043516 020105 044527 * TO..; CHANGE WI*
     MRP    BEG    IQ4    J/X    JQ2    KPN    EFU    K/O
420/ 044124 052517 020124 047516 044524 042503 040440 042116 *THOUT NOTICE AND*
     KV6    MY9    EF6    LSV    K/L    KCK    JP2    J7F
440/ 051440 047510 046125 020104 047516 020124 042502 041440 * SHOULD NOT BE C*
     ML     LSP    LHU    EFT    LSV    EF6    KCJ    J/X
460/ 047117 052123 052522 042105 005015 020073 051501 040440 *ONSTRUED..; AS A*
     LUG    MSS    MZB    J67    AXM    EFK    ML3    JP2
500/ 041440 046517 044515 046524 047105 020124 054502 042040 * COMMITMENT BY D*
     J/X    LN1    K/E    LN6    LT7    EF6    NKJ    J6
520/ 043511 040511 020124 050505 044525 046520 047105 020124 *IGIAT EQUIPMENT *
     KPI    JQ3    EF6    M E    K/M    LN2    LT7    EF6
540/ 047503 050122 051117 052101 047511 027116 005015 020073 *CORPORATION...; *
     LSK    L4B    MF1    MSI    LSQ    GPN    AXM    EFK
560/ 042504 020103 051501 052523 042515 020123 047516 051040 *DEC ASSUMES NO R*
     KCL    EFS    ML3    MZC    KCU    EFS    LSV    MEX
600/ 051505 047520 051516 041111 046111 052111 020131 047506 *ESPONSIBILITY FO*
     ML7    LSX    MMF    JXI    LHI    MSQ    EGA    LSN
620/ 020122 047101 020131 051105 047522 051522 052040 040510 *R ANY ERRORS THA*
     EF4    LT3    EGA    MFU    LSZ    MMJ    MRP    JQ2
640/ 006524 035412 046440 054501 040440 050120 040505 020122 *T..; MAY APPEAR *
     BEL    IQ4    LMX    NKI    JP2    L4     JQ/    EF4
```

```
660/ 047111 052040 044510 020123 047504 052503 042515 052116 *IN THIS DOCUMENT*
     LUA    MRP    K/     EF5    LSL    MYS    KCU    MSV
700/ 006456 035412 005015 020073 044124 051511 051440 043117 *...;..; THIS SOF*
     BDN    IQ4    AXM    EFK    KV6    MMA    ML     KI9
720/ 053524 051101 020105 051511 043040 051125 044516 044123 *TwARE IS FURNISH*
     M86    MFQ    EFU    MMA    KH2    MF7    K/F    KV5
740/ 042105 052040 020117 052520 041522 040510 042523 020122 *ED TO PURCHASER *
     J67    MRP    EF1    MZ     J04    JQ2    KCS    EF4


760/ 047125 042504 020122 006501 035412 046040 041511 047105 *UNDER A..; LICEN*
     LUM    KCL    EF4    BD3    IQ4    LGH    J0Y    LT7
```

I.2 DUMP ERROR MESSAGES

The following errors may occur when using DUMP:

Message	Meaning
?IN ERR?	A hardware error occurred while reading the input file and /G was not specified in the command line.
?OUT ERR?	A hardware error occurred while writing an output file, or the output device was full.
?LP NOT FND?	A line printer handler is not available on the system.

APPENDIX J

FILEX

J.1 FILEX OVERVIEW

FILEX is a general file transfer program used to convert files among devices for various operating systems. Transfers may be initiated between any block-replaceable RT-11 directory-structured device and PDP-11 DOS/BATCH DECtape (as input or output), DOS/BATCH disk (as input only), RSTS DECtape (as input or output), and DECsystem-10 DECtape (as input only).

"Wild-card" names (the *.* construction explained in Chapter 4) are permitted. Magtape and cassette are not supported in any operation.

J.1.1 File Formats

FILEX can transfer files created by three different operating systems--RT-11, DECsystem-10, and DOS/BATCH (PDP-11 Disk Operating System/BATCH). Data formats that may be used in a transfer are ASCII, image, and packed image. Because the file structure and data formats for each system vary somewhat, switches are needed in the command line to indicate the file structures and the data formats involved in the transfer. These switches are defined in Section J.2.1. Devices are assumed to be in RT-11 file structure unless either a /S or /T switch (for DOS/BATCH and RSTS or DECsystem-10 respectively) is indicated. If both input and output devices are RT-11 format (or are not file-structured), FILEX will operate like PIP.

J.2 CALLING AND USING FILEX

FILEX is called from the RT-11 system device by typing:

```
R FILEX
```

in response to the dot printed by the Keyboard Monitor. An asterisk is printed when FILEX is loaded and ready to accept command string input.

Type CTRL C to halt FILEX at any time and return control to the monitor. To restart FILEX, type R FILEX or the REENTER command in response to the monitor's dot.

J.2.1 FILEX Switch Options

Table J-1 lists the switch options which are used for various FILEX operations. /S and /T must appear following the device and filename to which they apply; other switches may appear anywhere in the command line. These switches are explained in more detail following the table.

Table J-1
FILEX Switch Options

Switch	Meaning
/A	Indicates a character-by-character ASCII transfer in which rubouts and nulls are deleted; when /T is also used, each PDP-10 word is assumed to contain five 7-bit ASCII bytes. (The transfer terminates upon reaching a ↑Z for compatibility with RSTS; the ↑Z is not transferred.)
/D	Deletes the named file from the device; valid only for DOS/BATCH and RSTS DECtape.
/F	Causes a "fast" listing of the device directory by listing filenames only.
/I	Performs an image mode transfer; if the input is either DOS/BATCH, RSTS or RT-11, this is a word-for-word transfer; if the input is from DECsystem-10, /I indicates that the file resembles a file created on DECsystem-10 by MACY11, MACX11, or LNKX11 with the /I switch: in this case each DECsystem-10 36-bit word contains one PDP-11 8-bit byte in its low-order bits.
/L	Causes a complete listing of the device directory, including filenames, block lengths, and creation dates, to appear on the console terminal.
/P	Performs a packed image transfer; if the input is either DOS/BATCH, RSTS or RT-11, this is a word-for-word transfer; if the input is from DECsystem-10, /P indicates that the file resembles a file created on DECsystem-10 by MACY11, MACX11, or LNKX11 with the /P switch, in which case each DECsystem-10 36-bit word contains four PDP-11 8-bit bytes aligned on bits 0, 8, 18, and 26. This mode is assumed if no mode switch (/A, /I) is indicated in a command line.
/S	Indicates the device is a legal DOS/BATCH (or RSTS) block-replaceable device; the switch must appear on the

(continued on next page)

Table J-1 (Cont)
FILEX Switch Options

Switch	Meaning
	same side of the command line as the device to which it applies. NOTE Neglecting to include the /S switch in a command line involving DOS/BATCH (or RSTS) causes unpredictable results.
/T	Indicates the device is a legal DECsystem-10 block-replaceable device; the switch must appear on the same side of the command line as the device to which it applies.
/V	Types out version number of FILEX.
/Z	Zeroes the directory of the specified device in the proper format (valid only for DOS/BATCH and RSTS DECtape).

J.2.2 Transferring Files Between RT-11 and DOS/BATCH (or RSTS)

File transfers may be initiated between block-replaceable devices used by RT-11 and the PDP-11 DOS/BATCH system. Input from DOS/BATCH may be either disk or DECtape; both linked and contiguous files are supported as input. If the input device is a DOS/BATCH disk, the user should specify a DOS/BATCH user identification code (called a UIC; refer to the DOS/BATCH Handbook, DEC-11-ODBHA-A-D). This UIC then becomes default for all future transfers. If no UIC is specified, the current default UIC is used (see the description of UIC, following). Output to DOS/BATCH is limited to DECtape only. DECtape used under the RSTS system is also legal as both input and output, since its format is identical to DOS/BATCH DECtape; see the PDP-11 Resource Sharing Time-Sharing System User's Guide, DEC-11-ORSUA-D-D. Any valid RT-11 file storage device may be used for either input or output in the transfer. The RT-11 device DK: is assumed if no device is indicated.

NOTE

An RT-11 DECtape can hold more information than a DOS/BATCH or RSTS DECtape. Thus, caution should be observed in copying files from a full RT-11 tape to a DOS DECtape as some information may not transfer. In such a case an error message will be printed and the transfer will not be completed.

When a transfer from an RT-11 device to a DOS DECtape occurs, the block size of the file may increase. However, if the file is later transferred back to an RT-11 device, the block size does not decrease.

To transfer a file from a legal DOS/BATCH block-replaceable device (or RSTS DECtape) to a legal RT-11 device, the command string format is:

```
*dev:filnam.ext=dev:filnam.ext/S/s[UIC]
```

where:

dev:	= on the output side, any valid RT-11 device (if that device is not file-structured, filnam.ext may be omitted); on the input side, DOS/BATCH DECtape or disk, or RSTS DECtape.
filnam.ext	= for output, any valid RT-11 filename and extension; for input, any valid DOS/BATCH (RSTS) filename and extension.
/S	= the switch from Table J-1 which designates a DOS/BATCH (RSTS) block-replaceable device. (This switch MUST be included in the command line.)
/s	= optionally any other switch (one only) from Table J-1 (in this case, /A is the only meaningful switch which could be chosen); /A indicates an ASCII transfer is to be performed in which nulls and rubouts are deleted; if /A is not used, the transfer will be performed word-for-word.
[UIC]	= the DOS/BATCH user identification code in the format [nnn,nnn] where nnn represents 1 to 3 octal digits less than or equal to 377 specifying first the user-group number, and then the individual user number; this code may be placed anywhere on the side of the command line containing the DOS/BATCH device; whenever a UIC is entered, it becomes the default for any further DOS/BATCH transfers; the initial default value is [1,1].

NOTE

A UIC need not be indicated in any command line if accessing only DECtape since individual users do not "own" files on DECtape under DOS; no error will occur if a code is used, however.

To transfer files from an RT-11 storage device to a DOS/BATCH or RSTS DECtape, the command line format is as follows:

```
*DTn:filnam.ext/S/s=dev:filnam.ext
```

where:

DTn:	= a valid DOS/BATCH (RSTS) DECtape assignment (only DECtape is legal for output).
filnam.ext	= for output, any valid DOS/BATCH (RSTS) filename and extension; for input, any valid RT-11 filename and extension.
/S	= the switch from Table J-1 which designates a DOS/BATCH (RSTS) block-replaceable device. (This switch MUST be included in the command line.)

/s = optionally any other switch (one only) from Table J-1 (in this case, /A is the only meaningful switch which could be chosen); /A indicates an ASCII transfer is to be performed in which nulls and rubouts are deleted; if /A is not used, the transfer will be performed word-for-word.

dev: = any valid RT-11 device.

Examples:

```
*DT2:SORT.ABC/S=SORT.ABC
```

This command instructs FILEX to transfer a file called SORT.ABC from the RT-11 system device to a DOS/BATCH (or RSTS) formatted DECtape on unit DT2.

```
*PP:=DT2:TYPE.FIL/S/A
```

This command allows a file to be transferred from DOS/BATCH (or RSTS DECtape to the papertape punch under RT-11. The transfer is done in ASCII mode.

```
*DK:*.*=RK1:MACR1.MAC/S[200,200]
```

This command causes the file MACR1.MAC from the DOS/BATCH disk on unit 1 which is stored under the UIC [200,200] to be transferred to the RT-11 device DK. [200,200] becomes the default UIC for any further DOS/BATCH operations.

J.2.3 Transferring Files to RT-11 from DECsystem-10

Files may be transferred to RT-11 devices from a DECsystem-10 DECtape whenever a foreground job is not running (this restriction is due to the fact that when reading DECsystem-10 files, FILEX accesses the DECtape control registers directly rather than using the RT-11 DECtape control handler). Output may be to any valid RT-11 device; DECsystem-10 DECtape is the only valid input device. The format of the command line is:

```
*dev:filnam.ext=DTn:filnam.ext/T/s
```

where:

dev: = any valid RT-11 device; if that device is not file-structured, the filnam.ext may be omitted.

filnam.ext = for output, any valid RT-11 filename and extension; for input, any valid DECsystem-10 filename and extension (see NOTE below).

DTn: = a valid DECsystem-10 DECtape assignment (only DECtape is legal for input).

/T = the switch from Table J-1 which signifies a DECsystem-10 DECtape. When using /T, especially with /A, the time of day clock will lose time. It should be examined with the monitor TIME command, and reset if necessary.

/s = any other switch from Table J-1 which specifies the mode of transfer. Only one additional switch may be indicated per transfer; /P is assumed if no other mode is specified.

NOTE

RT-11 files may be indirectly converted to DECsystem-10 format by running RT-11 FILEX and converting the files to DOS formatted DECtape, and then running DECsystem-10 FILEX to read the DOS DECtape; however, it is currently not possible under RT-11 to convert files directly from RT-11 to DECsystem-10 format.

Examples:

```
*DT2:STAND.LIS=DT1:STAND.LIS/T/A
```

The ASCII file STAND.LIS is converted from DECsystem-10 ASCII format to RT-11 ASCII format and stored under RT-11 on DECtape 2 as STAND.LIS.

NOTE

Transfers from DECsystem-10 DECtape to RT-11 DECtape may cause an <UNUSED> block to appear after the file on the RT-11 device. This is a result of the method by which RT-11 handles the increased amount of information on a DECsystem-10 DECtape.

```
*SY:*.NEW=DT0:* LIS/T
```

This command indicates that all files on DECsystem-10 DECtape 0 with the extension .LIS are to be transferred to the RT-11 system device (disk or DECtape) using the same filename and an extension of .NEW. The /P switch is the assumed transfer mode.

J.2.4 Listing Directories

Device directories of any of the block-replaceable devices used in a FILEX transfer can be listed on the console terminal. The following FILEX command lines are used:

*dev:/L	for RT-11
*dev:/L/S	for DOS/BATCH (RSTS-11)
*DTn:/L/T	for DECsystem-10

where:

dev: = disk or DECtape (DK is assumed if no device is specified).

DTn: = DECsystem-10 DECtape.

/L = the switch from Table J-1 which indicates a listing is desired (/F may be substituted if a "fast listing" is preferred).

/S = the switch from Table J-1 which designates a DOS/BATCH or RSTS-11 block-replaceable device.

/T = the switch from Table J-1 which designates a DECsystem-10 block-replaceable device.

Examples:

```
*RK1:/L/S
BADB   .SYS      1  22-JUL-74
MONLIB.CIL    175C  22-JUL-74
DU11   .PAL     45  24-JUL-74
VERIFY.LDA     67C  22-JUL-74
CILUS  .LDA     39  22-JUL-74
```

This command lists the complete disk directory of the device RK1. The letter "C" following the file size on a DOS/BATCH or RSTS directory listing indicates the file is a contiguous file.

```
*DT1:*.PAL/L/S
```

This command lists all files with the extension .PAL which are on drive 1.

```
*DT1:*.*/F/T
```

All files on DECsystem-10 formatted DECtape 1, regardless of filename or extension, are listed; a fast directory is requested (/F) in which only filenames are printed.

J.2.5 Deleting Files from DOS/BATCH (RSTS) DECtapes

FILEX may be used to delete files from, and zero directories of, DOS/BATCH and RSTS formatted DECtapes. The format of these command lines is:

```
*dev:filnam.ext/S/D      to delete a file
```

or

```
*dev:/S/Z                to zero a directory
dev:/Z  ARE YOU SURE?
```

where:

dev: = DOS/BATCH or RSTS DECtape.

filnam.ext = a valid DOS/BATCH (RSTS) filename and extension.

/S = the switch from Table J-1 which designates that the device is a DOS/BATCH (RSTS) block-replaceable device.

/D = the switch from Table J-1 which designates that a file is to be deleted.

/Z = the switch from Table J-1 which designates that a directory is to be zeroed.

Examples:

```
*DT0:*.PAL/D/S
```

All files on DECtape 0 with the extension .PAL are deleted.

```
*DT2:TABLE.OBJ/D/S
```

The file TABLE.OBJ is deleted from the DECtape on unit 2.

```
*DT0:/Z/S
DT0:/Z  ARE YOU SURE ?Y
```

The DECtape on drive 0 is initialized in DOS/BATCH (RSTS) format so that it contains no files.

J.3 FILEX ERROR MESSAGES

Following is a list of error messages that may occur under FILEX:

Message	Meaning
?filnam.ext ALREADY EXISTS?	An attempt was made to create the named file (filnam.ext) on a DOS DECtape when a file already existed under the name specified. Use /D to delete the file, and retry the transfer.
?BAD PPN?	The DOS/BATCH user identification code was not in the form [nnn,nnn], where each nnn is an octal number less than or equal to 377(octal).
?COR OVR?	There was insufficient main storage for buffers and input list expansion. Try copying files one at a time, without using the "wild-card" (*.*) construction on input.
?DEV FULL?	There was no room in the directory for the filename or there was no room on the output device for the file (in which case the filename is not placed in the output device directory).
?DIR ERR?	An error occurred while reading or looking up the directory of the input device, or the

	input device does not have the proper file structure.
?FEATURE NOT IMP?	An operation was attempted which FILEX cannot perform (e.g., zeroing an RT-11 device).
?FG ACTIVE?	An attempt was made to use /T when a foreground job was active. The transfer is not allowed until the foreground job is terminated and unloaded via UNLOAD FG.
?FIL NAM?	The output filename was invalid or null.
?FIL NOT FND?	The input file was not found, or the "wild-card" (*.*) construction matched none of the existing files.
?ILL CMD?	The length of the command line exceeded 72 characters; the command line was not in proper CSI format; the UIC exceeded the allowed number of characters or [was used without]; a "wild-card" (*.*) construction was used on a sequential-access device; no output or no input file was specified for a copy operation, or more than one filename construction (dev:filnam.ext) was specified on either side of the = (<) sign.
?ILL DEV?	The device handler was not found, an invalid or illegal device name was used, or one of the following was attempted: RK or DT was not used for DOS/BATCH (RSTS) input in a copy operation; DT was not used for DOS/BATCH (RSTS) output in a zero or delete operation; DT was not used for DOS/BATCH (RSTS) output in a copy operation; DT was not used for DECsystem-10 input in any operation.
?ILL SWT?	An illegal switch was used in a command line (e.g., a switch not listed in Table J-1).
?IN ERR?	A device error occurred on input.
?NO UFD?	The specified UFD was not found on the DOS input disk.
?OUT ERR?	A device error occurred on output.
?SWT ERR?	An attempt was made to use more than one /S or /T switch in a command line (only one is allowed); an attempt was made to use more than one transfer mode switch (/I, /P, /A) or more than one operation switch (/D, /L, /F, /Z) in a command line (only one of each is allowed).

APPENDIX K

SOURCE COMPARE (SRCCOM)

The RT-11 Source Compare program (SRCCOM) is used to compare two ASCII files and to output any differences to a specified output device. It is particularly useful when the two files are different versions of a single program, in which case SRCCOM prints all the editing changes which transpired between the two versions.

K.1 CALLING AND USING SRCCOM

To run SRCCOM type the command:

```
R SRCCOM
```

followed by a carriage return in response to the dot printed by the Keyboard Monitor; the CSI prints an asterisk. Then enter the names of the files which are to be compared using a command string in the following format:

```
dev:output=dev:input1,dev:input2/s
```

where:

dev:	is any valid device specification.
output	is the filename and extension assigned to the output file. If no output file is indicated, output is directed to the terminal.
input1...	are the input source filenames and extensions to be compared.
/s	is one of the switches listed in Table K-1.

Source files are examined line by line for groups of lines which match. When a mismatch occurs, all differences are output until n successive lines in the first file are identical to n lines in the second file. The number (n) is a variable which the user can set with the /L switch.

K.1.1 Extensions

No default extension is assigned by SRCCOM to the output file. The default extension for an input file is .MAC, representing a source file in MACRO language.

K.1.2 Switches

Command switches are generally placed at the end of the command string but may follow any filename in the string. The following switches can appear in the command string:

Table K-1
SRCCOM Switches

Switch	Meaning
/B	Compare blank lines. Without this switch, blank lines are ignored.
/C	Ignore comments (all text on a line preceded by a semicolon) and spacing (spaces and tabs). This switch does not cause a line consisting entirely of a comment to become a blank line, and therefore ignored in the line count.
/F	Include form feeds in the output file. (Form feeds are still compared if /F is not used, but they are not included in the output of differences.)
/H	Type list of switches available (help text). No I/O device is necessary since /H always prints the help text on the terminal.
/L:n	Specify the number of lines that determines a match (n is an octal number <=310). All differences occurring before and after a match are output. In addition, the first line of the current match is output after the differences to aid in locating the place within each file at which the differences occurred. The default value for n is 3.
/S	Ignore spaces and tabs.

K.2 OUTPUT FORMAT

The first line of each file is always output as identification and is also compared. A blank line is then printed, followed by the differences between the files, in the following format:

Source Compare

```
        1)1         FILEA
        1)          A
        ****
        2)1         FILEB
        2)          A

        ***********

        % FILES ARE DIFFERENT
```

The different lines are listed followed by a reference line which is the same for both files. Note the example below.

The following example uses SRCCOM to compare an edited file and its backup version. The default value for a match is 3 lines. Blank lines are ignored but all other characters are compared.

Following the example is a coded explanation of the comparison.

```
     .R SRCCOM
     *RK1:LINK0.FB,LINK0.BAK
A {  1)1                .TITLE  RTLINK ROOT CODE    X03-16
  {  2)1                .TITLE  RTLINK ROOT CODE    X03-15

  {  1)1                SEVENK= 31452             ;MINIMUM CORE TO START LINKER
B {  1)
  {  1)
C {  1)                 .MCALL  .CSISPC,.CSIGEN,.SETTOP,.LOCK,.UNLOCK
     ****
  {  2)1                SEVENK= 31500             ;JUST BELOW 8K RESIDENT
B {  2)
  {  2)
C {  2)                 .MCALL  .CSISPC,.CSIGEN,.SETTOP,.LOCK,.UNLOCK
     **********
B {  1)2                .GLOBL  RSWIT,RELPTR,FBTXT,OVLNUM,RELOVL,RLSTRT
C {  1)                 .GLOBL  RELADR,PNRELO,RELID1,RSIZ1,OVSIZ1,OVLCDE
     ****
B {  2)2                .GLOBL  RSWIT,RELPTR,FBTXT,OVLNUM,RELOVL
C {  2)                 .GLOBL  RELADR,PNRELO,RELID1,RSIZ1,OVSIZ1,OVLCDE
     **********
B {  1)2      RLSTRT:   .BLKW             ;CURRENT REL BLK OVERLAY NUM
C {  1)       RELPTR:   .BLKW             ;POINTER TO CURRENT REL BLK LOCATION
     ****
C {  2)2      RELPTR:   .BLKW             ;POINTER TO CURRENT REL BLK LOCATION
     **********
B {  1)2      MTITLE:   .ASCII  /RT-11 LINK    X03-16/
C {  1)                 .ASCII  /    LOAD MAP /
     ****
B {  2)2      MTITLE:   .ASCII  /RT-11 LINK    X03-15/
C {  2)                 .ASCII  /    LOAD MAP /
     **********
  {  1)12               .IF DF FB
B {  1)                 MOV     OBLK,RLSTRT      ;IND START OF OVL FOR REL BLK
  {  1)                 .ENDC
C {  1)                 BR      1$
     ****
C {  2)12               BR      1$
     **********

D {  %FILES ARE DIFFERENT
```

A Headers, consisting of the first line of each file; for identification purposes.

B n)m. A notation where n is the number of the input file, and m is the page number (less than 256 decimal) of the input file on which the text appears. The right column lists the lines in the files which are different.

C Following a section of differences, a line identical to each file is output for reference purposes.

D Indicates that the files are different (this is printed on the system console, not in the output file).

This example uses the /L:n switch and sets the number of lines that determines a match to 2 lines. The first two columns represent the input files:

```
TEST FILE 1
LINE C
LINE E
LINE C
LINE D
LINE F
LINE H
LINE I
LINE J
```

```
TEST FILE 2
LINE C
LINE D
LINE C
LINE E
LINE F
LINE G
LINE H
LINE I
LINE J
```

The files are compared and differences listed on the line printer.

```
*LP:=TEST1,TEST2/L:2

1)1     TEST FILE 1
2)1     TEST FILE 2
```

```
1)1     LINE E
1)      LINE C
1)      LINE D
1)      LINE F
1)      LINE H
****
2)1     LINE D
2)      LINE C
2)      LINE E
2)      LINE F
2)      LINE G
2)      LINE H
***********
```

This message prints on the terminal indicating that the files are different.

```
%FILES ARE DIFFERENT
```

K.3 SRCCOM ERROR MESSAGES

The following errors may be reported by SRCCOM:

Messages	Meaning
?COR OVR?	Not enough memory to hold a particular difference section.
?IN ERR?	A hardware error occurred in reading input.
?OUT ERR?	A hardware error occurred in writing output file, or output device full.
?SWITCH ERROR?	An invalid switch was found or a switch other than /L was given a value.
?TOO MUCH DIFFERENCE?	More than 310 (octal) lines of difference between two files were found.

APPENDIX L

PATCH

The PATCH utility program is used to make code modifications to memory image (.SAV) files, including overlay-structured and monitor files. PATCH, like ODT, can be used to interrogate, and then to change, words or bytes in the file.

PATCH provides eight relocation registers. Before changing a program with PATCH, copy the old file to a backup file with PIP, as the old file is modified when PATCH is used.

L.1 CALLING AND USING PATCH

To run PATCH, type the command:

```
R PATCH
```

followed by the RETURN key in response to the dot printed by the monitor. PATCH prints a version number message:

```
PATCH V01-02
```

and then prints the message:

```
FILE NAME --
*
```

In response to the asterisk, enter the name of the file to be modified, using the following format:

```
dev:filnam.ext/M/O
```

where:

dev:	represents an optional device specification; if not specified, DK: is assumed.
filnam.ext	represents the name of the file which is to be patched, if an extension is not indicated, .SAV is assumed.
/M	must be used if the file is an RT-11 monitor file.
/O	must be used if the file is an overlay-structured file.

After this information has been entered, the Command String Interpreter prints an asterisk indicating that it is ready to accept a command. Note that /O and /M, if used, must be specified when the file name is typed; they are not legal at any other time.

L.2 PATCH COMMANDS

Table L-1 summarizes the PATCH commands.

Table L-1
PATCH Commands

Command	Action
Vr;nR	Set relocation register n to value Vr.
b;B	Set bottom address of overlay file to b.
[s:]r,o/	Open word location Vr + o in overlay segment s.
[s:]r,o\	Open byte location Vr + o in overlay segment s.
<CR>	Close currently open word/byte.
<LF>	Close currently open word/byte and open the next one.
↑ or ^	Close currently open word/byte and open the previous one.
@	Close the currently open word and open the word addressed by it.
F	Begin patching a new file.
E	Exit to RT-11 monitor.

Explanations of each command follow. An example of the use of the commands is provided in Section L.3.

L.2.1 Patch a New File

The F command causes PATCH to close the file being patched, and accept a new file name to be patched.

L.2.2 Exit from PATCH

The E command causes PATCH to close the file being patched and return control to the RT-11 monitor.

L.2.3 Examine, Change Locations in the File

For a non-overlay file, a word address may be opened, as with ODT, by typing:

```
[<relocation register>,]offset/
```

At this point, PATCH will type out the contents of the location and wait for the user to type in either new location contents (in octal) or another command.

In an overlay file, the format is:

[<segment number>:][<relocation register>,]offset/

Where <segment number> is the overlay segment number as it is printed on the link map for the file. If it is omitted, the root segment is assumed.

Similarly, to open a byte address in file, the format is:

```
[<relocation register>,]offset\
```

for non-overlay files, or

[<segment number>:][<relocation register>,]offset\

for overlay files.

Once a location has been opened, the user may optionally type in the new contents in the format:

```
[<relocation register>,]value
```

followed by one of these control characters:

<carriage return>	Close the current location by changing its contents to the new contents (if specified), and await more control input.
<line feed>	Close the current location, and open the next word/byte.
↑ or ^	Close the current location, and open the previous word/byte.
@	Close the current word location, and open the word addressed by it (in the same segment if an overlay file).

L.2.4 Set Bottom Address

To patch an overlay file, PATCH must know the bottom address at which the program was linked if it is different from the initial stack pointer. This is the case if the program sets location 42 in an .ASECT. To set the bottom address, type:

<bottom address>;B

Note that the B command must be issued before any locations are opened for modification.

L.2.5 Set Relocation Registers

The relocation registers 0-7 are set, as with ODT, by the R command. The R command has the format:

<relocation value>;<relocation register>R

Once one of the eight relocation registers has been set, the expression:

<relocation register>,<octal number>

typed as part of a command will have the value:

<relocation value> + <octal number>

L.3 EXAMPLES USING PATCH

The following example shows how to patch a non-overlaid file. Assume the following program (EXAM):

```
.MAIN.  RT-11 MACRO VM02-08    PAGE 1


        000015            CR=     15
        000012            LF=     12
        000000'           .CSECT  MAIN
                          .MCALL  .PRINT,.EXIT
                          .NLIST  BEX
 000000      124 MSG:     .ASCII  /THIS IS A SUCCESSFUL PATCH/<CR><LF>
                          .LIST   BEX
 00034  000403 START:     BR      EXIT
 00036                    .PRINT  #MSG
 00044           EXIT:    .EXIT
        000034'           .END    START
```

This program has been assembled with MACRO and linked with LINK; execution causes no output of text:

```
.R EXAM
```

To make a line of text print on the terminal, PATCH is used as follows:

```
 R PATCH

PATCH V01-02

FILE NAME--
*EXAM.SAV
*1000;0R
*0,34/  403     240
*E
```

Now when the program is executed:

```
 R EXAM
THIS IS A SUCCESSFUL PATCH
```

The next example demonstrates a similar situation, only includes an overlay file. These programs have been assembled and linked; the output of both operations is included:

```
 .MAIN.   RT-11 MACRO VM02-08   3-SEP-74 PAGE 1


 1
 2
 3         000015          CR=     15
 4         000012          LF=     12
 5         000007          PC=     %7
 6         000000'         .CSECT  MAIN
 7                         .GLOBL  ENTRY,MSG1
 8                         .MCALL  .PRINT,.EXIT
 9                         .NLIST  BEX
10 00000      124 MSG:     .ASCIZ  /THIS IS A SUCCESSFUL PATCH/<CR><LF>
11 00035      124 MSG1:    .ASCIZ  /THIS IS AN OVERLAY PATCH/
12                         .LIST   BEX
13 00066 000403 START:     BR      EXIT
14 00070                   .PRINT  #MSG
15 00076 004767 EXIT:      JSR     PC,ENTRY
         000000G
16 00102                   .EXIT
17         000066'         .END    START
```

```
.MAIN.  RT-11 MACRO VM02-08   3-SEP-74 PAGE 1

 1
 2
 3          000015          CR=     15
 4          000012          LF=     12
 5          000007          PC=     %7
 6          000000'         .CSECT  OVL
 7                          .MCALL  .PRINT
 8                          .GLOBL  MSG1
 9                          .GLOBL  ENTRY
10 00000 000403 ENTRY:      BR      RETURN
11 00002                    .PRINT  #MSG1
12 00010 000207 RETURN:     RTS     PC
13          000001'         .END
```

```
RT-11 LINK    X03-18      LOAD MAP
PTCH  .SAV                03-SEP-74

SECTION ADDR    SIZE      ENTRY   ADDR     ENTRY   ADDR     ENTRY   ADDR

. ABS.  000000  001122
MAIN    001122  000104    MSG1    001157

OVERLAY REGION 000001     SEGMENT 000001
OVL     001230  000012    ENTRY   001230

TRANSFER ADDRESS = 001210
HIGH LIMIT = 001242
```

Running the program (PTCH) produces no terminal output:

```
.R PTCH

.
```

But by using PATCH to modify the file as follows:

```
.R PATCH

PATCH V01-02

FILE NAME--
*PTCH.SAV/O
*1230;0R
*1:0,0/ 403     240
*E
```

the following line results:

```
.R PTCH
THIS IS AN OVERLAY PATCH
```

L.4 PATCH ERROR MESSAGES

Error messages which may occur under PATCH follow.

Message	Meaning
?ADDR NOT IN SEG?	The address is not in the specified overlay segment.
?BAD SWITCH?	Typed a switch other than /O or /M.
?BOTTOM ADDR WRONG?	The bottom address specifed or contained in location 42 of an overlay file is incorrect. Specify the correct one using the b:B command.
?INCORRECT FILE SPEC?	The response to the "FILE NAME --" message was not of the correct form. Try again.
?INSUFFICIENT CORE?	PATCH did not have enough memory to hold the file's device handler plus the internal "segment table." This message should not occur.
?INVALID RELOC REG?	Tried to reference a relocation register outside the range 0-7.
?INVALID SEG NO?	The segment number S: does not exist.
?MUST OPEN WORD?	The @ command was typed when a byte location was open.
?MUST SPECIFY SEG?	The address referenced is not in the root section; a segment number S: must be used.

?NO ADDR OPEN?	The <line feed>, ↑ or @ command was typed when no location was open.
?NOT IN PROGR BOUNDS?	Tried to open a location beyond the end of the file.
?ODD ADDRESS?	Tried to open a word address which was odd. (Use "\".)
?ODD BOTTOM ADDR?	The bottom address specified or contained in location 42 of an overlay file is odd.
?PROG HAS NO SEGS?	The file specified as an overlay file is not.
?READ ERROR?	File I/O error in reading.
?WRITE ERROR?	File I/O error in writing.

APPENDIX M

PATCHO

The RT-11 PATCHO program is used to correct and update object modules (files output by the assemblers or by the FORTRAN compiler). It is particularly useful when making corrections to routines that are in .OBJ format for which the source files are not available. PATCHO cannot be used to patch libraries built by LIBR, but it can be used to patch the OBJ modules from which a library is built.

M.1 CALLING AND USING PATCHO

To run PATCHO type the command:

```
R PATCHO
```

in response to the dot printed by the Keyboard Monitor. When PATCHO is ready to accept commands, an asterisk is printed. The standard RT-11 command string is not used for PATCHO. Specific commands (described in Section M.2) are typed in response to the asterisk.

Type CTRL C to halt PATCHO at any time and return control to the monitor. To restart PATCHO, type R PATCHO or the REENTER command in response to the monitor's dot.

M.2 PATCHO COMMANDS

There are nine commands and arguments accepted by PATCHO. All arguments to a command must be separated from the command name by one or more spaces (e.g., POINT TOP is acceptable, POINTTOP is not). Commands are terminated by a carriage return.

M.2.1 OPEN Command

The OPEN command sets input and output file names. When the command is given, PATCHO prints:

```
ENTER INPUT FILE*
```

on the console terminal, and waits for a standard RT-11 device and file specification to be entered (dev:filnam.ext) . After the input specification is given, followed by a carriage return, PATCHO responds with:

ENTER OUTPUT FILE*

A device and file specification (followed by a carriage return) for the desired output file is now accepted from the console.

NOTE

There is no default extension. Therefore, the user must explicitly specify the filename and extension.

Example:

```
*OPEN
ENTER INPUT  FILE *RK1:OTS.OBJ
ENTER OUTPUT FILE *RK1:NEWOTS.OBJ
```

M.2.2 POINT Command

The POINT command locates an object module of a given name (used with concatenated object modules) and prepares it for subsequent WORD, BYTE, or DUMP operations. POINT takes one argument--the module name to be located.

Example:

```
*POINT OTINIT
```

M.2.3 WORD Command

The WORD command modifies a given word in an object module. There may be several arguments to the command, entered as illustrated below (the BYTE command is also included since its arguments are exactly the same; refer to Section M.2.4):

```
        Address Specifier           Value Specifier

{ WORD } CSECT + OFFSET = { # } NAME OF         { + } OFFSET
{ BYTE } NAME             { % } CSECT OR        { - }
                          {   } GLOBAL SYMBOL
```

where in the Address Specifier:

1. CSECT is the name of the CSECT which contains the word to be modified. The CSECT must be present in the current module (the one specified in the POINT command).

The CSECT NAME is optional; if omitted, the blank CSECT is assumed. To represent a CSECT name which contains embedded blanks (. ABS.), a backarrow (SHIFT O) or underscore character should be used in place of each embedded blank.

2. OFFSET is the octal location within the CSECT that is to be modified. OFFSET must be present.

3. = is a delimiter and must be present.

In the Value Specifier:

1. If # is used, the absolute value of the OFFSET in the Value Specifier field is placed in the target location (the address indicated by the Address Specifier).

   For example:

   ```
   *WORD PROG+24=#4
   ```

   This command patches location 24 in CSECT PROG to contain the value 4.

2. If % is used, displaced relocation is generated on the Value Specifier. This mode is used for PC-relative references. For example:

   ```
   *WORD PROG+24=%PROG1+24
   ```

   This command patches PROG+24 to contain the value:

   Address (PROG1+24)-Address (PROG+26)

3. If neither # or % are specified, the address of the Value Specifier is placed in the target location. For example:

   ```
   *WORD PROG+24=PROG+14
   ```

   This causes the word at PROG+24 to contain the address of the word at PROG+14, while:

   ```
   *WORD PROG+24=SYMBOL+0
   ```

   causes the word at PROG+24 to contain the address of the global symbol "SYMBOL".

4. GLOBAL is optional and is the name of a global symbol whose value (address) is to be used in the word to be modified.

5. The characters + or - are optional and indicate the sign of the following OFFSET. If a global symbol is indicated, a + (or -) and OFFSET must be present.

M.2.4 BYTE Command

The BYTE command modifies a given byte in an object module. The arguments are the same as for WORD, explained in Section M.2.3.

Example:

```
*BYTE CSECTA+21=#101
```

The byte in CSECTA whose offset is 21 is patched to contain octal 101 (ASCII "A").

M.2.5 DUMP Command

The DUMP command prints the contents of the object module currently being pointed to and causes an automatic POINT to the next module in the input file, if any. Use the monitor SET LP CR command first for correct output format.

NOTE

> LIST and DUMP go to LP: (lineprinter) by default. If the user wants LIST and DUMP to go to the terminal, he must execute an ASSIGN TT:6 command before typing R PATCHO.

Refer to Section M.4 for an example of the DUMP command.

M.2.6 LIST Command

The LIST command lists the names of all the object modules in the input file in the order in which they appear in the file. A POINT command should be given after the LIST command to assure that PATCHO is positioned at the desired module for the next operation. LIST lists the names of all object modules in a file without changing the module being pointed to.

Refer to Section M.4 for an example of the LIST command.

M.2.7 EXIT Command

The EXIT command returns to the operating system terminating a patch session and closing a file. The EXIT command takes no argument. As the EXIT command is executed, ENTER CHECKSUM:- - - is displayed. The user should type a six-digit octal number corresponding to the appropriate checksum for the patch (if any). If the checksum agrees, PATCHO prints STOP-- and returns control to the monitor. If the checksum does not agree, PATCHO prints PAUSE -- ?BAD PATCH? and waits for user input. The user has two options: typing a CTRL C at this point aborts PATCHO without closing the output file, thus allowing PATCHO to be rerun and the patch to be reentered. For example:

```
:
*EXIT
ENTER CHECKSUM: 115570

PAUSE -- ?BAD PATCH?
^C

.
```

Typing a carriage return causes the output file to be closed, despite the bad patch, and returns control to the monitor. Note that in the latter case, if the output and input files were called by the same name and extension, the input file is destroyed and the output file contains an incorrect patch.

M.2.8 DEC Command

The DEC command is used generally by maintainers when creating system patches and when the proper checksum for the patch being made is unknown. If a DEC command is issued during a patch session, the EXIT command is automatically modified to display:

```
ENTER CHECKSUM:
```

followed by the correct checksum for the patch just completed. The checksum computed by PATCHO is derived from all of the commands entered during the session, thereby providing a safeguard against typographical errors. If the DEC command is used, it is the first command given.

```
*DEC
.
.
.
*EXIT
ENTER CHECKSUM: 115570
STOP --
.
```

M.2.9 HELP Command

The HELP command prints an explanation of PATCHO commands. The HELP command takes no arguments.

M.3 PATCHO LIMITATIONS

Note the following limitations on the use of PATCHO:

1. PATCHO only works with object modules or concatenated object modules, not with libraries.

2. PATCHO always copies its input file to its output file as it makes changes.

3. As a consequence of 2 (above) the POINT command can only be used to point to modules that physically follow the current module in the input file.

4. If changes are being made to a file, PATCHO must be given an EXIT command when all the changes are complete in order to assure that the entire input is copied to the output (i.e., CTRL C should not be used).

5. The LIST and DUMP commands will work in an 8K system only if an ASSIGN TT:6 command has been entered prior to running PATCHO. Otherwise 12K is required for these two commands.

M.4 EXAMPLES

The following is an example of the PATCHO command LIST in which a few of the names of object modules in the library file (OTS.OBJ) are listed:

```
*LIST

OBJECT MODULES:
ERRSS
IVEC
IVECP
IPVEC
FVEC
FVECP
FPVEC
DVEC
DVECP
DPVEC
LVEC
LVECP
LPVEC
ERRS
AOTS
OTINIT
```

The next example is a sample of output produced by the PATCHO DUMP Command. The module is dumped by formatted binary block.

```
*DUMP

DUMP OF MODULE LPS0
--------------------

BLOCK TYPE GSD
GLOBAL   USAGE      DEFINED    RELOC      EXTERNAL  SIZE/ADRS
LPS0     MOD NAME    NO         NO         NO              0
. ABS.   CSECT       YES        NO         NO              0
ERRARG   GLOBAL      NO         NO         YES             0
ERRPDL   GLOBAL      NO         NO         YES             0
ERRSYN   GLOBAL      NO         NO         YES             0

BLOCK TYPE GSD
GLOBAL   USAGE      DEFINED    RELOC      EXTERNAL  SIZE/ADRS
EVAL     GLOBAL      NO         NO         YES             0
           .
           .
           .
BLOCK TYPE GSD
GLOBAL   USAGE      DEFINED    RELOC      EXTERNAL  SIZE/ADRS
USE      GLOBAL      YES        YES        YES           104
. ABS.   TRAN ADR    YES        NO         NO              1

BLOCK TYPE GSD END


BLOCK TYPE RLD
ADDRESS RLD TYPE  GLOBAL     OFFSET

  1772  LCTR DEF                  0
```

```
BLOCK TYPE TXT
ADDRESS CONTENTS
      0         0      0    0
      2         0      0    0
      4         0      0    0
      6         0      0    0
     10         0      0    0
     12         0      0    0
      .
      .
      .
   1476     12602    202   25
   1500     12600    200   25
   1502       207    207    0

BLOCK TYPE MOD END
```

M.5 PATCHO ERROR MESSAGES

Message	Explanation
?BAD CHECKSUM?	A formatted binary block in the input file has a checksum which does not agree with that calculated for its contents.
?BAD OBJ?	The input file contains information which cannot be interpreted as an object module.
?DUMP ERROR?	An input/output error occurred while dumping a module.
?ILLEGAL COMMAND?	A command line was not recognized, or it was not in the proper format for the particular operation.
?MODULE NOT FOUND?	The module requested in a POINT command was not found in the input file between the position of the file at the time of the POINT and the file's end.
?MORE THAN 15 CHANGES?	Too many changes have been specified for a particular module. The patch should be broken up into several steps.
?MORE THAN 5 CSECTS REQUIRE CHANGE?	An attempt has been made to patch locations in too many different CSECTS. The patch should be made in several steps.
?NO FILE OPEN?	An attempt was made to use a command other than "DEC" or "HELP" before an OPEN command was issued.
?OFFSET?	The offset supplied in a WORD or BYTE command is not an octal number, or is in improper format.

Message	Explanation
?OUTPUT ERROR?	A hardware error (or possibly a write-lock condition) occurred while attempting to write the output file.
?OUTPUT FILE TOO SMALL?	The space allocated to the output file is too small. This may be corrected by compressing the device with PIP (if enough total space is free on the device), or by using another device for output.
PAUSE -- ?BAD PATCH?	The checksum entered on exit does not agree with that calculated by PATCHO.

M.5.1 Run-Time Error Messages

Because PATCHO is a FORTRAN program, run-time error messages may occur. To find a complete explanation of each run-time error refer to the RT-11/RSTS/E FORTRAN IV User's Guide or the RT-11 System Message Manual. Listed below are four of the most important run-time error messages which may be encountered.

23	FATAL	HARDWARE I/O ERROR A hardware error has been detected during an I/O operation.
28	FATAL	OPEN FAILED FOR FILE A file could not be found, there was no room in the device, or FORTRAN selected a channel already in use.
29	FATAL	NO ROOM FOR DEVICE HANDLER There is not enough free memory left to accommodate a specific device handler.
30	FATAL	NO ROOM FOR BUFFERS There is not enough free memory left to set up required I/O buffers.

APPENDIX N

DISPLAY FILE HANDLER

This appendix describes the assembly language graphics support provided under RT-11 for the GT40, GT44, and DECLab-11 display hardware systems.

The following manuals are suggested for additional reference:

For GT40 users:

1. GT40 User's Guide (DEC-11-HGTGA-A-D)

For GT44 users:

1. GT44 User's Guide (DEC-11-HGT44-A-D)

For DECLab-11 users:

1. VT11 Graphics Display Processor Manual (DEC-11-HVGTA-A-D)
2. RT-11 BASIC Reference Manual (DEC-11-LBACA-B-D)

N.1 DESCRIPTION

The GT40, GT44, and DECLab-11 have hardware configurations that include a display processor and CRT (cathode ray tube) display. The GT44 has a 17-inch tube; the GT40 and DECLab-11 use a 12-inch tube. Both systems are equipped with light pens and hardware character and vector generators, and are capable of high-quality graphics. The Display File Handler supports this graphics hardware at the assembly language level under the RT-11 monitor.

N.1.1 Assembly Language Display Support

The Display File Handler is not an RT-11 device handler, since it does not use the I/O structure of the RT-11 monitor. For example, it is not possible to use PIP to transfer a text file to the display via the Display File Handler. Rather, the Display File Handler provides the graphics programmer the means for the display of graphics files and the easy management of the display processor. Included in its capabilities are such services as interrupt handling, light pen support, tracking object, and starting and stopping of the display processor.

The Display File Handler manages the display processor by means of a base segment (called VTBASE) which contains interrupt handlers, an internal display file and some pointers and flags. The display processor cycles through the internal display file; any user graphics files to be displayed are accessed via display subroutine calls from the Handler's display file. In this way, the Display File Handler exerts control over the display processor, relieving the assembly language user of the task.

Through the Display File Handler, the programmer can insert and remove calls to display files from the Handler's internal display file. Up to two user files may be inserted at one time, and that number may be increased by re-assembling the Handler. Any user file inserted for display may be blanked (the subroutine call to it bypassed) and unblanked by macro calls to the Display File Handler.

Since the Handler treats all user display files as graphics subroutines to its internal display file, a display processor subroutine call is required. This is implemented with software, using the display stop instruction, and is available for user programs. This instruction and several other extended instructions implemented with the display stop instruction are described in Section N.3.

The facilities of the Display File Handler are accessed through a library of macros (VTMAC) which generate calls to a set of subroutines in VTLIB. VTMAC is the MACRO library, and its call protocol is similar to that of the RT-11 macros. The expansion of the macros is shown in Section N.6. VTMAC also contains, for convenience in programming, the set of recommended display processor instruction mnemonics and their values. The mnemonics are listed in Section N.7 and are used in the examples throughout this appendix.

VTCAL1 through VTCAL4 are the set of subroutines which service the VTMAC calls. They include functions for display file and display processor management. These are described in detail in Section N.2. VTCAL1 through VTCAL4 are currently constructed, along with the base segment VTBASE, as a library of (five) modules, called VTLIB.

N.1.2 Monitor Display Support

The RT-11 monitor, under Version 2, directly supports the display as console device. A keyboard monitor command, GT ON (GT OFF) described in Section 2.7.1, permits the selection of the display as console device. Selection results in the allocation of approximately 1.25K words of memory for text buffer and code. The buffer holds approximately 2000 characters.

The text display includes a blinking cursor to indicate the position in the text where a character is added. The cursor initially appears at the top left corner of the text area. As lines are added to the text the cursor moves down the screen. When the maximum number of lines are on the screen, the top line is deleted from the text buffer when the line feed terminating a new line is received. This causes the appearance of "scrolling", as the text disappears off the top of the display.

When the maximum number of characters have been inserted in the text buffer, the scroller logic deletes a line from the top of the screen to make room for additional characters. Text may appear to move (scroll) off the top of the screen while the cursor is in the middle of a line.

The Display File Handler can operate simultaneously with the scroller program, permitting graphic displays and monitor dialogue to appear on the screen at the same time. It does this by inserting its internal display file into the display processor loop through the text buffer. However, the following should be noted. Under the Single-Job Monitor, if a program using the display for graphics is running with the scroller in use (that is, GT ON is in effect), and the program does a soft exit (.EXIT with R0 not equal to 0) with the display stopped, the display remains stopped until a CTRL C is typed at the keyboard.

This can be recognized by failure of the monitor to echo on the screen when expected. If the scroller text display disappears after a program exit, always type CTRL C to restore. If CTRL C fails to restore the display, the running program probably has an error.

Four scroller control characters provide the user with the capability of halting the scroller, advancing the scrolling in page sections, and printing hard copy from the scroller. These are described in Chapter 2.

NOTE

The scroller logic does not limit the length of a line, but the length of text lines affects the number of lines which may be displayed, since the text buffer is finite. As text lines become longer, the scroller logic may delete extra lines to make room for new text, temporarily decreasing the number of lines displayed.

N.2 DESCRIPTION OF GRAPHICS MACROS

The facilities of the Display File Handler are accessed through a set of macros, contained in VTMAC, which generate assembly language calls to the Handler at assembly time. The calls take the form of subroutine calls to the subroutines in VTLIB. Arguments are passed to the subroutines through register 0 and, in the case of the .TRACK call, through both register 0 and the stack.

This call convention is similar to Version 1 RT-11 I/O macro calls, except that the subroutine call instruction is used instead of the EMT instruction. If a macro requires an argument but none is specified, it is assumed that the address of the argument has already been placed in register 0. The programmer should not assume that R0 is preserved through the call.

N.2.1 .BLANK

The .BLANK request temporarily blanks the user display file specified in the request. It does this by by-passing the call to the user display file, which prevents the display processor from cycling through the user file, effectively blanking it. This effect can later be cancelled by the .RESTR request, which restores the user file. When the call returns, the user is assured the display processor is not in the file that was blanked.

Macro Call: .BLANK .faddr

where: .faddr is the address of the user display file to be blanked.

Errors:

No error is returned. If the file specified was not found in the Handler file or has already been blanked, the request is ignored.

N.2.2 .CLEAR

The .CLEAR request initializes the Display File Handler, clearing out any calls to user display files and resetting all of the internal flags and pointers.

After initialization with .LNKRT (Section N.2.4), the .CLEAR request can be used any time in a program to clear the display and to reset pointers. All calls to user files are deleted and all pointers to status buffers are reset. They must be re-inserted if they are to be used again.

Macro Call: .CLEAR

Errors:

None.

Example:

This example uses a .CLEAR request to initialize the Handler, then later uses the .CLEAR to re-initialize the display. The first .CLEAR is used for the case when a program may be restarted after a CTRL C or other exit.

```
        BR      RSTRT
EX1:    BIS     #20000,@#44     ;SET RENTER BIT IN JSW
RSTRT:  .UNLNK                  ;CLEARS LINK FLAG FOR RESTART
        .LNKRT                  ;SET UP VECTORS, START DISPLAY
        .CLEAR                  ;INITIALIZE HANDLER.
        .INSRT  #FILE1          ;DISPLAY A PICTURE
1$:     .TTYIN                  ;WAIT FOR A KEY STRIKE
        CMPB    #12,R0          ;LINE FEED?
        BNE     1$              ;NO,LOOP
        .CLEAR                  ;YES, CLEAR DISPLAY
        .INSRT  #FILE2          ;DISPLAY NEW PICTURE
                .
                .
FILE1:  POINT                   ;AT POINT (0,500)
        0
        500
        LONGV                   ;DRAW A LINE
        500!INTX                ;TO (500,500)
        0
        DRET
        0
```

```
FILE2:  POINT           ;AT POINT (500,0)
        500
        0
        LONGV           ;DRAW A LINE
        0!INTX          ;TO (500,500)
        500
        DRET
        0

        .END    EX1
```

N.2.3 .INSRT

The .INSRT request inserts a call to the user display file specified in the request into the Display File Handler's internal display file. .INSRT causes the display processor to cycle through the user file as a subroutine to the internal file. The handler permits two user files at one time. The call inserted in the handler looks like the following:

```
DJSR        ;DISPLAY SUBROUTINE
.+4         ;RETURN ADDRESS
.faddr      ;SUBROUTINE ADDRESS
```

The call to the user file is removed by replacing its address with the address of a null display file. The user file is blanked by replacing the DJSR with a DJMP instruction, bypassing the user file.

Macro Call: .INSRT .faddr

where: .faddr is the address of the user display file to be inserted.

Errors:

The .INSRT request returns with the C bit set if there was an error in processing the request. An error occurs only when the Handler's display file is full and cannot accept another file. If the user file specified exists, the request is not processed. Two display files with the same starting address cannot be inserted.

Example:

See the examples in Sections N.2.2 and N.2.4.

N.2.4 .LNKRT

The .LNKRT request sets up the display interrupt vectors and possibly links the Display File Handler to the scroll text buffer in the RT-11 monitor. It must be the first call to the Handler, and is used whether or not the RT-11 monitor is using the display for console output (i.e., the KMON command GT ON has been entered).

The .LNKRT request used with the Version 2 RT-11 monitor enables a display application program to determine the environment in which it is operating. Error codes are provided for the situations where there is no display hardware present on the system or the display hardware is already being used by another task (e.g., a foreground job in the Foreground/Background version).

The existence of the monitor scroller and the size of the Handler's subpicture stack are also returned to the caller. If a previous call to .LNKRT was made without a subsequent .UNLNK, the .LNKRT call is ignored and an error code is returned.

Macro Call: .LNKRT

Errors:

Error codes are returned in R0, with the N condition bit set.

Code	Meaning
-1	No VT11 display hardware is present on this system.
-2	VT11 hardware is presently in use.
-3	Handler has already been linked.

On completion of a successful .LNKRT request, R0 will contain the display subroutine stack size, indicating the depth to which display subroutines may be nested. The N bit will be zero.

If the RT-11 monitor scroll text buffer was not in memory at the time of the .LNKRT, the C bit will be returned set. The KMON commands GT ON and GT OFF cannot be issued while a task is using the display.

Example:

```
START:  .LNKRT                          ;LINK TO MONITOR
        BMI     ERROR                   ;ERROR DOING LINK
        BCS     CONT                    ;NO SCROLL IF C SET
        .SCROL  #SBUF                   ;ADJUST SCROLL PARAMETERS
CONT:   .INSRT  #FILE1                  ;DISPLAY A PICTURE
1$:     .TTYIN                          ;WAIT FOR KEY STRIKE
        CMPB    #12,R0                  ;LINE FEED?
        BNE     1$                      ;NO, LOOP
        .UNLNK                          ;YES, UNLINK AND EXIT
        .EXIT

SBUF:   .BYTE   5                       ;LINE COUNT OF 5
        .BYTE   7                       ;INTENSITY 7(SCALE OF 1-8)
        .WORD   1000                    ;POSITION OF TOP LINE

FILE1:  POINT                           ;AT POINT (500,500)
        500
        500
        CHAR                            ;DISPLAY SOME TEXT
        .ASCII  /THIS IS FILE1. TYPE CR TO EXIT/
        .EVEN
        DRET
        0

ERROR:  Error routine
```

N.2.5 .LPEN

The .LPEN request transfers the address of a light pen status data buffer to VTBASE. Once the buffer pointer has been passed to the Handler, the light pen interrupt handler in VTBASE will transfer display processor status data to the buffer, depending on the state of the buffer flag.

The buffer must have seven contiguous words of storage. The first word is the buffer flag, and it is initially cleared (set to zero) by the .LPEN request. When a light pen interrupt occurs, the interrupt handler transfers status data to the buffer and then sets the buffer flag non-zero. The program can loop on the buffer flag when waiting for a light pen hit (although doing this will tie up the processor, and in a Foreground/Background environment, timed waits would be more desirable). No further data transfers take place, despite the occurrence of numerous light pen interrupts, until the buffer flag is again cleared to zero. This permits the program to process the data before it is destroyed by another interrupt.

The buffer structure looks like this:

Buffer Flag
Name
Subpicture Tag
Display Program Counter (DPC)
Display Status Register (DSR)
X Status Register (XSR)
Y Status Register (YSR)

The Name value is the contents of the software Name Register (described in N.3.5) at the time of interrupt. The Tag value is the tag of the subpicture being displayed at the time of interrupt. The last four data items are the contents of the display processor status registers at the time of interrupt. They are described in detail in Table N-1.

Macro Call: .LPEN .baddr

where: .baddr is the address of the 7-word light pen status data buffer.

Errors:

None.

If a .LPEN was already issued and a buffer specified, the new buffer address replaces the previous buffer address. Only one light pen buffer can be in use at a time.

Example:

```
        .INSRT  #LFILE          ;DISPLAY LFILE
        .LPEN   #LBUF           ;SET UP LPEN BUFFER
LOOP:   TST     LBUF            ;TEST LBUF FLAG, WHICH
        BEQ     LOOP            ;WILL BE SET NON-ZERO
                                ;ON LIGHT PEN HIT.
; PROCESS DATA IN LBUF HERE.
        CLR     LBUF            ;CLEAR THE BUFFER FLAG,
                                ;PERMITTING ANOTHER "HIT".
        BR      LOOP            ;GO WAIT FOR IT.
```

```
LBUF:    .BLKW   7               ;SEVEN WORD LPEN BUFFER
LFILE:
```

.
.
.

Table N-1
Description of Display Status Words

Bits	Significance
DISPLAY PROGRAM COUNTER (DPC=172000)	
0-15	Address of display processor program counter at time of interrupt.
DISPLAY STATUS REGISTER (DSR=172002)	
0-1	Line Type
2	Spare
3	Blink
4	Italics
5	Edge Indicator
6	Shift Out
7	Light Pen Flag
8-10	Intensity
11-14	Mode
15	Stop Flag
X STATUS REGISTER (XSR=172004)	
0-9	X Position
10-15	Graphplot Increment
Y STATUS REGISTER (YSR=172006)	
0-9	Y Position
10-15	Character Register

N.2.6 .NAME

The .NAME request has been added to the Version 2 Handler. The contents of the name register are now stacked when a subpicture call is made. When a light pen interrupt occurs, the contents of the name register stack may be recovered if the user program has supplied the address of a buffer through the .NAME request.

The buffer must have a size equal to the stack depth (default is 10) plus one word for the flag. When the .NAME request is entered, the address of the buffer is passed to the Handler and the first word (the flag word) is cleared. When a light pen hit occurs, the stack's contents are transferred and the flag is set non-zero.

Macro Call: .NAME .baddr

where:	.baddr	is the address of the name register buffer.

Errors:

None.

If a .NAME request has been previously issued, the new buffer address replaces the previous buffer address.

N.2.7 .REMOV

The .REMOV request removes the call to a user display file previously inserted in the handler's display file by the .INSRT request. All reference to the user file is removed, unlike the .BLANK request, which merely bypasses the call while leaving it intact.

Macro Call: .REMOV .faddr

where:	.faddr	is the address of the display file to be removed.

Errors:

No errors are returned. If the file address given cannot be found, the request is ignored.

N.2.8 .RESTR

The .RESTR request restores to view a user display file that was previously blanked by a .BLANK request. It removes the by-pass of the call to the user file, so that the display processor once again cycles through the user file.

Macro Call: .RESTR .faddr

where:	.faddr	is the address of the user file that is to be restored to view.

Errors:

No errors are returned. If the file specified cannot be found, the request is ignored.

N.2.9 .SCROL

This request is used to modify the appearance of the Display Monitor's text display. The .SCROL request permits the programmer to change the maximum line count, intensity and the position of the top line of text of the scroller. The request passes the address of a two-word buffer which contains the parameter specifications. The first byte is the line count, the second byte is the intensity, and the second word is the Y position. Both line count and Y position must be octal numbers. The intensity may be a number from 1 to 8, ranging from lowest to highest intensity. If an intensity of 1 is specified, the scroller text will be almost unnoticeable at a BRIGHTNESS knob setting less than one-half. The scroller parameter change is temporary, since an .UNLNK or CTRL C restores the previous values.

Macro Call: .SCROL .baddr

where: .baddr is the address of the two-word scroll parameters buffer.

Errors:

No errors are returned. No checking is done on the values of the parameters. A zero argument is interpreted to mean that the parameter value is not to be changed. A negative argument causes the default parameter value to be restored.

Example:

```
                .SCROL  #SCBUF          ;ADJUST SCROLL PARAMETERS
                        .
                        .
        SCBUF:  .BYTE   5               ;DECREASE # LINES TO 5.
                .BYTE   0               ;LEAVE INTENSITY UNCHANGED.
                .WORD   300             ;TOP LINE AT Y = 300.
```

N.2.10 .START

The .START request starts the display processor if it was stopped by a .STOP directive. If the display processor is running, it is stopped first, then restarted. In either case, the subpicture stack is cleared and the display processor is started at the top of the handler's internal display file.

Macro Call: .START

Errors:

None.

N.2.11 .STAT

The .STAT request transfers the address of a seven-word status buffer to the display stop interrupt routine in VTBASE. Once the transfer has been made, display processor status data is transferred to the buffer by the display stop interrupt routine in VTBASE whenever a

.DSTAT or .DHALT instruction is encountered (see Sections N.3.3 and N.3.4). The transfer is made only when the buffer flag is clear (zero). After the transfer is made, the buffer flag is set non-zero and the .DSTAT or .DHALT instruction is replaced by a .DNOP (Display NOP) instruction.

The status buffer must be a seven-word, contiguous block of memory. Its contents are the same as the light pen status buffer. For a detailed description of the buffer and an explanation of the status words, see section N.2.5 and Table N-1.

Macro Call: .STAT .baddr

where: .baddr is the address of the status buffer receiving the data.

Errors:

No errors are indicated. If a buffer was previously set up, the new buffer address is replaced as the old buffer address.

N.2.12 .STOP

The .STOP request "stops" the display processor. It actually effects a stop by preventing the DPU from cycling through any user display files. It is useful for stopping the display during modification of a display file, a risky task when the display processor is running. However, a .BLANK could be equally useful for this purpose, since the .BLANK request does not return until the display processor has been removed from the user display file being blanked.

Macro Call: .STOP

Errors:

None.

NOTE

Since the display processor must cycle through the text buffer in the Display Monitor in order for console output to be processed, the text buffer remains visible after a .STOP request is processed, but all user files disappear.

N.2.13 .SYNC/.NOSYN

The .SYNC and .NOSYN requests provide program access to the power line synchronization feature of the display processor. The .SYNC request enables synchronization and the .NOSYN request disables it (the default case).

Synchronization is achieved by stopping the display and restarting it when the power line frequency reaches a trigger point, e.g., a peak or zero-crossing. Synchronization has the effect of fixing the display refresh time. This may be useful in some cases where small amounts of material are displayed but the amount frequently changes, causing changes in intensity. In most cases, however, using synchronization increases flicker.

Macro Calls: .SYNC
.NOSYN

Errors:

None.

N.2.14 .TRACK

The .TRACK request causes the tracking object to appear on the display CRT at the position specified in the request. The tracking object is a diamond-shaped display figure which is light-pen sensitive. If the light pen is placed over the tracking object and then moved, the tracking object follows the light pen, trying to center itself on the pen.

The tracking object first appears at a position specified in a two-word buffer whose address was supplied with the .TRACK request. As the tracking object moves to keep centered on the light pen, the new center position is returned to the buffer. A new set of X and Y values is returned for each light pen interrupt.

The tracking object cannot be lost by moving it off the visible portion of the display CRT. When the edge flag is set, indicating a side of the tracking object is crossing the edge of the display area, the tracking object stops until moved toward the center. To remove the tracking object from the screen, repeat the .TRACK request without arguments.

The .TRACK request may also include the address of a completion routine as the second argument. If a .TRACK completion routine is specified, the light pen interrupt handler passes control to the completion routine at interrupt level. The completion routine is called as a subroutine and the exit statement must be an RTS PC. The completion routine must also preserve any registers it may use.

Macro Call: .TRACK .baddr, .croutine

where:

.baddr	is the address of the two-word buffer containing the X and Y position for the track object.
.croutine	is the address of the completion routine.

Errors:

None.

Example:

See Section N.10.

N.2.15 .UNLNK

The .UNLNK request is used before exiting from a program. In the case where the scroller is present, .UNLNK breaks the link, established by .LNKRT, between the Display File Handler's internal display file and the scroll file in the Display Monitor. The display processor is started cycling in the scroll text buffer, and no further graphics may be done until the link is established again. In the case where no scroller exists, the display processor is simply left stopped.

Macro Call: .UNLNK

Errors:

No errors are returned. An internal link flag is checked to determine if the link exists. If it does not exist, the request is ignored.

N.3 EXTENDED DISPLAY INSTRUCTIONS

The Display File Handler offers the assembly language graphics programmer an extended display processor instruction set, implemented in software through the use of the Load Status Register A (LSRA) instruction. The extended instruction set includes: subroutine call, subroutine return, display status return, display halt, and load name register.

N.3.1 DJSR Subroutine Call Instruction

The DJSR instruction (octal code is 173400) simulates a display subroutine call instruction by using the display stop instruction (LSRA instruction with interrupt bits set). The display stop interrupt handler interprets the non-zero word following the DJSR as the subroutine return address, and the second word following the DJSR as the address of the subroutine to be called. The instruction sequence is:

```
DJSR
Return address
Subroutine address
```

Example:

To call a subroutine SQUARE:

```
POINT           ;POSITION BEAM
100             ;AT (100,100)
100
DJSR            ;THEN CALL SUBROUTINE
.+4
SQUARE          ;TO DRAW A SQUARE
DRET
0
```

The use of the return address preceding the subroutine address offers several advantages. BASIC/GT uses the return address to branch around subpicture tag data stored following the subpicture address. This structure is described in Section N.5.3. In addition, a subroutine may be temporarily bypassed by replacing the DJSR code with a DJMP instruction, without the need to stop the display processor to make the by-pass.

The address of the return address is stacked by the display stop interrupt handler on an internal subpicture stack. The stack depth is conditionalized and has a default depth of 10. If the stack bottom is reached, the display stop interrupt handler attempts to protect the system by rejecting additional subroutine calls. In that case, the portions of the display exceeding the legal stack depth will not be displayed.

N.3.2 DRET Subroutine Return Instruction

The DRET instruction provides the means for returning from a display file subroutine. It uses the same octal code as DJSR, but with a single argument of zero. The DRET instruction causes the display stop interrupt handler to pop its subpicture stack and fetch the subroutine return address.

Example:

```
SQUARE: LONGV                    ;DRAW A SQUARE
        100!INTX
        0
        0!INTX
        100
        100!INTX!MINUSX
        0
        0!INTX
        100!MINUSX
        DRET                     ;RETURN FROM SUBPICTURE
        0
```

N.3.3 DSTAT Display Status Instruction

The DSTAT instruction (octal code is 173420) uses the LSRA instruction to produce a display stop interrupt, causing the display stop interrupt handler to return display status data to a seven-word user status buffer. The status buffer must first have been set up with a .STAT macro call (if not, the DSTAT is ignored and the display is resumed). The first word of the buffer is set non-zero to indicate the transfer has taken place, and the DSTAT is replaced with a DNOP (display NOP). The first word is the buffer flag and the next six words contain name register contents, current subpicture tag, display program counter, display status register, display X register, and display Y register. After transfer of status data, the display is resumed.

N.3.4 DHALT Display Halt Instruction

The DHALT instruction (octal code is 173500) operates similarly to the DSTAT instruction. The difference between the two instructions is that the DHALT instruction leaves the display processor stopped when exiting from the interrupt. A status data transfer takes place provided the buffer was initialized with a .STAT call. If not, the DHALT is ignored.

Example:

```
        .STAT   #SBUF           ;INIT BUFFER
        MOV     #DHALT,STPLOC   ;INSERT DHALT
        .INSRT  #DFILE          ;DISPLAY THE PICTURE
1$:     TST     SBUF            ;DHALT PROCESSED?
        BEQ     1$              ;NO, WAIT
          .
          .
SBUF:   .BLKW   7               ;STATUS BUFFER
DFILE:  POINT                   ;POSITION NEAR TOP OF 12" TUBE
        .WORD   500,1350
        LONGV                   ;DRAW A LINE, MAYBE OVER EDGE
        .WORD   0,400           ;IF IT IS A 12" SCOPE.
STPLOC: DNOP                    ;STATUS WILL BE RETURNED AT THIS POINT
        DRET
        0
```

N.3.5 DNAME Load Name Register Instruction

The Display File Handler provides a name register capability through the use of the display stop interrupt. When a DNAME instruction (octal code is 173520) is encountered, a display stop interrupt is generated. The display stop handler stores the argument following the DNAME instruction in an internal software register called the "name register". The current name register contents are returned whenever a DSTAT or DHALT is encountered, and more importantly, whenever a light pen interrupt occurs. The use of a "name" (with a valid range from 1 to 77777) enables the programmer to label each element of the display file with a unique name, permitting the easy identification of the particular display element selected by the light pen.

The name register contents are stacked on a subpicture call and restored on return from the subpicture.

Example:

The SQUARE subroutine with "named" sides:

```
SQUARE: DNAME                   ;NAME IS
        10                      ;10
        LONGV                   ;DRAW A SIDE
        100!INTX
        0
        DNAME                   ;THIS SIDE IS NAMED
        11                      ;11
        0!INTX                  ;STILL IN LONG VECTOR MODE
        100
        DNAME
        12
        100!INTX!MINUSX
        0
        DNAME
```

```
13
0;INTX
100;MINUSX
DRET                    ;RETURN FROM SUBPICTURE
0
```

N.4 USING THE DISPLAY FILE HANDLER

Graphics programs which intend to use the Display File Handler for display processor management can be written in MACRO assembly language. The display code portions of the program may use the mnemonics described in Section N.7. Calls to the Handler should have the format described in Section N.6.

The Display File Handler is supplied in two pieces, a library of MACRO calls and a library containing the Display File Handler modules.

MACRO Library: VTMAC.MAC

Display File Handler: VTHDLR.OBJ (consisting of:)

VTBASE.OBJ
VTCAL1.OBJ
VTCAL2.OBJ
VTCAL3.OBJ
VTCAL4.OBJ

N.4.1 Assembling Graphics Programs

To assemble a graphics program using the display processor mnemonics or the Display Handler macro calls, the file VTMAC.MAC must be assembled with the program, and must precede the program in the assembler command string.

Example:

Assume PICTUR.MAC is a user graphics program to be assembled. An assembler command string would look like this:

```
.R MACRO
*PICTUR=VTMAC,PICTUR
```

N.4.2 Linking Graphics Programs

Once assembled with VTMAC, the graphics program must be linked with the Display File Handler, which is supplied as a single concatenated object module, VTHDLR.OBJ. The Handler may optionally be built as a library, following the directions in N.8.5. The advantage of using the library when linking is that the Linker will select from the library only those modules actually used. Linking with VTHDLR.OBJ results in all modules being included in the link.

To link a user program called PICTUR.OBJ using the concatenated object module supplied with RT-11:

```
.R LINK
*PICTUR=PICTUR,VTHDLR
*
```

To link a program called PICTUR.OBJ using the VTLIB library built by following the directions in N.8.5, be sure to use the Version 2 Linker:

```
.R LINK
*PICTUR=PICTUR,VTLIB
*
```

VTLIB (Handler Modules):

Module	CSECT	Contains
VTCAL1	$GT1	.CLEAR .START .STOP .INSRT .REMOV
VTCAL2	$GT2	.BLANK .RESTR
VTCAL3	$GT3	.LPEN .NAME .STAT .SYNC .NOSYN .TRACK
VTCAL4	$GT4	.LNKRT .UNLNK .SCROL
VTBASE	$GTB	Memory resident base module containing interrupt handlers and internal display file

To link a display program using overlays, the modules must be specified individually. The user must acquire the sources and follow the assembly directions in Section N.8. The modules VTCAL1 and VTCAL2 must be simultaneously resident. For example:

```
.R LINK
*PICTUR=PICTUR,VTBASE/C
*VTCAL1,VTCAL2,VTCAL3/O:1/C
*VTCAL4/O:1
```

To link a display program to run as a foreground task:

```
.R LINK
*PICTUR=PICTUR,VTLIB/R
```

N.5 DISPLAY FILE STRUCTURE

The Display File Handler supports a variety of display file structures, takes over the job of display processor management for the programmer, and may be used for both assembly language graphics programming and for systems program development. For example, the

Handler supports the tagged subpicture file structure used by BASIC/GT, as well as simple file structures. These are discussed in this section.

N.5.1 Subroutine Calls

A subroutine call instruction, with the mnemonic DJSR, is implemented using the display stop (DSTOP) instruction with an interrupt. The display stop interrupt routine in the Display File Handler simulates the DJSR instruction, and this allows great flexibility in choosing the characteristics of the DJSR instruction.

The DJSR instruction stops the display processor and requests an interrupt. The DJSR instruction may be followed by two or more words, and in this implementation the exact number may be varied by the programmer at any time. The basic subroutine call has this form:

```
DJSR
Return Address
Subroutine Address
```

In practice, simple calls to subroutines could look like:

```
DJSR
.WORD     .+4
.WORD     SUB
```

where SUB is the address of the subroutine. Control will return to the display instruction following the last word of the subroutine call. This structure permits a call to the subroutine to be easily by-passed without stopping the display processor, by replacing the DJSR with a display jump (DJMP) instruction:

```
DJMP
.WORD     .+4
.WORD     SUB
```

A more complex display file structure is possible if the return address is generalized:

```
.DJSR
.WORD     NEXT
.WORD     SUB
```

where NEXT is the generalized return address. This is equivalent to the sequence:

```
DJSR
.WORD     .+4
.WORD     SUB
DJMP
.WORD     NEXT
```

It is also possible to store non-graphic data such as tags and pointers in the subroutine call sequence, such as is done in the tagged subpicture display file structure of BASIC/GT. This technique looks like:

```
             DJSR
             .WORD      NEXT
             .WORD      SUB
             DATA
     NEXT:      .
                .
                .
```

For simple applications where the flexibility of the DJSR instruction described above is not needed and the resultant overhead not desired, the Display File Handler (VTBASE.MAC and VTCALL.MAC) can be conditionally re-assembled to produce a simple DJSR call. If NOTAG is defined during the assembly, the Handler will be configured to support this simple DJSR call:

```
     DJSR
     .WORD      SUB
```

where SUB is the address of the subroutine. Defining NOTAG will eliminate the subpicture tag capability, and with it the tracking object, which uses the tag feature to identify itself to the light pen interrupt handler.

Whatever the DJSR format used, all subroutines and the user main file must be terminated with a subroutine return instruction. This is implemented as a display stop instruction (given the mnemonic DRET) with an argument of zero. A subroutine then has the form:

```
     SUB: Display Code
          DRET
          .WORD 0
```

N.5.2 Main File/Subroutine Structure

A common method of structuring display files is to have a main file which calls a series of display subroutines. Each subroutine will produce a picture element and may be called many times to build up a picture, producing economy of code. If the following macros are defined:

```
     .MACRO     CALL <ARG>
     DJSR
     .WORD      .+4
     .WORD      ARG
     .ENDM
     .MACRO     RETURN
     DRET
     .WORD      0
     .ENDM
```

then a main file/subroutine file structure would look like:

```
     ;MAIN DISPLAY FILE
     ;
     MAIN:      Display Code
                CALL SUB1          ;CALL SUBROUTINE 1
                Display Code
                CALL SUB2          ;CALL SUBROUTINE 2
                                   ;ETC
```

```
                .
                .
                RETURN
;
;DISPLAY        SUBROUTINES
;
SUB1:           Display Code    ;SUBROUTINE 1
                RETURN
;
SUB2:           Display Code    ;SUBROUTINE 2
                RETURN
                .               ;ETC.
                .
                .
```

N.5.3 BASIC/GT Subroutine Structure

An example of another approach to display file structure is the tagged subpicture structure used by BASIC/GT. The display file is divided into distinguishable elements called subpictures, each of which has its own unique tag.

The subpicture is constructed as a subroutine call followed by the subroutine. It is essentially a merger of the main file/subroutine structure into an in-line sequence of calls and subroutines. As such, it facilitates the construction of display files in real time, one of the important advantages of BASIC/GT.

The following is an example of the subpicture structure. Each subpicture has a call to a subroutine with the return address set to be the address of the next subpicture. The subroutine called may either immediately follow the call, or may be a subroutine defined as part of a subpicture created earlier in the display file. This permits a subroutine to be used by several subpictures without duplication of code. Each subpicture has a tag to identify it and it is this tag which is returned by the light pen interrupt routine. To facilitate finding subpictures in the display file, they are made into a linked list by inserting a forward pointer to the next tag.

```
SUB1:   DJSR                    ;START OF SUBPICTURE 1
        .WORD   SUB2            ;NEXT SUBPICTURE
        .WORD   SUB1+12         ;JUMP TO THIS SUBPICTURE
        .WORD   1               ;TAG = 1
        .WORD   SUB2+6          ;POINTER TO NEXT TAG

; BODY OF SUBPICTURE 1

        DRET                    ;RETURN FROM
        0                       ;SUBPICTURE 1

SUB2:   DJSR                    ;START SUBPICTURE 2
        .WORD   SUB3            ;NEXT SUBPICTURE
        .WORD   SUB2+12         ;JUMP TO THIS SUBPICTURE
        .WORD   2               ;TAG = 2
        .WORD   SUB3+6          ;PTR TO NEXT TAG

; BODY OF SUBPICTURE 2
        DRET                    ;RETURN FROM
        .WORD   0               ;SUBPICTURE 2
```

```
SUB3:   DJSR                    ;START SUBPICTURE 3
        .WORD   SUB4            ;NEXT SUBPICTURE
        .WORD   SUB1+12         ;COPY SUBPICTURE 1
                                ;FOR THIS SUBPICTURE
        .WORD   3               ;BUT TAG IT 3.
        .WORD   SUB4+6          ;PTR TO NEXT TAG

SUB4:   DJSR                    ;START SUBPICTURE 4
          .                     ;ETC.
          .
          .
```

N.6 SUMMARY OF GRAPHICS MACRO CALLS

Mnemonic	Function	MACRO Call (See Note 1)	Assembly Language Expansion (see Note 2)
.BLANK	Temporarily blanks a user display file.	.BLANK .faddr	.GLOBL $VBLNK .IF NB, .faddr MOV .faddr, %↑100 .ENDC JSR %↑07, $VBLNK
.CLEAR	Initializes handler.	.CLEAR	.GLOBL $VINIT JSR %↑07, $VINIT
.INSRT	Inserts a call to user display file in handler's master display file.	.INSRT .faddr	.GLOBL $VNSRT .IF NB, .faddr MOV .faddr, %↑00 .ENDC JSR %↑07, $VNSRT
.LNKRT	Sets up vectors and links display file handler to RT-11 scroller.	.LNKRT	.GLOBL $VRTLK JSR %↑07, $VRTLK
.LPEN	Sets up light pen status buffer.	.LPEN .baddr	.GLOBL $VLPEN .IF NB, .baddr MOV .baddr, %↑00 .ENDC JSR %↑07, $VLPEN
.NAME	Sets up buffer to receive name register stack contents.	.NAME .baddr	.GLOBL $NAME .IF NB, .baddr MOV .BEDDR, %↑00 .ENDC JSR %↑07, $NAME
.NOSYN	Disable power line sync.	.NOSYN	.GLOBL $NOSYN JSR %↑07, $NOSYN
.REMOV	Removes the call to a user display file.	.REMOV .faddr	.GLOBL $VRMOV .IF NB, .faddr MOV .faddr, %↑00 .ENDC JSR %↑07, $VRMOV

.RESTR	Unblanks the user display file.	.RESTR .faddr	.GLOBL $VRSTR IF NB, .faddr MOV .faddr, %↑00 .ENDC JSR %↑07, $VRSTR
.SCROL	Adjusts monitor scroller parameters.	.SCROL .baddr	.GLOBL $VSCRL .IF NB, .baddr MOV .baddr, %↑00 .ENDC JSR %↑07, $VSCRL
.START	Starts the display.	.START	.GLOBL $VSTRT JSR %↑07, $VSTRT
.STAT	Sets up status buffer.	.STAT .baddr	.GLOBL $VSTPM .IF NB, .baddr MOV .baddr, %↑00 .ENDC JSR %↑07, $VSTPM
.STOP	Stops the display.	.STOP	.GLOBL $VSTOP JSR %↑07, $VSTOP
.SYNC	Enables power line sync.	.SYNC	.GLOBL $SYNC JSR %↑07, $SYNC

.TRACK	Enables the track object.	.TRACK .baddr, .croutine	.GLOBL $VTRAK
			.IF NB, .baddr
			MOV .baddr, %↑00
			.ENDC
			.IF NB, .croutine
			MOV .croutine,-
			(%↑06)
			.IFF
			CLR-(%↑06)
			.ENDC
			.NARG T
			.IF EQ, T
			CLR %↑00
			.ENDC
			JSR %↑07, $VTRAK

.UNLNK	Unlinks display handler from RT-11 if linked, otherwise leaves display stopped.	.UNLNK	.GLOBL $VUNLK JSR %↑07, $VUNLK

NOTE 1

.baddr	Address of data buffer.
.faddr	Address of start of user display file.
.croutine	Address of .TRACK completion routine.

NOTE 2

The lines preceded by a dot will not be assembled. The code they enclose may or may not be assembled depending on the conditionals.

N.7 DISPLAY PROCESSOR MNEMONICS

Mnemonic	=	Value	Function
CHAR	=	100000	Character Mode
SHORTV	=	104000	Short Vector Mode
LONGV	=	110000	Long Vector Mode
POINT	=	114000	Point Mode
GRAPHX	=	120000	Graphplot X Mode
GRAPHY	=	124000	Graphplot Y Mode
RELATV	=	130000	Relative Point Mode
INT0	=	2000	Intensity 0 (Dimmest)
INT1	=	2200	Intensity 1
INT2	=	2400	Intensity 2
INT3	=	2600	Intensity 3
INT4	=	3000	Intensity 4
INT5	=	3200	Intensity 5
INT6	=	3400	Intensity 6
INT7	=	3600	Intensity 7 (Brightest)
LPOFF	=	100	Light Pen Off
LPON	=	140	Light Pen On
BLKOFF	=	20	Blink Off
BLKON	=	30	Blink On
LINE0	=	4	Solid Line
LINE1	=	5	Long Dash
LINE2	=	6	Short Dash
LINE3	=	7	Dot Dash
DJMP	=	160000	Display Jump
DNOP	=	164000	Display No Operation
STATSA	=	170000	Load Status A Instruction
LPLITE	=	200	Light Pen Hit On
LPDARK	=	300	Light Pen Hit Off
ITAL0	=	40	Italics Off
ITAL1	=	60	Italics On
SYNC	=	4	Halt and Resume in Sync
STATSB	=	174000	Load Status B Instruction

Display File Handler

Mnemonic		Value	Function
INCR	=	100	Graphplot Increment
(Vector/Point Mode)			
INTX	=	40000	Intensity Vector or Point
MAXX	=	1777	Maximum X Component
MAXY	=	1377	Maximum Y Component
MINUSX	=	20000	Negative X Component
MINUSY	=	20000	Negative Y Component
(Short Vector Mode)			
SHIFTX	=	200	
MAXSX	=	17600	Maximum X Component
MAXSY	=	77	Maximum Y Component
MISVX	=	20000	Negative X Component
MISVY	=	100	Negative Y Component

N.8 ASSEMBLY INSTRUCTIONS

N.8.1 General Instructions

All programs can be assembled in 16K, using RT-11 MACRO. All assemblies and all links should be error free. The following conventions are assumed:

1. Default extensions are not explicitly typed. These are .MAC for source files, .OBJ for assembler output, and .SAV for Linker output.
2. The default device (DK) is used for all files in the example command strings.
3. Listings and link maps are not generated in the example command strings.

N.8.2 VTBASE

To assemble VTBASE with RT-11 link-up capability:

```
.R MACRO
*VTBASE=VTBASE
```

N.8.3 VTCAL1 - VTCAL4

To assemble the modules VTCAL1 through VTCAL4:

```
.R MACRO
*VTCAL1=VTCAL1
*VTCAL2=VTCAL2
*VTCAL3=VTCAL3
*VTCAL4=VTCAL4
```

N.8.4 VTHDLR

To create the concatenated handler module:

```
.R PIP
*VTHDLR.OBJ=VTCAL1.OBJ,VTCAL2.OBJ,VTCAL3.OBJ,VTCAL4.OBJ,VTBASE.OBJ/B
```

N.8.5 Building VTLIB.OBJ

To build the VTLIB library:

```
.R LIBR
*VTLIB=VTHDLR.OBJ
```

N.9 VTMAC

```
        .NLIST
; VTMAC
; LIBRARY OF MACRO CALLS AND MNEMONIC DEFINITIONS
; FOR THE VT11 DEVICE SUPPORT PACKAGE

; DEC-11-OVTMA-B-LA

; COPYRIGHT (C) 1974
; DIGITAL EQUIPMENT CORPORATION
; MAYNARD, MASSACHUSETTS 01754

; MAY 1974

; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
; DEC ASSUMES NO RESPONSIBILITY FOR ANY ERRORS THAT
; MAY APPEAR IN THIS DOCUMENT.

; DEC ASSUMES NO RESPONSIBILITY FOR THE USE
; OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
; WHICH IS NOT SUPPLIED BY DEC.

; THIS SOFTWARE IS FURNISHED TO PURCHASER UNDER A
; LICENSE FOR USE ON A SINGLE COMPUTER SYSTEM AND
; CAN BE COPIED (WITH INCLUSION OF DEC'S COPYRIGHT
; NOTICE) ONLY FOR USE IN SUCH A SYSTEM, EXCEPT AS MAY
; OTHERWISE BE PROVIDED IN WRITING BY DEC.
```

```
; VTMAC IS A LIBRARY OF MACRO CALLS WHICH PROVIDE SUPPORT
; OF THE VT11 DISPLAY PROCESSOR. THE MACROS PRODUCE CALLS
; TO THE VT11 DEVICE SUPPORT PACKAGE, USING GLOBAL REFER-
; ENCES.

; MACRO TO GENERATE A MACRO WITH ZERO ARGUMENTS.

        .MACRO MAC0 NAME,CALL
        .MACRO NAME
        .GLOBL  CALL
        JSR     %^O7,CALL
        .ENDM
        .ENDM

; MACRO TO GENERATE A MACRO WITH ONE ARGUMENT

        .MACRO MAC1 NAME,CALL
        .MACRO NAME ARG
        .IF NB,ARG
        MOV     ARG,%^O0
        .ENDC
        .GLOBL  CALL
        JSR     %^O7,CALL
        .ENDM
        .ENDM

; MACRO TO GENERATE A MACRO WITH TWO OPTIONAL ARGUMENTS

        .MACRO MAC2 NAME,CALL
        .MACRO NAME ARG1,ARG2
        .GLOBL  CALL
        .IF NB,ARG1
        MOV     ARG1,%^O0
        .ENDC
        .IF NB,ARG2
        MOV     ARG2,-(%^O6)
        .IFF
        CLR     -(%^O6)
        .NARG   T
        .IF EQ,T
        CLR     %^O0
        .ENDC
        .ENDC
        JSR     %^O7,CALL
        .ENDM
        .ENDM
```

Display File Handler

```
; MACRO LIBRARY FOR VT11:

        MAC0    <.CLEAR>,<$VINIT>
        MAC0    <.STOP>,<$VSTOP>
        MAC0    <.START>,<$VSTRT>
        MAC0    <.SYNC>,<$SYNC>
        MAC0    <.NOSYN>,<$NOSYN>
        MAC0    <.UNLNK>,<$VUNLK>
        MAC1    <.INSRT>,<$VNSRT>
        MAC1    <.REMOV>,<$VRMOV>
        MAC1    <.BLANK>,<$VBLNK>
        MAC1    <.RESTR>,<$VRSTR>
        MAC1    <.STAT>,<$VSTPM>
        MAC1    <.LPEN>,<$VLPEN>
        MAC1    <.SCROL>,<$VSCRL>
        MAC1    <.NAME>,<$NAME>
        MAC2    <.TRACK>,<$VTRAK>
        MAC0    <.LNKRT>,<$VRTLK>
; MNEMONIC DEFINITIONS FOR THE VT11 DISPLAY PROCESSOR
;
DJMP=160000     ;DISPLAY JUMP
DNOP=164000     ;DISPLAY NOP
DJSR=173400     ;DISPLAY SUBROUTINE CALL
DRET=173400     ;DISPLAY SUBROUTINE RETURN
DNAME=173520    ;SET NAME REGISTER
DSTAT=173420    ;RETURN STATUS DATA
DHALT=173500    ;STOP DISPLAY AND RETURN STATUS DATA
;
CHAR=100000     ;CHARACTER MODE
SHORTV=104000   ;SHORT VECTOR MODE
LONGV=110000    ;LONG VECTOR MODE
POINT=114000    ;POINT MODE
GRAPHX=120000   ;GRAPH X MODE
GRAPHY=124000   ;GRAPH Y MODE
RELATV=130000   ;RELATIVE VECTOR MODE
;
INT0=2000       ;INTENSITY 0
INT1=2200
INT2=2400
INT3=2600
INT4=3000
INT5=3200
INT6=3400
INT7=3600
;
LPOFF=100       ;LIGHT PEN OFF
LPON=140        ;LIGHT PEN ON
BLKOFF=20       ;BLINK OFF
BLKON=30        ;BLINK ON
LINE0=4         ;SOLID LINE
LINE1=5         ;LONG DASH
LINE2=6         ;SHORT DASH
LINE3=7         ;DOT DASH
;
```

```
STATSA=170000       ;LOAD STATUS REG A
LPLITE=200          ;INTENSIFY ON LPEN HIT
LPDARK=300          ;DON'T INTENSIFY
ITAL0=40            ;ITALICS OFF
ITAL1=60            ;ITALICS ON
SYNC=4              ;POWER LINE SYNC
;
STATSB=174000       ;LOAD STATUS REG B
INCR=100            ;GRAPH PLOT INCREMENT
INTX=40000          ;INTENSIFY VECTOR OR POINT
MAXX=1777           ;MAXIMUM X INCR. - LONGV
MAXY=1377           ;MAXIMUM Y INCR. - LONGV
MINUSX=20000        ;NEGATIVE X INCREMENT
MINUSY=20000        ;NEGATIVE Y INCREMENT
MAXSX=17600         ;MAXIMUM X INCR. - SHORTV
MAXSY=77            ;MAXIMUM Y INCR. - SHORTV
MISVX=20000         ;NEGATIVE X INCR. - SHORTV
MISVY=100           ;NEGATIVE Y INCR. - SHORTV
        .LIST
```

N.10 EXAMPLES USING GTON

```
EXAMPLE #1       RT-11 MACRO VM02-06    9-AUG-74 PAGE 4

                                .TITLE  EXAMPLE #1
                        ;
                        ; THIS EXAMPLE USES THE .LPEN STATUS BUFFER AND THE
                        ; NAME REGISTER TO MODIFY A DISPLAY FILE WITH THE LIGHT PEN.
                        ;
        000000                  R0=%0
        000001                  R1=%1
        000007                  PC=%7
        000044                  JSW=44                          ;JOB STATUS WORD

                                .MCALL  .TTINR,.EXIT,.PRINT
  00000                 START:  .LNKRT                          ;LINK TO MONITOR
  00004 100004                  BPL     1$                      ;LINK UP ERROR?
  00006                         .PRINT  #EMSG                   ;YES, PRINT MESSAGE
  00014                         .EXIT                           ;AND EXIT.
  00016                 1$:     .SCROL  #SCBUF                  ;ADJUST SCROLL
  00026                         .PRINT  #MSG
  00034                         .INSRT  #DFILE                  ;INSERT DISPLAY FILE
  00044                         .LPEN   #LBUF                   ;SET UP LPEN BUFFER
  00054 052737                  BIS     #100,@#JSW              ;SET JSW FOR TTINR
        000100
        000044
  00062 005767          LTST:   TST     LBUF                    ;LIGHT PEN HIT?
        000070
  00066 001003                  BNE     1$                      ;YES
  00070                         .TTINR                          ;NO, ANY TT INPUT?
  00072 103023                  BCC     EXIT                    ;YES, EXIT
  00074 000772                  BR      LTST                    ;NO, LOOP AGAIN
  00076 016777          1$:     MOV     I2,@IPTR                ;RESTORE PREVIOUS CODE
        000074
        000102
  00104 016701                  MOV     LBUF+2,R1               ;GET NAME VALUE
```

```
28           000050
29 00110 005301              DEC     R1                  ;SUBTRACT ONE
30 00112 006301              ASL     R1                  ;MULTIPLY BY TWO
31 00114 060701              ADD     PC,R1               ;USE TO INDEX
32 00116 062701              ADD     #DTABL-.,R1         ;OFF TABLE DTABL.
32           000062
33 00122 011167              MOV     (R1),IPTR           ;MOVE ADDR INTO IPTR
33           000060
34 00126 016777              MOV     I1,@IPTR            ;MODIFY THAT CODE
34           000042
34           000052
35 00134 005067              CLR     LBUF                ;CLEAR BUFFER FLAG TO
35           000016
36                                                       ;ENABLE ANOTHER LP HIT.
37 00140 000750              BR      LTST                ;LOOP AGAIN
38 00142 022700 EXIT:        CMP     #12,R0              ;LINE FEED?
38           000012
39 00146 001345              BNE     LTST                ;NO, GET ANOTHER
40 00150                     .UNLNK                      ;UNLINK FROM MONITOR
41 00154                     .EXIT
42 00156        LBUF:        .BLKW   7                   ;LPEN STATUS BUFFER
43 00174 103370 I1:          .WORD   CHAR!INT5!BLKON!LPON
44 00176 103160 I2:          .WORD   CHAR!INT4!BLKOFF!LPON
45 00200 000252'DTABL:       .WORD   D1,D2,D3            ;TABLE OF DISPLAY FILE
45 00202 000272'
```

```
EXAMPLE #1        RT-11 MACRO VM02-06    9-AUG-74 PAGE 4+
45 00204 000312'
46                                                      ;LOCATIONS TO BE MODIFIED
47 00206 000252'IPTR:        .WORD   D1                 ;PREVIOUS LOCATION MODIFIED
48 00210 000002 SCBUF:       .WORD   2                  ;SCROLL LINE COUNT
49 00212 001000              .WORD   1000               ;SCROLL TOP Y POS.
50 00214    041 EMSG:        .ASCIZ  /!ERROR!/          ;ERROR MESSAGE
50 00215    105
50 00216    122
50 00217    122
50 00220    117
50 00221    122
50 00222    041
50 00223    000
51              .EVEN
52 00224    105 MSG:         .ASCIZ  /EXAMPLE #1/       ;I.D. MESSAGE
52 00225    130
52 00226    101
52 00227    115
52 00230    120
52 00231    114
52 00232    105
52 00233    040
52 00234    043
52 00235    061
52 00236    000
53                           .EVEN
54              ;
55              ; DISPLAY FILE FOR EXAMPLE #1
56              ;
57 00240 114000 DFILE:       POINT
58 00242 000100              100
59 00244 000500              500
60 00246 173520              DNAME
61 00250 000001              1
62 00252 103160 D1:          CHAR!BLKOFF!INT4!LPON
```

```
63 00254      117          .ASCII   /ONE./
63 00255      116
63 00256      105
63 00257      056
64 00260 114000            POINT
65 00262 000100            100
66 00264 000300            300
67 00266 173520            DNAME
68 00270 000002            2
69 00272 103160  D2:       CHAR!BLKOFF!INT4!LPON
70 00274      124          .ASCII   /TWO./
70 00275      127
70 00276      117
70 00277      056
71 00300 114000            POINT
72 00302 000100            100
73 00304 000100            100
74 00306 173520            DNAME
75 00310 000003            3
76 00312 103160  D3:       CHAR!BLKOFF!INT4!LPON
77 00314      124          .ASCII   /THREE./
77 00315      110

EXAMPLE #1       RT-11 MACRO VM02-06    9-AUG-74 PAGE 4+
77 00316      122
77 00317      105
77 00320      105
77 00321      056
78 00322 173400            DRET
79 00324 000000            0
80        000000'          .END     START

EXAMPLE #1       RT-11 MACRO VM02-06    9-AUG-74 PAGE 4+
SYMBOL TABLE
BLKOFF= 000020             BLKON  = 000030             CHAR   = 100000
DFILE   000240R            DHALT  = 173500             DJMP   = 160000
DJSR   = 173400            DNAME  = 173520             DNOP   = 164000
DRET   = 173400            DSTAT  = 173420             DTABL    000200R
D1      000252R            D2       000272R            D3       000312R
EMSG    000214R            EXIT     000142R            GRAPHX= 120000
GRAPHY= 124000             INCR   = 000100             INTX   = 040000
INT0   = 002000            INT1   = 002200             INT2   = 002400
INT3   = 002600            INT4   = 003000             INT5   = 003200
INT6   = 003400            INT7   = 003600             IPTR     000206R
ITAL0  = 000040            ITAL1  = 000060             I1       000174R
I2      000176R            JSW    = 000044             LBUF     000156R
LINE0  = 000004            LINE1  = 000005             LINE2  = 000006
LINE3  = 000007            LONGV  = 110000             LPDARK= 000300
LPLITE= 000200             LPOFF  = 000100             LPON   = 000140
LTST    000062R            MAXSX  = 017600             MAXSY  = 000077
MAXX   = 001777            MAXY   = 001377             MINUSX= 020000
MINUSY= 020000             MISVX  = 020000             MISVY  = 000100
MSG     000224R            PC     =%000007             POINT  = 114000
RELATV= 130000             R0     =%000000             R1     =%000001
SCBUF   000210R            SHORTV= 104000              START    000000R
STATSA= 170000             STATSB= 174000              SYNC   = 000004
$VLPEN= ****** G           $VNSRT= ****** G            $VRTLK= ****** G
$VSCRL= ****** G           $VUNLK= ****** G
. ABS.  000000     000
        000326     001
ERRORS DETECTED: 0
FREE CORE: 15117. WORDS

MANEX1,LP:=VTMAC,MANEX1
```

```
EXAMPLE #2      RT-11 MACRO VM02-06   9-AUG-74 PAGE 4

                                .TITLE  EXAMPLE #2
                        ;
                        ; THIS EXAMPLE USES THE TRACKING OBJECT AND THE TRACK
                        ; COMPLETION ROUTINE TO CAUSE A VECTOR TO FOLLOW
                        ; THE LIGHT PEN FROM A SET POINT AT (500,500).
                        ;
            000000              R0=%0
            000001              R1=%1
            000006              SP=%6
            000007              PC=%7
                                .MCALL  .EXIT,.TTYIN,.PRINT
      00000             START:  .LNKRT                  ;LINK TO MONITOR
      00004 100004              BPL     1$              ;LINK UP ERROR?
      00006                     .PRINT  #EMSG           ;YES, INFORM USER
      00014                     .EXIT                   ;AND EXIT
      00016             1$:     .INSRT  #DFILE          ;INSERT DISPLAY FILE
      00026                     .TRACK  #TBUF,#TCOM     ;DISPLAY TRACK OBJECT
      00042 004767              JSR     PC,WAIT         ;WAIT FOR <CR>
            000006
      00046                     .UNLNK                  ;UNLINK FROM MONITOR
      00052                     .EXIT
      00054             WAIT:   .TTYIN                  ;GET CHAR. FROM TTY
      00060 022700              CMP     #12,R0          ;LINE FEED?
            000012
      00064 001373              BNE     WAIT            ;NO, GET ANOTHER
      00066 000207              RTS     PC
      00070 000500      TBUF:   .WORD   500,500         ;TRACK BUFFER INITED TO
      00072 000500
                                                        ;START TRACK AT (500,500)
                        ;
                        ; TRACK COMPLETION ROUTINE ENTERED AT INTERRUPT LEVEL
                        ; FROM DISPLAY FILE HANDLER WITH DISPLAY STOPPED.
                        ; USED TO UPDATE DISPLAY FILE WITH DATA FROM TBUF.
                        ;
      00074 010146      TCOM:   MOV     R1,-(SP)        ;SAVE R1
      00076 016701              MOV     TBUF,R1         ;NEW X
            177766
      00102 166701              SUB     OX,R1           ;NEW X - OLD X
            000052
      00106 100003              BPL     1$              ;POSITIVE DIFFERENCE?
      00110 005401              NEG     R1              ;NO, SO MAKE POSITIVE
      00112 052701              BIS     #MINUSX,R1      ;BUT SET MINUS BIT
            020000
      00116 052701      1$:     BIS     #INTX,R1        ;ALSO SET INTENSIFY BIT
            040000
      00122 010167              MOV     R1,DX           ;THEN STORE IN DFILE.
            000040
      00126 016701              MOV     TBUF+2,R1       ;NEW Y
            177740
      00132 166701              SUB     OY,R1           ;NEW Y - OLD Y
            000024
      00136 100003              BPL     2$              ;POSITIVE DIFFERENCE?
      00140 005401              NEG     R1              ;NO,SO MAKE POSITIVE
      00142 052701              BIS     #MINUSX,R1      ;AND SET MINUS BIT
            020000
      00146 010167      2$:     MOV     R1,DY           ;THEN STORE IN DFILE
            000016
```

```
EXAMPLE #2      RT-11 MACRO VM02-06    9-AUG-74 PAGE 4+

47 00152 012601            MOV     (SP)+,R1          ;RESTORE R1
48 00154 000207            RTS     PC                ;EXIT FROM COMPLETION ROUTINE
49                 ;
50                 ; DISPLAY FILE FOR EXAMPLE #2
51                 ;
52 00156 114000 DFILE:     POINT                     ;SET POINT AT
53 00160 000500 DX:        500
54 00162 000500 DY:        500                       ;(500,500)
55 00164 113000            LONGV!INT4                ;DRAW A VECTOR
56 00166 000000 DX:        .WORD   0                 ;INITIALLY NOWHERE
57 00170 000000 DY:        .WORD   0
58 00172 173400            DRET                      ;DISPLAY FILE END
59 00174 000000            0
60 00176    123 EMSG:      .ASCIZ  /SORRY, THERE SEEMS TO BE A PROBLEM/
60 00177    117
60 00200    122
60 00201    122
60 00202    131
60 00203    054
60 00204    040
60 00205    124
60 00206    110
60 00207    105
60 00210    122
60 00211    105
60 00212    040
60 00213    123
60 00214    105
60 00215    105
60 00216    115
60 00217    123
60 00220    040
60 00221    124
60 00222    117
60 00223    040
60 00224    102
60 00225    105
60 00226    040
60 00227    101
60 00230    040
60 00231    120
60 00232    122
60 00233    117
60 00234    102
60 00235    114
60 00236    105
60 00237    115
60 00240    000
61                 .EVEN
62        000000'          .END    START
```

```
EXAMPLE #2      RT-11 MACRO VM02-06   9-AUG-74 PAGE 4+
SYMBOL TABLE

BLKOFF= 000020          BLKON = 000030          CHAR  = 100000
DFILE   000156R         DHALT = 173500          DJMP  = 160000
DJSR  = 173400          DNAME = 173520          DNOP  = 164000
DRET  = 173400          DSTAT = 173420          DX      000166R
DY      000170R         EMSG    000176R         GRAPHX= 120000
GRAPHY= 124000          INCR  = 000100          INTX  = 040000
INT0  = 002000          INT1  = 002200          INT2  = 002400
INT3  = 002600          INT4  = 003000          INT5  = 003200
INT6  = 003400          INT7  = 003600          ITAL0 = 000040
ITAL1 = 000060          LINE0 = 000004          LINE1 = 000005
LINE2 = 000006          LINE3 = 000007          LONGV = 110000
LPDARK= 000300          LPLITE= 000200          LPOFF = 000100
LPON  = 000140          MAXSX = 017600          MAXSY = 000077
MAXX  = 001777          MAXY  = 001377          MINUSX= 020000
MINUSY= 020000          MISVX = 020000          MISVY = 000100
OX      000160R         OY      000162R         PC    =%000007
POINT = 114000          RELATV= 130000          R0    =%000000
R1    =%000001          SHORTV= 104000          SP    =%000006
START   000000R         STATSA= 170000          STATSB= 174000
SYNC  = 000004          TBUF    000070R         TCOM    000074R
WAIT    000054R         $VNSRT= ****** G        $VRTLK= ****** G
$VTRAK= ****** G        $VUNLK= ****** G
. ABS.  000000    000
        000242    001
ERRORS DETECTED: 0
FREE CORE: 15022. WORDS

MANEX2,LP:=VTMAC,MANEX2
```

APPENDIX O

SYSTEM SUBROUTINE LIBRARY

O.1 INTRODUCTION

The RT-11 FORTRAN System Subroutine Library (SYSLIB) is a collection of FORTRAN-callable routines which allow a FORTRAN user to utilize various features of RT-11 Foreground/Background (F/B) and Single-Job (SJ) Monitors. SYSLIB also provides various utility functions, a complete character string manipulation package, and 2-word integer support. SYSLIB is provided as a library of object modules to be combined with FORTRAN programs at link-time.

The user of SYSLIB should be familiar with Chapters 1, 2, and 9 of this manual. This appendix assumes that FORTRAN users are familiar with the PDP-11 FORTRAN Language Reference Manual and the RT-11/RSTS/E FORTRAN IV User's Guide.

The following are some of the functions provided by SYSLIB.

- Complete RT-11 I/O facilities, including synchronous, asynchronous, and completion-driven modes of operation. FORTRAN subroutines may be activated upon completion of an input/output operation.
- Timed scheduling of asynchronous subjobs (completion routines). (F/B only)
- Complete facilities for interjob communication between Foreground and Background jobs (F/B only).
- FORTRAN interrupt service routines.
- Complete timer support facilities, including timed suspension of execution (F/B only), conversion of different time formats, and time-of-day information. These timer facilities support either 50- or 60-cycle clocks.
- All auxiliary input/output functions provided by RT-11, including the capabilities of opening, closing, renaming, creating, and deleting files from any device.
- All monitor-level informational functions, such as job partition parameters, device statistics, and input/output channel statistics.
- Access to the RT-11 Command String Interpreter (CSI) for acceptance and parsing of standard RT-11 command strings.
- A character string manipulation package supporting variable-length character strings.
- INTEGER*4 support routines that allow 2-word integer computations.

SYSLIB allows the FORTRAN user to write almost all application programs completely in FORTRAN with no assembly language coding.

Assembly language programs may also utilize SYSLIB routines (see Section 0.1.3).

0.1.1 Conventions and Restrictions

In general, the SYSLIB routines were written for use with RT-11 V2B and FORTRAN IV V1B. The use of this SYSLIB package with prior versions of RT-11 or FORTRAN will lead to unpredictable results.

Programs using IPEEK, IPOKE, and/or ISPY to access FORTRAN, monitor, hardware, or other system specific addresses are not guaranteed to run under future releases or on different configurations. Suitable care should be taken with this type of coding to document precisely the use of these access functions and to check a referenced location's usage against the current documentation.

The following must be considered when coding a FORTRAN program that uses SYSLIB.

1. Various functions in the SYSLIB package return values that are of type integer, real, and double precision. If the user specifies an IMPLICIT statement that changes the defaults for external function typing, he must explicitly declare the type of those SYSLIB functions that return integer or real results. Double precision functions must always be declared to be type DOUBLE PRECISION (or REAL*8). Failure to observe this requirement will lead to unpredictable results.

2. All names of subprograms external to the routine being coded that are being passed to scheduling calls (such as ISCHED, ITIMER, IREADF, etc.) must be specified in an EXTERNAL statement in the FORTRAN program unit issuing the call.

3. Certain arguments (noted as such in the individual routine descriptions) to SYSLIB calls must be located in such a manner as to prohibit the RT-11 USR (User Service Routine) from swapping over them at execution time. This is accomplished by issuing a SET USR NO SWAP command before the job is entered, by using the /U switch at compile time, or by assuring that the routine argument is located in the memory image in such a way as to not be swapped over. The latter may be accomplished by changing the order of the object modules and libraries as specified to the linker. If the USR is swapping, it swaps over the first 2K of the program. The default case, for example, swaps the USR at 1000 octal to 11000 octal; arguments being passed to the USR must be kept out of this area. (See Section 0.1.3 for further information.)

4. Quoted-string literals are useful as arguments of calls to routines in the SYSLIB package, notably the character string routines. These literals are allowed in subroutine calls (i.e., those invoked by the CALL statement), but they are explicitly prohibited in function calls (i.e., those invoked by the appearance of the function name followed by an argument list in an expression). See Sections 0.1.2 and 0.2.4 for further information.

5. Certain restrictions apply to completion or interrupt routines; see Section 0.2.1 for these restrictions.

O.1.2 Calling SYSLIB Subprograms

SYSLIB subprograms are called in the same manner as user-written subroutines. SYSLIB includes both FUNCTION subprograms and SUBROUTINE subprograms. FUNCTION subprograms receive control by means of a function reference, represented in this appendix as:

```
i = function name ({arguments})
```

SUBROUTINE subprograms are invoked by means of a CALL statement, i.e.,

```
CALL subroutine name {(arguments)}
```

All routines in SYSLIB may be called as FUNCTION subprograms if the return value is desired, or as SUBROUTINE subprograms if no return value is desired. For example, the LOCK subroutine can be referenced as either:

```
CALL LOCK

  or

I = LOCK()
```

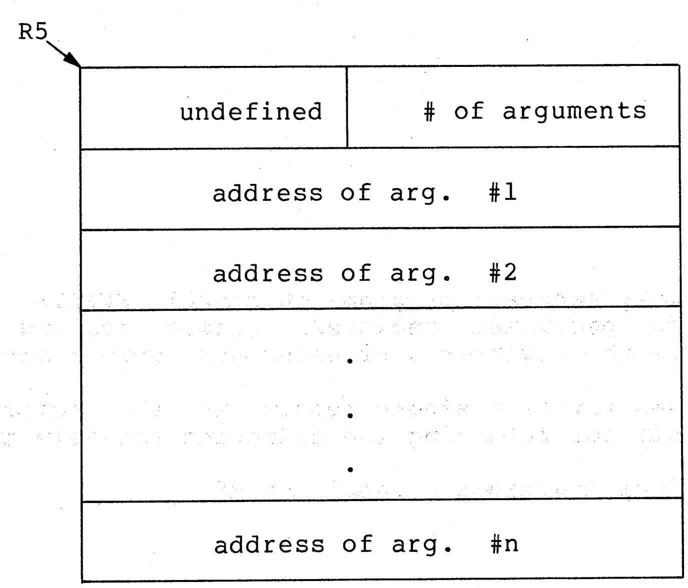
Note that routines that do not explicitly return function results will produce meaningless values if referenced as functions. In the following descriptions, the more common usage (function or subroutine) is shown.

O.1.3 Using SYSLIB with MACRO

The calling sequence is standard for all subroutines, including user-written FORTRAN subprograms and assembly language subprograms. SYSLIB routines may be used with MACRO programs by passing control to the SYSLIB routine with the following instruction:

```
     JSR     PC,routine
```

Register five points to an argument list having the following format:



Control is returned to the calling program by use of the instruction:

```
        RTS     PC
```

The following is an example of calling a SYSLIB function from an assembly language routine.

```
             .GLOBL JMUL       ;GLOBAL FOR JMUL

             .
             .
             .

             MOV #LIST,R5      ;POINT R5 TO ARG LIST
             JSR PC,JMUL       ;CALL JMUL
             CMP #-2,R0        ;CHECK FOR OVERFLOW
             BEQ OVRFL         ;BRANCH IF ERROR

             .
             .
             .

     LIST:   .WORD 3           ;ARG LIST,3 ARGS
             .WORD OPR1        ;ADDR OF 1ST ARG
             .WORD OPR2        ;ADDR OF 2ND ARG
             .WORD RESULT      ;ADDR OF 3RD ARG
     OPR1:   .WORD 100         ;LOW-ORDER VALUE OF 1ST ARG
             .WORD 0           ;HIGH-ORDER VALUE OF 1ST ARG
     OPR2:   .WORD 10          ;LOW-ORDER VALUE OF 2ND ARG
             .WORD 10          ;HIGH-ORDER VALUE OF 2ND ARG
     RESULT: .BLKW 2           ;2-WORD RESULT (LOW ORDER, HIGH ORDER)
             .END
```

The following routines can be used only with FORTRAN:

GETSTR
IASIGN
ICDFN
IFETCH
IGETC
ILUN
INTSET
IQSET
IRCVDF
IREADF
ISCHED
ISDATF
ISPFNF
ITIMER
IWRITF
PUTSTR
SECNDS

User-written assembly language programs that call SYSLIB subprograms must preserve any pertinent registers before calling the SYSLIB routine and restore the registers, if necessary, upon return.

Function subprograms return a single result in the registers. The register assignments for returning the different variable types are:

Integer, Logical functions - result in R0

Real functions - high-order result in R0, low-order result in R1

Double Precision functions - result in R0-R3, lowest order result in R3

Complex functions - high-order real result in R0, low-order real result in R1, high-order imaginary result in R2, low-order imaginary result in R3

User-written assembly language routines which interface to the FORTRAN Object Time System (OTS) must be aware of the location of the RT-11 USR (User Service Routine). If a user routine requests a USR function (e.g., IENTER, LOOKUP), or if the USR is invoked by the FORTRAN OTS, the USR will be swapped into memory if it is nonresident. The FORTRAN OTS is designed so that the USR can swap over it. User routines must be written to allow the USR to swap over them or must be located outside the region of memory into which the USR will swap.

User interrupt service routines and completion routines, because of their asynchronous nature, must be further restricted to be located where the USR will not swap. The USR (if in a swapping state) will always swap over the area of memory that starts at the program initial stack pointer address; the USR occupies 2K words. Interrupt and completion routines (and their data areas) must not be located in this area. The best way to accomplish this is to examine the link map, determine whether the USR will swap over an assembly language or FORTRAN asynchronous routine, and, if so, change the order of object modules and libraries as specified to the linker. Continue this process until a suitable arrangement is obtained.

To remove these restrictions, the user must make the USR resident either by specifying the /U switch to the FORTRAN compiler (when compiling a program to be run in the background of F/B or under S-J) or by issuing the SET USR NOSWAP command before executing the program.

O.1.4 Running a FORTRAN Program in the Foreground

The FRUN monitor command must be modified to include various SYSLIB functions. Section G.1 explains the formula used to allocate the needed space when running a FORTRAN program as a foreground job. This formula:

x = {1/2{378+(29*N)+(R-136)+A*512)}}

must be modified for SYSLIB functions as follows:

The IQSET function requires the formula to include additional space for queue elements (qleng) to be added to the queue:

x = {1/2{378+(29*N)+(R-136)+A*512)}}+{7*qleng}

The ICDFN function requires the formula to include additional space for the integer number of channels (num) to be allocated.

x = {1/2{378+(29*N)+(R-136)+A*512)}}+{6*num}

The INTSET function requires the formula to include additional space for the number of INTSET calls (INTSET) issued in the program.

x = {1/2 {378+(29*N)+(R-136)+A*512}}+{25*INTSET}

Any SYSLIB calls, including INTSET, that invoke completion routines must include 64(10) words plus the number of words needed to allocate the second record buffer (default is 68(10) words). The length of the record buffer is controlled by the /R switch to the FORTRAN compiler. If the /R switch is not used, the allocation in the formula must be 132(10) words.

x = {1/2{378+(29*N)+(R-136)+A*512}}+{64+R/2}

If the /N option does not allocate enough space in the foreground on the initial call to a completion routine, the following message appears:

```
?ERR 0, NON-FORTRAN ERROR CALL
```

This message also appears if there is not enough free memory for the background job or if a completion routine in the Single-Job monitor is activated during another completion routine. In the latter case, the job aborts. F/B should be used for multiple active completion routines.

O.1.5 Linking with SYSLIB

SYSLIB is provided on the distribution media as a file of concatenated object modules (SYSF4.OBJ). If this file is linked directly with the FORTRAN program, all SYSLIB modules will be included whether they are used or not. For example:

```
.R LINK
*PROG=PROG,SYSF4/F
```

A library can be created by using the librarian to transform SYSF4 into a library file (SYSLIB) as follows:

```
.R LIBR
*SYSLIB=SYSF4
```

When a library is used, only the modules called will be linked with the program. For example:

```
.R LINK
*PROG=PROG,SYSLIB/F
```

The following example links the object module EXAMPL.OBJ into a single save image file EXAMPL.SAV and produces a load map file on LP:. SYSLIB and the default FORTRAN library (FORLIB.OBJ) are searched for any routines that are not found in other object modules.

```
.R LINK
*EXAMPL,LP:=EXAMPL,SYSLIB/F
```

If the FORTRAN library is explicitly specified in the command string, SYSLIB must precede it, i.e.,

```
.R LINK
*TEST=TEST,SYSLIB,FPULIB
```

is an acceptable command string, but the following is not:

```
*TEST=TEST,FPULIB,SYSLIB
```

O.2 TYPES OF SYSLIB SERVICES

Ten types of services are available to the user through SYSLIB. These are:

1. File-Oriented Operations
2. Data Transfer Operations
3. Channel-Oriented Operations
4. Device and File Specifications
5. Timer Support Operations
6. RT-11 Service Operations
7. INTEGER*4 Support Functions
8. Character String Functions
9. Radix-50 Conversion Operations
10. Miscellaneous Services

Table O-1 alphabetically summarizes the SYSLIB subprograms in each of these categories. Those marked with an asterisk (*) are allowed only in a Foreground/Background environment.

Table O-1
Summary of SYSLIB Subprograms

Function Call	Section	Purpose
File-Oriented Operations		
CLOSEC	O.3.3	Closes the specified channel.
IDELET	O.3.19	Deletes a file from the specified device.
IENTER	O.3.22	Creates a new file for output.
IRENAM	O.3.37	Changes the name of the indicated file to a new name.
LOOKUP	O.3.66	Opens an existing file for input and/or output via the specified channel.
Data Transfer Functions		
*IRCVD *IRCVDC *IRCVDF *IRCVDW	O.3.35	Receives data. Allows a job to read messages or data sent by another job in an F/B environment. The four modes correspond to the IREAD, IREADC, IREADF, and IREADW modes.

(continued on next page)

Table O-1 (cont.)
Summary of SYSLIB Subprograms

Function Call	Section	Purpose
Data Transfer Functions (cont.)		
IREAD	0.3.36	Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. No special action is taken upon completion of I/O.
IREADC	0.3.36	Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the read, control transfers to the assembly language routine specified in the IREADC function call.
IREADF	0.3.36	Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the read, control transfers to the FORTRAN subroutine specified in the IREADF function call.
IREADW	0.3.36	Transfers data via the specified channel to a memory buffer and returns control to the program only after the transfer is complete.
*ISDAT *ISDATC *ISDATF *ISDATW	0.3.41	Allows the user to send messages or data to the other job in an F/B environment. The four modes correspond to the IWRITE, IWRITC, IWRITF, and IWRITW modes.
ITTINR	0.3.47	Inputs one character from the console keyboard.
ITTOUR	0.3.48	Transfers one character to the console terminal.
IWAIT	0.3.51	Waits for completion of all I/O on a specified channel. (Commonly used with the IREAD and IWRITE functions.)
IWRITC	0.3.52	Transfers data via the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the write, control transfers to the assembly language routine specified in the IWRITC function call.

(continued on next page)

Table O-1 (cont.)
Summary of SYSLIB Subprograms

Function Call	Section	Purpose
Data Transfer Functions (cont.)		
IWRITE	0.3.52	Transfers data via the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. No special action is taken upon completion of the I/O.
IWRITF	0.3.52	Transfers data via the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the write, control transfers to the FORTRAN subroutine specified in the IWRITF function call.
IWRITW	0.3.52	Transfers data via the specified channel to a device and returns control to the user program only after the transfer is complete.
*MWAIT	0.3.68	Waits for messages to be processed.
PRINT	0.3.69	Outputs an ASCII string to the console terminal.
Channel-Oriented Operations		
ICDFN	0.3.14	Defines additional I/O channels.
*ICHCPY	0.3.15	Allows access to files currently open in the other job's environment.
*ICSTAT	0.3.18	Returns the status of a specified channel.
IFREEC	0.3.24	Returns the specified RT-11 channel to the available pool of channels for the FORTRAN I/O system.
IGETC	0.3.25	Allocates an RT-11 channel and marks it in use to the FORTRAN I/O system.
ILUN	0.3.27	Returns the RT-11 channel number with which a FORTRAN logical unit is associated.
IREOPN	0.3.38	Restores the parameters stored via an ISAVES function and reopens the channel for I/O.
ISAVES	0.3.39	Stores five words of channel status information into a user-specified array.
PURGE	0.3.70	Deactivates a channel.

(continued on next page)

Table O-1 (cont.)
Summary of SYSLIB Subprograms

Function Call	Section	Purpose
Device and File Specifications		
IASIGN	O.3.13	Sets information in the FORTRAN logical unit table.
ICSI	O.3.17	Calls the RT-11 CSI in special mode to decode file specifications and switches.
Timer Support Operations		
CVTTIM	O.3.5	Converts a 2-word internal format time to hours, minutes, seconds, and ticks.
GTIM	O.3.9	Gets time of day.
*ICMKT	O.3.16	Cancels an unexpired ISCHED, ITIMER, or MRKT request.
*ISCHED	O.3.40	Schedules the specified FORTRAN subroutine to be entered at the specified time of day as an asynchronous completion routine.
*ISLEEP	O.3.42	Suspends main program execution of the running job for a specified amount of time; completion routines continue to run.
*ITIMER	O.3.45	Schedules the specified FORTRAN subroutine to be entered as an asynchronous completion routine when the time interval specified has elapsed.
*ITWAIT	O.3.49	Suspends the running job for a specified amount of time; completion routines continue to run.
*IUNTIL	O.3.50	Suspends the main program execution of the running job until a specified time-of-day; completion routines continue to run.
JTIME	O.3.63	Converts hours, minutes, seconds, and ticks into 2-word internal format time.
*MRKT	O.3.67	Marks time, i.e., schedules an assembly language routine to be activated as an asynchronous completion routine after a specified interval.

(continued on next page)

Table O-1 (cont.)
Summary of SYSLIB Subprograms

Function Call	Section	Purpose
Timer Support Operations (cont.)		
SECNDS	O.3.80	Returns the current system time in seconds past midnight minus the value of a specified argument.
TIMASC	O.3.84	Converts a specified 2-word internal format time into an 8-character ASCII string.
TIME	O.3.85	Returns the current system time-of-day as an 8-character ASCII string.
RT-11 Services		
CHAIN	O.3.2	Chains to another program (in the background job only).
*DEVICE	O.3.6	Specifies actions to be taken on normal or abnormal program termination, such as turning off interrupt enable on foreign devices, etc.
GTJB	O.3.10	Returns the parameters of this job.
IDSTAT	O.3.21	Returns the status of the specified device.
IFETCH	O.3.23	Loads a device handler into memory.
IQSET	O.3.33	Expands the size of the RT-11 monitor queue from the free space managed by the FORTRAN system.
ISPFN ISPFNC ISPFNF ISPFNW	O.3.43	Issues special function requests to various handlers, e.g., magtape. The four modes correspond to the IWRITE, IWRITC, IWRITF, and IWRITW modes.
*ITLOCK	O.3.46	Indicates whether the USR is currently in use by another job and performs a LOCK if the USR is available.
LOCK	O.3.65	Makes the RT-11 monitor User Service Routine (USR) permanently resident until an UNLOCK function is executed. A portion of the user's program is swapped out to make room for the USR if necessary.
RCHAIN	O.3.74	Allows a program to access variables passed across a chain.

(continued on next page)

Table O-1 (cont.)
Summary of SYSLIB Subprograms

Function Call	Section	Purpose
RT-11 Services (cont.)		
RCTRLO	O.3.75	Enables output to the terminal by cancelling the effect of a previously typed CTRL O, if any.
*RESUME	O.3.77	Causes the main program execution of a job to resume where it was suspended by a SUSPND function call.
*SUSPND	O.3.83	Suspends main program execution of the running job; completion routines continue to execute.
UNLOCK	O.3.88	Releases the USR if a LOCK was performed; the user program is swapped in if required.
INTEGER*4 Support Functions		
AJFLT	O.3.1	Converts a specified INTEGER*4 value to REAL*4 and returns the result as the function value.
DJFLT	O.3.7	Converts a specified INTEGER*4 value to REAL*8 and returns the result as the function value.
IAJFLT	O.3.12	Converts a specified INTEGER*4 value to REAL*4 and stores the result.
IDJFLT	O.3.20	Converts a specified INTEGER*4 value to REAL*8 and stores the result.
IJCVT	O.3.26	Converts a specified INTEGER*4 value to INTEGER*2.
JADD	O.3.53	Computes the sum of two INTEGER*4 values.
JAFIX	O.3.54	Converts a REAL*4 value to INTEGER*4.
JCMP	O.3.55	Compares two INTEGER*4 values and returns an INTEGER*2 value that reflects the signed comparison result.
JDFIX	O.3.56	Converts a REAL*8 value to INTEGER*4.
JDIV	O.3.57	Computes the quotient and remainder of two INTEGER*4 values.
JICVT	O.3.58	Converts an INTEGER*2 value to INTEGER*4.

(continued on next page)

Table O-1 (cont.)
Summary of SYSLIB Subprograms

Function Call	Section	Purpose
INTEGER*4 Support Functions (cont.)		
JJCVT	O.3.59	Converts 2-word internal time format to INTEGER*4 format, and vice versa.
JMOV	O.3.60	Assigns an INTEGER*4 value to a variable.
JMUL	O.3.61	Computes the product of two INTEGER*4 values.
JSUB	O.3.62	Computes the difference between two INTEGER*4 values.
Character String Functions		
CONCAT	O.3.4	Concatenates two variable-length strings.
GETSTR	O.3.8	Reads a character string from a specified FORTRAN logical unit.
INDEX	O.3.28	Returns the location in one string of the first occurrence of another string .
INSERT	O.3.29	Replaces a portion of one string with another string.
ISCOMP	O.3.78	Compares two character strings.
IVERIF	O.3.89	Indicates whether characters in one string appear in another.
LEN	O.3.64	Returns the number of characters in a specified string.
PUTSTR	O.3.71	Writes a variable-length character string on a specified FORTRAN logical unit.
REPEAT	O.3.76	Concatenates a specified string with itself to provide an indicated number of copies and stores the resultant string.
SCOMP	O.3.78	Compares two character strings.
SCOPY	O.3.79	Copies a character string from one array to another.
STRPAD	O.3.81	Pads a variable-length string on the right with blanks to create a new string of a specified length.

(concluded on next page)

Table O-1 (cont.)
Summary of SYSLIB Subprograms

Function Call	Section	Purpose
Character String Functions (cont.)		
SUBSTR	O.3.82	Copies a substring from a specified string.
TRANSL	O.3.86	Replaces one string with another after performing character modification.
TRIM	O.3.87	Removes trailing blanks from a character string.
VERIFY	O.3.89	Indicates whether characters in one string appear in another.
Radix-50 Conversion Operations		
IRAD50	O.3.34	Converts ASCII characters to Radix-50, returning the number of characters converted.
R50ASC	O.3.72	Converts Radix-50 characters to ASCII.
RAD50	O.3.73	Converts six ASCII characters, returning a REAL*4 result which is the 2-word Radix-50 value.
Miscellaneous Services		
IADDR	O.3.11	Obtains the memory address of a specified entity.
INTSET	O.3.30	Establishes a specified FORTRAN subroutine as an interrupt service routine at a specified priority.
IPEEK	O.3.31	Returns the value of a word located at a specified absolute memory address.
IPOKE	O.3.32	Stores an integer value in an absolute memory location.
ISPY	O.3.44	Returns the integer value of the word located at a specified offset from the beginning of the RT-11 resident monitor.

Routines requiring the USR (see Section 9.2.5) differ between the Single-Job and F/B Monitors. The following functions require the use of the USR:

CLOSEC
GETSTR (only if first I/O operation on logical unit)
ICDFN (Single-Job only)

```
ICSI
IDELET
IDSTAT
IENTER
IFETCH
IQSET
IRENAM
ITLOCK (only if USR is not in use by the other job)
LOCK   (only if USR is in a swapping state)
LOOKUP
PUTSTR (only if first I/O operation on logical unit)
```

Certain requests require a queue element taken from the same list as the I/O queue elements. These are:

```
IRCVD/IRCVDC/IRCVDF/IRCVDW
IREAD/IREADC/IREADF/IREADW
ISCHED
ISDAT/ISDATC/ISDATF/ISDATW
ISLEEP
ISPFN/ISPFNC/ISPFNF/ISPFNW
ITIMER
ITWAIT
IUNTIL
IWRITC/IWRITE/IWRITF/IWRITW
MRKT
MWAIT
```

O.2.1 Completion Routines

Completion routines are subprograms that execute asynchronously with a main program. A completion routine is scheduled to run as soon as possible after the event for which it has been waiting has completed (e.g., the completion of an I/O transfer, or the lapsing of a specified time interval). All completion routines of the current job have higher priority than other parts of the job; therefore, once a completion routine becomes runnable because of its associated event, it interrupts execution of the job and continues to execute until it relinquishes control.

Completion routines are handled differently in the Single-Job and the F/B versions of RT-11. In the Single-Job version, completion routines are totally asynchronous and can interrupt one another. In F/B, completion routines do not interrupt each other but are queued and made to wait until the correct job is running. (For further information on completion routines, see Sections 9.2.8 and O.1.4.)

A FORTRAN completion routine can have a maximum of two arguments:

```
SUBROUTINE  crtn {(iarg1,iarg2)}
```

where:

- iarg1 is equivalent to R0 on entry to an assembly language completion routine.
- iarg2 is equivalent to R1 on entry to an assembly language completion routine.

If an error occurs in a completion routine or in a subroutine at completion level, the error handler will trace back normally through to the original interruption of the main program. Thus the traceback is shown as though the completion routine were called from the main program and lets the user know where the main program was executing if a fatal error occurs.

Certain restrictions apply to completion routines, i.e., those routines that are activated by the following calls:

INTSET
IRCVDC
IRCVDF
IREADC
IREADF
ISCHED
ISDATC
ISDATF
ISPFNC
ISPFNF
ITIMER
IWRITC
IWRITF
MRKT

These restrictions are:

1. The first subroutine call that references a FORTRAN completion routine must be issued from the main program.

2. No channels may be allocated (by calls to IGETC) or freed (by calls to IFREEC) from a completion routine. Channels to be used by completion routines should be allocated and placed in a COMMON block for use by the routine.

3. The completion routine may not perform any call which requires the use of the USR, e.g., LOOKUP and IENTER. See Section O.2 for a list of SYSLIB functions that call the USR.

4. Files to be operated upon in completion routines must be opened and closed by the main program. There are, however, no restrictions on the input or output operations that may be performed in the completion routine. If many files must be made available to the completion routine, they may be opened by the main program and saved for later use (without tying up RT-11 channels) by the ISAVES call. The completion routine may later make them available by reattaching the file to a channel with an IREOPN call.

5. FORTRAN subprograms are reusable but not reentrant. A given subprogram can be used many times as a completion routine or as a routine in the main program but a subprogram executing as main program code will not work properly if interrupted at the completion level. This restriction applies to all subprograms that can be invoked at the completion level and can be active at the same time in the main program.

6. Only one completion function should be active at any time under the Single-Job Monitor (see Section O.1.4).

7. Assembly language completion routines must be exited via an RTS PC.

8. FORTRAN completion routines must be exited by execution of a RETURN or END statement in the subroutine.

O.2.2 Channel-Oriented Operations

An RT-11 channel being used for input/output with SYSLIB must be allocated in one of the following two ways:

1. The channel is allocated and marked in use to the FORTRAN I/O system by a call to IGETC and is later freed by a call to IFREEC.

2. An ICDFN call is issued to define more channels (up to 256). All channels numbered greater than 17 (octal) can be freely used by the programmer; the FORTRAN I/O system uses only channels 0 through 17 (octal).

Channels must be allocated in the main program routine or its subprograms, not in routines that are activated as the result of I/O completion events or ISCHED or ITIMER calls.

O.2.3 INTEGER*4 Support Functions

INTEGER*4 variables are allocated two words of storage. INTEGER*4 values are stored in two's complement representation. The first word (lower address) contains the low-order part of the value, and the second word (higher address) contains the sign and the high-order part of the value. The range of numbers supported is $-2^{31}+1$ to $2^{31}-1$.

Note that this format differs from the 2-word internal time format which stores the high-order part of the value in the first word and the low-order part in the second. The JJCVT function (Section O.3.59) is provided for conversion between the two internal formats.

Integer and real arguments to subprograms are indicated in the following manner in this appendix.

```
i = INTEGER*2 arguments
j = INTEGER*4 arguments
a = REAL*4 arguments
d = REAL*8 arguments
```

When the DATA statement is used to initialize INTEGER*4 variables, it must specify both the low- and high-order parts, i.e.,

```
INTEGER*4 J
DATA J/3/
```

only initializes the first word.

The correct way to initialize an INTEGER*4 variable to a constant (e.g., 3) is shown below:

```
INTEGER*4 J
INTEGER*2 I(2)
EQUIVALENCE (J,I)
DATA I/3,0/        !INITIALIZE J TO 3
```

If initializing an INTEGER*4 variable to a negative value (e.g., -4), the high-order (second word) part must be the continuation of the two's complement of the low-order part. For example:

```
INTEGER*4 J
INTEGER*2 I(2)
EQUIVALENCE (J,I)
DATA I/-4,-1/     !INITIALIZE J TO -4
```

The following form is suitable for INTEGER*4 arguments to subprograms:

```
INTEGER*2 J(2)
DATA J/3,0/       !LOW-ORDER,HIGH-ORDER
```

O.2.4 Character String Functions

The SYSLIB character string functions and routines provide variable-length string support for RT-11 FORTRAN. SYSLIB calls are provided to perform the following character string operations:

- Read character strings from a specified FORTRAN logical unit (GETSTR).
- Write character strings to a specified FORTRAN logical unit (PUTSTR).
- Concatenate variable-length strings (CONCAT).
- Return the position of one string in another (INDEX).
- Insert one string into another (INSERT).
- Return the length of a string (LEN).
- Repeat a character string (REPEAT).
- Compare two strings (SCOMP).
- Copy a character string (SCOPY).
- Pad a string with rightmost blanks (STRPAD).
- Copy a substring from a string (SUBSTR).
- Perform character modification (TRANSL).
- Remove trailing blanks (TRIM).
- Verify the presence of characters in a string (VERIFY).

Strings are stored in LOGICAL*1 arrays which are defined and dimensioned by the FORTRAN programmer. Strings are stored in these arrays as one character per array element plus a zero element to indicate the current end of the string (ASCIZ format).

The length of a string may vary at execution time, ranging from zero characters in length to one less than the size of the array which stores the string. The maximum size of any string is 32767 characters. Strings may contain any of the 7-bit ASCII characters except null (0), as the null character is used to mark the end of the string. Bit 7 of each character must be off (0); therefore, the valid characters are those whose decimal representations range from 1 to 127, inclusive.

The ASCII code used in this string package is the same as that employed by FORTRAN for A-type FORMAT items, ENCODE/DECODE strings, and object-time FORMAT strings. ASCIZ strings in the form used by these routines are generated by the FORTRAN compiler whenever quoted strings are used as arguments in the CALL statement. Note that a null string (a string containing no characters) may be represented in

FORTRAN by a variable or constant of any type which contains the value zero, or by a LOGICAL variable or constant with the .FALSE. value.

The SYSLIB user should ensure that a string never overflows the array that contains it by being aware of the length of the string result produced by each routine. In many routines where the result string length may vary or is difficult to determine, an optional integer argument may be specified to the subroutine to limit the length. In the sections describing the character string routines, this argument is called "len". The length of an output string is limited to the value specified for "len" plus one (for the null terminator); therefore the array receiving the result must be at least "len" plus one elements in size.

The optional argument "err" may be included when "len" is specified. "Err" is a logical variable which should be initialized by the FORTRAN program to the .FALSE. value. If a string function is given the arguments "len" and "err", and "len" is actually used to limit the length of the string result, then "err" is set to the .TRUE. value. If "len" is not used to truncate the string, "err" is unchanged, i.e., remains .FALSE..

"Len" and "err" are always optional arguments. "Len" may appear alone; however, "len" must appear if "err" is specified.

Several routines use the concept of character position. Each character in a string is assigned a position number which is one greater than the position of the character immediately to its left. The first character in a string is in position one.

O.2.4.1 Allocating Character String Variables -- A 1-dimensional LOGICAL*1 array may be used to contain a single string whose length may vary from zero characters to one less than the dimensioned length of the array. For example:

```
LOGICAL*1 A(45)      !ALLOCATE SPACE FOR STRING VARIABLE A
```

The preceding example allows array A to be used as a string variable that can contain a string of 44 or less characters. Similarly, a 2-dimensional LOGICAL*1 array may be used to contain a 1-dimensional array of strings. Each string in the array may have a length up to one less than the first dimension of the the LOGICAL*1 array. There may be as many strings as the number specified for the second dimension of the LOGICAL*1 array, e.g.,

```
LOGICAL*1 W(21,10)     !ALLOCATE AN ARRAY OF STRINGS
```

The preceding example creates a string array W which has 10 string elements, each of which may contain up to 20 characters. String I in array W is referenced in subroutine or function calls as W(1,I).

A 2-dimensional string array may be allocated, e.g.,

```
LOGICAL*1 T(14,5,7)      !ALLOCATE A 5 BY 7 STRING ARRAY
```

In the preceding example, each string in array T may vary in length to a maximum of 13 characters. String I,J of the array may be referenced as T(1,I,J). Note that T is the same as T(1,1,1). This dimensioning process may be continued to create string arrays of up to six dimensions (represented by LOGICAL*1 arrays of up to seven dimensions).

O.2.4.2 Passing Strings to Subprograms -- The LOGICAL*1 arrays which contain strings may be placed in a COMMON block and referenced by any or all routines with a similar COMMON declaration. However, care should be taken when a LOGICAL*1 array is placed in a COMMON block, for if such an array has an odd length, it may cause all succeeding variables in the COMMON block to be assigned odd addresses. This situation will be detected by the RT-11 FORTRAN compiler, resulting in a warning message if the /W switch is specified.

A LOGICAL*1 array has an odd length only if the product of its dimensions is odd, e.g.,

```
    LOGICAL*1 B(10,7)       !(10*7)=70; EVEN LENGTH
    LOGICAL*1 H(21)         !21 IS ODD; ODD LENGTH
```

If odd length arrays are to be placed in a COMMON block, they should either be placed at the end of the block or they should be paired to result in an effective even length. For example:

```
    COMMON A1,A2,A3(10),H(21)         !PLACE ODD-SIZED ARRAY AT END

         or

    COMMON A1,A2,H(21),H1(7),A3(10)   !PAIR ODD-SIZED ARRAYS H AND H1
```

Note that these cautions apply only to LOGICAL*1 variables and arrays.

The second method of passing strings to subprograms is through arguments and formal parameters. A single string may be passed by using its array name as an argument. For example:

```
    LOGICAL*1 A(21)       !STRING VARIABLE "A", 20 CHARACTERS MAXIMUM
    CALL SUBR(A)          !PASS STRING A TO SUBROUTINE SUBR
```

If the maximum length of a string argument is unknown in a subroutine or function, or if the routine is used to handle many different length strings, the dummy argument in the routine should be declared as a LOGICAL*1 array with a dimension of one, e.g., LOGICAL*1 ARG(1). In this case, the string routines will correctly determine the length of ARG whenever it is used, but it will not be possible to determine the maximum size string which may be stored in ARG. If a multi-dimensional array of strings is passed to a routine, it must be declared in the called program with the same dimensions as were specified in the calling program.

NOTE

> The length argument specified in many of the character string functions refers to the maximum length of the string excluding the necessary null byte terminator. The length of the LOGICAL*1 array to receive the string must be at least one greater than the length argument.

0.2.4.3 Using Quoted-String Literals -- Quoted-strings may be used as arguments to any of the string routines that are invoked by the CALL statement. They cannot be used for routines invoked as functions. For example:

```
CALL SCOMP(NAME,'DOE, JOHN',M)
```

compares the string in the array NAME to the constant string DOE, JOHN and sets the value of the integer variable M accordingly. Although the above form of the routine reference is permitted, the statement:

```
M=ISCOMP(NAME,'DOE, JOHN')
```

is not allowed since the routine is not invoked by the CALL statement and includes a quoted-string literal.

0.3 LIBRARY FUNCTIONS AND SUBROUTINES

This section presents all SYSLIB functions and subroutines in alphabetic order. To reference these subprograms by usage, see Table 0-1.

AJFLT

0.3.1 AJFLT

The AJFLT function converts an INTEGER*4 value to a REAL*4 value and returns that result as the function value.

Form: a = AJFLT (jsrc)

where: jsrc is the INTEGER*4 variable to be converted.

Function Results:

The function result is the REAL*4 value that is the result of the operation.

Example:

The following example converts the INTEGER*4 value contained in JVAL to single precision (REAL*4), multiplies it by 3.5, and stores the result in VALUE.

```
REAL*4 VALUE,AJFLT
INTEGER*4 JVAL
.
.
.
VALUE=AJFLT(JVAL)*3.5
```

CHAIN

O.3.2 CHAIN

The CHAIN subroutine allows a background program (or any program in the Single-Job system) to transfer control directly to another background program, passing it specified information. CHAIN cannot be called from a completion or interrupt routine. CHAIN does not close any of the FORTRAN logical units. When CHAINing to any other program, the user should explicitly close the opened logical units with calls to the CLOSE routine. Any USEREX routine specified to the FORTRAN run-time system will not be executed if a CHAIN is accomplished.

Form: CALL CHAIN (dblk,var,wcnt)

where:

dblk — is the address of a 4-word Radix-50 descriptor of the file specification for the program to be run. (See the FORTRAN Language Reference Manual, Section 2.2.8.)

var — is the first variable in a sequence of variables with increasing memory addresses to be passed between programs in the chain parameter area (absolute locations 510 up to 700). A single array or a COMMON block (or portion of a COMMON block) is a suitable sequence of variables.

wcnt — is a word count (up to 60 words) specifying the number of words (beginning at var) to be passed to the called program.

If the size of the chain parameter area is insufficient, it may be increased by specifying the /B (bottom) switch to LINK for both the program executing the CHAIN call and the program receiving control.

The data passed may be accessed through a call to the RCHAIN routine (see Section O.3.74). For more information on chaining to other programs, see Section 9.4.2.

Errors:

None.

Example:

The following example transfers control from the main program to PROG.SAV, on DT0, passing it variables.

```
     REAL*4 PROGNM                          !RAD50 FOR PROGRAM NAME
     COMMON /BLK1/ A,B,C,D                  !DATA TO BE PASSED
     DATA PROGNM/6RDT0PRO,6RG  SAV/
     .
     .
     .
     CALL CHAIN(PROGNM,A,8)                 !RUN DT0:PROG.SAV
```

CLOSEC

0.3.3 CLOSEC

The CLOSEC subroutine terminates activity on the specified channel and frees it for use in another operation. The handler for the associated device must be in memory. CLOSEC cannot be called from a completion or interrupt routine.

Form: CALL CLOSEC (chan)

where: chan is the channel number to be closed. This argument must be located so that the USR cannot swap over it.

A CLOSEC or PURGE must eventually be issued for any channel opened for either input or output. A CLOSEC call specifying a channel that is not open is ignored.

A CLOSEC performed on a file that was opened via an IENTER causes the device directory to be updated to make that file permanent. If the device associated with the specified channel already contains a file with the same name and extension, the old copy is deleted when the new file is made permanent. A CLOSEC on a file opened via LOOKUP does not require any directory operations.

When an entered file is CLOSECed, its permanent length reflects the highest block of the file written since the file was entered; for example, if the highest block written is block number 0, the file is given a length of 1; if the file was never written, it is given a length of 0. If this length is less than the size of the area which was allocated at IENTER time, the unused blocks are reclaimed as an empty area on the device.

Errors:

CLOSEC does not generate any errors. If the device handler for the operation is not in memory, a fatal monitor error is generated.

Example:

The following example creates and processes a 56-block file.

```
        REAL*4 DBLK(2)
        DATA DBLK/6RSY0NEW,6RFILDAT/
        DATA ISIZE/56/
        .
        .
        .
        ICHAN=IGETC()
        IF(ICHAN.LT.0) GOTO 100
        IF(IENTER(ICHAN,DBLK,ISIZE)-1) 10,110,120
   10   .
        .
        .
        CALL CLOSEC(ICHAN)
        CALL IFREEC(ICHAN)
        CALL EXIT
  100   STOP 'NO AVAILABLE CHANNELS'
  110   STOP 'CHANNEL ALREADY IN USE'
  120   STOP 'NOT ENOUGH ROOM ON DEVICE'
        END
```

CONCAT

0.3.4 CONCAT

The CONCAT subroutine is used to concatenate character strings.

Form: CALL CONCAT (a,b,out{,len{,err}})

where:

a	is the array containing the left string.
b	is the array containing the right string.
out	is the array into which the concatenated result is placed. This array must be at least one element longer than the maximum length of the result string (i.e., one greater than the value of len, if specified).
len	is the integer number of characters representing the maximum length of the output string. The effect of len is to truncate the output string to a given length, if necessary.
err	is the Logical error flag set if the output string is truncated to the length specified by len.

The string in array "a" immediately followed on the right by the string in array "b" and a terminating null character replaces the string in array "out". Any combination of string arguments is allowed so long as "b" and "out" do not specify the same array. Concatenation stops either when a null character is detected in "b" or when the number of characters specified by "len" have been moved.

If either the left or right string is a null string, the other string is copied to "out". If both are null strings, then "out" is set to a null string. The old contents of "out" are lost when this routine is called.

Errors:

Error conditions are indicated by "err", if specified. If "err" is given and the output string would have been longer than "len" characters, then "err" is set to .TRUE.; otherwise, "err" is unchanged.

Example:

The following example concatenates the string in array STR and the string in array IN and stores the resultant string in array OUT. OUT cannot be larger than 29 characters.

```
LOGICAL*1 IN(30),OUT(30),STR(7)
.
.
.
CALL CONCAT(STR,IN,OUT,29)
```


O.3.5 CVTTIM

The CVTTIM subroutine converts a 2-word internal format time to hours, minutes, seconds, and ticks.

Form: CALL CVTTIM (time,hrs,min,sec,tick)

where: time is the 2-word internal format time to be converted. If time is considered as a 2-element INTEGER*2 array, then:

time (1) is the high-order time.
time (2) is the low-order time.

hrs is the integer number of hours.

min	is the integer number of minutes.
sec	is the integer number of seconds.
tick	is the integer number of ticks (1/60 of a second for 60-cycle clocks; 1/50 of a second for 50-cycle clocks).

Errors:

None.

Example:

```
INTEGER*4 ITIME
.
.
.
CALL GTIM(ITIME)                     !GET CURRENT TIME-OF-DAY
CALL CVTTIM(ITIME,IHRS,IMIN,ISEC,ITCK)
IF(IHRS.GE.12) GOTO 100              !TIME FOR LUNCH
```

DEVICE

0.3.6 DEVICE (F/B only)

The DEVICE subroutine allows the user to set up a list of addresses to be loaded with specified values when the program is terminated. If a job terminates or is aborted with a CTRL C from the terminal, this list is picked up by the system and the appropriate addresses are set up with the corresponding values.

This function is primarily designed to allow user programs to load device registers with necessary values. In particular, it is used to turn off a device's interrupt enable bit when the program servicing the device terminates.

Only one address list may be active at any given time; hence, if multiple DEVICE calls are issued, only the last one has any effect. The list must not be modified by the FORTRAN program after the DEVICE call has been issued, and the list must not be located in an overlay or an area over which the USR will swap.

Form: CALL DEVICE (ilist)

where: ilist is an integer array containing address/value pairs, terminated by a zero word. On program termination, each value is moved to the corresponding address.

For more information on loading values into device registers, see the assembly language .DEVICE request, Section 9.4.11.

Errors:

None.

Example:

```
    INTEGER*2 IDR11(3)                  !DEVICE ARRAY SPEC
    DATA IDR11(1)/"167770/              !DR11 CSR ADDRESS (OCTAL)
    DATA IDR11(2)/0/                    !VALUE TO CLEAR INTERRUPT ENABLE
    DATA IDR11(3)/0/                    !AND END-OF-LIST FLAG
    CALL DEVICE(IDR11)                  !SET UP FOR ABORT
```

DJFLT

O.3.7 DJFLT

The DJFLT function converts an INTEGER*4 value into a REAL*8 (DOUBLE PRECISION) value and returns that result as the function value.

Form: d = DJFLT (jsrc)

where: jsrc specifies the INTEGER*4 variable which is to be converted.

Notes:

If DJFLT is used, it must be explicitly defined (REAL*8 DJFLT) or implicitly defined (IMPLICIT REAL*8 (D)) in the FORTRAN program. If this is not done, its type will be assumed as REAL*4 (single precision).

Function Results:

The function result is the REAL*8 value that is the result of the operation.

Example:

```
        INTEGER*4 JVAL
        REAL*8 DJFLT,D
        .
        .
        .
        D=DJFLT(JVAL)
```

GETSTR

O.3.8 GETSTR

The GETSTR subroutine reads a formatted ASCII record from a specified FORTRAN logical unit into a specified array. The data is truncated (trailing blanks removed) and a null byte is inserted at the end to form a character string.

GETSTR can be used in main program routines or in completion routines but cannot be used in both at the same time. If GETSTR is used in a completion routine, it cannot be the first I/O operation on the specified logical unit.

Form: CALL GETSTR (lun,out,len{,err})

where:

- lun — is the integer FORTRAN logical unit number of a formatted sequential file from which the string is to be read.
- out — is the array which is to receive the string; this array must be one element longer than len.
- len — is the integer number representing the maximum length of the string to be input.
- err — is the Logical error flag that is set to .TRUE. if the string specified by "out" exceeds the value of "len" in length; err must be present if the input string can exceed "len".

Errors:

Error conditions are indicated by "err". If "err" is given and the output string was truncated to the length specified by "len", then "err" is set to .TRUE.; otherwise, "err" remains unchanged.

Example:

The following example reads a string of up to 80 characters from logical unit 5 into the array STRING.

```
LOGICAL*1 STRING(81),ERR
.
.
.
CALL GETSTR(5,STRING,80,ERR)
```

GTIM

0.3.9 GTIM

THE GTIM subroutine allows user programs to access the current time of day. The time is returned in two words and is given in terms of clock ticks past midnight. If the system does not have a line clock, a value of zero is returned. If an RT-11 monitor TIME command has not been entered, the value returned will be the time elapsed since the system was bootstrapped rather than the time of day.

Form: CALL GTIM (itime)

where: itime is the 2-word area to receive the time of day.

The high-order time is returned in the first word, the low-order time in the second word. The SYSLIB routine CVTTIM (Section 0.3.5) can be used to convert the time into hours, minutes, seconds and ticks. CVTTIM performs the conversion based on the monitor configuration word for 50- or 60-cycle clocks (see Section 9.2.6). Under an F/B Monitor, the time-of-day is automatically reset after 24:00 when a GTIM is executed; under the Single-Job Monitor, it is not.

Errors:

None.

Example:

```
INTEGER*4 JTIME
.
.
.
CALL GTIM(JTIME)
```

GTJB

0.3.10 GTJB

The GTJB subroutine passes certain job parameters back to the user program.

Form: CALL GTJB (addr)

where: addr is an 8-word area to receive the job parameters. This area, considered as an 8-element INTEGER*2 array, has the following format:

addr(1)	job number. (0=Background, 2=Foreground)
addr(2)	high memory limit
addr(3)	low memory limit
addr(4)	beginning of I/O channel space
addr(5)-addr(8)	reserved for future use

For more information on passing job parameters, see the assembly language .GTJB request, Section 9.4.17.

Errors:

None.

Example:

```
      INTEGER*2 PARAMS(8)
      CALL GTJB(PARAMS)
      IF(PARAMS(1).EQ.0) TYPE 99
99    FORMAT (' THIS IS THE BACKGROUND JOB')
```

IADDR

O.3.11 IADDR

The IADDR function returns the 16-bit absolute memory address of its argument as the integer function value.

Form: i = IADDR (arg)

where: arg is the variable, constant, or expression whose memory address is to be obtained.

Errors:

None.

Example:

IADDR can be used to find the address of an assembly language global area. For example:

```
      EXTERNAL CAREA
      J=IADDR(CAREA)
```

IAJFLT

O.3.12 IAJFLT

The IAJFLT function converts an INTEGER*4 value to a REAL*4 value and stores the result.

Form: i = IAJFLT (jsrc,ares)

where: jsrc is the INTEGER*4 variable to be converted.

ares is the REAL*4 variable or array element to receive the converted value.

Function Results:

The function result indicates the following:

i = -2	Significant digits were lost during the conversion.
= -1	Normal return; the result is negative.
= 0	Normal return; the result is zero.
= 1	Normal return; the result is positive.

Example:

```
      INTEGER*4 JVAL
      REAL*4 RESULT
      .
      .
      .
      IF(IAJFLT(JVAL,RESULT).EQ.-2) TYPE 99
99    FORMAT (' OVERFLOW IN INTEGER*4 TO REAL CONVERSION')
```

IASIGN

O.3.13 IASIGN

The IASIGN function sets information in the FORTRAN logical unit table (overriding the defaults) so that the specified information is used when the FORTRAN Object Time System (OTS) opens the logical unit. This function can be used with ICSI (see Section O.3.17) to allow a FORTRAN program to accept a standard CSI input specification. IASIGN must be called before the unit is opened, i.e., before any READ, WRITE, PRINT, TYPE, or ACCEPT statements are executed that reference the logical unit.

Form: i = IASIGN (lun,idev{,ifilex{,isize{,itype}}})

where:

lun — is an INTEGER*2 variable, constant, or expression specifying the FORTRAN logical unit for which information is being specified.

idev — is a 1-word Radix-50 device name; this can be the first word of an ICSI input or output file specification.

ifilex — is a 3-word Radix-50 filename and extension; this can be words 2 through 4 of an ICSI input or output file specification.

isize — is the length (in number of blocks) to allocate for an output file; this can be the fifth word of an ICSI output specification. If zero, the larger of either one-half the largest empty segment or the entire second largest empty segement is allocated (see Section 9.4.13). If the value specified for length is -1, the entire largest empty segment is allocated.

itype — is an integer value determining the optional attributes to be assigned to the file. This value is obtained by adding the values that correspond to the desired operations:

1 use double buffering for output
2 open the file as a temporary file
4 perform a lookup on the file (otherwise, the first FORTRAN I/O operation determines how the file is opened)

8	expand carriage control information (see Notes below)
16	do not expand carriage control information
32	file is read-only

Notes:

Expanded carriage control information applies only to formatted output files and means that the first character of each record is used as a carriage control character when processing a write operation to the given logical unit. The first character is removed from the record and converted to the appropriate ASCII characters to simulate the requested carriage control.

If carriage control information is not expanded, the first character of each record is unmodified and the FORTRAN OTS outputs a line feed, followed by the record, followed by a carriage return.

If carriage control is unspecified, the FORTRAN OTS sends expanded carriage control information to the terminal and line printer and sends unexpanded carriage control information to all other devices and files. See the PDP-11 FORTRAN Language Reference Manual, Section 6.3, for further carriage control information.

Function Results:

i = 0	Normal return.
<> 0	The specified logical unit is already in use or there is no space for another logical unit association.

Example:

The following example creates an output file on logical unit 3 using the first output file given to the RT-11 command string interpreter (CSI), sets it up for double buffering, creates an input file on logical unit 4 based on the first input file specification given to the RT-11 CSI, and makes it available for read-only access.

```
          INTEGER*2 SPEC(39)
          REAL*4 EXT(2)
          DATA EXT/6RDATDAT,6RDATDAT/        !DEFAULT EXTENSION IS DAT
          .
          .
          .
     10   IF(ICSI(SPEC,EXT,,,0).NE.0) GOTO 10
     C
     C    DO NOT ACCEPT ANY SWITCHES
     C
          CALL IASIGN(3,SPEC(1),SPEC(2),SPEC(5),1)
          CALL IASIGN(4,SPEC(16),SPEC(17),0,32)
```

ICDFN

0.3.14 ICDFN

The ICDFN function increases the number of input/output channels. Note that ICDFN defines new channels; the previously-defined channels are not used. Thus, an ICDFN for 20 channels (while the 16 original channels are defined) causes only 20 I/O channels to exist; the space for the original 16 is unused. The space for the new channel area is allocated out of the free space managed by the FORTRAN system.

Form: i = ICDFN (num)

where:	num	is the integer number of channels to be allocated. The number of channels must be greater than 16 and can be a maximum of 256. SYSLIB can use all new channels greater than 16 without a call to IGETC; the FORTRAN system input/output uses only the first 16 channels. This argument must be positioned so that the USR cannot swap over it.

Notes:

1. ICDFN cannot be issued from a completion or interrupt routine.
2. It is recommended that the ICDFN function be used at the beginning of the main program before any I/O operations are initiated.
3. If ICDFN is executed more than once, a completely new set of channels is created each time ICDFN is called.
4. ICDFN requires that extra memory space be allocated to foreground programs (see Section 0.1.4).

Function Results:

i	= 0	Normal return.
	= 1	An attempt was made to allocate fewer channels than already exist.
	= 2	Not enough free space is available for the channel area.

Example:

```
IF(ICDFN(24).NE.0) STOP 'NOT ENOUGH MEMORY'
```

ICHCPY

0.3.15 ICHCPY (F/B only)

The ICHCPY function opens a channel for input, logically connecting it to a file which is currently open by another job for either input or output. This function may be used by either the foreground or the background. An ICHCPY must be done before the first read or write for the given channel.

Form: i = ICHCPY (chan,ochan)

where:

	chan	is the channel which the job will use to read the data.
	ochan	is the channel number of the other job which is to be copied.

Notes:

1. If the other job's channel was opened via an IENTER function or a .ENTER programmed request to create a file, the copier's channel indicates a file which extends to the highest block that the creator of the file had written at the time the ICHCPY was executed.

2. A channel which is open on a sequential-access device should not be copied, because buffer requests may become intermixed.

3. A program can write to a file (which is being created by the other job) on a copied channel just as it could if it were the creator. When the copier's channel is closed, however, no directory update takes place.

Errors:

i = 0	Normal return.
= 1	Other job does not exist or does not have the specified channel (ochan) open.
= 2	Channel (chan) is already open.

ICMKT

O.3.16 ICMKT (F/B only)

The ICMKT function causes one or more scheduling requests (made by an ISCHED, ITIMER or MRKT routine) to be cancelled.

Form: i = ICMKT (id,time)

where:

	id	is the identification integer of the request to be cancelled. If id is equal to zero, all scheduling requests are cancelled.
	time	is the name of a 2-word area in which the monitor will return the amount of time remaining in the cancelled request.

For further information on cancelling scheduling requests, see the assembly language .CMKT request, Section 9.4.5.

Errors:

i = 0	Normal return.
= 1	id was not equal to zero and no schedule request with that identification could be found.

Example:

```
      INTEGER*4 J
      .
      .
      .
      CALL ICMKT(0,J)      !ABORT ALL TIMER REQUESTS NOW
      .
      .
      .
      END
```

ICSI

O.3.17 ICSI

The ICSI function calls the RT-11 command string interpreter in special mode to parse a command string and return file descriptors and switches to the program. In this mode, the CSI does not perform any handler fetches, CLOSEs, ENTERs, or LOOKUPs. ICSI cannot be called from a completion or interrupt routine.

Form: i = ICSI (filspc,defext,{cstring},{switch},x)

where: filspc is the 39-word area to receive the file specifications. The format of this area (considered as a 39-element INTEGER*2 array) is:

filspc(1)- filspc(4)	output file number 1 specification
filspc(5)	output file number 1 length
filspc(6)- filspc(9)	output file number 2 specification
filspc(10)	output file number 2 length
filspc(11)- filspc(14)	output file number 3 specification
filspc(15)	output file number 3 length
filspc(16)- filspc(19)	input file number 1 specification
filspc(20)- filspc(23)	input file number 2 specification
filspc(24)- filspc(27)	input file number 3 specification
filspc(28)- filspc(31)	input file number 4 specification
filspc(32)- filspc(35)	input file number 5 specification
filspc(36)- filspc(39)	input file number 6 specification

defext is the table of Radix-50 default extensions to be assumed when a file is specified without an extension.

defext(1)	is the default for all input file extensions.
defext(2)	is the default extension for output file number 1.
defext(3)	is the default extension for output file number 2.
defext(4)	is the default extension for output file number 3.

cstring — is the area which contains the ASCIZ command string to be interpreted; the string must end in a zero byte. If this argument is omitted, the system prints the prompt character (*) at the terminal and accepts a command string.

switch — is the name of an INTEGER*2 array dimensioned (4,n) where n represents the number of switches which are defined to the program. This argument must be present if the value specified for "x" is non-zero. This array has the following format for the nth switch described by the array.

switch(1,n)	is the 1-character ASCII name of the switch.
switch(2,n)	is set by the routine to 0, if the switch did not occur; to 1, if the switch occurred without a value; to 2, if the switch occurred with a value.
switch(3,n)	is set to the file number on which the switch is specified.
switch(4,n)	is set to the specified value if switch(2,n) is equal to 2.

x — is the number of switches defined in the array "switch".

Notes:

The array "switch" must be set up to contain the names of the valid switches. For example, use the following to set up names for five switches:

```
INTEGER*2 SW(4,5)
DATA SW(1,1)/'S'/,SW(1,2)/'M'/,SW(1,3)/'I'/
DATA SW(1,4)/'L'/,SW(1,5)/'E'/
```

Multiple occurrences of the same switch are supported by allocating an entry in the switch array for each occurrence of the switch. Each time the switch occurs in the switch array, the next unused entry for the named switch will be used.

The arguments of ICSI must be positioned so that the USR cannot swap over them.

For more information on calling the Command String Interpreter, see the assembly language .CSISPC request, Section 9.4.8.

Errors:

i = 0	Normal return.
= 1	Illegal command line; no data was returned.
= 2	An illegal device specification occurred in the string.
= 3	An illegal switch was specified, or a given switch was specified more times than allowed for in the switch array.

Example:

The following example causes the program to loop until a valid command is typed at the console terminal.

```
        INTEGER*2 SPEC(39)
        REAL*4 EXT(2)
        DATA EXT/6RDATDAT,6RDATDAT/
        .
        .
        .
10      TYPE 99
99      FORMAT (' ENTER VALID CSI STRING WITH NO SWITCHES')
        IF(ICSI(SPEC,EXT,,,0).NE.0) GOTO 10
```

ICSTAT

O.3.18 ICSTAT (F/B only)

The ICSTAT function furnishes the user with information about a channel. It is supported only in the F/B environment; no information is returned when operating under the Single-Job Monitor.

Form: i = ICSTAT (chan,addr)

where: chan is the channel whose status is desired.

addr is a 6-word area to receive the status information. The area, as a 6-element INTEGER*2 array, has the following format.

addr(1)	channel status word (see Section 9.4.34)
addr(2)	starting block number of file on this channel
addr(3)	length of file
addr(4)	highest block number written since file was opened (see Section 9.4.9)

addr(5)	unit number of device with which this channel is associated
addr(6)	Radix-50 of device name with which the channel is associated

Errors:

i = 0	Normal return.
= 1	Channel specified is not open.

Example:

The following example obtains channel status information about channel I.

```
        INTEGER*2 AREA(6)
        I=7
        IF(ICSTAT(I,AREA).NE.0) TYPE 99,I
99      FORMAT(1X,'CHANNEL',I4,'IS NOT OPEN')
```

IDELET

O.3.19 IDELET

The IDELET function deletes a named file from an indicated device. IDELET cannot be issued from a completion or interrupt routine.

Form: i = IDELET (chan,dblk{,count})

where:

chan	is the channel to be used for the delete operation.
dblk	is the 4-word Radix-50 specification (dev:filnam.ext) for the file to be deleted.
count	is used by magtape/cassette only to prevent a rewind before the operation. (Refer to Appendix H for more information concerning the magtape and cassette handlers.)

NOTE

The arguments of IDELET must be located so that the USR cannot swap over them.

The specified channel is left inactive when the IDELET is complete. IDELET requires that the handler to be used be resident (via an IFETCH call) at the time the IDELET is issued. If it is not, a monitor error occurs.

For further information on deleting files, see the assembly language .DELETE request, Section 9.4.10.

Errors:

i = 0	Normal return.
= 1	Channel specified is already open.
= 2	File specified was not found.

Example:

The following example deletes a file named FTN5.DAT from SY0.

```
          REAL*4 FILNAM(2)
          DATA FILNAM/6RSY0FTN,6R5  DAT/
          .
          .
          .
          I=IGETC()
          IF(I.LT.0) STOP 'NO CHANNEL'
          CALL IDELET(I,FILNAM)
          CALL IFREEC(I)
```

IDJFLT

O.3.20 IDJFLT

The IDJFLT function converts an INTEGER*4 value into a REAL*8 (DOUBLE PRECISION) value and stores the result.

Form: i = IDJFLT (jsrc,dres)

where:

jsrc	specifies the INTEGER*4 variable that is to be converted.
dres	specifies the REAL*8 (or DOUBLE PRECISION) variable to receive the converted value.

Function Results:

The function result indicates the following:

i = -1	Normal return; the result is negative.
= 0	Normal return; the result is zero.
= 1	Normal return; the result is positive.

Example:

```
        INTEGER*4 JJ
        REAL*8 DJ
        .
        .
        .
        IF(IDJFLT(JJ,DJ).LE.0) TYPE 99
  99    FORMAT (' VALUE IS NOT POSITIVE')
```

IDSTAT

O.3.21 IDSTAT

The IDSTAT function is used to obtain information about a particular device. IDSTAT cannot be issued from a completion or interrupt routine.

Form: i = IDSTAT (devnam,cblk)

where:

devnam — is the Radix-50 device name.

cblk — is the 4-word area used to store the status information. The area, as a 4-element INTEGER*2 array, has the following format:

cblk(1)	device status word (see Section 9.4.12)
cblk(2)	size of handler in bytes
cblk(3)	entry point of handler (non-zero implies that the handler is in memory)
cblk(4)	size of the device (in 256-word blocks) for block-replaceable devices; zero for sequential-access devices

NOTE

The arguments of IDSTAT must be positioned so that the USR cannot swap over them.

IDSTAT looks for the device specified by devnam and, if found, returns four words of status in cblk.

Errors:

i = 0	Normal return.
= 1	Device not found in monitor tables.

Example:

The following example determines whether the line printer handler is in memory. If it is not, the program stops and prints a message to indicate that the handler must be loaded.

```
REAL*4 IDNAM
INTEGER*2 CBLK(4)
DATA IDNAM/3RLP /
DATA CBLK/4*0/
CALL IDSTAT(IDNAM,CBLK)
IF(CBLK(3).EQ.0) STOP 'LOAD THE LP HANDLER AND RERUN'
```

IENTER

O.3.22 IENTER

The IENTER function allocates space on the specified device and creates a tentative directory entry for the named file. If a file of the same name already exists on the specified device, it is not deleted until the tentative entry is made permanent by CLOSEC. The file is attached to the channel number specified.

Form: i = IENTER (chan,dblk,length{,count})

where:

chan	is the integer specification for the RT-11 channel to be associated with the file.
dblk	is the 4-word Radix-50 descriptor of the file to be operated upon.
length	is the integer number of blocks to be allocated for the file. If zero, the larger of either one-half the largest empty segment or the entire second largest empty segment is allocated (see Section 9.4.13). If the value specified for length is -1, the entire largest empty segment is allocated.
count	is used by magtape/cassette only to prevent a rewind before the operation (see Appendix H).

Notes:

1. IENTER cannot be issued from a completion or interrupt routine.

2. IENTER requires that the appropriate device handler be in memory.

3. The arguments of IENTER must be positioned so that the USR does not swap over them.

For further information on creating tentative directory entries, see the assembly language .ENTER request, Section 9.4.13.

Errors:

i = n	Normal return; number of blocks actually allocated (n = 0 for nonfile-structured IENTER).
= -1	Channel (chan) is already in use.
= -2	In a fixed-length request, no space greater than or equal to length was found.

Example:

The following example allocates a channel for file TEMP.TMP on Y0. If no channel is available, the program prints a message and halts.

```
        REAL*4 DBLK(2)
        DATA DBLK/6RSY0TEM,6RP  TMP/
        ICHAN=IGETC()
        IF(ICHAN.LT.0) STOP 'NO AVAILABLE CHANNEL'
C
C       CREATE TEMPORARY WORK FILE
C
        IF(IENTER(ICHAN,DBLK,20).LT.0) STOP 'ENTER FAILURE'
        .
        .
        .
        CALL PURGE(ICHAN)
        CALL IFREEC(ICHAN)
```

IFETCH

O.3.23 IFETCH

The IFETCH function loads a device handler into memory from the system device, making the device available for input/output operations. The handler is loaded into the free area managed by the FORTRAN system. Once the handler is loaded, it cannot be released and the memory in which it resides cannot be reclaimed. IFETCH cannot be issued from a completion or interrupt routine.

Form: i = IFETCH (devnam)

where: devnam is the 1-word Radix-50 name of the device for which the handler is desired. This argument can be the first word of an ICSI input or output file specification. This argument must be positioned so that the USR cannot swap over it.

For further information on loading device handlers into memory, see the assembly language .FETCH request, Section 9.4.15.

Errors:

i = 0	Normal return.
= 1	Device name specified does not exist.
= 2	Not enough room exists to load the handler.
= 3	No handler for the specified device exists on the system device.

Example:

The following example requests the DX1 handler to be loaded into memory; execution stops if the handler cannot be loaded.

```
        REAL*4 IDNAM
        DATA IDNAM/3RDX1/
        .
        .
        .
        IF (IFETCH(IDNAM).NE.0) STOP 'FATAL ERROR FETCHING HANDLER'
```

IFREEC

O.3.24 IFREEC

The IFREEC function returns a specified RT-11 channel to the available pool of channels. Before IFREEC is called, the specified channel must be closed or deactivated with a CLOSEC (see Section O.3.3) or a PURGE (see Section O.3.70) call. IFREEC cannot be called from a completion or interrupt routine. IFREEC calls must be issued only for channels that have been successfully allocated by IGETC calls; otherwise, unpredictable results will occur.

Form: i = IFREEC (chan)

where: chan is the integer number of the channel to be freed.

Errors:

i = 0	Normal return.
= 1	Specified channel is not currently allocated.

Example:

See the example under IGETC, (Section 0.3.25).

IGETC

0.3.25 IGETC

The IGETC function allocates an RT-11 channel (in the range 0-17 octal) and marks it in use for the FORTRAN I/O system. IGETC cannot be issued from a completion or interrupt routine.

Form: i = IGETC()

Function Results:

i = -1	No channels are available.
= n	Channel n has been allocated.

Example:

```
ICHAN=IGETC()              !ALLOCATE CHANNEL
IF(ICHAN.LT.0) STOP 'CANNOT ALLOCATE CHANNEL'
.
.
.
CALL IFREEC(ICHAN)         !FREE IT WHEN THROUGH
.
.
.
END
```

IJCVT

0.3.26 IJCVT

The IJCVT function converts an INTEGER*4 value to INTEGER*2 format. If ires is not specified, the result returned is the INTEGER*2 value of jsrc. If ires is specified, the result is stored there.

Form: i = IJCVT (jsrc{,ires})

where:

where:	jsrc	specifies the INTEGER*4 variable or array element whose value is to be converted.
	ires	specifies the INTEGER*2 entity to receive the conversion result.

Function results if ires is specified:

i = -2	An overflow occurred during conversion.
= -1	Normal return; the result is negative.
= 0	Normal return; the result is zero.
= 1	Normal return; the result is positive.

Example:

```
        INTEGER*4 JVAL
        INTEGER*2 IVAL
        .
        .
        .
        IF(IJCVT(JVAL,IVAL).EQ.-2) TYPE 99
  99    FORMAT(' NUMBER TOO LARGE IN IJCVT CONVERSION')
```

ILUN

0.3.27 ILUN

The ILUN function returns the RT-11 channel number with which a FORTRAN logical unit is associated.

Form: i = ILUN (lun)

where: lun is an integer expression whose value is a FORTRAN logical unit number in the range 1-99.

Function Results:

i = -1	Logical unit is not open.
= -2	Logical unit is opened to console terminal.
= +n	RT-11 channel number n is associated with lun.

Example:

```
      PRINT 99
99    FORMAT(' PRINT DEFAULTS TO LOGICAL UNIT 6, WHICH FURTHER DEFAULTS TO LP:')
      LUNRT=ILUN(6)                    !WHICH RT-11 CHANNEL IS RECEIVING I/O?
```

INDEX

O.3.28 INDEX

The INDEX subroutine searches a string for the occurrence of another string and returns the character position of the first occurrence of that string.

Form: CALL INDEX (a,pattrn,{i},m)

or

m = INDEX (a,pattrn{,i})

where:

- a — is the array containing the string to be searched.
- pattrn — is the string being sought.
- i — is the integer starting character position of the search in a. If i is omitted, a is searched beginning at the first character position.
- m — is the integer result of the search; m equals the starting character position of pattrn in a, if found; otherwise it is zero.

Errors:

None.

Example:

The following example searches the array STRING for the first occurrence of strings EFG and XYZ and searches the string ABCABCABC for the occurrence of string ABC after position 5.

```
CALL SCOPY('ABCDEFGHI',STRING)      !INITIALIZE STRING
CALL INDEX(STRING,'EFG',,M)         !M=5
CALL INDEX(STRING,'XYZ',,N)         !N=0
CALL INDEX('ABCABCABC','ABC',5,L)   !L=7
```

INSERT

0.3.29 INSERT

The INSERT subroutine replaces a portion of one string with another string.

Form: CALL INSERT (in,out,i{,m})

where:

- in — is the array containing the string being inserted.
- out — is the array containing the string being modified.
- i — is the integer specifying the character position in "out" at which the insertion begins.
- m — is the integer maximum number of characters to be inserted.

If the maximum number of characters (m) is not specified, all characters to the right of the specified character position (i) in the string being modified are replaced by the string being inserted. The insert string (in) and the string being modified (out) may be in the same array only if the maximum number of characters (m) is specified and is less than or equal to the difference between the position of the insert (i) and the maximum string length of the array.

Errors:

None.

Example:

```
CALL SCOPY('ABCDEFGHIJ',S1)      !INITIALIZE STRING 1
CALL SCOPY(S1,S2)                !INITIALIZE STRING 2
CALL INSERT('123',S1,6,3)        !S1 = 'ABCDE123IJ'
CALL INSERT('123',S2,4)          !S2 = 'ABC123'
```

INTSET

0.3.30 INTSET

The INTSET function establishes a FORTRAN subroutine as an interrupt service routine, assigns it a priority, and attaches it to a vector. INTSET requires that extra memory be allocated to foreground programs that use it (see Section 0.1.4).

Form: i = INTSET (vect,pri,id,crtn)

where:

where:	vect	is the integer specifying the address of the interrupt vector to which the subroutine is to be attached.
	pri	is the integer specifying the actual priority level (4-7) at which the device interrupts.
	id	is the identification integer to be passed as the single argument to the FORTRAN routine when an interrupt occurs. This allows a single crtn to be associated with several INTSET calls.
	crtn	is a FORTRAN subroutine to be established as the interrupt routine. This name should be specified in an EXTERNAL statement in the FORTRAN program that calls INTSET. The subroutine has one argument:

```
SUBROUTINE crtn(id)
INTEGER id
```

When the routine is entered, the value of the integer argument will be the value specified for id in the appropriate INTSET call.

Notes:

1. The "id" argument may be used to distinguish between interrupts from different vectors if the routine to be activated services multiple devices.

2. When using INTSET in F/B, the SYSLIB call DEVICE must be used in almost all cases to prevent interrupts from interrupting beyond program termination.

3. If the interrupt routine (crtn) has control for a period of

time longer than the time in which two more interrupts using the same vector occur, interrupt overrun is considered to have occurred. The error message:

?SYSLIB- FATAL INTERRUPT OVERRUN

is printed and the job is aborted. Tasks requiring very fast interrupt response may not be viable using FORTRAN as FORTRAN overhead lowers RT-11's interrupt response rate.

4. The interrupt routine (crtn) is actually run as a completion routine via use of the RT-11 .SYNCH request (Section 9.3.1.4). The "pri" argument is used for the RT-11 .INTEN request (Section 9.3.1.2).

5. A .PROTECT request is issued for the vector, but no attempt is made to report an error if the vector is already protected; furthermore, the vector will be taken over unconditionally. See Section 9.4.25 for more information.

6. The FORTRAN interrupt service subroutine (crtn) cannot call the USR.

7. INTSET cannot be called from a completion or interrupt routine.

8. Interrupt enable should not be set on the associated device until the INTSET call has been successfully executed.

Errors:

```
i = 0    Normal return.
  = 1    Invalid vector specification.
  = 2    Reserved for future use.
  = 3    No space is available for the linkage setup.
```

Example:

```
EXTERNAL CLKSUB                  !SUBR TO HANDLE KW11-P CLOCK
.
.
.
I=INTSET("104,6,0,CLKSUB)        !ATTACH ROUTINE
IF (I.NE.0) GOTO 100             !BRANCH IF ERROR
.
.
.
END
SUBROUTINE CLKSUB(ID)
.
.
.
END
```

IPEEK

O.3.31 IPEEK

The IPEEK function returns the contents of the word located at a specified absolute 16-bit memory address. This function may be used to examine device registers or any location in memory.

Form: i = IPEEK (iaddr)

where: iaddr is the integer specification of the absolute address to be examined. If this argument is not an even value, a trap will result.

Function Result:

The function result (i) is set to the value of the word examined.

Example:

```
ISWIT = IPEEK("177570)          !GET VALUE OF CONSOLE SWITCHES
```

IPOKE

O.3.32 IPOKE

The IPOKE subroutine stores a specified 16-bit integer value into a specified absolute memory location. This subroutine may be used to store values in device registers.

Form: CALL IPOKE (iaddr,ivalue)

where: iaddr is the integer specification of the absolute address to be modified. If this argument is not an even value, a trap will result.

ivalue   is the integer value to be stored in the given address (iaddr).

Errors:

None.

Example:

The following example displays the value of IVAL in the console display register.

```
CALL IPOKE("177570,IVAL)
```

To set bit 12 in the JSW without zeroing any other bits in the JSW, use the following procedure.

```
CALL IPOKE("44,"10000.OR.IPEEK("44))
```

IQSET

O.3.33 IQSET

The IQSET function is used to make the RT-11 queue larger (i.e., add available elements to the queue). These elements are allocated out of the free space managed by the FORTRAN system. IQSET cannot be called from a completion or interrupt routine.

Form: i = IQSET (qleng)

where: qleng   is the integer number of elements to be added to the queue. This argument must be positioned so that the USR does not swap over it.

All RT-11 I/O transfers are done through a centralized queue management system. If I/O traffic is very heavy and not enough queue elements are available, the program issuing the I/O requests may be suspended until a queue element becomes available. In an F/B system, the other job runs while the first program waits for the element. When IQSET is used in a program to be run in the foreground, the FRUN command must be modified to allocate space for the queue elements (see Section O.1.4).

A general rule to follow is that each program should contain one more queue element than the total number of I/O and timer requests that will be active simultaneously. Timing functions such as ITWAIT and MRKT also cause elements to be used and must be considered when allocating queue elements for a program. Note that if synchronous I/O is done (i.e. IREADW/IWRITW, etc.) and no timing functions are done, no additional queue elements need be allocated. See Section 0.2 for a list of SYSLIB calls that use queue elements.

For further information on adding elements to the queue, see the assembly language .QSET request, Section 9.4.27.

Function Results:

i = 0	Normal return.
= 1	Not enough free space is available for the number of queue elements to be added; no allocation was made.

Example:

```
IF(IQSET(5).NE.0) STOP 'NOT ENOUGH FREE SPACE FOR QUEUE ELEMENTS'
```

IRAD50

0.3.34 IRAD50

The IRAD50 function converts a specified number of ASCII characters to Radix-50 and returns the number of characters converted. Conversion stops on the first non-Radix-50 character encountered in the input or when the specified number of ASCII characters have been converted.

Form: n = IRAD50 (icnt,input,output)

where:	icnt	is the number of ASCII characters to be converted.
	input	is the area from which input characters are taken.
	output	is the area into which Radix-50 words are stored.

Three characters of text are packed into each word of output. The number of output words modified is computed by the expression (in integer words):

(icnt+2)/3

Thus, if a count of 4 is specified, two words of output are written even if only a 1-character input string is given as an argument.

Function Results:

The integer number of input characters actually converted (n) is returned as the function result.

Example:

```
REAL*8 FSPEC
CALL IRAD50(12,'SY0TEMP  DAT',FSPEC)
```

IRCVD/IRCVDC/IRCVDF/IRCVDW

O.3.35 IRCVD/IRCVDC/IRCVDF/IRCVDW(F/B only)

There are four forms of the receive data function; these are used in conjunction with the ISDAT (send data) functions to allow a general data/message transfer system. The receive data functions issue RT-11 receive data programmed requests (see Section 9.4.29). These functions require a queue element; this should be considered when the IQSET function (Section O.3.33) is executed.

IRCVD

The IRCVD function is used to receive data and continue execution. The operation is queued and the issuing job continues execution. At some point when the job must receive the transmitted message, an MWAIT should be executed. This causes the job to be suspended until the message has been received.

Form: i = IRCVD (buff,wcnt)

where:

	buff	is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word will contain the integer number of words actually transmitted when IRCVD is complete.
	wcnt	is the maximum integer number of words that can be received.

Errors:

i = 0	Normal return.
= 1	No foreground job exists in the system.

Example:

```
        INTEGER*2 MSG(41)
        .
        .
        .
        CALL IRCVD(MSG,40)
        .
        .
        .
        CALL MWAIT
```

IRCVDC

The IRCVDC function receives data and enters an assembly language completion routine when the message is received. The IRCVDC is queued and program execution stays with the issuing job. When the other job sends a message, the completion routine specified is entered.

Form: i = IRCVDC (buff,wcnt,crtn)

where:

where:	buff	is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word will contain the integer number of words actually transmitted when IRCVDC is complete.
	wcnt	is the maximum integer number of words to be received.
	crtn	is the assembly language completion routine to be entered. This name must be specified in a FORTRAN EXTERNAL statement in the routine that issues the IRCVDC call.

Errors:

i = 0		Normal return.
= 1		No foreground job exists in the system.

IRCVDF

The IRCVDF function receives data and enters a FORTRAN completion subroutine (see Section O.2.1) when the message is received. The IRCVDF is queued and program execution continues with the issuing job. When the other job sends a message, the FORTRAN completion routine specified is entered.

Form: i = IRCVDF (buff,wcnt,area,crtn)

where:	buff	is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word will contain the integer number of words actually transmitted when IRCVDF is complete.

wcnt	is the maximum integer number of words to be received.
area	is a 4-word area to be set aside for linkage information. This area must not be modified by the FORTRAN program and the USR must not swap over it. This area may be reclaimed by other FORTRAN completion routines when crtn has been entered.
crtn	is the FORTRAN completion routine to be entered. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the IRCVDF call.

Errors:

i = 0 Normal return.
= 1 No foreground job exists in the system.

Example:

```
INTEGER*2 MSG(41),AREA(4)
EXTERNAL RMSGRT
.
.
.
CALL IRCVDF(MSG,40,AREA,RMSGRT)
```

IRCVDW

The IRCVDW function is used to receive data and wait. This function queues a message request and suspends the job issuing the request until the other job sends a message. When execution of the issuing job resumes, the message has been received, and the first word of the buffer indicates the number of words which were transmitted.

Form: i = IRCVDW (buff,wcnt)

where:	buff	is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word will contain the integer number of words actually transmitted when IRCVDW is complete.
	wcnt	is the maximum integer number of words to be received.

Errors:

i = 0 Normal return.
= 1 No foreground job exists in the system.

Example:

```
INTEGER*2 MSG(41)
IF(IRCVDW(MSG,40).NE.0) STOP 'UNEXPECTED ERROR'
```

IREAD/IREADC/IREADF/IREADW

0.3.36 IREAD/IREADC/IREADF/IREADW

SYSLIB provides four modes of I/O: IREAD/IWRITE, IREADC/IWRITC, IREADF/IWRITF, and IREADW/IWRITW. Section 0.3.52 explains the output operations. These functions require a queue element; this should be considered when the IQSET function (Section 0.3.33) is executed.

IREAD

The IREAD function transfers a specified number of words from the file associated with the indicated channel into memory. Control returns to the user program immediately after the IREAD function is initiated. No special action is taken when the transfer is completed.

Form: i = IREAD (wcnt,buff,blk,chan)

where:

wcnt	is the integer number of words to be transferred.
buff	is the array to be used as the buffer. This array must contain at least wcnt words.
blk	is the integer block number of the file to be read. The user program normally updates blk before it is used again.
chan	is the integer specification for the RT-11 channel to be used.

NOTE

The "blk" argument must be updated, if necessary, by the user program. For example, if reading two blocks at a time, update blk by 2.

When the user program needs to access the data read on the specified channel, an IWAIT function should be issued. This ensures that the IREAD operation has been completed. If an error occurred during the transfer, the IWAIT function indicates the error.

Errors:

i = n	Normal return; n equals the number of words read (0 for nonfile-structured read, multiple of 256 for file-structured read). For example: If wcnt is a multiple of 256 and less than that number of words remain in the file, n is shortened to the number of words that remain in the file; e.g., if wcnt is 512 and only 256 words remain, i = 256. If wcnt is not a multiple of 256 and more than wcnt words remain in the file, n is rounded up to the next block; e.g., if wcnt is 312 and more than 312 words remain, i = 512, but only 312 are read. If wcnt is not a multiple of 256 and less than wcnt words remain in the file, n equals a multiple of 256 that is the actual number of words being read.
= -1	Attempt to read past end-of-file; no words remain in the file.
= -2	Hardware error occurred on channel.
= -3	Specified channel is not open.

Example:

```
      INTEGER*2 BUFFER(256),RCODE,BLK
      .
      .
      .
      RCODE = IREAD(256,BUFFER,BLK,ICHAN)
      IF(RCODE+1) 1010,1000,10
C     IF NO ERROR, START HERE
10    .
      .
      .
      IF(IWAIT(ICHAN).NE.0) GOTO 1010
      .
      .
      .
1000  CONTINUE
C     END OF FILE PROCESSING
      .
      .
      .
      CALL EXIT           !NORMAL END OF PROGRAM
1010  STOP 'FATAL READ'
      END
```

IREADC

The IREADC function transfers a specified number of words from the indicated channel into memory. Control returns to the user program immediately after the IREADC function is initiated. When the operation is complete, the specified assembly language routine (crtn) is entered as an asynchronous completion routine.

Form: i = IREADC (wcnt,buff,blk,chan,crtn)

where:

	wcnt	is the integer number of words to be transferred.
	buff	is the array to be used as the buffer. This array must contain at least wcnt words.
	blk	is the integer block number of the file to be read. The user program normally updates blk before it is used again.
	chan	is the integer specification for the RT-11 channel to be used.
	crtn	is the assembly language routine to be activated when the transfer is complete. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the IREADC call.

Errors:

See Errors under IREAD.

Example:

```
INTEGER*2 IBUF(256),RCODE,IBLK
EXTERNAL RDCMP
.
.
.
RCODE=IREADC(256,IBUF,IBLK,ICHAN,RDCMP)
```

IREADF

The IREADF function transfers a specified number of words from the indicated channel into memory. Control returns to the user program immediately after the IREADF function is initiated. When the operation is complete, the specified FORTRAN subprogram (crtn) is entered as an asynchronous completion routine (see Section O.2.1).

Form: i = IREADF (wcnt,buff,blk,chan,area,crtn)

where:

	wcnt	is the integer number of words to be transferred.
	buff	is the array to be used as the buffer. This array must contain at least wcnt words.

blk — is the integer block number of the file to be used. The user program normally updates blk before it is used again.

chan — is the integer specification for the RT-11 channel to be used.

area — is a 4-word area to be set aside for linkage information; this area must not be modified by the FORTRAN program or swapped over by the USR. This area may be reclaimed by other FORTRAN completion functions when crtn has been activated.

crtn — is the FORTRAN routine to be activated on completion of the transfer. This name must be specified in an EXTERNAL statement in the routine that issues the IREADF call. The subroutine has two arguments:

SUBROUTINE crtn (iarg1,iarg2)

iarg1 — is the channel status word (see Section 9.4.34) for the operation just completed. If bit 0 is set, a hardware error occurred during the transfer.

iarg2 — is the octal channel number used for the operation just completed.

Errors:

See Errors under IREAD.

Example:

```
        INTEGER*2 DBLK(4),BUFFER(256),BLKNO
        DATA DBLK/3RDX0,3RINP,3RUT ,3RDAT/,BLKNO/0/
        EXTERNAL RCMPLT
        .
        .
        .
        ICHAN=IGETC()
        IF(ICHAN.LT.0) STOP 'NO CHANNEL AVAILABLE'
        IF(IFETCH(DBLK).NE.0) STOP 'BAD FETCH'
        IF(LOOKUP(ICHAN,DBLK).LT.0) STOP 'BAD LOOKUP'
        .
        .
        .
   20   IF(IREADF(256,BUFFER,BLKNO,ICHAN,DBLK,RCMPLT).LT.0) GOTO 100
   C    PERFORM OVERLAP PROCESSING
        .
        .
        .
   C    SYNCHRONIZER
        CALL IWAIT(ICHAN)   !WAIT FOR COMPLETION ROUTINE TO RUN
        BLKNO=BLKNO+1       !UPDATE BLOCK NUMBER
        GOTO 20
        .
        .
        .
```

```
C     END OF FILE PROCESSING
100   CALL CLOSEC(ICHAN)
      CALL IFREEC(ICHAN)
      .
      .
      .
      CALL EXIT
      END
      SUBROUTINE RCMPLT(I,J)
C     THIS IS THE COMPLETION ROUTINE
      .
      .
      .
      RETURN
      END
```

IREADW

The IREADW function transfers a specified number of words from the indicated channel into memory. Control returns to the user program when the transfer is complete or when an error is detected.

Form: i = IREADW (wcnt,buff,blk,chan)

where:

wcnt	is the integer number of words to be transferred.
buff	is the array to be used as the buffer. This array must contain at least wcnt words.
blk	is the integer block number of the file to be read. The user program normally updates blk before it is used again.
chan	is the integer specification for the RT-11 channel to be used.

Errors:

See Errors under IREAD.

Example:

```
      INTEGER*2 IBUF(1024)
      .
      .
      .
      ICODE=IREADW(1024,IBUF,IBLK,ICHAN)
      IF(ICODE.EQ.-1) GOTO 100     !END OF FILE PROCESSING AT 100
      IF(ICODE.LT.-1) GOTO 200     !ERROR PROCESSING AT 200
C
C     MODIFY BLOCKS
C
      .
      .
      .
C
C     WRITE THEM OUT
C
      ICODE=IWRITW(1024,IBUF,IBLK,ICHAN)
```

IRENAM

O.3.37 IRENAM

The IRENAM function causes an immediate change of the name of a specified file. An error occurs if the channel specified is already open.

Form: i = IRENAM (chan,dblk)

where:

chan — is the integer specification for the RT-11 channel to be used for the operation.

dblk — is the 8-word area specifying the name of the existing file and the new name to be assigned. If considered as an 8-element INTEGER*2 array, dblk has the form:

dblk(1)-dblk(4)	specify the Radix-50 file descriptor for the old file name.
dblk(5)-dblk(8)	specify the Radix-50 file descriptor for the new file name.

NOTE

The arguments of IRENAM must be positioned so that the USR does not swap over them.

If a file with the same name as the new file name specified already exists on the indicated device, it is deleted. The specified channel is left closed when the IRENAM is complete. IRENAM requires that the handler to be used be resident at the time the IRENAM is issued. If it is not, a monitor error occurs. The device names specified in the file descriptors must be the same.

For more information on renaming files, see the assembly language .RENAME request, Section 9.4.32.

Errors:

i	= 0	Normal return.
	= 1	Specified channel is already open.
	= 2	Specified file was not found.

Example:

```
        REAL*8 NAME(2)
        DATA NAME/12RDK0FTN2  DAT,12RDK0FTN2  OLD/
        .
        .
        .
        ICHAN=IGETC()
        IF(ICHAN.LT.0) STOP 'NO CHANNEL'
        CALL IRENAM(ICHAN,NAME)     !PRESERVE OLD DATA FILE
        CALL IFREEC(ICHAN)
```

IREOPN

O.3.38 IREOPN

The IREOPN function reassociates a specified channel with a file on which an ISAVES was performed (see Section O.3.39). The ISAVES/IREOPN combination is useful when a large number of files must be operated on at one time. As many files as are needed can be opened with LOOKUP and their status preserved with ISAVES. When data is required from a file, an IREOPN enables the program to read from the file. The IREOPN need not be done on the same channel as the original LOOKUP and ISAVES.

Form: i = IREOPN (chan,cblk)

where:

chan — is the integer specification for the RT-11 channel to be associated with the reopened file. This channel must be initially in an inactive state.

cblk — is the 5-word block where the channel status information was stored by a previous ISAVES. This block, considered as a 5-element INTEGER*2 array, has the following format:

cblk(1)	Channel status word (see Section 9.4.33).
cblk(2)	Starting block number of the file; zero for nonfile-structured devices.
cblk(3)	Length of file (in 256-word blocks).
cblk(4)	(Reserved for future use.)
cblk(5)	Two information bytes. Even byte: I/O count of the number of requests made on this channel. Odd byte: unit number of the device associated with the channel.

Errors:

```
i = 0    Normal return.
  = 1    Specified channel is already in use.
```

Example:

```
      INTEGER*2 SAVES(5,10)
      DATA ISVPTR/1/
      .
      .
      .
      CALL ISAVES(ICHAN,SAVES(1,ISVPTR))
      .
      .
      .
      CALL IREOPN(ICHAN,SAVES(1,ISVPTR))
```

ISAVES

O.3.39 ISAVES

The ISAVES function stores five words of channel status information into a user-specified array. These words contain all the information that RT-11 requires to completely define a file. When an ISAVES is finished, the data words are placed in memory and the specified channel is closed and is again available for use. When the saved channel data is required, the IREOPN function (Section O.3.38) is used.

ISAVES can be used only if a file was opened with a LOOKUP call (see Section O.3.66). If IENTER was used, ISAVES is illegal and returns an error. Note that ISAVES is not legal on magtape or cassette files.

Form: i = ISAVES (chan,cblk)

where:

	chan	is the integer specification for the RT-11 channel whose status is to be saved.
	cblk	is a 5-word block into which the channel status information describing the open file is stored. See Section O.3.38 for the format of this block.

The ISAVES/IREOPN combination is very useful, but care must be observed when using it. In particular, the following cases should be avoided.

1. If an ISAVES is performed on a file and the same file is then deleted before it is reopened, the space occupied by the file becomes available as an empty space which could then be used

by the IENTER function. If this sequence occurs, the contents of the file whose status was supposedly saved will change.

2. Although the handler for the required peripheral need not be in memory for execution of an IREOPN, a fatal error is generated if the handler is not in memory when an IREAD or IWRITE is executed.

Errors:

i = 0	Normal return.
= 1	The specified channel is not currently associated with any file.
= 2	The file was opened with an IENTER call; an ISAVES is illegal.

Example:

```
INTEGER*2 BLK(5)
.
.
.
IF(ISAVES(ICHAN,BLK).NE.0) STOP 'ISAVES ERROR'
```

ISCHED

O.3.40 ISCHED (F/B Only)

The ISCHED function schedules a specified FORTRAN subroutine to be run as an asynchronous completion routine at a specified time-of-day.

Form: i = ISCHED (hrs,min,sec,tick,area,id,crtn)

where:

hrs	is the integer number of hours.
min	is the integer number of minutes.
sec	is the integer number of seconds.
tick	is the integer number of ticks (1/60 of a second on 60-cycle clocks; 1/50 of a second on 50-cycle clocks).
area	is a 4-word-area which must be provided for linkage information; this area must never be modified by the FORTRAN program, and the USR must not swap over it. This area may be reclaimed by other FORTRAN completion functions when crtn has been activated.

id — is the identification integer to be passed to the routine being scheduled.

crtn — is the name of the FORTRAN subroutine to be entered at the time-of-day specified. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISCHED call. The subroutine has one argument, e.g.,

```
SUBROUTINE crtn(id)
INTEGER id
```

When the routine is entered, the value of the integer argument will be the value specified for id in the appropriate ISCHED call.

Notes:

1. The scheduling request made by this function may be cancelled at a later time by an ICMKT function call.

2. If the system is busy, the actual time-of-day that the completion routine is run may be greater than the requested time-of-day.

3. A FORTRAN subroutine may periodically reschedule itself by issuing its own ISCHED or ITIMER calls from within the routine.

4. ISCHED requires a queue element; this should be considered when the IQSET function (Section O.3.33) is executed.

Errors:

i = 0	Normal return.
= 1	No queue elements available; unable to schedule request.

Example:

```
      INTEGER*2 LINK(4)              !LINKAGE AREA
      EXTERNAL NOON                  !NAME OF ROUTINE TO RUN
      .
      .
      .
      I=ISCHED(12,0,0,0,LINK,0,NOON) !RUN SUBR NOON AT 12 PM
      .
      .   (rest of main program)
      .
      END
      SUBROUTINE NOON(ID)
C
C     THIS ROUTINE WILL TERMINATE EXECUTION AT LUNCHTIME,
C     IF THE JOB HAS NOT COMPLETED BY THAT TIME.
C
      STOP     'ABORT JOB -- LUNCHTIME'
      END
```

ISDAT/ISDATC/ISDATF/ISDATW

O.3.41 ISDAT/ISDATC/ISDATF/ISDATW (F/B only)

These functions are used with the IRCVD/IRCVDC/IRCVDF, and IRCVDW calls to allow message transfers under the F/B Monitor. Note that the buffer containing the message should not be modified or reused until the message has been received by the other job. These functions require a queue element; this should be considered when the IQSET function (Section O.3.33) is executed.

ISDAT

The ISDAT function transfers a specified number of words from one job to the other. Control returns to the user program immediately after the transfer is queued. This call is used with the MWAIT routine (see Section O.3.68).

Form: i = ISDAT (buff,wcnt)

where:

	buff	is the array containing the data to be transferred.
	wcnt	is the integer number of data words to be transferred.

Errors:

i = 0	Normal return.
= 1	No foreground job currently exists in the system.

Example:

```
      INTEGER*2 MSG(40)
      .
      .
      .
      CALL ISDAT(MSG,40)
      .
      .
      .
      CALL MWAIT
C     PUT NEW MESSAGE IN BUFFER
```

ISDATC

The ISDATC function transfers a specified number of words from one job to another. Control returns to the user program immediately after the transfer is queued. When the other job accepts the message through a receive data request, the specified assembly language routine (crtn) is activated as an asynchronous completion routine.

Form: i = ISDATC (buff,wcnt,crtn)

where:

buff	is the array containing the data to be transferred.
wcnt	is the integer number of data words to be transferred.
crtn	is the name of an assembly language routine to be activated on completion of the transfer. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISDATC call.

Errors:

i = 0	Normal return.
= 1	No foreground job currently exists in the system.

Example:

```
      INTEGER*2 MSG(40)
      EXTERNAL RTN
      .
      .
      .
      CALL ISDATC(MSG,40,RTN)
```

ISDATF

The ISDATF function transfers a specified number of words from one job to the other. Control returns to the user program immediately after the transfer is queued and execution continues. When the other job accepts the message through a receive data request, the specified FORTRAN subprogram (crtn) is activated as an asynchronous completion routine (see Section O.2.1).

Form: i = ISDATF (buff,wcnt,area,crtn)

where:

buff	is the array containing the data to be transferred.
wcnt	is the integer number of data words to be transferred.
area	is a 4-word area to be set aside for linkage information; this area must not be modified by the FORTRAN program and the USR must not swap over it. This area may be reclaimed by other FORTRAN completion functions when crtn has been activated.

crtn — is the name of a FORTRAN routine to be activated on completion of the transfer. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISDATF call.

Errors:

i = 0 Normal return.
= 1 No foreground job currently exists in the system.

Example:

```
INTEGER*2 MSG(40),SPOT(4)
EXTERNAL RTN
.
.
.
CALL ISDATF(MSG,40,SPOT,RTN)
```

ISDATW

The ISDATW function transfers a specified number of words from one job to the other. Control returns to the user program when the other job has accepted the data through a receive data request.

Form: i = ISDATW (buff,wcnt)

where:

buff — is the array containing the data to be transferred.

wcnt — is the integer number of data words to be transferred.

Errors:

i = 0 Normal return.
= 1 No foreground job currently exists in the system.

Example:

```
INTEGER*2 MSG(40)
.
.
.
IF (ISDATW(MSG,40).NE.0) STOP 'FOREGROUND JOB NOT RUNNING'
```

ISLEEP

0.3.42 ISLEEP (F/B Only)

The ISLEEP function suspends the main program execution of a job for a specified amount of time. The specified time is the sum of hours, minutes, seconds, and ticks specified in the ISLEEP call. All completion routines continue to execute.

Form: i = ISLEEP (hrs,min,sec,tick)

where:

- hrs — is the integer number of hours.
- min — is the integer number of minutes.
- sec — is the integer number of seconds.
- tick — is the integer number of ticks (1/60 of a second on 60-cycle clocks; 1/50 of a second on 50-cycle clocks).

Notes:

1. ISLEEP requires a queue element; this should be considered when the IQSET function (Section 0.3.33) is executed.
2. If the system is busy, the time that execution is suspended may be greater than that specified.

Errors:

i = 0	Normal return.
= 1	No queue element available.

Example:

```
        .
        .
        .
        CALL IQSET(2)
        .
        .
        .
        CALL ISLEEP(0,0,0,4)       !GIVE BACKGROUND JOB SOME TIME
```

ISPFN/ISPFNC/ISPFNF/ISPFNW

O.3.43 ISPFN/ISPFNC/ISPFNF/ISPFNW

These functions are used in conjunction with special functions to various handlers (cassette, diskette, and magtape). They provide a means of doing device-dependent functions, such as rewind and backspace, to those devices. If ISPFN function calls are made to any other devices, the function call is ignored.

To use these functions, the handler must be in memory and a channel associated with a file via a non-file structured LOOKUP call. These functions require a queue element; this should be considered when the IQSET function (Section O.3.33) is executed. Refer to Appendix H for details of CT, DX, and MT handlers.

ISPFN

The ISPFN function queues the specified operation and immediately returns control to the user program. The IWAIT function may be used to ensure completion of the operation.

Form: i = ISPFN (code,chan{,wcnt,buff,blk})

where:

code	is the integer numeric code of the function to be performed (see Table O-2).
chan	is the integer specification for the RT-11 channel to be used for the operation.
wcnt	is the integer number of data words in the operation.[1]
buff	is the array to be used as the data buffer.[1]
blk	is the integer block number of the file to be operated upon.[1]

[1]These optional parameters are required only when doing a write with extended record gap to MT or a read or write to DX. If specified and not required, these arguments must be 0.

Table O-2
Special Function Codes (Octal)

Function	MT	CT	DX
Read absolute			377
Write absolute			376
Write absolute with deleted data			375
Backspace to last file		377	
Backspace to last block		376	
Forward to next file		375	
Forward to next block		374	
Rewind to load point	373	373	
Write file gap		372	
Write EOF	377		
Forward 1 record	376		
Backspace 1 record	375		
Write with extended record gap	374		
Offline	372		

Errors:

i	= 0	Normal return.
	= 1	Attempt to read or write past end-of-file.
	= 2	Hardware error occurred on channel.
	= 3	Channel specified is not open.

Example:

```
CALL ISPFN("373,ICHAN)     !REWIND
```

ISPFNC

The ISPFNC function queues the specified operation and immediately returns control to the user program. When the operation is complete, the specified assembly language routine (crtn) is entered as an asynchronous completion routine.

Form: i = ISPFNC (code,chan,wcnt,buff,blk,crtn)

where:

code — is the integer numeric code of the function to be performed (see Table O-2).

chan — is the integer specification for the RT-11 channel to be used for the operation.

wcnt — is the integer number of data words in the operation. This argument must be 0 if not required.

buff — is the array to be used as the data buffer. This argument must be 0 if not required.

blk — is the integer block number of the file to be operated upon. This argument must be 0 if not required.

crtn — is the name of an assembly language routine to be activated on completion of the operation. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISPFNC call.

Errors:

i = 0	Normal return.
= 1	Attempt to read or write past end-of-file.
= 2	Hardware error occurred on channel.
= 3	Channel specified is not open.

ISPFNF

The ISPFNF function queues the specified operation and immediately returns control to the user program. When the operation is complete, the specified FORTRAN subprogram (crtn) is entered as an asynchronous completion routine.

Form: i = ISPFNF (code,chan,wcnt,buff,blk,area,crtn)

where:

code — is the integer numeric code of the function to be performed (see Table O-2).

chan — is the integer specification for the RT-11 channel to be used for the operation.

wcnt — is the integer number of data words in the operation. This argument must be 0 if not required.

buff — is the array to be used as the data buffer. This argument must be 0 if not required.

blk — is the integer block number of the file to be operated upon. This argument must be 0 if not required.

area — is a 4-word area to be set aside for linkage information; this area must not be modified by the FORTRAN program and the USR must not swap over it. This area may be reclaimed by other FORTRAN completion functions when crtn has been activated.

crtn — is the name of a FORTRAN routine to be activated on completion of the operation. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISPFNF call. The subroutine has two arguments:

SUBROUTINE crtn (iarg1,iarg2)

iarg1 — is the channel status word (see Section 9.4.34) for the operation just completed. If bit 0 is set, a hardware error occurred during the transfer.

iarg2 — is the channel number used for the operation just completed.

Errors:

```
i = 0    Normal return.
  = 1    Attempt to read or write past end-of-file.
  = 2    Hardware error occurred on channel.
  = 3    Channel specified is not open.
```

Example:

```
          REAL*4 MTNAME(2),AREA(2)
          DATA MTNAME/3RMT0,0./
          EXTERNAL DONSUB
          .
          .
          .
          I=IGETC()                  !ALLOCATE CHANNEL
          CALL IFETCH(MTNAME)        !FETCH MT HANDLER
          CALL LOOKUP(I,MTNAME)      !NON-FILE STRUCTURED LOOKUP ON MT0
          IERR=ISPFNF("373,I,0,0,0,AREA,DONSUB)    !REWIND MAGTAPE
          .
          .
          .
          END
          SUBROUTINE DONSUB
    C
    C     RUNS WHEN MT0 HAS BEEN REWOUND
    C
          .
          .
          .
          END
```

ISPFNW

The ISPFNW function queues the specified operation and returns control to the user program when the operation is complete.

Form: i = ISPFNW (code,chan{,wcnt,buff,blk})

where:

- code — is the integer numeric code of the function to be performed (see Table O-2).
- chan — is the integer specification for the RT-11 channel to be used for the operation.
- wcnt — is the integer number of data words in the operation.[1]
- buff — is the array to be used as the data buffer.[1]
- blk — is the integer block number of the file to be operated upon.[1]

---

[1]These optional parameters are required only when doing a write with extended record gap to MT or a read or write to DX. If specified and not required, these arguments must be 0.

Errors:

```
i = 0    Normal return.
  = 1    Attempt to read or write past end-of-file.
  = 2    Hardware error occurred on channel.
  = 3    Channel specified is not open.
```

Example:

```
      INTEGER*2 BUF(65),TRACK,SECTOR,DBLK(4)
      DATA DBLK/3RDX0,0,0,0/
      .
      .
      .
      ICHAN=IGETC()
      IF(ICHAN.LT.0) STOP 'NO CHANNEL AVAILABLE'
      IF(LOOKUP(ICHAN,DBLK).LT.0) STOP 'BAD LOOKUP'
      .
      .
      .
C     READ AN ABSOLUTE TRACK AND SECTOR FROM THE FLOPPY
C
      ICODE=ISPFNW("377,ICHAN,TRACK,BUF,SECTOR)
C
C     BUF(1) IS THE DELETED DATA FLAG
C     BUF(2-65) IS THE DATA
```

ISPY

O.3.44 ISPY

The ISPY function returns the integer value of the word at a specified offset from the RT-11 resident monitor. (See Section 9.2.6 for information on fixed offset references.)

Form: i = ISPY (ioff)

where: ioff is the offset (from the base of RMON) to be examined.

Function Result:

The function result (i) is set to the value of the word examined.

Example:

```
C
C     BRANCH TO 200 IF RUNNING UNDER F/B MONITOR
C
      IF(ISPY("300).AND.1) GOTO 200
C
C     WORD AT OCTAL 300 FROM RMON IS
C     THE CONFIGURATION WORD.
```

0.3.45 ITIMER (F/B Only)

The ITIMER function schedules a specified FORTRAN subroutine to be run as an asynchronous completion routine after a specified time interval has elapsed.

Form: i = ITIMER (hrs,min,sec,tick,area,id,crtn)

where:

hrs	is the integer number of hours.
min	is the integer number of minutes
sec	is the integer number of seconds.
tick	is the integer number of ticks (1/60 of a second on 60-cycle clocks; 1/50 of a second on 50-cycle clocks).
area	is a 4-word area which must be provided for linkage information; this area must never be modified by the FORTRAN program, and the USR must never swap over it. This area may be reclaimed by other FORTRAN completion functions when crtn has been activated.
id	is the identification integer to be passed to the routine being scheduled.
crtn	is the name of the FORTRAN subroutine to be entered when the specified time interval elapses. This name must be specified in an EXTERNAL statement in the FORTRAN routine that references ITIMER. The subroutine has one argument, e.g.,

```
SUBROUTINE crtn(id)
INTEGER id
```

When the routine is entered, the value of the integer argument will be the value specified for id in the appropriate ITIMER call.

Notes:

1. This function may be cancelled at a later time by an ICMKT function call.

2. If the system is busy, the actual time interval at which the completion routine is run may be greater than the time interval requested.

3. FORTRAN subroutines can periodically re-schedule themselves by issuing ISCHED or ITIMER calls.

4. ITIMER requires a queue element; this should be considered when the IQSET function (Section O.3.33) is executed.

For more information on scheduling completion routines, see Section O.2.1 and the assembly language .MRKT request, Section 9.4.22.

Errors:

i = 0		Normal return
	= 1	No queue elements available; unable to schedule request.

Example:

```
      INTEGER*2 AREA(4)
      EXTERNAL WATCHD
      .
      .
      .
C     IF THE CODE FOLLOWING ITIMER DOES NOT REACH THE ICMKT CALL
C     IN 12 MINUTES, WATCH DOG COMPLETION ROUTINE WILL BE
C     ENTERED WITH ID OF 3
C
      CALL ITIMER(0,12,0,0,AREA,3,WATCHD)
      .
      .
      .
      CALL ICMKT(3,AREA)
      .
      .
      .
      END
      SUBROUTINE WATCHD(ID)
C
C     THIS IS CALLED AFTER 12 MINUTES
      .
      .
      .
      RETURN
      END
```

ITLOCK

0.3.46 ITLOCK (F/B Only)

The ITLOCK function is used in an F/B system to attempt to gain ownership of the USR. It is similiar to LOCK (Section 0.3.65) in that if successful, the user job returns with the USR in memory. However, if a job attempts to LOCK the USR while the other job is using it, the requesting job is suspended until the USR is free. With ITLOCK, if the USR is not available, control returns immediately and the lock failure is indicated. ITLOCK cannot be called from a completion or interrupt routine.

Form: i = ITLOCK()

For further information on gaining ownership of the USR, see the assembly language .TLOCK request, Section 9.4.41.

Errors:

```
     i = 0    Normal return.
       = 1    USR is already in use by another job.
```

Example:

```
          IF(ITLOCK().NE.0) GOTO 100     !GOTO 100 IF USR BUSY
```

ITTINR

0.3.47 ITTINR

The ITTINR function transfers a character from the console terminal to the user program. If no characters are available, return is made with an error flag set.

Form: i = ITTINR()

If the function result (i) is less than zero when execution of the ITTINR function is complete, it indicates that no character was available; the user has not yet typed a valid line. Under the F/B Monitor, ITTINR does not return a result of less than zero unless bit 6 of the Job Status Word was on when the request was issued (see below).

There are two modes of doing console terminal input. The mode is governed by bit 12 of the Job Status Word (JSW). The JSW is at octal location 44. If bit 12 equals 0, normal I/O is performed. In this mode, the following conditions apply:

1. The monitor echoes all characters typed; lower case characters are converted to upper case unless JSW bit 14 is set, in which case lower case characters are not translated.
2. CTRL U (^U) AND RUBOUT perform line deletion and character deletion, respectively.
3. A carriage return, line feed, CTRL Z, or CTRL C must be struck before characters on the current line are available to the program. When one of these is typed, characters on the line typed are passed one-by-one to the user program. Both carriage return and line feed are passed to the program.
4. ALTMODEs (octal codes 175 and 176) are converted to ESCAPEs (octal 33).

If bit 12 equals 1, the console is in special mode. The effects are:

1. The monitor does not echo characters typed except for CTRL C and CTRL O.
2. CTRL U and RUBOUT do not perform special functions.
3. Characters are immediately available to the program.
4. No ALTMODE conversion is done.

In special mode, the user program must echo the characters desired. However, CTRL C and CTRL O are acted on by the monitor in the usual way. Bits 12 and 14 in the JSW must be set by the user program if special console mode or lower case characters are desired (see the example under Section O.3.32). These bits are cleared when control returns to RT-11.

NOTE

> To set and/or clear bits in the JSW, do an IPEEK and then an IPOKE. In special terminal mode (JSW bit 12 set), normal FORTRAN formatted I/O from the console is undefined.

In the F/B Monitor, CTRL F and CTRL B are not affected by the setting of bit 12. The monitor always acts on these characters if the SET TTY FB command was issued.

Under the F/B Monitor, if a terminal input request is made and no character is available, job execution is suspended until a character is ready. If a program really requires execution to continue and ITTINR to return a result of less than zero, it must turn on bit 6 of the JSW before the ITTINR. Bit 6 is cleared when a program terminates.

Function Results:

i	>0	Normal return; character read.
	<0	Error return; no character available.

Example:

```
ICHAR=ITTINR()          !READ A CHARACTER FROM THE CONSOLE
IF(ICHAR.LT.0) GOTO 100 !CHARACTER NOT AVAILABLE
```

ITTOUR

O.3.48 ITTOUR

The ITTOUR function transfers a character from the user program to the console terminal if there is room for the character in the monitor buffer. If it is not currently possible to output a character, an error flag is returned.

Form: i = ITTOUR (char)

where: char is the character to be output, right-justified in the integer (may be LOGICAL*1 entity if desired).

If the function result (i) is 1 when execution of the ITTOUR function is complete, it indicates that there is no room in the buffer and that no character was output. Under the F/B Monitor, ITTOUR normally does not return a result of 1. Instead, the job is blocked until room is available in the output buffer. If a job really requires execution to continue and a result of 1 to be returned, it must turn on bit 6 of the JSW (location 44) before issuing the request.

The ITTINR and ITTOUR have been supplied as a help to those users who do not wish to suspend program execution until a console operation is complete. With these modes of I/O, if a no-character or no-room condition occurs, the user program can continue processing and try the operation again at a later time.

Errors:

i	= 0	Normal return; character was output.
	= 1	Error return; ring buffer is full.

Example:

```
        DO 20 I=1,5
10      IF(ITTOUR("007).NE.0) GOTO 10       !RING BELL 5 TIMES
20      CONTINUE
```

ITWAIT

O.3.49 ITWAIT (F/B Only)

The ITWAIT function suspends the main program execution of the current job for a specified time interval. All completion routines continue to execute.

Form: i = ITWAIT (itime)

where: itime is the 2-word internal format time interval.

itime (1) is the high-order time.
itime (2) is the low-order time.

Notes:

1. ITWAIT requires a queue element; this should be considered when the IQSET function (Section O.3.33) is executed.

2. If the system is busy, the actual time interval that execution is suspended may be greater than the time interval specified.

Errors:

i = 0	Normal return.
= 1	No queue element available.

Example:

```
        INTEGER*2 TIME(2)
        .
        .
        .
        CALL ITWAIT(TIME)     !WAIT FOR TIME TIME
```

IUNTIL

O.3.50 IUNTIL (F/B Only)

The IUNTIL function suspends main program execution of the job until the time-of-day specified. All completion routines continue to run.

Form: i = IUNTIL (hrs,min,sec,tick)

where:

hrs	is the integer number of hours.
min	is the integer number of minutes.
sec	is the integer number of seconds.
tick	is the integer number of ticks (1/60 of a second on 60-cycle clocks; 1/50 of a second on 50-cycle clocks).

Notes:

1. IUNTIL requires a queue element; this should be considered when the IQSET function (Section O.3.33) is executed.

2. If the system is busy, the actual time-of-day that the program resumes execution may be later than that requested.

Errors:

i = 0	Normal return.
= 1	No queue element available.

Example:

```
C     TAKE A LUNCH BREAK
      CALL IUNTIL(13,0,0,0)      !START UP AGAIN AT 1 P.M.
```

IWAIT

0.3.51 IWAIT

The IWAIT function suspends execution of the main program until all input/output operations on the specified channel are complete. This function is used with IREAD, IWRITE, and ISPFN calls. Completion routines continue to execute.

Form: i = IWAIT (chan)

where: chan is the integer specification for the RT-11 channel to be used.

For further information on suspending execution of the main program, see the assembly language .WAIT request, Section 9.4.46.

Errors:

i = 0	Normal return.
= 1	Channel specified is not open.
= 2	Hardware error occurred during the previous I/O operation on this channel.

Example:

```
IF(IWAIT(ICHAN).NE.0) CALL IOERR(4)
```

IWRITC/IWRITE/IWRITF/IWRITW

0.3.52 IWRITC/IWRITE/IWRITF/IWRITW

These functions transfer a specified number of words from memory to the specified channel. The IWRITE functions require queue elements; this should be considered when the IQSET function (Section 0.3.33) is executed.

IWRITC

The IWRITC function transfers a specified number of words from memory to the specified channel. The request is queued and control returns to the user program. When the transfer is complete, the specified assembly language routine (crtn) is entered as an asynchronous completion routine.

Form: i = IWRITC (wcnt,buff,blk,chan,crtn)

where:

wcnt	is the integer number of words to be transferred.
buff	is the array to be used as the output buffer.
blk	is the integer block number of the file to be written. The user program normally updates blk before it is used again.
chan	is the integer specification for the RT-11 channel to be used.
crtn	is the name of the assembly language routine to be activated upon completion of the transfer. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the IWRITC call.

NOTE

The "blk" argument must be updated, if necessary, by the user program. For example, if writing two blocks at a time, update blk by 2.

Errors:

i = n	Normal return; n equals the number of words written, rounded to a multiple of 256 (0 for nonfile-structured writes).

NOTE

If the word count returned is less than that requested, an implied end-of-file has occurred although the normal return is indicated.

= -1	Attempt to write past end-of-file; no more space is available in the file.
= -2	Hardware error occurred.
= -3	Channel specified is not open.

Example:

```
        INTEGER*2 IBUF(256)
        EXTERNAL CRTN
        .
        .
        .
        ICODE=IWRITC(256,IBUF,IBLK,ICHAN,CRTN)
```

IWRITE

The IWRITE function transfers a specified number of words from memory to the specified channel. Control returns to the user program immediately after the request is queued. No special action is taken upon completion of the operation.

Form: i = IWRITE (wcnt,buff,blk,chan)

where:

	wcnt	is the integer number of words to be transferred.
	buff	is the array to be used as the output buffer.
	blk	is the integer block number of the file to be written. The user program normally updates blk before it is used again.
	chan	is the integer specification for the RT-11 channel to be used.

Errors:

See Errors under IWRITC.

Example:

See the example under IREAD, Section O.3.36.

IWRITF

The IWRITF function transfers a number of words from memory to the specified channel. The transfer request is queued and control returns to the user program. When the operation is complete, the specified FORTRAN subprogram (crtn) is entered as an asynchronous completion routine (see Section O.2.1).

Form: i = IWRITF (wcnt,buff,blk,chan,area,crtn)

where:

	wcnt	is the integer number of words to be transferred.
	buff	is the array to be used as the output buffer.
	blk	is the integer block number of the file to be written. The user program normally updates blk before it is used again.
	chan	is the integer specification for the RT-11 channel to be used.

area — is a 4-word area to be set aside for linkage information; this area must not be modified by the FORTRAN program and the USR must not swap over it. This area may be reclaimed by other FORTRAN completion functions when crtn has been activated.

crtn — is the name of the FORTRAN routine to be activated upon completion of the transfer. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the IWRITF call. The subroutine has two arguments:

SUBROUTINE crtn (iarg1,iarg2)

iarg1 — is the channel status word (see Section 9.4.34) for the operation just completed. If bit 0 is set, a hardware error occurred during the transfer.

iarg2 — is the channel number used for the operation just completed.

Errors:

See Errors under IWRITC.

Example:

See the example under IREADF, Section O.3.36.

IWRITW

The IWRITW function transfers a specified number of words from memory to the specified channel. Control returns to the user program when the transfer is complete.

Form: i = IWRITW (wcnt,buff,blk,chan)

where:

wcnt — is the integer number of words to be transferred.

buff — is the array to be used as the output buffer.

blk — is the integer block number of the file to be written. The user program normally updates blk before it is used again.

chan — is the integer specification for the RT-11 channel to be used.

Errors:

See Errors under IWRITC.

Example:

See the example under IREADW, Section O.3.36.

JADD

0.3.53 JADD

The JADD function computes the sum of two INTEGER*4 values.

Form: i = JADD (jopr1,jopr2,jres)

where:

where:	jopr1	is an INTEGER*4 variable.
	jopr2	is an INTEGER*4 variable.
	jres	is an INTEGER*4 variable that receives the sum of jopr1 and jopr2.

Function Results:

i = -2	An overflow occurred while computing the result.
= -1	Normal return; the result is negative.
= 0	Normal return; the result is zero.
= 1	Normal return; the result is positive.

Example:

```
INTEGER*4 JOP1,JOP2,JRES
.
.
.
IF(JADD(JOP1,JOP2,JRES).EQ.-2) GOTO 100
```

JAFIX

0.3.54 JAFIX

The JAFIX function converts a REAL*4 value to INTEGER*4.

Form: i = JAFIX (asrc,jres)

where:	asrc	is a REAL*4 variable, constant, or expression to be converted to INTEGER*4.
	jres	is an INTEGER*4 variable that is to contain the result of the conversion.

Function Results:

i = -2	An overflow occurred while computing the result.
= -1	Normal return; the result is negative.
= 0	Normal return; the result is zero.
= 1	Normal return; the result is positive.

Example:

```
      INTEGER*4 JOP1
C     READ A LARGE INTEGER FROM THE TERMINAL
      ACCEPT 99,A
99    FORMAT (F15.0)
      IF(JAFIX(A,JOP1).EQ.-2) GOTO 100
      .
      .
      .
```

JCMP

O.3.55 JCMP

The JCMP function compares two INTEGER*4 values and returns an INTEGER*2 value that reflects the signed comparison result.

Form: i = JCMP (jopr1,jopr2)

where:	jopr1	is the INTEGER*4 variable or array element that is the first operand in the comparison.
	jopr2	is the INTEGER*4 variable or array element that is the second operand in the comparison.

Function Result:

i = -1	If jopr1 is less than jopr2
= 0	If jopr1 is equal to jopr2
= 1	If jopr1 is greater than jopr2

Example:

```
      INTEGER*4 JOPX,JOPY
      .
      .
      .
      IF(JCMP(JOPX,JOPY)) 10,20,30
```

JDFIX

0.3.56 JDFIX

The JDFIX function converts a REAL*8 (DOUBLE PRECISION) value to INTEGER*4.

Form: i = JDFIX (dsrc,jres)

where: dsrc is a REAL*8 variable, constant, or expression to be converted to INTEGER*4.

jres is an INTEGER*4 variable to contain the conversion result.

Function Results:

i = -2	An overflow occurred while computing the result.
= -1	Normal return; the result is negative.
= 0	Normal return; the result is zero.
= 1	Normal return; the result is positive.

Example:

```
      INTEGER*4 JNUM
      REAL*8 DPNUM
      .
      .
      .
20    TYPE 98
98    FORMAT(' ENTER POSITIVE INTEGER')
      ACCEPT 99,DPNUM
99    FORMAT(F20.0)
      IF(JDFIX(DPNUM,JNUM).LT.0) GOTO 20
      .
      .
      .
```

JDIV

0.3.57 JDIV

The JDIV function computes the quotient of two INTEGER*4 values.

Form: i = JDIV (joprl,jopr2,jres{,jrem})

where:	joprl	is an INTEGER*4 variable that is the dividend of the operation.
	jopr2	is an INTEGER*4 variable that is divisor of joprl.
	jres	is an INTEGER*4 variable that receives the quotient of the operation (i.e., jres=joprl/jopr2).
	jrem	is an INTEGER*4 variable that receives the remainder of the operation. The sign is the same as that for joprl.

Function Results:

i = -3	An attempt was made to divide by zero.
= -2	(not used)
= -1	Normal return; the quotient is negative.
= 0	Normal return; the quotient is zero.
= 1	Normal return; the quotient is positive.

Example:

```
INTEGER*4 JN1,JN2,JQUO
.
.
.
CALL JDIV(JN1,JN2,JQUO)
.
.
.
```

JICVT

0.3.58 JICVT

The JICVT function converts a specified INTEGER*2 value to INTEGER*4.

Form: i = JICVT (isrc,jres)

Where: isrc is the INTEGER*2 quantity to be converted.

jres is the INTEGER*4 variable or array element which will receive the result.

Function Results:

i = -1	Normal return;	the result is negative.
= 0	Normal return;	the result is zero.
= 1	Normal return;	the result is positive.

Example:

```
INTEGER*4 JVAL
CALL JICVT(478,JVAL)      !FORM A 32-BIT CONSTANT
```

JJCVT

0.3.59 JJCVT

The JJCVT function interchanges words of an INTEGER*4 value to form an internal format time or vice versa. This is necessary when the INTEGER*4 variable is to be used as an argument in a timer-support function such as ITWAIT. When a 2-word internal format time is specified to a function such as ITWAIT, it must have the high-order time as the first word and the low-order time as the second word.

Form: CALL JJCVT (jsrc)

where: jsrc is the INTEGER*4 variable whose contents are to be interchanged.

Errors:

None.

Example:

```
INTEGER*4 TIME
.
.
.
CALL GTIM(TIME)       !GET TIME OF DAY
CALL JJCVT(TIME)      !TURN IT INTO INTEGER*4 FORMAT
```

JMOV

O.3.60 JMOV

The JMOV function assigns the value of an INTEGER*4 variable to another INTEGER*4 variable and returns the sign of the value moved.

Form: i = JMOV (jsrc,jdest)

where: jsrc is the INTEGER*4 variable whose contents are to be moved.

jdest is the INTEGER*4 variable which is the target of the assignment.

Function Result:

The value of the function is an INTEGER*2 value which represents the sign of the result as follows:

i = -1	Normal return;	the result is negative.
= 0	Normal return;	the result is zero.
= 1	Normal return;	the result is positive.

Example:

The JMOV function allows an INTEGER*4 quantity to be compared with zero by using it in a logical IF statement, e.g.,

```
INTEGER*4 INT1
.
.
.
IF(JMOV(INT1,INT1)) 300,100,300 !GO TO STMT 300 IF INT1 IS NOT 0
```

JMUL

0.3.61 JMUL

The JMUL function computes the product of two INTEGER*4 values.

Form: i = JMUL (jopr1,jopr2,jres)

where

jopr1	is an INTEGER*4 variable that is the multiplicand.
jopr2	is an INTEGER*4 variable that is the multiplier.
jres	is an INTEGER*4 variable that receives the product of the operation.

Function Results:

i = -2	An overflow occurred while computing the result.
= -1	Normal return; the product is negative.
= 0	Normal return; the product is zero.
= 1	Normal return; the product is positive.

Example:

```
      INTEGER*4 J1,J2,JRES
      .
      .
      .
      IF(JMUL(J1,J2,JRES)+1) 100,10,20
C     GOTO 100 IF OVERFLOW
C     GOTO 10 IF RESULT IS NEGATIVE
C     GOTO 20 IF RESULT IS POSITIVE OR ZERO
```

JSUB

0.3.62 JSUB

The JSUB function computes the difference between two INTEGER*4 values.

Form: i = JSUB (jopr1,jopr2,jres)

where:

jopr1	is an INTEGER*4 variable that is the minuend of the operation.
jopr2	is an INTEGER*4 variable that is the subtrahend of the operation.
jres	is an INTEGER*4 variable that is to receive the difference between iopr1 and iopr2 (i.e., jres=jopr1-jopr2).

Function Results:

i = -2	An overflow occurred while computing the result.
= -1	Normal return; the result is negative.
= 0	Normal return; the result is zero.
= 1	Normal return; the result is positive.

Example:

```
INTEGER*4 JOP1,JOP2,J3
.
.
.
CALL JSUB(JOP1,JOP2,J3)
```

JTIME

0.3.63 JTIME

The JTIME subroutine converts the time specified to the internal 2-word format time.

Form: CALL JTIME (hrs,min,sec,tick,time)

where:

hrs	is the integer number of hours.
min	is the integer number of minutes.
sec	is the integer number of seconds.
tick	is the integer number of ticks (1/60 of a second for 60-cycle clocks; 1/50 of a second for 50-cycle clocks).
time	is the 2-word area to receive the internal format time: time(1) is the high-order time. time(2) is the low-order time.

Errors:

None.

Example:

```
      INTEGER*4 J1
      .
      .
      .
C     CONVERT 3 HRS, 7 MIN, 23 SECONDS TO INTEGER *4 VALUE
      CALL JTIME(3,7,23,0,J1)
      CALL JJCVT(J1)
```

LEN

O.3.64 LEN

The LEN function returns the number of characters currently in the string contained in a specified array. This number is computed as the number of characters preceding the first null byte encountered. If the specified array contains a null string, a value of zero is returned.

Form: i = LEN (a)

where: a specifies the array containing the string: a must not be a quoted-string literal.

Errors:

None.

Example:

```
        LOGICAL*1 STRNG(73)
        .
        .
        .
        TYPE 99,(STRNG(I),I=1,LEN(STRNG))
   99   FORMAT('0',132A1)
```

LOCK

O.3.65 LOCK

The LOCK subroutine is issued to "lock" the USR in memory for a series of operations. The USR (User Service Routine) is the section of the RT-11 system which performs various file management functions.

If all the conditions which cause swapping are satisfied, a portion of the user program is written into scratch blocks and the USR is loaded. Otherwise, the USR which is in memory is used, and no swapping occurs. The USR is not released until an UNLOCK (see Section O.3.88) is given. (Note that in an F/B System, calling the CSI may also perform an implicit UNLOCK.) A program which has many USR requests to make can LOCK the USR in memory, make all the requests, and then UNLOCK the USR; no time is spent doing unnecessary swapping.

In an F/B environment, a LOCK inhibits the other job from using the USR. Thus, the USR should be locked only as long as necessary.

Form: CALL LOCK

Note that the LOCK routine reduces time spent in file handling by eliminating the swapping of the USR. If the USR is currently resident, LOCK involves no I/O. After a LOCK has been executed, the UNLOCK routine must be executed to release the USR from memory. The LOCK/UNLOCK routines are complementary and must be matched. That is, if three LOCKs are issued, at least three UNLOCKs must be done, otherwise the USR will not be released. More UNLOCKs than LOCKs may occur without error; the extra UNLOCKs are ignored.

Notes:

1. It is vital that the LOCK call not come from within the area into which the USR will be swapped. If this should occur, the return from the USR request would not be to the user program, but to the USR itself, since the LOCK function causes part of the user program to be saved on disk and replaced in memory by the USR. Furthermore, subroutines, variables, and arrays in the area where the USR is swapping should not be referenced while the USR is LOCKed in memory.

2. Once a LOCK has been performed, it is not advisable for the program to destroy the area the USR is in, even though no further use of the USR is required. This causes unpredictable results when an UNLOCK is done.

3. LOCK cannot be called from a completion or interrupt routine.

4. If a SET USR NO SWAP command has been issued, LOCK and UNLOCK will not cause the USR to swap but, in F/B, LOCK will still inhibit the other job from using the USR and UNLOCK will allow the other job access to the USR.

5. Argument lists, such as device file name specifications, being passed to the USR must not be in the area into which the USR has just been locked.

Errors:

None.

LOOKUP

0.3.66 LOOKUP

The LOOKUP function associates a specified channel with a device and/or file for the purpose of performing I/O operations. The channel used is then busy until one of the following functions is executed.

CLOSEC
ISAVES
PURGE

Form: i = LOOKUP (chan,dblk{,count})

where:

chan — is the integer specification for the RT-11 channel to be associated with the file.

dblk — is the 4-word area specifying the Radix-50 file descriptor. Note that unpredictable results will occur if the USR swaps over this 4-word area.

count — is an optional argument used for the magtape and cassette handlers (See Appendix H).

NOTE

The arguments of LOOKUP must be positioned so that the USR does not swap over them.

The handler for the selected device must be in memory for a LOOKUP. If the first word of the file name in dblk is zero and the device is a file-structured device, absolute block 0 of the device is designated as the beginning of the "file". This technique, called a nonfile-structured lookup, allows I/O to any physical block on the device. If a file name is specified for a device which is not file-structured (i.e. LP:FILE.EXT), the name is ignored.

NOTE

Since nonfile-structured lookups allow I/O to any physical block on the device, the user must be aware that, in this mode, it is possible to overwrite the RT-11 device directory, thus destroying all file information on the device.

Errors:

i = n	Normal return; n equals the number of blocks in the file (0 for nonfile-structured lookups).
= -1	Channel specified is already open.
= -2	File specified was not found on the device.

Example:

```
INTEGER*2 DBLK(4)
DATA DBLK/3RDK0,3RFTN,3R44 ,3RDAT/
.
.
.
ICHAN=IGETC()
IF(ICHAN.LT.0) STOP 'NO CHANNEL'
IF(IFETCH(DBLK).NE.0) STOP 'BAD FETCH'
IF(LOOKUP(ICHAN,DBLK).LT.0) STOP 'BAD LOOKUP'
.
.
.
CALL CLOSEC(ICHAN)
CALL IFREEC(ICHAN)
.
.
.
```

MRKT

O.3.67 MRKT (F/B only)

The MRKT function schedules an assembly language completion routine to be entered after a specified time interval has elapsed.

Form: i = MRKT (id,crtn,time)

where:

id	is an integer identification number to be passed to the routine being scheduled.
crtn	is the name of the assembly language routine to be entered when the time interval elapses. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the MRKT call.
time	is the 2-word internal format time interval; when this interval elapses, the routine will be entered. If considered as a 2-element INTEGER*2 array:

time (1) is the high-order time.

time (2) is the low-order time.

Notes:

1. MRKT requires a queue element; this should be considered when the IQSET function (Section O.3.33) is executed.

2. If the system is busy, the time interval that elapses before the completion routine is run may be greater than that requested.

For further information on scheduling completion routines, see the assembly language .MRKT request, Section 9.4.22.

Errors:

i = 0	Normal return
= 1	No queue element was available; unable to schedule request.

Example:

```
INTEGER*2 TINT(2)
EXTERNAL ARTN
.
.
.
CALL MRKT(4,ARTN,TINT)
```

MWAIT

O.3.68 MWAIT (F/B Only)

The MWAIT subroutine suspends main program execution of the current job until all messages sent to or from the other job have been transmitted or received. It provides a means for ensuring that a required message has been processed. MWAIT is used primarily in conjunction with the IRCVD and ISDAT calls, where no action is taken when a message transmission is completed. This subroutine requires a queue element; this should be considered when the IQSET function (Section O.3.33) is executed.

Form: CALL MWAIT

Errors:

None.

Example:

See the example under ISDAT, Section 0.3.41.

PRINT

0.3.69 PRINT

The PRINT subroutine causes output (from a specified string) to be printed at the console terminal. This routine can be used to print messages from completion routines without using the FORTRAN formatted I/O system. Control returns to the user program after all characers have been placed in the output buffer.

The string to be printed may be terminated with either a null (0) byte or a 200 (octal) byte. If the null (ASCIZ) format is used, the output is automatically followed by a <CR><LF>. If a 200 byte terminates the string, no <CR><LF> is generated.

In the F/B Monitor, a change in the job that is controlling terminal output is indicated by a B> or F>. Any text following the message has been printed by the job indicated (foreground or background) until another B> or F> is printed. When PRINT is used by the foreground job, the message appears immediately, regardless of the state of the background job. Thus, for urgent messages, PRINT should be used rather than ITTOUR.

Form: CALL PRINT (string)

where: string is the string to be printed. Note that all quoted literals used in FORTRAN subroutine calls are in ASCIZ format as are all strings produced by the SYSLIB string handling package.

Errors:

None.

Example:

```
CALL PRINT ('THE COFFEE IS READY')
```

PURGE

O.3.70 PURGE

The PURGE subroutine is used to deactivate a channel without performing an ISAVES or a CLOSEC. Any tentative file currently associated with the channel is not made permanent. This subroutine is useful for keeping ENTERed (IENTER or .ENTER) files from becoming permanent directory entries.

Form: CALL PURGE (chan)

where: chan is the integer specification for the RT-11 channel to be deactivated.

Errors:

None.

Example:

See the example under IENTER, Section O.3.22.

PUTSTR

O.3.71 PUTSTR

The PUTSTR subroutine writes a variable-length character string to a specified FORTRAN logical unit. PUTSTR may be used in main program routines or in completion routines but not in both in the same program at the same time. If PUTSTR is used in a completion routine, it must not be the first I/O operation on the specified logical unit.

Form: CALL PUTSTR (lun,in,char)

where: lun is the integer specification of the FORTRAN logical unit number to which the string is to be written.

in — is the array containing the string to be written.

char — is an ASCII character that will be appended to the beginning of the string before it is output. If 0, the first character of "in" will be the first character of the record. This character is used primarily for carriage control purposes (see Section O.3.13).

Errors:

None.

Example:

```
LOGICAL*1 STRNG(81)
.
.
.
CALL PUTSTR(7,STRNG,'0')   !OUTPUT STRING WITH DOUBLE SPACING
```

R50ASC

O.3.72 R50ASC

The R50ASC subroutine converts a specified number of Radix-50 characters to ASCII.

Form: CALL R50ASC (icnt,input,output)

where:

icnt — is the integer number of ASCII characters to be produced.

input — is the area from which words of Radix-50 values to be converted are taken. Note that (icnt+2)/3 words are read for conversion.

output — is the area into which the ASCII characters are stored.

Errors:

If an input word contains illegal Radix-50 codes, i.e., if the input word is greater (unsigned) than 174777(octal), the routine outputs question marks for the value.

Example:

```
REAL*8 NAME
LOGICAL*1 OUTP(12)
.
.
.
CALL R50ASC(12,NAME,OUTP)
```

RAD50

0.3.73 RAD50

The RAD50 function provides a method of encoding RT-11 file descriptors in Radix-50 notation. The RAD50 function converts six ASCII characters from the specified area, returning a REAL*4 result which is the 2-word Radix-50 value.

Form: a = RAD50 (input)

where: input is the area from which the ASCII input characters are taken.

The RAD50 call:

```
A = RAD50 (LINE)
```

is exactly equivalent to the IRAD50 call:

```
CALL IRAD50 (6,LINE,A)
```

Function Results:

The 2-word Radix-50 value is returned as the function result.

RCHAIN

0.3.74 RCHAIN

The RCHAIN subroutine allows a program to determine whether it has been chained to and to access variables passed across a chain. If RCHAIN is used, it must be used in the first executable FORTRAN statement in a program. RCHAIN cannot be called from a completion or interrupt routine.

Form: CALL RCHAIN (flag,var,wcnt)

where:

flag — is an integer variable which will be set to -1 if the program has been chained to; otherwise, it is 0.

var — is the first variable in a sequence of variables with increasing memory addresses to receive the information passed across the chain (see Section O.3.2).

wcnt — is the number of words to be moved from the chain parameter area to the area specified by var. RCHAIN moves "wcnt" words into the area beginning at "var".

Errors:

None.

Example:

```
INTEGER*2 PARMS(50)
CALL RCHAIN(IFLAG,PARMS,50)
IF(IFLAG) GOTO 10       !GOTO 10 IF CHAINED TO
.
.
.
```

RCTRLO

O.3.75 RCTRLO

The RCTRLO subroutine resets the effect of any console terminal CTRL O (^O) command which was typed. After an RCTRLO call, any output directed to the console terminal will print until another CTRL O is typed.

Form: CALL RCTRLO

Errors:

None.

Example:

```
CALL RCTRLO
CALL PRINT ('THE REACTOR IS ABOUT TO BLOW UP')
```

REPEAT

O.3.76 REPEAT

The REPEAT subroutine concatenates a specified string with itself to produce the indicated number of copies. REPEAT places the resulting string in a specified array.

Form: CALL REPEAT (in,out,i{,len{,err}})

where:

- in — is the array containing the string to be repeated.
- out — is the array into which the resultant string is placed. This array must be at least one element longer than the value of len, if specified.
- i — is the integer number of times to repeat the string.
- len — is the integer number representing the maximum length of the output string.
- err — is the Logical error flag set if the output string is truncated to the length specified by len.

Input and output strings may only specify the same array if the repeat count (i) is 1 or 0. When the repeat count is 1, this routine is the equivalent of SCOPY; when the repeat count is 0, "out" is replaced by a null string. The old contents of "out" are lost when this routine is called.

Errors:

Error conditions are indicated by "err", if specified. If "err" is given and the output string would have been longer than "len" characters, then "err" is set to .TRUE.; otherwise, "err" is unchanged.

Example:

```
LOGICAL*1 SIN(21),SOUT(101)
.
.
.
CALL REPEAT(SIN,SOUT,5)
```

RESUME

0.3.77 RESUME (F/B Only)

The RESUME subroutine allows a job to resume execution of the main program. A RESUME call is normally issued from an asynchronous FORTRAN routine entered on I/O completion or because of a schedule request. (See Section 0.3.83 for more information.)

Form: CALL RESUME

Errors:

None.

Example:

See the example under SUSPND, Section 0.3.83.

SCOMP

0.3.78 SCOMP

The SCOMP routine compares two character strings and returns the integer result of the comparison.

Form: CALL SCOMP (a,b,i)

or

i = ISCOMP (a,b)

where:

a		is the array containing the first string.
b		is the array containing the second string.
i		is the integer variable that receives the result of the comparison.

The strings are compared left-to-right, one character at a time, using the collating sequence specified by the ASCII codes for each character. If the two strings are not equal, the absolute value of variable i (or the result of the function ISCOMP) is the character position of the first inequality found in scanning left-to-right.

If the strings are not the same length, the shorter one is treated as if it were padded on the right with blanks to the length of the other string. A null string argument is equivalent to a string containing only blanks.

Notes:

References to ISCOMP may not have quoted-string literals as arguments.

Function Result:

```
i <0          if a is less than b
  =0          if a is equal to b
  >0          if a is greater than b
```

Example:

```
LOGICAL*1 INSTR(81)
.
.
.
CALL GETSTR(5,INSTR,80)
CALL SCOMP('YES',INSTR,IVAL)
IF(IVAL) GOTO 10     !IF INPUT STRING IS NOT YES GOTO 10
```

SCOPY

O.3.79 SCOPY

The SCOPY routine copies a character string from one array to another. Copying stops either when a null (zero) character is encountered or when a specified number of characters have been moved.

Form: CALL SCOPY (in,out{,len{,err}})

where:	in	is the array containing the string to be copied.
	out	is the array to receive the copied string. This array must be at least one element longer than the value of len, if specified.

len — is the integer number representing the maximum length of the output string. The effect of len is to truncate the output string to a given length, if necessary.

err — is a Logical variable that receives the error indication if the output string was truncated to the length specified by len.

The input (in) and output (out) arguments may specify the same array. The string previously contained in the output array is lost when this subroutine is called.

Errors:

Error conditions are indicated by "err", if specified. If "err" is given and the output string was truncated to the length specified by "len", then "err" is set to .TRUE.; otherwise, "err" is unchanged.

Example:

SCOPY is useful for initializing strings to a constant value, for example:

```
LOGICAL*1 STRING(80)
CALL SCOPY('THIS IS THE INITIAL VALUE',STRING)
```

SECNDS

O.3.80 SECNDS

The SECNDS function returns the current system time, in seconds past midnight, minus the value of a specified argument. Thus, SECNDS can be used to calculate elapsed time. The value returned is single-precision floating point (REAL*4).

Form: a = SECNDS (atime)

where: atime — is a REAL*4 variable, constant, or expression whose value is subtracted from the current time-of-day to form the result.

Notes:

This function does floating-point arithmetic. Elapsed times may also be calculated by using the GTIM call and the INTEGER*4 support functions.

Function Result:

The function result (a) is the REAL*4 value returned.

Example:

```
     C     START OF TIMED SEQUENCE
           T1=SECNDS(0.)
     C
     C     CODE TO BE TIMED GOES HERE
     C
           DELTA=SECNDS(T1)    !DELTA IS ELAPSED TIME
```

STRPAD

O.3.81 STRPAD

The STRPAD routine pads a character string with rightmost blanks until that string is a specified length. This padding is done in place; the result string is contained in its original array. If the present length of the string is greater than or equal to the specified length, no padding occurs.

Form: CALL STRPAD (a,i{,err})

where:

a	is the string to be padded.
i	is the integer length of the desired result string.
err	is the Logical error flag that is set to .TRUE. if the string specified by a exceeds the value of i in length.

Errors:

Error conditions are indicated by "err", if specified. If "err" is given and the string indicated is longer than "i" characters, "err" is set to .TRUE.; otherwise, the value of "err" is unchanged.

Example:

This routine is especially useful for preparing strings to be output in A-type FORMAT fields. For example:

```
           LOGICAL*1 STR(81)
           .
           .
           .
           CALL STRPAD(STR,80)        !ASSURE 80 VALID CHARACTERS
           PRINT 100,(STR(I),I=1,80) !PRINT STRING OF 80 CHARACTERS
     100   FORMAT(80A1)
```

SUBSTR

O.3.82 SUBSTR

The SUBSTR routine copies a substring from a specified position in a character string. If desired, the substring may then be placed in the same array as the string from which it was taken.

Form: CALL SUBSTR (in,out,i{,len})

where:

where:	in	is the array from which the substring is taken.
	out	is the array to contain the substring result. This array must be one element longer than "len", if specified.
	i	is the integer character position in the input string of the first character of the desired substring.
	len	is the integer number of characters representing the maximum length of the substring.

If a maximum length (len) is not given, the substring contains all characters to the right of character position "i" in array "in". If "len" is equal to zero, "out" is replaced by the null string. The old contents of array "out" are lost when this routine is called.

Errors:

None.

SUSPND

0.3.83 SUSPND (F/B Only)

The SUSPND subroutine suspends main program execution of the current job and allows only completion routines (for I/O and scheduling requests) to run.

Form: CALL SUSPND

Notes:

1. The monitor maintains a suspension counter for each job. This count is decremented by SUSPND and incremented by RESUME (see Section 0.3.77). A job will actually be suspended only if this counter is negative. Thus, if a RESUME is issued before a SUSPND, the latter function will return immediately.

2. A program must issue an equal number of SUSPNDs and RESUMEs.

3. A RESUME subroutine call from the main program or from a completion routine increments the suspension counter.

4. A SUSPND subroutine call from a completion routine decrements the suspension counter but does not suspend the main program. If a completion routine does a SUSPND, the main program continues until it also issues a SUSPND, at which time it is suspended and will require two RESUMEs to proceed.

5. Because SUSPND and RESUME are used to simulate an ITWAIT (see Section 0.3.49) in the monitor, a RESUME issued from a completion routine and not matched by a previously executed SUSPND may cause the main program execution to continue past a timed wait before the entire time interval has elapsed.

For further information on suspending main program execution of the current job, see the assembly language .SPND request, Section 9.4.39.

Errors:

None.

Example:

```
          INTEGER IAREA(4)
          COMMON /RDBLK/ IBUF(256)
          EXTERNAL RDFIN
          .
          .
          .
          IF(IREADF(256,IBUF,IBLK,ICHAN,IAREA,RDFIN).NE.0) GOTO 1000
     C    GOTO 1000 FOR ANY TYPE OF ERROR
     C
     C    DO OVERLAPPED PROCESSING
          .
          .
          .
          CALL SUSPND     !SYNCHRONIZE WITH COMPLETION ROUTINE
          .
          .
          .
          END
          SUBROUTINE RDFIN(IARG1,IARG2)
          COMMON /RDBLK/ IBUF(256)
          .
          .
          .
          CALL RESUME     !CONTINUE MAIN PROGRAM
          .
          .
          .
          END
```

TIMASC

0.3.84 TIMASC

The TIMASC subroutine converts a 2-word internal format time into an ASCII string of the form:

hh:mm:ss

where:	hh	is the 2-digit hours indication
	mm	is the 2-digit minutes indication
	ss	is the 2-digit seconds indication

Form: CALL TIMASC (itime,strng)

where: itime is the 2-word internal format time to be converted. itime (1) is the high-order time. itime (2) is the low-order time.

strng is the 8-element array to contain the ASCII time.

Errors:

None.

Example:

The following example determines the amount of time until 5 p.m. and prints it.

```
        INTEGER*4 J1,J2,J3
        LOGICAL*1 STRNG(8)
        .
        .
        .
        CALL JTIME(17,0,0,0,J1)
        CALL GTIM(J2)
        CALL JJCVT(J1)
        CALL JJCVT(J2)
        CALL JSUB(J1,J2,J3)
        CALL JJCVT(J3)
        CALL TIMASC(J3,STRNG)
        TYPE 99,(STRNG(I),I=1,8)
99      FORMAT(' IT IS ',8A1,' TILL 5 P.M.')
        .
        .
        .
```

TIME

O.3.85 TIME

The TIME subroutine returns the current system time-of-day as an 8-character ASCII string of the form:

hh:mm:ss

where: hh is the 2-digit hours indication

mm is the 2-digit minutes indication

ss is the 2-digit seconds indication

Form: CALL TIME (strng)

where: strng is the 8-element array to receive the ASCII time.

Notes:

A 24-hour clock is used, e.g., 1:00 p.m. is represented as 13:00:00.

Errors:

None.

Example:

```
        LOGICAL*1 STRNG(8)
        .
        .
        .
        CALL TIME(STRNG)
        TYPE 99,(STRNG(I),I=1,8)
  99    FORMAT (' IT IS NOW ',8A1)
```

TRANSL

O.3.86 TRANSL

The TRANSL routine performs character translation on a specified string. The TRANSL routine requires approximately 64 words on the R6 stack for its execution. This space should be considered when allocating stack space.

Form: CALL TRANSL (in,out,r{,p})

where:

in	is the array containing the input string.
out	is the array to receive the translated string.
r	is the array containing the replacement string.
p	is the array containing the characters in "in" to be translated.

The string specified by array "out" is replaced by the string specified by array "in", modified by the character translation process specified by arrays "r" and "p". If any character position in "in" contains a character which appears in the string specified by "p", it is replaced in "out" by the corresponding character from string "r".

If the array "p" is omitted, it is assumed to be the 127 7-bit ASCII characters arranged in ascending order, beginning with the character whose ASCII code is 001. If strings "r" and "p" are given and differ in length, the longer string is truncated to the length of the shorter. If a character appears more than once in string "p", only the last occurrence is significant. A character may appear any number of times in string "r".

Errors:

None.

Examples:

The following example causes the string in array A to be copied to array B. All periods within A become minus signs, and all question marks become exclamation points.

```
     CALL TRANSL(A,B,'-!','.?')
```

The following is an example of TRANSL being used to format character data.

```
      LOGICAL*1 STRING(27),RESULT(27),PATRN(27)
C     SET UP THE STRING TO BE REFORMATTED
C
      CALL SCOPY('THE HORN BLOWS AT MIDNIGHT',STRING)
C
C     00000000011111111112222222
C     12345678901234567890123456
C     THE HORN BLOWS AT MIDNIGHT
C     NOW SET UP PATRN TO CONTAIN THE FOLLOWING PATTERN:
C     16,17,18,19,20,21,22,23,24,25,26,15,1,2,3,4,5,6,7,8,9,10,11,12,13,14,0
C
      DO 10 I=16,26
10    PATRN(I-15)=I
      PATRN(12)=15
      DO 20 I=1,14
20    PATRN(I+12)=I
      PATRN(27)=0
C
C     THE FOLLOWING CALL TO TRANSL REARRANGES THE CHARACTERS OF
C     THE INPUT STRING TO THE ORDER SPECIFIED BY PATRN:
C
      CALL TRANSL(PATRN,RESULT,STRING)
C
C     RESULT NOW CONTAINS THE STRING 'AT MIDNIGHT THE HORN BLOWS'
C     IN GENERAL, THIS METHOD CAN BE USED TO FORMAT INPUT STRINGS
C     OF UP TO 127 CHARACTERS.  THE RESULTANT STRING WILL BE
C     AS LONG AS THE PATTERN STRING (AS IN THE ABOVE EXAMPLE).
```

TRIM

O.3.87 TRIM

The TRIM routine shortens a specified character string by removing all trailing blanks. A trailing blank is a blank that has no non-blanks to its right. If the specified string contains all blank characters, it is replaced by the null string. If the specified string has no trailing blanks, it is unchanged.

Form: CALL TRIM (a)

where: a is the array containing the string to be trimmed.

Errors:

None.

Example:

```
        LOGICAL*1 STRING(81)
        ACCEPT 100,(STRING(I),I=1,80)
    100 FORMAT(80A1)
        CALL SCOPY(STRING,STRING,80)         !MAKE ASCIZ
        CALL TRIM(STRING)                    !TRIM TRAILING BLANKS
```

UNLOCK

O.3.88 UNLOCK

The UNLOCK subroutine releases the User Service Routine (USR) from memory if it was placed there by the LOCK routine. If the LOCK required a swap, the UNLOCK loads the user program back into memory. If the USR does not require swapping, the UNLOCK involves no I/O.

Form: CALL UNLOCK

Notes:

1. It is important that at least as many UNLOCKS are given as LOCKs. If more LOCKs were done, the USR remains locked in memory. It is not harmful to give more UNLOCKS than are required; those that are extra are ignored.

2. When running two jobs in the F/B system, use the LOCK/UNLOCK pairs only when absolutely necessary. If one job LOCKs the USR, the other job cannot use the USR until it is UNLOCKed. Thus, the USR should not be LOCKed unnecessarily, as this may degrade performance in some cases.

3. In an F/B System, calling the CSI (ICSI) with input coming from the console terminal performs an implicit UNLOCK.

For further information on releasing the USR from memory, see the assembly language .LOCK/.UNLOCK requests, Section 9.4.20.

Errors:

None.

Example:

```
      .
      .
      .
C     GET READY TO DO MANY USR OPERATIONS
      CALL LOCK     !DISABLE USR SWAPPING
C     PERFORM THE USR CALLS
      .
      .
      .
C     FREE THE USR
      CALL UNLOCK
      .
      .
      .
```

VERIFY

O.3.89 VERIFY

The VERIFY routine determines whether each character of a specified string occurs anywhere in another string. If a character does not exist in the string being examined, VERIFY returns its character position in string "b". If all characters exist, VERIFY returns a zero.

Form: CALL VERIFY (a,b,i)

or

i = IVERIF (a,b)

where:

a	is the array containing the string to be scanned.
b	is the array containing the string of characters to be accepted in a.
i	is the integer result of the verification.

Notes:

Quoted-string literals must not be used with the IVERIF function subprogram. They may be used with the VERIFY subroutine subprogram.

Function Result:

i = 0	if all characters of "a" exist in "b"; also if "a" is a null string.
= n	where n is the character position of the first character in "a" that does not appear in "b"; if "b" is a null string and "a" is not, i equals 1.

Example:

The following example accepts a 1- to 5-digit unsigned decimal number and returns its value.

```
      LOGICAL*1 INSTR(81)
      .
      .
      .
      CALL VERIFY(INSTR(IPOS),'0123456789',I)
      IF(I.EQ.1) STOP 'NUMBER MISSING'
      IF(I.EQ.0) I=LEN(INSTR)-IPOS+1
      IF(I.GT.5) STOP 'TOO MANY DIGITS'
      NUM=IVALUE(INSTR(IPOS),I)
      .
      .
      .
      END
      FUNCTION IVALUE(ARRAY,I)
      LOGICAL*1 ARRAY(1)
      DECODE(I,99,ARRAY) IVALUE
99    FORMAT(I5)
      END
```

APPENDIX P

ERROR MESSAGE SUMMARY

The information that formerly appeared in this appendix has now been incorporated into the RT-11 System Message Manual, DEC-11-ORMEA-A-D.

GLOSSARY

Absolute address	A binary number that is assigned as the address of a fixed memory storage location.
Absolute block numbers	Any blocks which use block 0 of a physical device as a base. Relative blocks use the start of a file as a base.
Absolute section	That portion of a program in which the programmer specifies the physical (absolute) locations of data items, using the .ASECT assembler directive.
Addressing mode	The portion of a PDP-11 machine instruction which specifies how the argument is to be referenced.
Allocation	Assigning a resource to a job.
Alphanumeric character string	A command string containing any of the 26 alphabetic characters A-Z, the numeric characters 0-9, space and certain special characters.
Argument	A variable or constant which is given in the call of a subroutine as information to it; a variable upon whose value the value of a function depends; the known reference factor necessary to find an item in a table or array (i.e., the index).
Argument block	A block of memory used to transmit programmed request arguments to the monitor.
Array	A set or list of elements, usually subscripted variables or data.
ASCII	American Standard Code for Information Interchange. Established by American National Standards Institute to standardize a binary code for alphanumeric characters.
Assembler	A program that translated symbolic opcodes into machine language and assigns memory locations for variables and constants.
Assembler directives	The mnemonics used in the assembly language programs to control or direct the assembly process.

Assembly language	A symbolic language that translates directly into machine language instructions. Usually there is a one-to-one relation between assembly language instructions and machine language instructions.
Asynchronous	1. Pertaining to the scheduling of hardware operations by ready and done signals rather than by time intervals. 2. Pertaining to the method of data transmission in which each character is sent with its own synchronizing information.
Autoincrement mode	Mode of operation wherein the contents of the register are incremented immediately after being used as the address of the operand.
Background program	A program operating automatically, under low priority, when higher priority (foreground) programs are not using system resources.
Backup file	A copy of a file created for protection in case the primary file is inadvertently destroyed.
Bad blocks	A defective block of storage on any type of magnetic storage medium that produces a hardware error when attempting to read or write in the block.
Base address	A given address from which an absolute address is derived by combination with a relative address. An address constant.
Base segment	The portion of an overlayed program that is always memory resident; same as root segment.
Baud	A unit of signaling speed. In a code in which all characters have the same length, one baud corresponds to a rate of one signal element per second, usually one bit per second.
Binary	Pertaining to the number system having a base or radix of two. In this system numbers are represented by 1s and 0s. Counting to ten in binary: 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010.

Bit	Contraction of "Binary digit", a bit is the smallest unit of information in a binary system of notation. It is the choice between two possible states, usually designated one (1) and zero (0).
Block-replaceable	See Random-Access.
Blocks	A set of consecutive machine words, characters or digits handled as a unit, particularly in reference to I/O. Each type of mass storage has its own block size, its own smallest unit of storage (e.g., PDP-11 DECtape has 256 (decimal) 16-bit words per block). See Data Block.
Bootstrap	A program whose purpose is to load and (usually) start a complex system of programs.
Bottom address	The lowest memory address in which a program is located.
Bounds	The limits a program may operate within.
Breakpoint	A preset point in a user program where control passes to a debugging routine.
Buffer	A temporary storage area which may be a special register or an area of storage. Buffers are often used to hold data being passed between processes or devices which operate at different speeds or different times.
Byte	A group of binary digits usually operated upon as a unit; 8 bits is usually a byte (1/2 a PDP-11 word).
Carry bit	A bit in the program status word indicating a carry from the most significant bit in the operation; also a common method of indicating a program request failure.
Central processing unit	The part of the computer containing the Arithmetic and Logical Unit, the Instruction Control Unit, timing generators and I/O interfaces of the basic system.
Chaining	A programming technique which involves dividing a routine into sections with each section terminated by a call to the next section.
Channel number	Logical identifier in the range 0 to 255 (decimal) assigned to a file used by RT-11 monitor. Data blocks in an open file may be referred to by channel number.

Channnel status word	A word associated with each I/O channel in RT-11 that perserves the status of that channel.
Character-oriented	Referring to editing operations on a single character. See also Line-oriented.
Checksum	A value representing the sum of all bytes in a program. When the program is loaded, the sum of the bytes can be compared with the checksum to make sure that the entire program has been loaded correctly.
Clock	A time-keeping or frequency-measuring device within the computer system.
Closed location	A location whose contents are not available for examination and change.
Co-resident overlay routines	Overlay routines that are simultaneously resident in memory.
Command string	A series of characters which specify the input/output devices, files, and switches.
Command String Interpreter	The portion of the RT-11 system software which accepts an ASCII string input and interprets the string as input and output files and switches.
Completion routine	An optional user-supplied routine that is called at the completion of an operation and is generally used to allow the user to handle asynchronous events.
Compress	To collect into one area all the free (unused) blocks that are interspersed in the directory and files on a specified device.
Concatenate	To combine many files into one file.
Condition codes	The four bits of the Program Status Word in the PDP-11 Processor that preserve the results (negative, zero, carry, overflow) of the instruction just completed.
Configuration	A particular selection of computer, peripherals, and interfacing equipment that are functioning together. A list of the devices of a computer system.
Console device	The interface, or communication device, between the operator and the computer, that contains indicator lights, switches, knobs and sometimes a keyboard to permit manual operation of the device.
Constant register	A logical register in ODT or PATCH that is used to store an often-used constant.

Context switching	The saving of key registers and other memory areas prior to switching between jobs, as in timesharing or multiprogramming.
Contiguous	Code that resides in memory or on a peripheral device immediately adjacent to other sections of code or data.
Control Section (CSECT)	A named, contiguous part of a program. CSECTs are denoted by the directives ".CSECT" and ".ASECT" in the MACRO Assembler Language. CSECTs are collected and assigned addresses by the Linker.
Control statement	A job statement that is used to issue commands to the system through the input stream.
Core memory	The main high-speed storage of a computer in which binary data is represented by the switching polarity of magnetic cores. Semi-conductor memory is higher speed than core memory, but semi-conductor memory is more expensive and volatile.
Cross reference table (CREF)	A list of all or a subset of symbols in source program and statements where they were defined or used.
Cursor	On a display device, a symbol that appears to indicate the location of the pointer.
Data block	A logical grouping of data, usually associated with input or output. Typical data block involved in RT-11 are 256 words long.
Data format	The form or structure of information, particularly in an I/O file. RT-11 has several standard data formats such as ASCII and formatted binary.
Data image	A job statement that is used to define data for system programs and user programs through the input stream.
Debug	To detect, locate, and correct mistakes in a program.
Default	A specification assigned by a system program when user specification is omitted.
Delimiter	A character that separates, terminates and organizes elements of a statement or program.
Device block	A section of code that specifies a physical device and filename for an RT-11 programmed request.

Device directory listing	A list of all files on a specified device. The list contains all files with associated creation dates, total free blocks on the device, and number of blocks used by files. Magtape and cassette directories omit some information. See Directory.
Device handlers	Routines that perform I/O for specific storage devices and translate logical block numbers to physical disk, tape or drum addresses. These routines also handle error recovery and provide device independence in conjunction with operating systems.
Direct assignment	User definition of a symbol and its associated value.
Directive	See Assembler directives.
Directory	An area of a mass storage device that describes the layout of the data on that device in terms of file names, length and location.
Disk	A mass storage device. Basic unit is an electromagnetic platter on which data is magnetically recorded. Features random access and faster access time than magnetic tape.
Display unit	A device that provides a visual representation of data.
Double buffered I/O	An input (or output) operation using two buffers to achieve more efficient usage of computer time. While one buffer is being used by the CPU program, the other is being read from (or written to) by an I/O device.
Dummy file	A file used only to hold space on a storage device.
Editor	A program which interacts with the programmer to enter new programs into the computer and edit them as well as modify existing programs. Editors are language independent and will edit anything in alphanumeric representation.
EMT	A PDP-11 machine instruction (operation codes from 104000-1043777). EMTs are most often used for monitor communication.
Entry point	A point in a subroutine to which control is transferred when the subroutine is called.

Entry point table	A table, kept by the Librarian program, of the names and locations of the routines available in the library.
Entry symbol	A global symbol defined in an object module that can be referred to by other object module.
Error flag	Condition indicating an error has occurred. If the C bit is set on return from an RT-11 call, an error occurred. Thus, the C bit set is the RT-11 error flag.
Exception	An unusual condition (e.g., a floating point exception is an overflow).
Expressions	A combination of variables, constants, and operators (as in a mathematical expression).
External symbol	A symbolic name which is defined in one assembly or compilation and can be referenced in another. The .GLOBL directive is used to indicate external symbols to the MACRO assembler.
Fatal error	An error which makes continued processing impossible (e.g., running out of symbol table space is usually a fatal error).
File allocation scheme	The method used to store data on I/O devices. RT-11 uses a contiguous structure.
File gap	A fixed length of blank tape separating files on a recording medium.
File-structured device	A device on which data is given names and arranged in files (e.g., disk, DECtape, magtape, or cassette).
Filler characters	Null characters output to a device to give it time to perform unusually long operations (such as return the carriage) without explicitly waiting.
Flag	A variable or register used to record the status of a program or device. In the latter case it is sometimes called a device flag.
Floating-point	A number system in which the position of the radix point is indicated by one part (the exponent part), and another part represents the significant digits, the fractional part (e.g., 5.39X10(8) for Decimal; 137.3X8(4) for Octal; 101.10X2(13) for Binary).

Foreground program	A program of high priority that utilizes machine facilities when and as needed, but allows less critical background work to be performed in otherwise unused time.
Formatted binary block	A data structure used to hold binary information, usually to be read or written to a data file. For example, an object module produced by the MACRO assembler consists of formatted binary blocks.
Fragmented	Having many empty entries scattered over a device.
Free blocks	Areas of a file structured device which are unused.
General registers	A set of eight general purpose registers available for use as accumulators, as auto index registers or as pointers. General registers 6 and 7 serve as the hardware stack pointer and program counter respectively.
Global symbol	Any symbol accessible to other programs. Linkage must be supplied by a linker.
Global symbol directory	The portion of an object module which describes all external symbols and CSECT names which are defined or referenced by the object module.
Handler	See Device handler
Hardware mode	For magnetic tapes and cassettes. Allows full user control over position and record size (as opposed to software mode.)
High level language	A language in which single statements may result in more than one machine language instruction, e.g., BASIC, FORTRAN, COBOL.
Image mode	A mode of data transfer in PIP in which a file is copied without change of any kind.
Implied argument	An argument which is assumed by the program, whether or not it is explicitly stated by the user.
Indexing	Using a variable index register as an offset into a table.
Initialize	To set counters, switches, addresses and variables to zero or other starting values.

Input stream	The sequence of control statements and data images submitted to the operating system on an input device especially dedicated for this purpose.
Internal symbol	A symbolic name which is known only within the assembly or compilation where it is defined. Symbols are internal by default.
Interrupt service routine	Routine entered when an external interrupt occurs.
Interrupt	1. To break the normal operation of the routine being executed and pass control to a specific location, generally accompanied by saving the state of the current routine so that control can return later. 2. The signal which causes the break.
Iterations	Repetitions of a portion of a program.
Job	The largest definable collection of elements describing the work to be performed. The purpose of the job is to collect and describe the logical work which is to be performed. A job is composed of control statements and data images.
Job status word	A word in the RT-11 communications region containing bit flags indicating the status of the program currently in memory.
Keyboard monitor (KMON)	The program that provides communication between the user at the console and the RT-11 system.
Keyword	A symbol that identifies a parameter.
Label	One or more characters used to identify a source language statement or line.
LDA format file	See Load image file.
Library	A collection of standard routines which can be incorporated into other programs.
Library header	Section of code that contains the current status of the library, including version, date and time of creation or update, relative starting address of entry point table, number of EPT entries and placing of next module to be inserted into the library file.
Light pen	A device resembling a pencil or stylus which can detect a fluorescing spot on a CRT screen. Used to input information to a CRT display system.

Line-oriented	Referring to editing operations on an entire line of text. See also Character-oriented.
Load image file	A program that can be executed in stand-alone environment without the aid of relocation.
Local symbol	A symbol used only within the program in which it is defined (all non-global symbols).
Location counter	A counter kept by an assembler to determine the address assigned to an instruction or constant being assembled.
Logical name	A user-defined name assigned as an alternate name for a physical device. Useful for redirecting I/O when device originally specified is unavailable.
Loop	A sequence of instructions that is executed repeatedly until a terminal condition occurs. Also, used as a verb meaning to execute this circle of instructions while waiting for the ending condition.
Macro	An instruction in a source language that is equivalent to a specified sequence of machine instructions, or a command in a command language that is equivalent to a specified sequence of commands.
Macro symbol	Symbol used as macro name in operator field.
Mainstream code	Any code which is not executing as a result of a completion routine.
Mask word	A combination of bits that is used to access selected portions of any word, character, byte, or register while retaining other parts for use. Also, to clear these selected locations with a mask.
Memory address	An address in memory used to store data.
Memory image	An exact copy of a portion of memory. RT-11 save image files (.SAV) are memory images.
Mnemonic	Alphabetic representation of a function or octal machine instruction.
Mode	A state or method of program operation. Command mode causes user input to be interpreted as a command; text mode causes user input to be interpreted as alphanumeric data, etc.

Module	A routine which handles a particular function.
Monitor	The collection of routines which schedules resources, I/O, system programs, and user programs, and obeys keyboard commands in a Monitor System.
Monitor System	Editors, assemblers, compilers, loaders, interpreters, data management programs and other utility programs all automated for the user by a monitor.
Multi-region	A term describing the RT-11 overlay structure in which there may exist multiple independent areas of high-speed memory (regions) in which overlays may occur.
Nesting	1. Including a program loop inside another program loop, or other similar occurrences within one another. 2. Algebraic nesting such as (A+B+(C+D)) where execution proceeds from the innermost level to the outermost level.
Nonfile-structured device	A device that is used only for input or output, and not storage, of a file (e.g., line printer, card reader, terminal, or paper tape).
Nonfile-structured .LOOKUP (or .ENTER)	A .LOOKUP (or .ENTER) with a null file-name, that permits access to a device by reference to absolute block numbers, rather than block numbers relative to a file.
NonRT-11 directory-structured	A device having no standard RT-11 directory at its beginning. This includes magtape and cassette, and any devices for which user-defined directories have been written.
Null	Characters with ASCII code 000.
Object code	The result after assembling or compiling source code. Machine code.
Off-line	Pertaining to equipment, devices or events which are not under direct control of the computer.
Offset	The difference between a location of interest and some known base location.
On-line	Pertaining to equipment, devices and events which are in direct communication with the CPU and thereby under its control in some way.
Open location	A location whose contents have been printed for examination; contents can be changed.

Operand	1. A quantity which is affected, manipulated or operated upon. 2. The contents of the field following the operator of an assembler instruction.
Operating system	A program for automating a programmer's use of software. A Monitor System.
Operator	That symbol or code which indicates an action or operation to be performed.
Output stream	Output data issued by the operating system on the processing program to an output device especially designated for this purpose.
Overlay	The technique of repeatedly using the same area of memory during different stages of a program. When one routine is no longer needed in memory, another routine can replace all or part of it. Over-laying replaces parts of programs; chaining replaces the whole program.
Overlay segment	A section of code handled as a unit. This segment of code can overlay code already in memory or can be overlaid by another overlay segment.
Overlay-structured	A term describing a program which does not entirely reside in high-speed memory at any instant. Portions of the program are brought into memory when needed.
Page of text	That portion of a file delimited by form feed characters, generally 50-60 lines long: corresponds approximately to a physical page of program listing.
Parameter	A variable or an arbitrary constant appearing in a mathematical expression, each value of which restricts or determines the specific form of the expression.
Patch	To modify a routine in a rough or expedient way, usually by modifying the binary code rather than reassembling it.
Peripheral devices	In a data processing system, any device distinct from the central processing unit, which may provide the system with outside storage or communication.
Permanent file	An output file that is stored in memory for later use.
Permanent symbol	Instruction mnemonics, assembler directives, and macro directives incorporated in the assembler.

Physical device	An input, output, or storage device associated with the Central Processing Unit.
Physical name	A 2- or 3-character name identifying a physical device. The first two characters are alphabetic and identify the device; the third character is numeric and identifies the unit.
PIC code	Abbreviation for Position Independent Code. (Code which can operate properly wherever it may be loaded in memory).
Pointer	1. A location containing the address to another location. 2. In the EDIT program, a movable reference pointer normally located between the character most recently operated upon and the next character in the buffer; represents the current position of the Editor in the text.
Positional parameter	A parameter that must appear in a specified position in an ordered group of parameters.
Proceed count	In a program loop, the number of times the breakpoint is to be encountered before suspension of program execution.
Processor status register	A register that indicates the current priority of the processor, the condition of the previous operation, and other basic control items.
Program counter	A register in the CPU that holds the address of the current instruction being executed plus two; in other words, it holds the address of the next instruction unless the current instruction causes a jump.
Program sections	See Control section.
Programmed requests	Machine language instruction which is used to invoke a monitor service for the issuing program.
Purge	Deactivate a channel without taking any other action.
Radix	The base of a number system; the number of digit symbols required by a number system.
Random-access	A means of accessing data in a random order (independently of the data's physical or relative position on the device).

Real time	1. Pertaining to the actual time during which a physical process takes place. 2. Pertaining to the performance of computer activity which occurs fast enough to influence the related physical process.
Real time system	A system in which computation is performed while a related physical process is occurring so that the results of the computation can be used in guiding the physical process.
Record	A collection of related items of data treated as a unit. Example: A line of source code.
Reentry address	The Start address -2. Allows the user program to reset itself internally, and resume operation.
Region number	A number which is used to identify a portion of memory to the Linker for the purpose of describing an overlay structure.
Relative address	A number specifying the difference between an absolute address and a base address.
Relocatable code	Code written so that it can be located and executed in any part of memory, once it has been linked by Linker.
Relocatable object modules	A set of instructions which are written so that the module can be loaded and executed in any memory area.
Relocatable symbols	Symbolic names whose associated value is an offset from the base (beginning) of a control section. Such a symbol's value depends on the address of the control section and must be relocated each time the control section is assigned an address by the Linker.
Relocation	To move a routine from one portion of storage to another and to adjust the necessary address references so that the routine can be executed in the new location.
Relocation directory	A portion of an object module which describes and identifies all occurrences of relocatable symbols in the object module.
Repeat block	Block of code to be repeated a defined number of times.
Repeat count	The number of times a block of code is to be repeated.

Resident	In memory, as opposed to being stored externally.
Resident monitor (RMON)	The permanent memory-resident part of RT-11. Contains console terminal service, the error processor, the system device handler, the EMT processor, and system tables.
Root segment	See Base segment.
RT-11 directory-structured	A device having a standard RT-11 directory at its beginning that is updated as files on the device are manipulated (for example, disk and DECtape).
Scratch area	Any memory or registers used for temporary storage of partial results.
Scrolling	On a display screen, when the maximum number of lines are on the screen, the top line is deleted, and all text moves up one line, permitting one new line of display at the bottom of the screen. The visual effect is similar to the rolling up of a scroll.
Sentinel file	The last file on cassette tape; contains only a header record and represents logical end-of-tape.
Sequence number	The number in a cassette directory which designates where a file is placed (in sequence), when that file has been continued on more than one cassette.
Sequential-access	A means of accessing data in which records or blocks are read one after another from the device (as opposed to random-access).
Software mode	For magnetic tapes and cassettes, the mode of operation when device access is through any RT-11 system program.
Source code	Text, usually in the form of an ASCII formatted file, which represents a program. Such a file can be processed by an appropriate system program (MACRO or FORTRAN) to produce an object module.
Source file	A file to be used as input to a translating program such as MACRO or BASIC.
Stack	An area of memory set aside by the programmer for temporary storage or subroutine interrupt service linkage. The stack uses the "last-in, first-out" concept. The stack starts at the highest location reserved for it and expands linearly downward as items are added to the stack.

Stack pointer	A general register used to keep track of the last location where data is entered into the stack.
Storage device	A general term for any device capable of retaining information.
Subconditionals	Directives that indicate: 1. assembly of an alternate body of code; 2. assembly of a non-contiguous body of code within a conditional block; 3. unconditional assembly of a body of code within a conditional block.
Subpicture address	Address of a set of display processor instructions which display text on a GT40 (42,44).
Suspend	To temporarily halt execution of a task while another task of different priority runs.
Swap	The process of saving user memory in system device scratch blocks, reading in and executing a USR function, and then restoring the user code.
Switch register	A location in the CPU which can be loaded with a value by the operator by his setting switches on the computer console for each bit he wants to enter.
Switches	1. in a command string to the CSI a switch is a slash followed by an ASCII character which can be given a value by typing a colon after the character followed by an octal number or from 1-3 ASCII characters. 2. Two or three position mechanisms used for operating computer or devices.
Symbols	Names which can be assigned values or which can be used to indicate specific locations in a program.
Symbol table	An array which contains all defined symbols and the binary value associated with each one. Mnemonic operators, labels and user defined symbols are all placed in the symbol table. (Mnemonic operators stay in the table permanently.)
System configuration	See Configuration.
System device	A peripheral file-structured storage device on which the monitor resides.

System programs	DEC-supplied programs which come in the basic software packages. These include editors, PIP, assemblers, compilers, loaders, etc.
Time-critical job	A job which demands response within a fixed time period.
Transfer address	A program entry point.
Trap	1. An automatic transfer of control to a prespecified routine that can be caused by software or hardware. The trap instruction is an example of a hardware implementation. 2. A conditional jump to a known location performed automatically by hardware as a side effect to executing a processor instruction. The address location from which the jump is made is recorded. It is distinguished from an interrupt which is caused by an external event.
Truncation	The reduction of precision by ignoring one or more of the digits (not rounding off).
Two's complement	A number used to represent the negative of a given value in many computers. This number is formed from the given binary value by changing all 1s to 0s and all 0s to 1s, and then adding 1.
Type-ahead	The ability to type information at the console terminal and have it remembered by the system for later use.
Unary operation	Operation affecting a single operand. Examples: negation, radix indicator.
User-defined sybmbol	1. A label; 2. A symbol defined by direct assignment.
Utility	Any program which performs useful functions, i.e., PIP.
Vector	Two words describing 1) where to go when external interrupt occurs, and 2) the contents of the processor status word when the interrupt is acknowledged.
Wild card operation	A shorthand method of transferring all files with the same name, extension, or both.
Word-for-word transfer	A transfer in which no alteration of data is performed.

Words — In the PDP-11 a 16-bit unit of data which may be stored in an addressable location.

Write-lock condition — The condition of a device that is protected against any transfers which would write information to it.

Zero — To initialize a device (e.g., DECtape, cassette) so that it contains no information.

INDEX

Please cut along this line.

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form.

Did you find errors in this manual? If so, specify by page.

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

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

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

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name ______________________ Date ______________

Organization ______________________________

Street ______________________________

City ______________ State __________ Zip Code or Country __________

If you require a written reply, please check here. ☐

-- Fold Here --

-- Do Not Tear - Fold Here and Staple --

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

digital

digital equipment corporation

Printed in U.S.A.