RSTS/E
RT11 Utilities Manual

Order No. AA–M213A–TC

digital software

RSTS/E
RT11 Utilities Manual

Order No. AA–M213A–TC

December 1981

This document describes RT11–based utilities that you use while programming under the RSTS/E operating system.

OPERATING SYSTEM AND VERSION:	RSTS/E	V7.1
SOFTWARE VERSION:	RSTS/E	V7.1

digital equipment corporation – maynard, massachusetts

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

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

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

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

The following are trademarks of Digital Equipment Corporation:

DEC	VT	IAS
DECUS	DECsystem–10	MASSBUS
DECnet	DECSYSTEM 20	PDT
PDP	DECwriter	RSTS
UNIBUS	DIBOL	RSX
VAX	EduSystem	VMS

Commercial Engineering Publications typeset this manual using DIGITAL's TMS–11 Text Management System.

1/82/14

Contents

Chapter 4 Librarian (LIBR)

Figures

Tables

Preface

In previous releases of RSTS/E, this document was called the *RSTS/E FORTRAN IV Utilities Manual.* The title has been changed to reflect more accurately the manual's content and its use in the RSTS/E programming environment.

Audience Description

Users of this manual should be familiar with either the FORTRAN IV or MACRO computer language and have a working knowledge of the RSTS/E operating system.

Purpose of Document

This manual describes the RT11–based utilities that MACRO and FORTRAN IV programmers need to develop programs on RSTS/E, on an RT11 system, or both.

Associated Documents

Refer to Chapter 1 for a description of the documents you need to develop MACRO or FORTRAN IV programs on a RSTS/E system.

Document Structure

To better understand and use this manual, read the material in Chapter 1 before using the utilities in Chapters 2 through 5 and before referencing the appendixes. The introductory chapter contains information you need to understand more thoroughly the use of RT11 utilities on a RSTS/E system.

There are five chapters, a glossary, and two appendixes:

- Chapter 1 USING RT11 UTILITIES ON RSTS/E
  Introduces the reader to the RT11 utilities as they are used on a RSTS/E system.
- Chapter 2 MACRO–11 PROGRAM ASSEMBLY
  Describes how to use the MACRO assembler to create an object module that is input to the LINK utility.
- Chapter 3 LINKER (LINK)
  Contains information the MACRO and FORTRAN IV programmmer need to combine many object modules into an module the computer can execute.
- Chapter 4 LIBRARIAN (LIBR)
  Shows how to create, modify, maintain, and use library files containing FORTRAN IV and MACRO modules.

- Chapter 5 OBJECT MODULE PATCH UTILITY (PAT)
  Describes how to update code in a relocatable binary object module file.
- Appendix A SWITCH AND ARGUMENT SUMMARY
  Contains a summary list of MACRO, LINK, and LIBR switches (and arguments). Use this appendix for reference.
- Appendix B ERROR MESSAGE SUMMARY
  Lists the error messages you may encounter as you use the MACRO, LINK, LIBR, and PAT utilities. (Appendix B does not, however, contain a description of the MACRO programming-level error codes. You must refer to Chapter 2 for that information.)
- Glossary
  Defines the more commonly used terms in this manual.

Documentation Conventions

A description of the symbolic conventions used throughout this manual follows. Familiarize yourself with these conventions before you continue reading.

1. RT–11 (with the hyphen) refers to the RT–11 operating system. RT11 (without the hyphen) refers to the utilities based on the RT–11 utilities that have been modified to run on RSTS/E — the utilities described in this manual.
2. Examples consist of actual computer output wherever possible. Where necessary, user input is in red to distinguish it from computer output.
3. This manual uses the symbol RET to represent a carriage return. Unless the manual indicates otherwise, terminate all commands or command strings with a carriage return.
4. You produce several characters in system commands by typing a combination of keys concurrently. For example, while holding down the CTRL key, type O to produce the CTRL/O character. Key combinations such as this one are documented as CTRL/O, CTRL/C, and so on.
5. In discussions of command syntax, uppercase letters represent the command name, which you must type. Lowercase letters represent a variable, for which you must supply a value.

   Square brackets ([ ]) enclose an item that is optional: you may include the item in brackets, or you may omit it, as you choose.

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

* Teletype is a registered trademark of the Teletype Corporation.

Chapter 1
Introduction: Using RT11 Utilities on RSTS/E

The main text of this manual describes four RT–11 utilities that have been modified for RSTS/E: MACRO, LINK, LIBR, and PAT. These utilities give the MACRO or FORTRAN IV programmer tools to create executable programs on a RSTS/E system. This introductory chapter contains background information you need before using the RT11 utilities in the RSTS/E programming environment.

1.1 Program Development Cycle and the RT11 Utilities

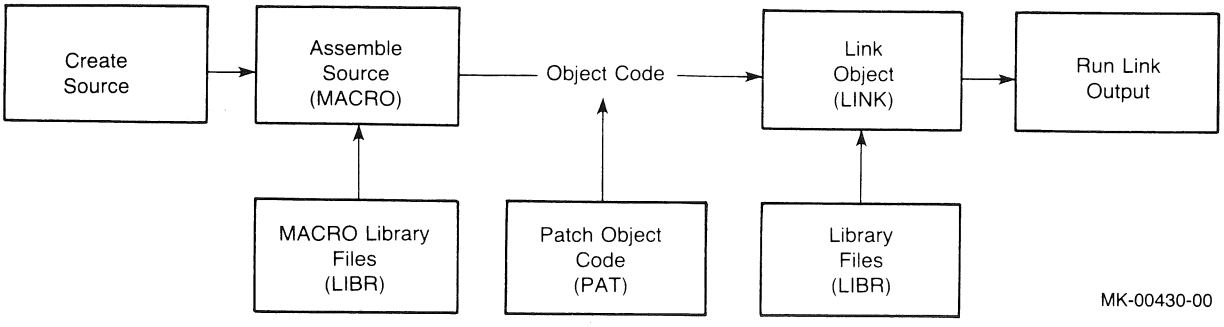
There are three major operating systems that run on the PDP–11 computer: RT11, RSX, and RSTS/E. Each provides an environment in which you can create and run programs. The RSTS/E operating system, in contrast to the others, allows the development of programs compatible with either RT11 or RSX execution-time structures, as well as programs for RSTS/E itself. In the case of RSX, compatibility is at the source and command level; for RT11 compatibility extends down to .SAV. There are certain restrictions, such as lack of the Extended Memory (XM) Monitor and Foreground/Background (FB) Monitor for RSTS/E's RT11 environment, and lack of asynchronous I/O for both RT11 and RSX; RSTS/E, in fact, emulates only the RT–11 Single-Job (SJ) Monitor. This manual describes the RT11 utilities you use to create RT11–based programs that can run either on an RT11 system or on your RSTS/E system. However, before beginning to program on RSTS/E, you must understand the "program development cycle."

1.1.1 Program Development Cycle

There are a number of steps to take before your computer can execute programs you write. These steps in the creation of your executable program are collectively called the "program development cycle." You have three paths you can take in developing programs on RSTS/E: RT11, RSX, and RSTS/E. (That is, you can develop RT11 programs that run on RT11 or RSTS/E systems and create RSX programs that run on RSX or RSTS/E, resulting in programs that can run in any one of three environments.)

While the operating systems are different, they have similar program cycles. A diagram (using the RT11 program names) can illustrate this process:

Figure 1–1: Developing an Executable Program



The utilities in this manual perform some of the operations in this cycle. The RSX environment on RSTS/E provides a parallel set of program development tools. Refer to the *RSTS/E Programming Utilities Manual* for a description of the utilities you use to create programs in the RSX RSTS/E environment.

1.1.2 Definition of the RT11 Utility Programs

The RT11 utility programs that RSTS/E provides for program development are:

- *Macro Assembly (MACRO)*
  The MACRO assembler accepts a MACRO source program as input to create an object module. You then use the object module as input to the LINK utility to make a file the system can execute.

- *Linker (LINK)*
  LINK accepts object modules from an assembler (and/or a compiler) to produce an executable file. LINK can also combine a number of object programs with any necessary library and assembly language subroutines. It is the output of the LINK operation that you specify with the system RUN command. For example, if MYPRGM were the name of the output from LINK, you could execute MYPRGM by typing RUN MYPRGM.

- *Librarian (LIBR)*
  LIBR lets you build and maintain object libraries of your frequently used FORTRAN IV or MACRO routines. It also maintains macro libraries for the MACRO–11 assembler. The librarian organizes the library files so that the linker and MACRO–11 assembler can access them rapidly.

- *Object Module Patch (PAT)*
  PAT helps you modify code in a relocatable binary object (.OBJ) module. This means you can change previously assembled code without reassembling it. You can also use PAT to update library files and to patch the compiler and FORTRAN Object Time System (OTS).

1.1.3 Languages that Use the RT11 Utilities

MACRO and FORTRAN IV are the two DIGITAL–supplied computer languages that use the RT11 utilities described in this manual. MACRO comes to you on the RSTS/E distribution kit; FORTRAN IV is distributed on its own kit, which means you must order it separately.

The RT11 utilities are compatible with programs written in either MACRO or FORTRAN IV. For example, object file formats from the RT11 MACRO assembler and the FORTRAN IV compiler are identical. This allows the RT11 LINK utility to accept both types of object files and process them in the same way. Furthermore, because the object file formats are identical, you can use the patch utility PAT to modify the object code of both languages. You must use the RSX utilities to process programs written in other languages, such as BASIC–PLUS–2.

Refer to the *RSTS/E System User's Guide* for a more complete discussion of program development on RSTS/E.

1.2 Additional Documents

To use this manual properly, you must have access to other documentation. The list that follows contains the documents either FORTRAN IV or MACRO programmers can use for developing FORTRAN IV or MACRO programs on RSTS/E:

- *PDP–11 MACRO Language Reference Manual*
  Describes how to use the MACRO–11 relocatable assembler to develop PDP–11 assembly language programs. This manual presents detailed descriptions of MACRO–11 features. These include source and command string control of assembly and listing functions, directives for conditional assembly and program sectioning, and user-defined and system macro libraries.
- *RSTS/E System Directives Manual*
  Contains general information on run-time systems and describes RSTS/E monitor, RSX emulator, and RT11 emulator directives for the assembly-language programmer.
- *RSTS/E System User's Guide*
  Contains a description of nonprivileged system programs and a discussion of the program development cycle on a RSTS/E system.
- *PDP–11 Programming Card*
  Summarizes, on a pocket-sized folding card, the PDP–11 machine instructions used by the various PDP–11 assembly language processors.
- *RT–11/RSTS/E FORTRAN IV User's Guide*
  Provides the information needed to compile, link, execute, and debug a FORTRAN program under RT–11.
- *PDP–11 FORTRAN Language Reference Manual*
  Describes the form of the basic elements of the FORTRAN language: the FORTRAN statements. The document is a reference manual for the inexperienced as well as the experienced programmer. It is not a tutorial manual.

1.3 Run-Time System Environment

The RSTS/E system places you under the control of the default keyboard monitor after you log in to the system. BASIC–PLUS, BASIC–PLUS–2, DCL, RT11, and RSX are keyboard monitors the system manager can select during system generation. You know what keyboard monitor is the default by the prompt the monitor prints at your terminal:

Ready	BASIC–PLUS
>	RSX
.	RT11
BASIC2	BASIC–PLUS–2
$	DIGITAL Command Language (DCL)

At this command level, you can run any of the RT11 utilities described in this manual. Each of the keyboard monitors understands the RUN command and passes the command line on to the RT11 utility program for processing. However, before using the utilities, you must learn the account in which they reside. Generally, you find MACRO.SAV, LINK.SAV, LIBR.SAV, and PAT.SAV in the system library account [1,2]. That is the account to which they are assigned on the RSTS/E distribution kit. Unless your system manager has moved them, you should be able to call the utilities from that account. Check with your system manager or simply try to run the utilities from the library account.

1.3.1 Running the Utility Programs

To call the utilities, respond to the keyboard monitor prompt by typing a command in the form:

```
RUN $utility RET
*
```

The dollar sign ($) indicates that the program resides in the RSTS/E system library account [1,2]. Utility represents the name of the program you want to run. When it is ready for you to enter a command string, the utility prints an asterisk (*) prompt. If you learn from the system manager that the utilities have been moved, then include the new account number in the file specification:

```
RUN [1,3]<utility>RET
*
```

In this case, the system searches account [1,3] and runs the utility, which prompts you for command input. (Your system manager may also have moved the utilities to a device other than one in the public structure. In that case, you will also need to include the device name along with the file specification of the utility.)

If you prefer to work under the RT11 run-time system, type:

```
RUN $SWITCH
Keyboard monitor to switch to? RT11 (RET)

.
```

RSTS/E changes your default keyboard monitor to RT11 if it was not the default already. To indicate the switch into RT11 has been made, the RT11 keyboard monitor dot prompt (.) appears on your teminal, showing its readiness to accept input from the keyboard.

1.3.2 Running the Utilities in DCL

You may want to run the RT11 utilities under the control of the DIGITAL Command Language (DCL) keyboard monitor. This command environment is compatible with other DIGITAL computer systems, allowing users of these systems to use RSTS/E without having to learn the RSTS/E command environment. For those not familiar with DCL, the language consists of words that suggest the operations performed. This makes using the computer less difficult and thus creates a more productive working environment.

To use DCL, you must run the RSTS/E SWITCH program (if DCL is not already your keyboard monitor). After you type "DCL" in response to the single program prompt, SWITCH places your terminal under DCL's control. The dollar prompt ($) printed by the DCL keyboard monitor indicates that you can begin to enter commands. DCL syntax is explained for each of the utilities described in following chapters. If you need more information about DCL, refer to the *RSTS/E DCL System User's Guide.*

1.4 Command String Specifications

A utility program prompts you with an asterisk prompt (*) when it is ready to accept a command string. (Pressing the RETURN key without entering a command line causes the utility to print its name and version number.) The first command string you enter has the general format:

```
output = input
```

The output and input side of your response have the following meaning:

output — Represents the output file specifications. You can in general specify up to three file names, although some utilities allow only one or two. If you do not include an output file specification, the utility chooses a default.

input — Represents the input file specifications. You can choose not to include an input file specification, but if you do, six is the maximum.

Use the following RSTS/E file specification format when responding to command prompts issued by the utilities:

dev:[p,pn]filename.type/pr[otect]:nn[/switch(es)]

dev: Identifies the device on which the file is stored or is to be written. You create a valid dev: field by combining a two-character device code with an optional unit number. Each dev: field must end with a colon (:). If, for example, you have a file on an RP06, on device unit number 1, the device field would be DB1:. Refer to the *RSTS/E System User's Guide* for a list of valid RSTS/E devices and their corresponding device codes. The logical DK: is the default; refer to Section 1.5.

[p,pn] Represents the account that contains the file you want to access. The number in square brackets ([ ]) consists of both a project (p) and a programmer number (pn), with each being assigned a decimal value from 0 to 254. Together they allow you to differentiate one user's files from another. The default [p,pn] is your own account. Examples of valid project-programmer numbers are [1,210] and [200,63].

filename Can have as many as six characters. It has no default and thus must be specified whenever you run a program.

file type Can contain up to three characters. It describes the type of data in a file. Some examples of file types that appear in this manual are:

.DAT FORTRAN IV data file

.FOR FORTRAN IV source file

.LLD Library listing file

.LST Listing file (MACRO, FORTRAN IV, and LIBR output)

.MAC MACRO source file (MACRO input, LIBR input and output)

.MAP Map file (linker output)

.OBJ Relocatable binary file (MACRO or FORTRAN IV output, LINK input, LIBR input and output)

.SAV Executable IV program file

.SML System MACRO library

.STB Symbol table file in object format containing all global symbols resolved during a link

.TMP Temporary cross-reference file, for communication from MACRO to CREF

/PR:nn Uses decimal values (such as 60) to restrict or permit access to a file. The code or combination of codes determines the degree of restriction. Protection codes have effect only when given to output files. Refer to the *RSTS/E System User's Guide* for more complete information on protection codes.

/switch Changes the way a utility program works. You will find the list of switches for a particular utility in the appropriate chapter.

The *RSTS/E System User's Guide* contains a more complete description of the RSTS/E file specification.

1.5 Logicals DK: and SY:

On an RT–11 system, programs and data files are often kept on separate devices, especially on very small systems. In RT–11, SY: is the logical name given the disk containing the system files. It is also the default boot device. The logical DK: refers to the disk that contains user work files and represents the disk to which the utility programs default when no device name is included in a file specification. The system also defaults to this device to find an executable file when you use the RUN command.

On RSTS/E, these two logicals exist and generally have the same meanings. (The exception is the RUN command, which searches the RSTS/E public structure by default.) The logical SY: usually refers to the disk(s) in the public structure that contain both the system files and user work files.

On RSTS/E, DK: is usually assigned to the public structure. Thus, DK: on a RSTS/E system may be synonymous with SY:, even though the utilities perform their input and output to and from DK:. This happens as long as you have not used the RSTS/E ASSIGN command to point DK: to another device. For example:

```
ASSIGN DB1:DK:
```

This command makes DB1: the disk that the RT11 utilities choose when you do not include a device name in your file specifications (which default to DK:).

1.6 Error Messages in the Appendix

Here are a few important facts you should know about the error messages described in this manual:

1. To make accessing error messages easier, all messages are in Appendix B. (Only Chapter 2 on the MACRO assembler contains any error information; input and output error messages for MACRO, however, are in Appendix B.)
2. You should read the introduction to Appendix B before using the RT11 utilities. It contains information that will help you interpret and correct the error conditions generated by the RT11 utilities.

Chapter 2
MACRO–11 Program Assembly

This chapter describes how to assemble MACRO–11 programs under RSTS/E.

Output from the MACRO–11 assembler includes any or all of the following:

- A binary object file — the machine-readable logical equivalent of the MACRO–11 assembly language source code
- A listing of the source input file
- A cross-reference file listing
- A table of contents listing
- A symbol table listing

To use the MACRO–11 assembler, you should understand how to:

1. Start and stop the MACRO–11 assembler (including how to format command strings to specify files MACRO–11 uses during assembly)
2. Assign temporary work files to nondefault devices, if necessary
3. Use file specification switches to override file control directives in the source program
4. Interpret error messages

The following sections describe these topics.

2.1 Running the MACRO–11 Assembler

This section describes how to run the MACRO–11 assembler:

- With the RUN command or a Concise Command Language (CCL) command
- From the DIGITAL Command Language (DCL) keyboard monitor

Use the method that best suits your needs and the prevailing conditions (default keyboard monitor for example) at your installation.

2.1.1 Running MACRO with the RUN Command or a CCL Command

To run the MACRO–11 assembler from the system library account [1,2], type:

```
RUN $MACRO(RET)
*
```

If your system manager has installed MACRO–11 as a Concise Command Language (CCL) command (for example, MACR–O), then you can also run the assembler with:

```
MACRO(RET)
*
```

In either case, MACRO–11 prints an asterisk prompt on your terminal. The assembler is now ready to accept command string input in the form:

output-filespec(s) = input-filespec(s)

As an alternative, if MACRO is a CCL on your system, you can enter a whole command line in the form:

MACRO output-filespec = input-filespec

The more detailed format for the command string is:

obj,list,cref/s:arg = sourcei,...,sourcen/s:arg

obj — The file specification of the binary object file that the assembly process produces (the device for this file should not be KBn: or LPn:).

list — The file specification of the assembly and symbol listing that the assembly process produces.

cref — The file specification of the CREF temporary cross-reference file that the assembly process produces. The dev:cref specification is necessary only if you must place the cref work file on a disk other than the default (which is DK:, or WF: if defined).

/s:arg — A set of file specification switches and arguments. (Section 2.3 describes these switches and associated arguments.)

sourcei — The file specifications for a MACRO–11 source file or MACRO library file. These files contain the MACRO language programs to be assembled. You can specify as many as six source files.

The complete format for a file specification is:

```
dev:[p,pn]filename.type/PR[otect]:nn
```

The default device for the files used by MACRO is DK:. On RSTS/E, DK: is usually synonymous with SY:, but you can assign DK: to another device and explicitly specify a device in your command string to MACRO. If you submit the listing file directly to a printing device, such as KBn: or LPn:, you can abbreviate the file specification for a listing file to include only the device code. In addition, RSTS/E allows you to use the default for the project-programmer number or the protection code, as described in Section 1.4. The format for an input (source) file specification is the same except for the protection code, which you can omit. MACRO ignores it on input files.

For example, the following command string calls for an assembly that uses one source file (SRC.MAC) plus the system MACRO library to produce an object file BINF.OBJ and a listing. The listing goes directly to the line printer.

```
*DK:BINF,OBJ,LP:=DK:SRC,MAC
```

All output file specifications are optional. MACRO does not produce an output file unless the command string contains a specification for that file. If you do not include an output file specification, you can omit the equal sign (=).

The system determines the file type of an output file specification by its position in the command string. Use commas in place of files you wish to omit. For example, to omit the object file, you must begin the command string with a comma. You need not include a comma after the final output file specification. The following command produces a listing, including cross-reference tables, but not binary object files:

```
*,LP:/C=MAIN,MAC
```

The next command produces an object file but no listing file or cross-reference listings; input files are on DK: (which is usually the system device, as described above):

```
*DB1:[240,129]BINF,OBJ<40>=SRC1,MAC,SRC2,MAC
```

MACRO assumes certain default values when you do not specify devices and file types in the command string. Table 2–1 lists these default values for each file specification.

Table 2–1: Default File Specification Values

File	Default Device	Default File Name	Default File Type
Object	DK:	None; must specify	.OBJ
Listing	Same as for object file	None; must specify	.LST
Cref	DK: (CF: if assigned)	CREF*	.TMP
First source	DK:	None; must specify	.MAC
Additional source	Same as for preceding source file	None; must specify	.MAC
System MACRO Library	DK:	SYSMAC	.SML
User MACRO Library	DK: if first file, otherwise same as for preceding source file	None; must specify	.MAC

* The default file name is DK:CREF.TMP if (1) you do not include a file specification or (2) you do not include a file name but do include the /C switch. Otherwise, you must enter a file name.

If you type RUN $MACRO and receive the asterisk prompt but do not enter a command string, you can exit the MACRO–11 assembler by typing CTRL/Z. This returns you to your keyboard monitor prompt. After you enter a command string and press the RETURN key (thus beginning an assembly), you can halt the assembly process at any time by typing CTRL/C. Control returns to the MACRO–11 asterisk prompt. Enter another command or type CTRL/Z to exit the MACRO–11 assembler and return control to your keyboard monitor.

To restart the assembly process, type RUN $MACRO in response to the keyboard monitor prompt.

2.1.2 Running MACRO in DCL

You can use the RUN command to invoke the MACRO–11 assembler when you are under the control of the DCL keyboard monitor. Just follow the procedures described in the previous section. But you can also run MACRO from DCL using DCL syntax rules and procedures. This section describes how to use these rules and procedures to execute MACRO programs.

You must first run the RSTS/E SWITCH program to switch control to the DCL keyboard monitor (unless it is already the default):

```
RUN $SWITCH<RET>
```

The SWITCH program asks one question only and then places you in the run-time system you select:

```
Keyboard Monitor? DCL<RET>
```

After you type DCL and press the RETURN key, DCL immediately prints a dollar ($) prompt on your terminal. The DCL keyboard monitor is now ready to accept command input.

At the prompt, type a single line command in the form:

```
$ MAC[RO]/RT11 input-filespeci+...+n/OBJECT[=obj-filespec]/LIST=[list-filespec]
```

The command line accepts up to six input file specifications (i through n). DCL syntax requires you to use a plus (+) sign instead of a comma (,) whenever you specify more than one input file. If you do not include an object file, MACRO creates one by default and gives it the same name as the first input file. The assembler also automatically creates a list file if you omit that specification. The default file types are .OBJ for the object file and .LST for the list file. Have MACRO create both files by typing a command line as follows:

```
$ MACRO/RT11 MAIN.MAC
```

MACRO creates the object file MAIN.OBJ and the list file MAIN.LST and places the files in your account on the system disk (SY:).

If you do not want MACRO to create an object file, you must use the /NOOBJ switch. Similarly, the /NOLIST switch tells MACRO not to create a list file. For example:

```
$ MACRO/RT11 MAIN.MAC/NOLIST
```

MACRO creates the object file MAIN.OBJ but does not create a list file.

When you need to see if an old MACRO program assembles, and you do not want to get an object or a list file (at least the first time you run the program), attach both the /NOOBJ and /NOLIST switches:

```
$ MACRO/RT11 OLDPRG.MAC/NOOBJ/NOLIST
```

When you want to give the object or list file a name other than the one MACRO assigns by default, enter the specific file name:

```
$ MACRO/RT11 MAIN.MAC/OBJECT=MODULE/LIST=MODLST
```

This command line tells MACRO to create the object file MODULE.OBJ and the list file MODLST.LST. If you do not include file types, the assembler assigns them for you. Even though an input file may be on a disk other than the system disk, DCL always places the object and list files on the system disk in your account, unless you specify otherwise. To place an object and a list file on DB0:, for example:

```
$ MACRO/RT11 DB0:MAIN.MAC/OBJECT=DB0:MODULE/LIST=DB0:MODLST
```

MACRO places both files in your account on DB0:.

If you include an object file but not a list file specification, DCL assigns the list file the same name, device, and account as the object file. For example:

```
$ MACRO/RT11 MAIN.MAC/OBJECT=MODULE
```

Your account on the system disk now contains the list file MODULE.LST as well as the object file MODULE.OBJ. MACRO uses the default file types .LST and .OBJ unless you enter your own.

When you attach the /LIBRARY switch to the particular input file specification, MACRO can tell that an input file is a library. For example:

```
$ MACRO MAIN.MAC+SECOND.LIB/LIBRARY/OBJECT=MODULE
```

The /LIBRARY switch marks the input file SECOND.LIB as a library. You must append the switch to the file you wish to mark. DCL allows you to specify up to five library files in a single command string. If you need to specify an object or list file, attach them as usual after the input file specifications and thus after any /LIBRARY switches you may have included. Note that you must use the plus (+) sign to separate input file specifications.

To abort the MACRO assembler and return to the DCL dollar prompt, type CTRL/C.

2.2 Temporary Work File

Some assemblies need more symbol table space than available memory contains. When this occurs, the system automatically creates a temporary work file called WRK.TMP to provide extended symbol table space.

The default device for WRK.TMP is DK:. To make the system assign a different device, enter the following command in response to the RT11 keyboard monitor prompt:

```
.ASSIGN dev: WF
```

Device (dev:) represents the physical name of a disk. MACRO creates WRK.TMP on this device. The period before the ASSIGN command indicates the command was typed in response to the RT11 keyboard monitor prompt (.). On RSTS/E, you switch into another run-time system, such as RT11, by running the SWITCH program.

2.3 File Specification Switches

At assembly time, you may need to override certain MACRO directives appearing in the source programs. You may also need to tell MACRO–11 how to handle certain files during assembly. You can satisfy these needs by including special switches in the MACRO–11 command string in addition to the file specifications. Table 2–2 lists the switches and describes their effects.

Table 2–2: File Specification Switches

Switch	Usage
/L[:arg]*	Listing control, overrides source program directives .LIST and .NLIST
/N[:arg]*	Listing control, overrides source program directives .LIST and .NLIST
/E:arg**	Object file function enabling, overrides source program directives .ENABL and .DSABL
/D:arg**	Object file function disabling, overrides source program directives .ENABL and .DSABL
/M	Indicates input file is a MACRO library file
/C[:arg]	Requests or controls contents of cross-reference listing
/P:arg	Specifies whether input source file is to be assembled in pass 1 or pass 2 only, rather than both passes

* Both /L and /N disable .LIST and .NLIST for the argument(s) specified; however, /L turns it on, and /N turns it off.

** Both /E and /D disable .ENABL and .DSABL for the argument(s) specified; however, /E turns it on, and /D turns it off.

The /M and /P switches affect only the particular source file specification to which they are directly appended in the command string. Other switches are unaffected by their placement in the command string. The /L switch, for example, affects the listing file, regardless of where in the command string you place it.

The following sections describe how to use the file specification switches.

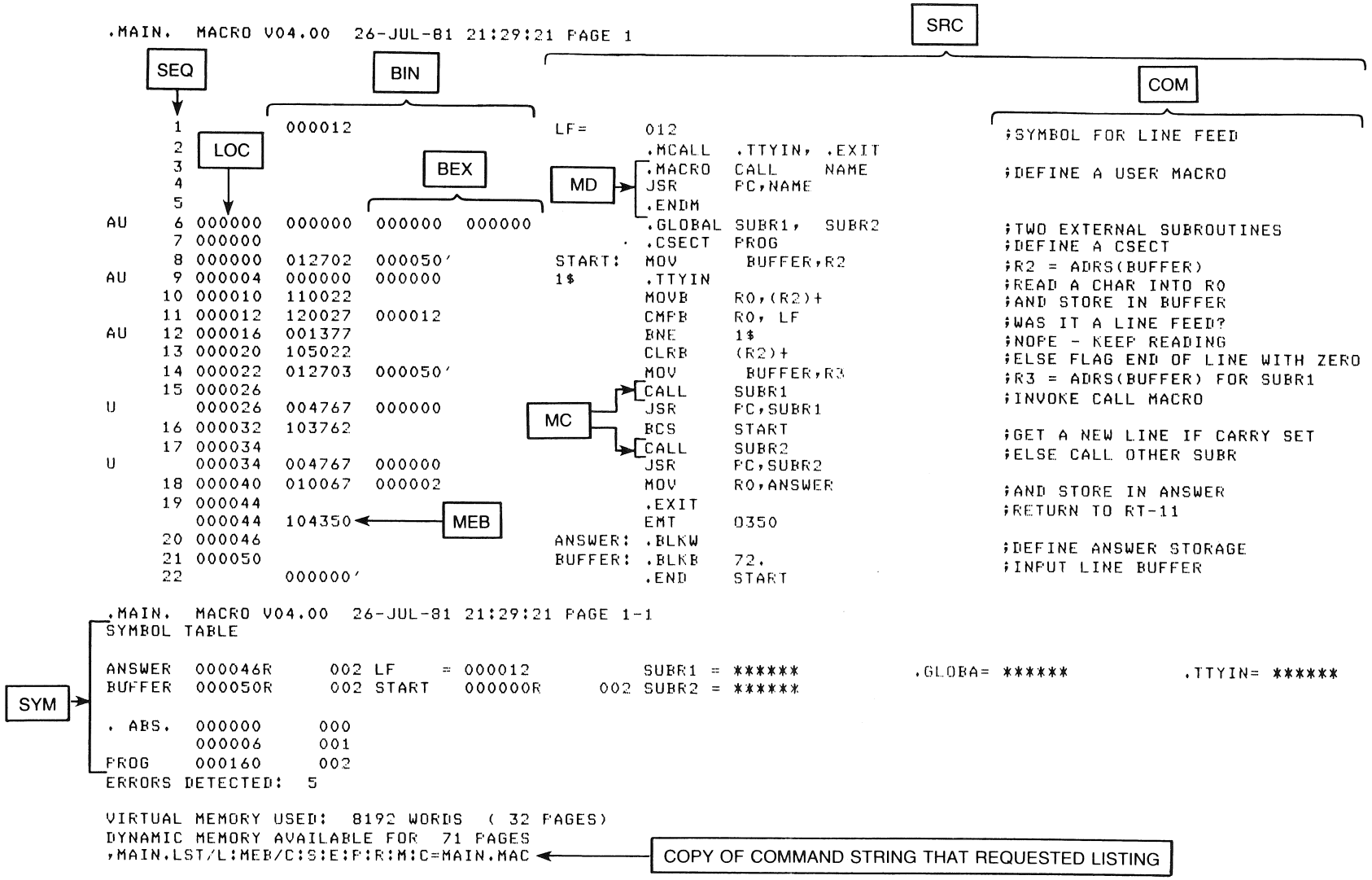
2.3.1 Listing Control Switches

Use the /L:arg and /N:arg switches with the set of arguments in Table 2–3 to control the content and format of assembly listings. At assembly time, you can override the arguments of .LIST and .NLIST directives in the source program. If you use the /L:arg or /N:arg switch, the directives .LIST and .NLIST cannot control that particular switch in the source. For example, specifying /N:CND disables the listing of unsatisfied conditionals even if .LIST CND is present in the source.

Figure 2–1 shows an assembly listing of a small program. This listing labels each important feature with the mnemonic name that determines its appearance on the listing; the argument SEQ, for instance, controls the appearance of the source line sequence numbers.

The system has default settings for the /L and /N switches when you do not include arguments:

- The /L switch without an argument causes the system to ignore .LIST and .NLIST directives that have no arguments.
- The /N switch without an argument causes the system to list only the symbol table, the table of contents, and error messages.



MK-00431-00

Figure 2–1: Sample Assembly Listing

The following example lists binary code throughout the assembly using the 132–column line printer format and suppresses the symbol table listing:

```
*I,LP:/L:MEB/N:SYM=FILE
```

Table 2–3: Arguments for /L and /N Switches

Argument	Default	Listing Control
SEQ	List	Source line sequence number
LOC	List	Address location counter
BIN	List	Generated binary code (includes BEX)
BEX*	List	Binary extensions
SRC	List	Source code
COM	List	Comments
MD	List	Macro definitions, repeat range definitions
MC	List	Macro calls, repeat range expansion
ME	No list	Macro expansions (includes MEB)
MEB	No list	Macro expansion binary code
CND	List	Unsatisfied conditionals, .IF and .ENDC statements
LD	No list	List control directives with no arguments
TOC	List	Table of Contents
TTM	No list	132–column line printer format when not specified, terminal mode (80–column mode) when specified
SYM	List	Symbol table

* This option applies to the listing of assembled binary code. There is room on a listing line to display three octal words (one if TTM is set) of assembled code. If you assemble a source statement that assembles to more than three words, only the first three are listed if .NLIST BEX is in effect. If .LIST BEX is in effect, MACRO uses additional lines to list all assembled words.

2.3.2 Function Control Switches

The /E:arg and /D:arg switches allow you to enable or disable functions at assembly time and thus influence the form and content of the binary object file. These functions override .ENABL and .DSABL directives in the source program; if you specify the /E:arg or /D:arg switch, the .ENABL arg or .DSABL arg directive no longer affect the particular argument that may occur in the source.

Table 2–4 summarizes the acceptable /E and /D function arguments, their normal default status, and the functions they control.

Table 2–4: Arguments for /E and /D Switches

Argument	Default Mode	Function
ABS	Disable	Produces output in paper tape absolute binary format instead of a standard object file.
AMA	Disable	Assembles all relative addresses as absolute addresses. Replaces all uses of relative addressing mode (mode 67) by absolute addressing (mode 37).
CDR	Disable	Ignores all source information beyond column 72.
CRF	Enable	Allows cross-reference listing. Disabling this function inhibits CREF output even if switch /C is active.
FPT	Disable	Truncates floating point values (instead of rounding).
GBL	Disable	Treats undefined symbols as globals.
LC	Disable	Allows lowercase ASCII source input.
LSB	Disable	Allows local symbol block (not recommended in /E:arg or /D:arg).
PNC	Enable	Allows binary output.
REG	Enable	Automatically defines register mnemonics if enabled. You should set or clear the REG argument at the beginning of the source module.

Use either the function control or listing control switch and arguments at assembly time to override corresponding listing or function control directives in the source program. For example, assume that the source program contains the following sequence:

```
.NLIST MEB
  .
  . (MACRO references)
  .
.LIST MEB
```

In this example, you disable the listing of MEB (Macro Expansion Binary) code for some portion of the code and subsequently resume MEB listing. If you indicate /L:MEB in the assembly command string, however, the system ignores both the .NLIST MEB and the .LIST MEB directives. This enables MEB listing throughout the program.

The *PDP–11 MACRO Language Reference Manual* contains more information on the arguments for both the listing control and function control switches.

2.3.3 Macro Library File Designation Switch

The /M switch is meaningful only if you append it to a source file specification. It designates the source file as a macro library.

If the command string does not include the standard system macro library SYSMAC.SML, the system automatically includes it as the last source file in the command string.

When the assembler encounters an .MCALL directive in the source code, it searches macro libraries according to their order of appearance in the command string. When it locates a macro record whose name matches that given in the .MCALL, it assembles the macro as indicated by that definition. Thus, if two or more macro libraries contain definitions of the same macro name, the macro library that appears leftmost in the command string takes precedence.

For example:

```
*<output file specification>=ALIB,MAC/M,BLIB,MAC/M,XIZ
```

Assume that each of the two macro libraries, ALIB and BLIB, contains a macro called .BIG, but with different definitions. Then, if source file XIZ contains a macro call .MCALL .BIG, the system includes the definition of .BIG in the program as it appears in the macro library ALIB.

Moreover, if macro library ALIB contains a definition of a macro called .READ, that definition of .READ overrides the standard .READ macro definition in SYSMAC.SML.

2.3.4 Cross-Reference (CREF) Table Generation Switch

A cross-reference (CREF) table lists all or a subset of the symbols in a source program, identifying the statements that define and use symbols.

2.3.4.1 Obtaining a Cross-Reference Table — To obtain a CREF table, you must include the /C[:arg] switch in the command string. Usually you include the /C[:arg] switch with the assembly listing file specification. However, you can place it anywhere in the command string.

If the command string does not include a CREF file specification, the system automatically generates a temporary file on the system device (DK:). (See Section 2.3.4.2.) To store the temporary CREF file on a device other than DK:, you must include the dev:cref field in the command string or assign CF: to another device.

A complete CREF listing contains the following six sections:

1. A cross reference of program symbols; that is, labels used in the program and symbols defined by a direct assignment statement.
2. A cross reference of register symbols. These normally include the symbols R0, R1, R2, R3, R4, R5, SP, and PC, unless the REG function has been disabled through a .DSABL REG directive or the /D:REG switch (in which case registers are treated as normal symbols and show up in the first symbol section).
3. A cross reference of macros; that is, those symbols defined by .MACRO and .MCALL directives.
4. A cross reference of permanent symbols; that is, all operation mnemonics and assembler directives.

5. A cross reference of program sections. These symbols include the names you specify as operands of the .CSECT, .ASECT, or .PSECT directive. Also included are the default program sections produced by the assembler, the blank p-sect, and the absolute p-sect, .ABS.

6. A cross reference of errors. MACRO detects certain types of programming and syntax errors in your source code and flags them with a one-letter error code. In the error section of a CREF table, MACRO groups and lists the errors by type.

   The one-letter error codes also appear in the assembly listings.

   Section 2.4 describes the one-letter codes MACRO prints to identify programmer level errors; Appendix B lists the input-output level error messages that may, for example, result from specifying an incorrect command string or from problems with I/O devices.

You can include any or all of these six sections on the cross-reference listing by specifying the appropriate arguments with the /C switch. Table 2–5 contains a description of these arguments.

Table 2–5: /C Switch Arguments

Argument	CREF Section
S	User-defined symbols
R	Register symbols
M	MACRO symbolic names
P	Permanent symbols, including instructions and directives
C	Control and program sections
E	Error code grouping

NOTE

Specifying /C with no arguments is equivalent to specifying /C:S:M:E:. Except for that special case, you must explicitly request each CREF section by including its arguments. The /C switch must be used to produce a cross-reference file even if the command string includes a CREF file specification.

2.3.4.2 Handling Cross-Reference Table Files — When you request a cross-reference listing with the /C switch, the system generates a temporary file, DK:CREF.TMP.

If device DK: is write-locked or if it contains insufficient free space for the temporary file, you can allocate another device for the file. To allocate another device, specify a third output file in the command string; that is, include a dev:cref specification. (You must still include the /C switch to control the form and content of the listing. The dev:cref specification is ignored if the /C switch is not also present in the command string.)

The system then uses the dev:cref file instead of DK:CREF.TMP and deletes it automatically after producing the CREF listing.

The following command string causes the system to use DB2:TEMP.TMP as the temporary CREF file:

```
*,LP:,DB2:TEMP.TMP/C=SOURCE
```

Another way to assign an alternate device for the CREF.TMP file is to enter the following command before typing RUN $MACRO:

```
.ASSIGN dev: CF
```

This method is preferred if you intend to do several assemblies, because it relieves you from having to include the dev: cref specification in each command string. If you enter the ASSIGN dev: CF command (there should be a space between dev: and CF) and later include a CREF file specification in a command string, the specification in the command string is in effect for that assembly only.

The system lists requested cross-reference tables following the MACRO assembly listing. Each table begins on a new page. (Figure 2–2 combines the tables to save space.)

Figure 2–2: Cross-Reference Table

```
.MAIN.  MACRO V04.00  26-JUL-81 21:29:21 PAGE S-1
CROSS REFERENCE TABLE (CREF V04.00 )

.GLOBA    1-6
.TTYIN    1-9
ANSWER    1-18*     1-20
BUFFER    1-8       1-14      1-21
LF        1-1       1-11
START     1-8       1-16      1-22
SUBR1     1-6       1-15
SUBR2     1-6       1-17

.MAIN.  MACRO V04.00  26-JUL-81 21:29:21 PAGE R-1
CROSS REFERENCE TABLE (CREF V04.00 )

PC        1-15*     1-17*
R0        1-10      1-11      1-18
R2        1-8*      1-10*     1-13*
R3        1-14*

.MAIN.  MACRO V04.00  26-JUL-81 21:29:21 PAGE M-1
CROSS REFERENCE TABLE (CREF V04.00 )

.EXIT     1-2       1-19
.TTYIN    1-2
CALL      1-3       1-15      1-17

.MAIN.  MACRO V04.00  26-JUL-81 21:29:21 PAGE C-1
CROSS REFERENCE TABLE (CREF V04.00 )

          0-0
. ABS.    0-0
PROG      1-7

.MAIN.  MACRO V04.00  26-JUL-81 21:29:21 PAGE E-1
CROSS REFERENCE TABLE (CREF V04.00 )

A         1-6       1-9       1-12
U         1-6       1-9       1-12      1-15      1-17
```

The system prints symbols and also symbol values, control sections, and error codes, if applicable, beginning at the left margin of the page. References to each symbol are listed on the same line, left-to-right across the page. The system lists references in the form p-l; where p is the page in which the symbol, control section, or error code appears, and l is the line number on the page.

A pound sign (#) next to a reference indicates a symbol definition. An asterisk (*) next to a reference indicates a destructive reference—that is, an operation that alters the contents of the addressed location or register.

2.3.5 Assembly Pass Switch

The /P:arg switch is meaningful only if you append it to a source input file specification. You must specify either of two arguments with it: 1 or 2.

The specification /P:1 calls for assembly of the file during pass 1 only. Some files consist entirely of code that is completely assembled at the end of pass 1. Definition files (prefix files containing only symbol definitions) are a good example. At the end of pass 1, all definitions have been processed; they are retained for pass 2 and do not need to be scanned again. (You should not put macro definitions for self-modifying macros in /P:1 files.) Note that listing, cref and object output take place in pass 2, so specifying /P:1 on a file causes it to be omitted from listing and cref, and should not be used on files that generate code. By specifying /P:1 for these files, you can cause MACRO–11 to skip processing of these files through pass 2. In some cases, this procedure can save considerable assembly time.

The specification /P:2 calls for assembly of the file during pass 2 only. (Note: Situations where the /P:2 switch can be meaningfully employed are unusual.)

2.4 MACRO–11 Error Codes and Messages

MACRO can detect errors on two levels: programming and input-output. Section 2.4.1 describes the single character codes that identify MACRO programming level errors. A brief explanation of input-output errors appears in Section 2.4.2. For descriptions of these error messages, refer to Appendix B.

2.4.1 Programming Level Errors

Programming level errors are mistakes in source code syntax or faulty program logic. MACRO indicates an error on this level with a single error code. These codes automatically appear on the assembly listings. Consider the following:

1. MACRO prints programming level error codes on the left margin of the assembly listing, preceding the source line sequence numbers, when the/L:TTM switch or the .LIST TTM directive has not been used.

2. MACRO prints error codes on the assembly listing following a field of six asterisk characters, when you request a listing in terminal, 80–column format (with /L:TTM or .LIST TTM). The source statement containing the error follows on the next line. For example:

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

3. MACRO also prints programming error codes on the cross-reference listing if you specify /C:E in the MACRO command string.

Table 2–6 shows the error codes that might appear on an assembly listing. For more information on error code interpretation and debugging, see the *PDP–11 MACRO–11 Language Reference Manual.*

Table 2–6: MACRO–11 Error Codes

Error Code	Meaning
A	Addressing or relocation error. This message can be generated by any of the following: 1. A branch instruction target that is too far above or below the current statement. Branch targets must be within –128 to –127 (decimal) words of the instruction. (A special case is the branch target for the SOB instruction which must be within 64 decimal words, backward only.) 2. A statement that makes an illegal change to the current location counter. For example, a statement that forces the current location counter to cross a .PSECT boundary generates this message. 3. A statement that contains an invalid address expression. For example, an absolute address expression that has a global symbol, relocatable value, or complex relocatable value generates this message. The directives .BLKB, .BLKW, and .REPT must have an absolute value or an expression that reduces to an absolute value. 4. Separate expressions in the statement that are not separated by commas. 5. A global definition error. If .ENABL GBL is set, MACRO–11 scans the symbol table at the end of the first pass and marks any undefined symbols as global references. If one of these symbols is subsequently defined in the second pass, a general addressing error occurs. 6. A global assignment statement that contains a forward reference to another symbol. 7. An expression that defines the value of the current location counter and contains a forward reference. 8. An illegal argument for an assembler directive. 9. An unmatched delimiter or illegal argument construction.
B	Instruction or word data are being assembled at an odd address. The assembler increments the location counter by 1 and continues.

(continued on next page)

Table 2–6: MACRO–11 Error Codes (Cont.)

Error Code	Meaning
D	Reference was made to a multiply defined nonlocal label.
E	The .END assembler directive at the end of the source input is missing. The assembler supplies a .END statement and completes the current assembly pass.
I	MACRO–11 has detected one or more illegal characters. This often occurs when a line feed does not follow a carriage return, which can easily happen while using a source editor. A question mark (?) replaces each illegal character on the assembly listing, and MACRO–11 continues after ignoring the character.
L	An input line is longer than 132 characters. In particular, this error occurs when the expansion of a macro causes excessive substitution of real arguments for dummy arguments.
M	A nonlocal label is the same as an earlier label (multiple definition of a label). For example, two labels whose first six characters are identical can generate this error. The error occurs on both definitions of the label.
N	A number is not in the current program radix. MACRO–11 processes this number as a decimal value.
O	Op-code error. Directive is out of context. Exceeding the permitted nesting level for conditional assemblies causes this error. Attempting to expand a macro that remains unidentified after a .MCALL search can also generate this message.
P	1. Phase error. The definition or value of a label differs from one assembler pass to the next, or a local symbol occurs more than once in a local symbol block. 2. Program-defined error. Generated if a .ERROR directive is assembled.
Q	Questionable syntax. Missing arguments, too many arguments, or an incomplete instruction scan generates this error message.
R	Register-type error. An invalid use of or reference to a register has been made, or an attempt has been made to redefine a standard register symbol without first issuing the .DSABL REG directive. For example, this error can be caused by illegal register numbers (such as FOO = %10) or by the use of a nonregister in a place where a register is required, such as MOV 10(FOO),BAR.
T	Truncation error. The expression is too large for the context. An expression generated more than 8 significant bits during the use of the .BYTE directive or trap (EMT, TRAP, and so forth) instruction or more than 6 bits on a SOB or MARK instruction or improper addressing mode — not register or registered deferred — in a floating point instruction.
U	Undefined symbol. An undefined symbol was encountered during the evaluation of an expression; the assembler assigns the undefined symbol a constant zero value.
Z	Incompatible instruction. This message is a warning that the instruction does not behave the same for all PDP–11 hardware configurations.

2.4.2 Input-Output Level Error Messages

Input-Output (I/O) level error messages appear when you specify incorrect command strings to MACRO or when problems arise with I/O devices. Error messages of this type have the following format:

```
?MACRO-n-message
```

Refer to Appendix B for the description of input-output level error messages that MACRO produces.

Chapter 3
Linker (LINK)

The linker (LINK) converts object modules to a format suitable for loading and execution. This chapter describes how to perform the link operation. The organization of this chapter is:

Section 3.1 Overview of the Linking Process explains some of the terms used exclusively in this chapter, the functions of the linker, how the linker structures your program to prepare it for execution, and the communication links between modules within your program.

Section 3.2 Running and Using the Linker describes how to run the linker from the keyboard monitors available to you on your RSTS/E system and describes the syntax for input and output file specifications. This section also summarizes the switches you use to adjust the output of the link operation.

Section 3.3 Input and Output lists and describes the files that are valid for input to and output from the linker. This section also explains how to use library files, and how the linker processes library files, which you create with the librarian utility (see Chapter 4).

Section 3.4 Creating an Overlay Structure describes how to design and implement overlay structures for your programs. This section provides descriptions and illustrations of how overlaid programs work and how they reside in memory.

Section 3.5 Switch Descriptions lists and describes the switches you can use with the linker.

Section 3.6 Linker Prompts lists and explains the prompts the linker prints at the terminal after you enter a command line.

3.1 Overview of the Linker Process

This chapter uses the following terms:

program section	A named, contiguous unit of code (instructions or data) that is considered an entity and that can be relocated separately without destroying the logic of the program. Also known as p-sect.
object module	The primary output of an assembler or compiler, which can be linked with other modules and loaded into memory as an executable program. The object module is composed of the relocatable machine language code, relocation information, and the corresponding global symbol table defining the use of the symbols within the program. Also known as a module.
load module	A program (in a format) ready for loading and executing.
library file	A file, generated by the librarian, containing one or more relocatable object modules that can be incorporated into other programs.
library module	A module from a library file.
root segment	The segment of an overlay structure that, when loaded, remains resident in memory during the execution of a program. Also known as the root.
overlay segment	A section of code treated as a unit that can overlay code already in memory and be overlaid by other overlay segments when called from the root segment or another overlay segment. Also known as an overlay.
global symbol	A global value or global label.
low memory	Physical memory from 0 to 28K words.

3.1.1 What the Linker Does

When the linker processes the loaded object modules, it:

- Relocates your program module and assigns absolute addresses
- Links the modules by correlating global symbols that are defined in one module and referenced in another
- Creates the initial control block for the linked program that the RUN command uses
- Creates an overlay structure, if specified, and includes the necessary run-time overlay handler and tables

- Searches the library files you specify to locate unresolved global symbols
- Produces a load map, if specified, that shows the layout of the load module
- Produces a symbol table definition file, if specified

The linker needs to make two passes over the input modules. During the first pass it constructs the symbol table, which includes all program section names and global symbols in the input modules. The linker then scans the library files to resolve undefined global symbols. It includes from the libraries only those modules that are required to resolve undefined global symbols. During the second pass, the linker reads in object modules, performs most of the functions listed above, and produces the load module.

3.1.2 How the Linker Structures the Load Module

When the linker processes the assembled or compiled object modules, it creates a load module, in which it has assigned all absolute addresses, has created an absolute section, and has allocated memory for the program sections.

3.1.2.1 Absolute Section — The absolute section is often called the ASECT because the assembler directive .ASECT allows information to be stored there. The absolute section appears in the load map with the name . ABS. and is always the first section in the listing. The absolute section ends at the assigned "base" address (by default octal 1000) and contains:

- A system communication area
- The user stack

The system communication area resides in locations 0–377 and contains data the linker uses to pass program control parameters and a memory usage bitmap. Section 3.3.3 provides a detailed description of each location in the system communication area.

The stack is an area that a program can use for temporary storage and subroutine linkage. General register 6, the stack pointer (SP), references the stack.

3.1.2.2 Program Sections — The program sections (p-sects) follow the absolute section. The set of attributes associated with each p-sect controls the allocation and placement of the section within the load module. The p-sect, as the basic unit of memory for a program, has:

- A name by which it can be referenced
- A set of attributes that define its contents, mode of access, allocation, and placement in memory
- A length that determines how much storage is reserved for the p-sect

You create p-sects by using a COMMON statement in FORTRAN or the .PSECT (or .CSECT) directive in MACRO. You can use the .PSECT directive to attach attributes to the section. (The .CSECT directive automatically supplies a fixed set of attributes.) Note that the attributes that follow the p-sect name in the load map are not part of the name; only the name itself distinguishes one p-sect from another. You should make sure, then, that p-sects of the same name that you want to link together also have the same attribute list. If the linker encounters p-sects with the same name but with different attributes, it prints a warning message and uses the attributes from the first time it encountered the p-sect.

Program Section Attributes

The linker collects from the input modules any references to a p-sect and combines them in a single area of the load module. The attributes, which are listed in Table 3–1, control the way the linker collects and places this unit of storage.

Table 3–1: P-sect Attributes

Attribute	Value	Explanation
access-code*	RW	Read/Write – data can be read from, and written into, the p-sect.
	RO	Read Only – data can be read from, but cannot be written into, the p-sect.
type-code	D	Data – the p-sect contains data, concatenated by byte.
	I	Instruction – the p-sect contains either instructions, or data and instructions, concatenated by word. (That is, each contribution to an I-type p-sect is rounded up to an even length.)
scope-code	GBL	Global – the p-sect name is recognized across segment boundaries. If all contributions to this p-sect are in a single segment, the p-sect is allocated in that segment. Otherwise, it is allocated in the root. In that case, all contributions are combined into the root regardless of which segment they occurred in.
	LCL	Local – the p-sect name is recognized only within each individual segment. The linker allocates storage for the p-sect in each segment from contributions within that segment only.
reloc-code	REL	Relocatable – the base address of the p-sect is relocated relative to the virtual base address of the program.
	ABS	Absolute – the base address of the p-sect is not relocated. It is always 0.
alloc-code	CON	Concatenate – all allocations to a given p-sect name are concatenated. The total allocation is the sum of the individual allocations.
	OVR	Overlay – all allocations to a given p-sect name overlay each other. The total allocation is the length of the longest individual allocation.

* Ignored by the linker

The scope-code is meaningful only when you define an overlay structure for the program. In an overlaid program, a global section is known throughout the entire program. Object modules contribute to only one global section of the same name. If two or more segments contribute to a global section, then the linker allocates that global section to the root segment of the program. In contrast to global sections, local sections are known only within a particular program segment. Because of this, several local sections of the same name can appear in different segments. Thus, several object modules contributing to a local section do so only within each segment. An example of a global section is named COMMON in FORTRAN. An example of a local section is the default blank section for each macro routine.

The alloc-code determines the starting address and length of memory allocated by modules that reference a common p-sect. If the alloc-code indicates that such a p-sect is to be overwritten, the linker stores the allocations from each module starting at the same location in memory. It determines the total size from the length of the longest contribution to the p-sect. Each module's allocation of memory to a location overwrites that of a previous module. If the alloc-code indicates that a p-sect is to be concatenated, the linker places the allocations from the modules one after the other in the load module; it determines the total allocation from the sum of the lengths of the contributions.

Any data (D) p-sect that contains references to word labels must start on a word boundary. This is done with the .EVEN assembler directive at the end of each module's concatenated p-sect. If this is not done, the program may fail to link, printing the message:

```
?LINK-F-Word relocation error
```

It may also fail at execution time with ?Odd address trap or some other similar message.

The allocation of memory for a p-sect always begins on a word boundary. If the p-sect has the D (data) and CON (concatenate) attributes, all storage contributed by subsequent modules is appended to the last byte of the previous allocation. This occurs whether or not that byte is on a word boundary. For a p-sect with the I (instruction) and CON attributes, however, all storage contributed by subsequent modules begins at the nearest following word boundary.

In any p-sect with the ABS attribute, except for the .ASECT, data is ignored. Thus, in the following example, the first case is treated like the second case:

Case 1

```
.PSECT   FOO,ABS
A::        .WORD       100
```

Case 2

```
.PSECT   FOO,ABS
A::        .BLKW       1
```

That is, the linker pretends the length is zero for allocation purposes. Such p-sects are used as a way of defining absolute global symbol values. (ABS p-sects primarily facilitate symbolic memory layouts.) For each ABS p-sect, address assignment starts at zero. But if CON is used, the individual contributions are assigned one after the other.

The .CSECT directive of MACRO is converted internally by MACRO to an equivalent .PSECT with fixed attributes. An unnamed CSECT (blank section) is the same as a blank PSECT with the attributes RW, I, LCL, REL, and CON. A named CSECT is equivalent to a named PSECT with the attributes RW, I, GBL, REL, and OVR. Table 3–2 shows these sections and their attributes.

Table 3–2: Section Attributes

Section	access-code	type-code	scope-code	reloc-code	alloc-code
CSECT	RW	I	LCL	REL	CON
CSECT name	RW	I	GBL	REL	OVR
ASECT (. ABS.)	RW	I	GBL	ABS	OVR
COMMON/name/	RW	D	GBL	REL	OVR

The names assigned to p-sects are not global symbols; you cannot reference them as such. For example:

```
MOV     #PNAME,R0
```

This statement, where PNAME is the name of a section, is invalid and generates the undefined global error message if no global symbol of PNAME exists. A name can be the same for both a p-sect name and a global symbol. The linker treats them separately.

Program Section Order

The linker determines the memory allocation of p-sects by the order of occurrence of the p-sects in the input modules. Table 3–3 shows the order in which p-sects appear for both overlaid and nonoverlaid files.

Table 3–3: P-sect Order

Nonoverlaid	Overlaid
Absolute (. ABS.)	Absolute (. ABS.)
Blank	Overlay handler ($OHAND)
Named (NAME)	Overlay table ($OTABL) Blank Named (NAME)

If there is more than one named section, the named sections appear in the order that they occur in the input files.

If the size of the blank p-sect is 0, it does not appear in the load map.

3.1.3 Global Symbols: Communication Links Between Modules

Global symbols provide the link, or communication, between object modules. You create global symbols with a double colon (::), with a double equal sign (==), with a double equal sign and a single colon (==:), or by specifying the name of a defined symbol in a .GLOBL directive. If you define the global symbol in an object module (as a label using :: or by direct assignment using ==), other object modules can reference it. If the global symbol is not defined in the object module, it is an external symbol and is assumed to be defined in some other object module. If you use a global symbol as a label in a routine, it is often called an entry point — that is, it is an entry point to that subroutine.

As the linker reads the object modules, it keeps track of all global symbol definitions and references. It then modifies the instructions and data that reference the global symbols. The linker always prints undefined globals on the console terminal after pass 1. If you request a load map on the terminal, undefined globals also appear at the end of the load map.

Table 3–4 shows how the linker resolves global references when it creates the load module.

Table 3–4: Global Reference Resolution

Module Name	Global Definition	Global Reference
IN1	B1 B2	A L1 C1 XXX
IN2	A B1	B2
IN3		B1

In processing the first module, IN1, the linker finds definitions for B1 and B2 and references to A, L1, C1, and XXX. Because no definition currently exists for these references, the linker defers the resolution of these global symbols. In processing the next module, IN2, the linker finds a definition for A that resolves the previous reference and a reference to B2 that can be immediately resolved.

When all of the object modules have been processed, the linker has three unresolved global references remaining: L1, C1, and XXX. A search of the default system library resolves XXX. The global symbols L1 and C1 remain unresolved and are, therefore, listed as undefined global symbols.

The relocatable global symbol, B1, is defined twice and is listed on the terminal as a global symbol with multiple definitions. The linker uses the first definition of such a symbol. An absolute global symbol can be defined more than once without being listed as having multiple definitions, as long as each occurrence of the symbol has the same value.

3.2 Running and Using the Linker

This section describes how to start the linker, how to create a valid command line, and how to use the switches to help you generate the output you need.

3.2.1 Running LINK

There are a number of ways to run the linker program. The method you use depends on the keyboard monitor that interprets your commands. The only exception to this rule is the RUN command; it starts the linker from any of the keyboard monitors that come with your RSTS/E system. To run the linker from the system device with the RUN command, respond to any of the RSTS/E keyboard monitor prompts by typing:

```
RUN $LINK (RET)
*
```

If your system manager has installed the LINK program as a Concise Command Language (CCL) command (such as LIN–K) and your keyboard monitor is not DCL, you can run LINK by typing:

```
LINK (RET)
*
```

In either case, LINK prints an asterisk prompt when the linker is ready to accept a command line. (If you press the RETURN key only, the linker prints its current version number.) When using the LINK CCL command, you have the added option of placing an entire command specification on one line, as in the format:

LINK <output-filespec> = <input-filespec>

After you press the RETURN key, LINK processes the command line and returns you to your keyboard monitor prompt. You must start the linker again if you have other files to link. Section 3.2.2 describes the input and output file specifications you use with the linker program.

As stated before, you can start the LINK program with the RUN command from the DCL keyboard monitor. But there are two commands you can use to run the linker when your keyboard monitor is DCL:

- With the CCL LINK
- Using the LINK/RT11 command

The first command actually consists of the "CCL" prefix and the name of a valid CCL command, in this case LINK. This syntax allows you to use CCL commands under DCL. The second command gives you the option to run the linker in DCL if DCL is your keyboard monitor and you plan to do most of your program development from this environment.

To use the CCL LINK command in DCL, you must type, while at the DCL dollar prompt ($), the three-letter prefix "CCL", a space, and then the CCL command that your system manager has assigned to the LINK program. If the assigned CCL is LIN–K, for example, you would run the linker by typing a command line in the form:

```
$ CCL LINK (RET)
*
```

The linker prints the asterisk prompt indicating that the program is ready to accept command input. As with using a CCL command under the control of a keyboard monitor other than DCL, you can place a complete command file specification on one line:

$ CCL LINK <output-filespec> = <input-filespec>[/switch],...

The linker returns you to the DCL dollar prompt after executing the command. Section 3.2.2 describes the formats to use for the input and output file specifications.

The DCL LINK/RT11 command consists of the command name "LINK" and the switch "/RT11". You must include the /RT11 switch when you need to use the RT11–based linker, unless your system manager has made it the default. The syntax for the command line is:

$ LINK/RT11 <filespeci[,...,filespecn[/switch(es)]]>

As an alternative, you can type LINK/RT11 in response to the DCL dollar prompt, and then press the RETURN key. DCL prints a prompt to which you enter file information in the following form:

$ LINK/RT11
Files: <filespeci[,...,filespecn[/switch(es)]]>

Whether you enter information on one line or in response to the FILES prompt, the form in which you enter information is the same. (Essentially, DCL treats the RETURN reponse like a space, after which it expects a command line.) DCL syntax accepts up to six input files (i thru n). If you do not include a file type, DCL assumes the file type is .OBJ. To suit the type of link you need to perform, use the switches described in Table 3–5.

The following example illustrates how to link FILE1 and FILE2 using the /EX=file and /MAP switches:

```
$ LINK/RT11
Files:      FILE1,FILE2/EX=FILE/MAP

$
```

Table 3–5: LINK/RT11 Command Switches

Switch	Meaning
/EXECUTABLE	Use the name of the first input file for the name of the executable file. If you do not include this switch, DCL assumes /EXECUTABLE as the default. The short form of this switch is /EX.
/EXECUTABLE=file	Use file for the name of the executable file. DCL selects .SAV if you do not include a file type. The short form of this switch is /EX=file.
/NOEXECUTABLE	Do not generate an executable file. You must specify /MAP when using the /NOEXECUTABLE switch. The short form of this switch is /NOEX.
/MAP	Use the name of the executable file (implicit or explicit file declaration) as the name of the map file. If you do not include a file type, DCL uses .MAP. The short form of this switch is /MA.
/MAP=file	Use file as the name of the map. DCL selects .MAP as the file type when you do not include a file type in the file specification. The short form of this switch is /MA=file.
/NOMAP	Do not generate a map file. DCL does not create a map file unless you explicitly include the /MAP switch. This means you do not need to use /NOMAP if you do not want a map file; /NOMAP is the default. You cannot use /NOMAP with /NOEXECUTABLE. DCL accepts /NOMA as the short form of /NOMAP.

The linker creates an executable file FILE.SAV and the map file FILE.MAP and returns to the DCL dollar prompt. You can create the same results with a LINK/RT11 command in the form:

```
$ LINK/RT11 FILE1,FILE2/EX=FILE/MAP

$
```

The linker again creates FILE.SAV and FILE.MAP and returns to DCL command level. Be sure, when you run the linker with a single command line, that you include a space between the /RT11 switch and the beginning of the file specification(s). The switches in Table 3–5 can be located anywhere on the command line but are generally placed at the end. (Refer to the *RSTS/E DCL User's Guide* for more information about the LINK command.)

Type CTRL/C to stop the linker at any time, or use CTRL/Z to stop the linker when it is waiting for input at its asterisk prompt (*). In either case, control returns to your keyboard monitor.

3.2.2 LINK Command Line Specification

If you are under the control of a keyboard monitor other than DCL, the format of the first command string you enter in response to the linker's prompt is:

[bin-filespec],[map-filespec],[sym-filespec] = obj-filespec[/switch...][,...obj-filespec[/switch...]]

The definitions for these file specifications are:

bin-filespec	Represents the file specification assigned to the linker's output load module file
map-filespec	Represents the file specification of the load map output file
sym-filespec	Represents the file specification of the symbol definition file
obj-filespec	Represents the file specifications for an object module, a library file, or a symbol table file created in a previous link
/switch	Represents one of the switches listed in Table 3–7

Chapter 1 describes the correct format for a RSTS/E file specification.

In each file specification above, the device should be a random-access device, except that the output device for the load map file can be any RSTS/E device. If you do not specify a device, the linker uses a default as shown in Table 3–6.

If you do not specify an output file, the linker assumes that you do not want the associated output. For example, if you do not specify the load module and load map (by using a comma in place of each file specification) or if you leave out the output side up to and including the equal sign, the linker prints only error messages, if any occur. Ordinarily, the linker generates at least one load module.

Table 3–6 shows the default values for each specification.

Table 3–6: Linker Defaults

Device		File Name	File Type
Load Module	DK:	None	SAV
Map Output	DK: or same as load module	None	MAP
Symbol Definition Output	DK: or same as previous output device	None	STB
Object Module	DK: or same as previous object module	None	OBJ

If you make a syntax error in a command string, the linker prints an error message and returns you to the asterisk prompt. You can then retype the new command string. Similarly, if you specify a nonexistent file, an error occurs; the linker prints an asterisk after which you must type the command string again.

3.2.3 LINK Switches Briefly Noted

The switches associated with the linker are described in Table 3–7. To properly use the switches, you must precede the letter representing each switch by the slash character (/). Switches must appear on the line indicated if you continue the input on more than one line, but you can position them anywhere on the line. The column titled Command Line lists on which line in the command string the switch can appear. Section 3.5 provides a more detailed explanation of each switch.

Table 3–7: Linker Switches

Switch Name	Command Line	Section	Explanation
/A	First	3.5.1	Lists global symbols in program sections in alphabetical order in the load map.
/B:n	First	3.5.2	Changes the bottom address of a program to n (invalid with /H).
/C	Any but last	3.5.3	Continues input specification on another command line. (You can also use /C with /O; however, do not use /C with the // switch.)
/E:n	First	3.5.4	Extends a particular program section in the root to a specific value.
/F	First	3.5.5	Instructs the linker to use the default FORTRAN library $FORLIB.OBJ to resolve any undefined global references. Do not specify this switch in the command line when $FORLIB has been incorporated into $SYSLIB.
/G	First	3.5.6	Adjusts the size of the linker's library directory buffer to accommodate the largest multiple definition library directory.
/H:n	First	3.5.7	Specifies the top (highest) address to be used by the relocatable code in the load module. Invalid with /B, /Y, or /Q.
/I	First	3.5.8	Allows you to specify additional external global symbols to be satisfied (typically from the libraries). In general, this is used to explicitly request the inclusion of additional library modules.
/K:n	First	3.5.9	Inserts the value you specify (the valid range for n is from 1 to 28) into word 56 of block 0 of the image file. This switch informs the RT11 run-time system that the program requires nK words of memory.

(continued on next page)

Table 3–7: Linker Switches (Cont.)

Switch Name	Command Line	Section	Explanation
/M[:n]	First	3.5.10	Causes the linker to prompt you for a global symbol that represents the initial stack address (if n is omitted) or that sets the initial stack address to the value n (if n is specified).
/O:n	Any but first	3.5.11	Indicates that the program is an overlay structure; n specifies the overlay region to which the module is assigned.
/P:n	First	3.5.12	Changes the default amount of space the linker uses for the library routines list.
/Q	First	3.5.13	Lets you specify the base addresses of up to eight root program sections. Invalid with /H.
/S	First	3.5.14	Makes the maximum amount of space in memory available for the linker's symbol table. (Use this switch only when a particular link stream causes a symbol table overflow.)
/T[:n]	First	3.5.15	Causes the linker to prompt you for a global symbol that represents the transfer address (if n is omitted) or that sets the transfer address to the value n (if n is specified).
/U:n	First	3.5.16	Rounds up the root program section you specify so that the size of the root segment is an integer multiple of the value you supply (n must be a power of 2).
/W	First	3.5.17	Directs the linker to produce a wide load map listing.
/X	First	3.5.18	Does not output the bitmap if the area normally used by the bitmap (location 360–377) is used by code.
/Y:n	First	3.5.19	Starts a specific program section in the root on a particular address boundary. Invalid with /H.
/Z:n	First	3.5.20	Sets unused locations in the load module to the value n (if n is omitted, the linker uses zero as the default).
//	First and last	3.5.3	Allows you to specify command string input on additional lines. Do not use this switch with /C.

3.3 Input and Output

Linker input and output is in the form of modules; the linker uses one or more input modules to produce a single output (load) module. The linker also accepts library modules and symbol table definition files as input and can produce a load map and/or symbol table definition file. The sections that follow describe all valid forms of linker input and output.

3.3.1 Input Object Modules

Object files, consisting of one or more object modules, are the input to the linker. (Entering files that are not object modules may result in a fatal error.) Object modules are created by language translators such as the FORTRAN compiler and the MACRO–11 assembler. The module name item declares the name of the object module (see Section 3.3.4).

The first six Radix–50 characters of the .TITLE assembler directive are used as the name of the object module. These six characters must be Radix–50 characters (the linker ignores any characters beyond the sixth character). The linker prints the first module name it encounters in the input file stream (normally the main routine of the program) on the second line of the map following TITLE:. The linker also uses the first identity label (issued by the .IDENT directive) for the load map. It ignores additional module names.

The linker reads each object module twice. During the first pass, it reads each object module to construct a global symbol table and to assign absolute values to the program section names and global symbols. The linker uses the library files to resolve undefined globals. It places their associated object modules in the root. On the second pass, the linker reads the object modules, links and relocates the modules, and outputs the load module.

Symbol table definition files are special object files that can serve as input to the linker anywhere other object files are allowed.

3.3.2 Input Library Modules

The linker can automatically search libraries. Libraries consist of library files, which are specially formatted files produced by the librarian program (described in Chapter 4). The files contain one or more object modules that provide routines and functions to aid you in meeting specific programming needs. (For example, FORTRAN has a set of modules containing all necessary computational functions — SQRT, SIN, COS, and so on.) You can use the librarian to create and update libraries. Then you can easily access routines that you use repeatedly or routines that different programs use. Selected modules from the appropriate library file are linked as needed with your program to produce one load module. Libraries are described in more detail in Chapter 4.

You specify libraries in a command string the same way you specify normal modules; you can include them anywhere in the command string. If you are creating an overlay structure, specify libraries before you specify the overlay structure. If a global symbol is undefined at the end of pass 1 and if a module in a library contains that global definition, then the linker pulls that module from the library and links it into the load image. Only the modules needed to resolve references are pulled from the library; unreferenced modules are not linked.

Modules in one library can call modules from another library; however, the libraries must appear in the command string in the order in which they are called. For example, assume module X in library ALIB calls Y from the BLIB library. To correctly resolve all globals, the order of ALIB and BLIB should appear in the command line as:

```
*Z=B,ALIB,BLIB
```

Module B is the root. It calls X from ALIB and brings X into the root. Module X in turn calls Y, which is brought from BLIB into the root.

Library Module Processing

The linker selectively relocates and links object modules from specific user libraries that were built by the librarian. Figure 3–1 illustrates this general process. During pass 1, the linker processes the input files in the order in which they appear in the input command line. If the linker encounters a library file during pass 1, it takes note of the library in an internal save status block, and then proceeds to the next file. The linker processes only nonlibrary files during the initial phase of pass 1. In the final phase of pass 1, the linker processes only library files. This is when it resolves the undefined globals that were referenced by the nonlibrary files.

The linker processes library files in the order in which they appear in the input command line. The default system library (DK:$SYSLIB.OBJ) is always processed last (if any undefined globals remain).

The search method the linker uses allows modules to appear in any order in the library. You can specify any number of libraries in a link and they can be positioned anywhere, with the exception of forward references between libraries, and they must come before the overlay structure. The default system library, DK:$SYSLIB.OBJ, is the last library file the linker searches to resolve any remaining undefined globals.

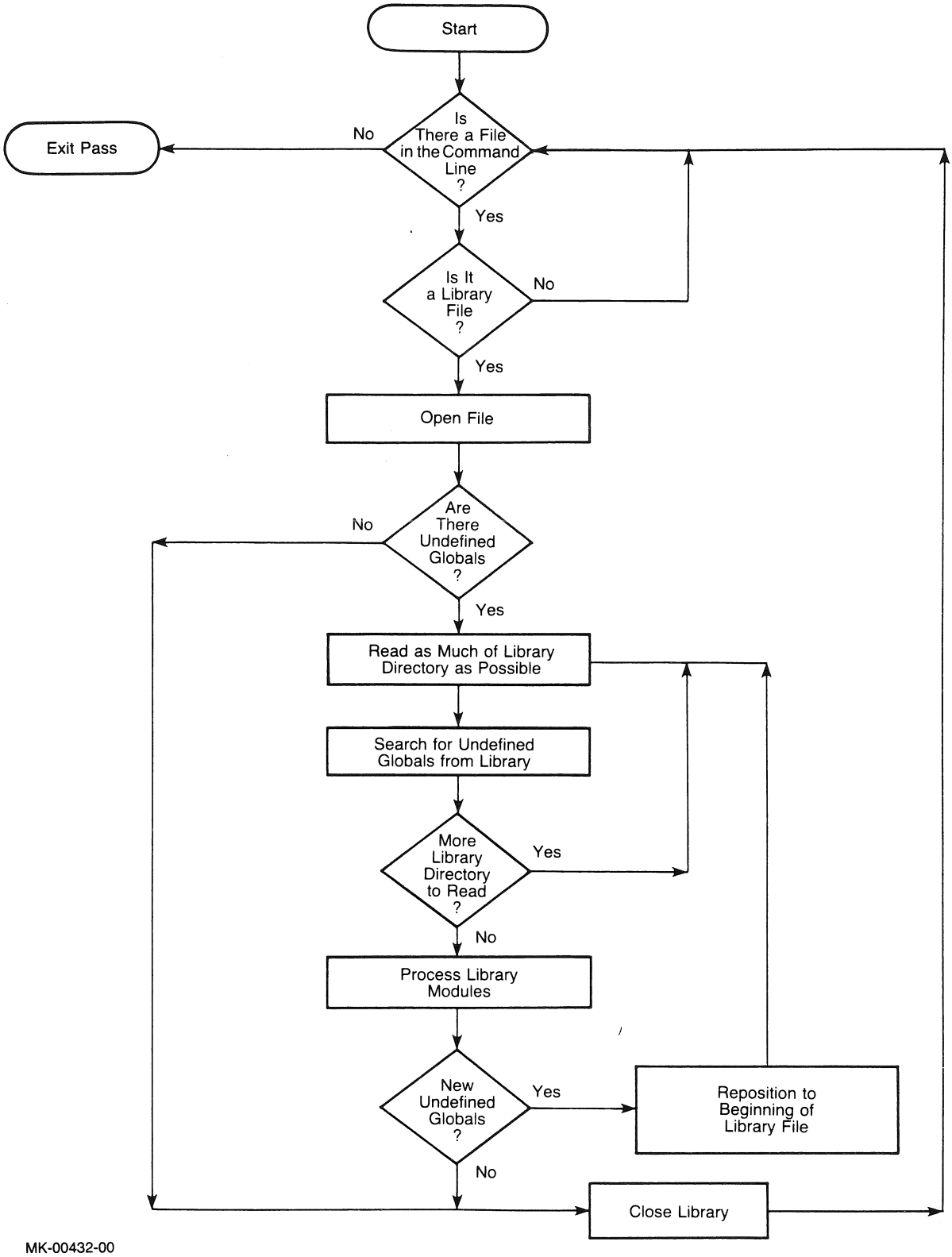
Some languages, such as FORTRAN, have an Object Time System (OTS) that the linker takes from a library and includes in the final module. The most efficient way to accomplish this is to include these OTS routines (such as NHD, OTSCOM, and V2NS for FORTRAN) in DK:$SYSLIB.OBJ.

Libraries are input to the linker the same way as other input files. For example:

```
*TASK01,LP:=MAIN,MEASUR
```

This causes program MAIN.OBJ to be read from DK: as the first input file. Any undefined symbols generated by program MAIN.OBJ should be satisfied by the library file MEASUR.OBJ specified in the second input file. The linker tries to satisfy any remaining undefined globals from the default library DK:$SYSLIB.OBJ. The load module, TASK01.SAV, is stored on DK: and a load map is printed on the line printer.

Figure 3–1: Library Searches



Multiple Definition Libraries

In addition to the libraries explained so far, the linker processes multiple definition libraries. Its primary purpose is to provide special functions for RSTS/E. These libraries differ from other libraries in that they can contain more than one definition for a given global. You specify multiple definition libraries in the command line the same way you specify normal libraries. Modules that the linker obtains from multiple definition libraries always appear in the root.

It is useful to know the differences between processing normal and multiple definition libraries. When you include modules from a multiple definition library, the linker has to store that library's directory in an internal buffer. A library's directory is called an entry point table (EPT). If a library EPT is too large to fit into the internal buffer, the linker prints a message asking you to use the /G switch. The /G switch changes the buffer's size to accommodate the largest EPT of all the multiple definition libraries you are using. Use the /G switch only when the linker indicates it is required.

When a global symbol in a module of a multiple definition library matches an undefined global, LINK removes from the undefined global list all other globals defined in the same module. LINK does this before it processes the library module. Thus, two modules with identical globals do not appear in the linked module.

NOTE

The order of modules in multiple definition libraries is very important and affects which modules LINK uses. The increased EPT size (due to duplicate entries, in addition to module name entries) also slows LINK down.

3.3.3 Output Load Module

The primary output of the linker is a load module that you can run under RSTS/E. The linker creates as a load module a memory image file (file type of .SAV) for use under the RT11 Emulator (RT11.RTS).

The load module for a memory image file is arranged as follows:

Root Segment	Overlay Segments (optional)

The first 256–word block of the root segment (main program) contains the memory usage bitmap and the locations the linker uses to pass program control parameters. The memory usage bitmap outlines the blocks of memory that the load module uses; it is located in locations 360 through 377.

Table 3–8 lists the parameters that appear in the absolute block, the addresses the parameters occupy, and the conditions under which they are set.

Table 3–8: Absolute Block Parameters Information

Address	Information	When Set
14,16	BPT trap vector	
20,22	IOT trap vector	
34,36	TRAP vector	
40	Start address of program	always
42	Initial setting of SP (stack pointer)	always
44	Job Status Word (overlay bit set by LINK)	always
50	Highest memory address used by the program (high limit)	always
56	Program size in K	with /K
64	Start address of overlay table	with /O
360–377	Memory usage bitmap	always, except with /X

The linker stores default values in locations 40, 42, and 50, unless you use switches to specify otherwise. The /T switch affects location 40, for example, and /M affects location 42. You can also use the .ASECT directive to change the defaults. The overlay bit is located in the job status word. LINK automatically sets this bit if the program is overlaid. Otherwise, the linker initially sets location 44 to 0.

You can assign initial values to memory locations 0–476 (which include the interrupt vectors and system communication area) by using an .ASECT assembler directive. The values appear in block 0 of the load module, but there are restrictions on the use of .ASECT directives in this region. You should not modify locations 360–377 because the memory usage map is passed in those locations, unless you use the /X switch.

You can use an .ASECT directive to set any location that is not restricted, but be careful if you change the system communication area. The program itself must initialize restricted areas, such as locations 360–377 at run time.

3.3.4 Output Load Map

The linker can produce a load map following the completion of the initial pass. This map, shown in Figure 3–2, illustrates the layout of memory for the load module.

The load map lists each program section that is included in the linking process. The line for a section includes the name and low address of the section and its size in bytes. The rest of the line lists the program section attributes, as shown in Table 3–2. The remaining columns contain the global symbols found in the section and their values.

The map begins with the linker version number, followed by the date and time the program was linked. The second line lists the file name of the program, its title (which is determined by the first module name record in the input file), and the first identification record found. The absolute section is always shown first, followed by any nonrelocatable symbols. The modules located in the root segment of the load module are listed next, followed by those modules that were assigned to overlays in order by their region number (see Section 3.4). Any undefined global symbols are then listed. The map ends with the transfer address (start address) and high limit of relocatable code in both octal bytes and decimal words.

Figure 3–2: Sample Load Map

```
1  RT-11 LINK V06.01       Load Map          Wed 26-Aug-81 12:01:10
2  TEST  .SAV     Title:   TEST     Ident:
3
4  Section  Addr    Size     Global   Value   Global  Value   Global  Value
5
6   . ABS.  000000 000000    (RW,I,GBL,ABS,OVR)
7           001000 000200    (RW,I,LCL,REL,CON)
8   TEST    001200 000174    (RW,I,LCL,REL,CON)
9                            START    001200  EXIT     001240
10
11 Transfer address = 001200, High limit = 001372 = 381.   words
```

Table 3–9 describes each line in the sample load map above.

Table 3–9: Line-by-Line Sample Load Map Description

Line	Contents
1	Load map header.
2	Program name, program title (.MAIN. default) and identity (default is blank).
4	P-sect description header. Section indicates the p-sect name; Addr indicates the p-sect start address; Size indicates p-sect length in octal bytes; Global and Value list the p-sect globals and their associated octal values.
6	Absolute p-sect, . ABS. This line includes the absolute p-sect's start address (always 0), length and attributes (for a complete description of these abbreviations, see Table 3–2).
7	Unnamed p-sect. This p-sect appears in the load map after the absolute p-sect. For overlaid programs, the unnamed (blank) p-sect appears in the load map after the overlay table p-sect.
8–9	TEST p-sect. Line 9 lists TEST's two globals, START and EXIT, with their associated values.
11	Transfer address indicates the address in memory where the program starts. High limit indicates the last address used by the program. The number of words in the program appears last.

3.4 Creating an Overlay Structure

The linker's ability to handle overlays gives you virtually unlimited memory space for an assembly language or FORTRAN program. A program using overlays can be much larger than would normally fit in the available memory space because portions of the program reside on a disk. To use this capability, you must define an overlay structure for your program.

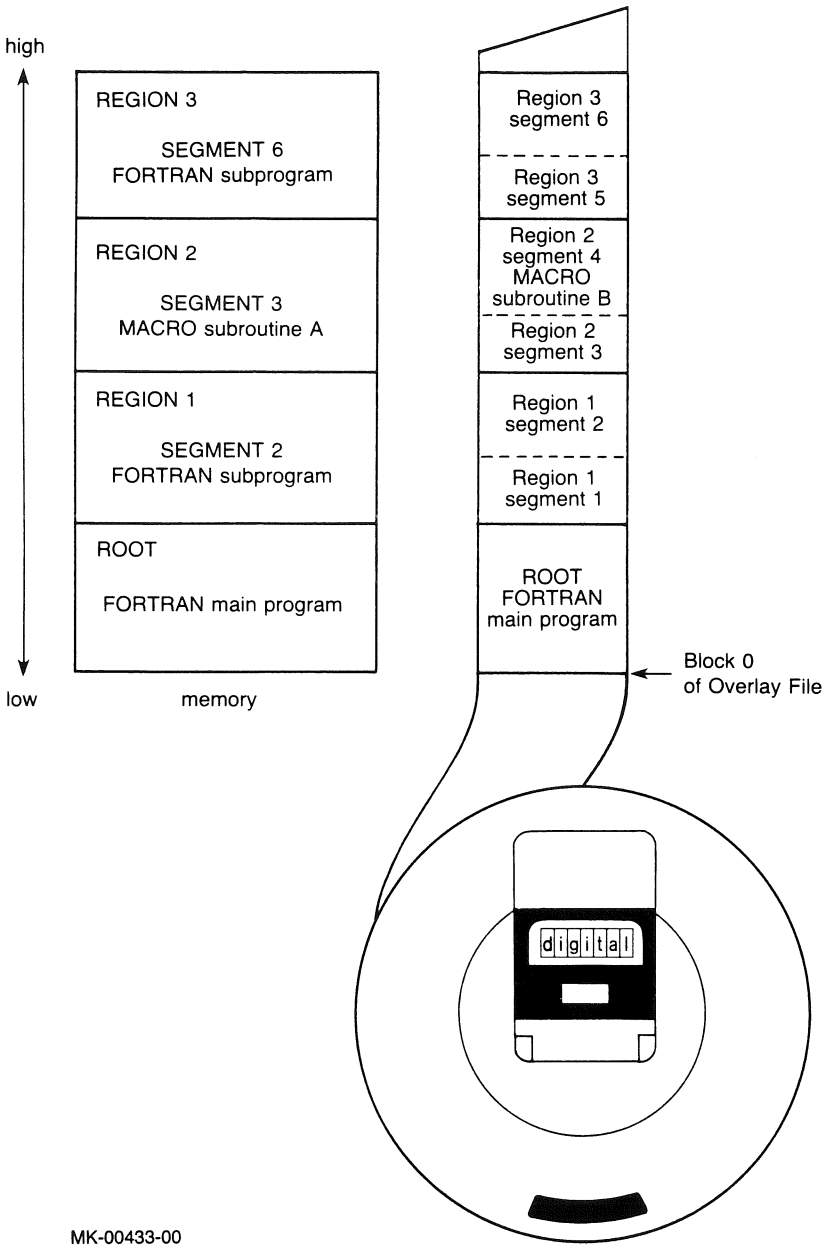
An overlay structure divides a program into segments. For each overlaid program, there is one root segment and a number of overlay segments. Each overlay segment is assigned to a particular area of available memory, called an overlay region. More than one overlay segment can be assigned to a given overlay region. Each region of memory, however, is occupied by one (and only one) of its assigned segments at a time. The other segments assigned to that region are stored on disk. They are brought into memory when called, replacing (overlaying) the segment previously stored in that region. The root segment, on the other hand, contains those parts of the program that must always be memory-resident. Therefore, the root is never overlaid by another segment.

Figure 3–3 diagrams an overlay structure for a FORTRAN program. The main program is placed in the root segment and is never overlaid. The various MACRO subroutines and FORTRAN subprograms are placed in overlay segments. Each overlay segment is assigned to an overlay region and stored on disk until called into memory. For example, region 2 is shared by the MACRO subroutine A currently in memory and the MACRO subroutine B in segment 4. When a call is made to subroutine B, segment 4 is brought into region 2 of memory, overlaying or replacing segment 3.

The overlay file shown in Figure 3–3 is created by the linker when you specify an overlay structure. The overlay file contains the root segment and each overlay segment, including those overlay segments currently in memory.

The linker calculates the size of any region to be the size of the largest segment assigned to that region. Thus, to reduce the size of a program (that is, the amount of memory it needs), you should first concentrate on reducing the size of the largest segment in each region. The linker delineates the overlay regions you specify and prefaces your program with the run-time overlay handler code shown in Figure 3–5. The linker also sets up links between the overlay handler and program references to routines that reside in overlays. When, at run time, a reference is made to a section of your program that is not currently in memory, these links cause an overlay to be read into memory. The overlay segment containing the referenced code becomes resident.

Figure 3–3: Sample Overlay Structure for a FORTRAN Program



You specify an overlay structure to the linker by using the /O switch (see Figure 3–4).

Figure 3–4: Overlay Scheme

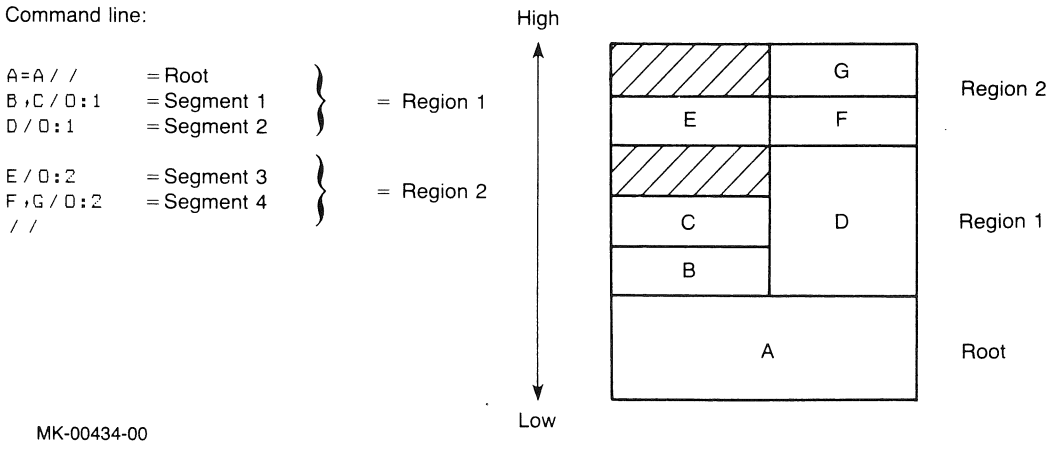


Figure 3–5: The Run-Time Overlay Handler

```
        .TITLE OHANDL.006 OVERLAY HANDLER
        .IDENT  /V01.00 /

; RT-11 OVERLAY HANDLER
;
; COPYRIGHT (C) 1979
; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS. 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 DEC.
;
; THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
; SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
;
; MAS


; EDIT LOG SINCE V01.00
; ADD NEW GLOBAL NAMES TO ALLOW RELOCATION OF HANDLER CODE                ;MAS01
```

(continued on next page)

Figure 3–5: The Run-Time Overlay Handler (Cont.)

```
.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.
; THE RUN-TIME OVERLAY HANDLER IS CALLED BY A DUMMY
; SUBROUTINE OF THE FOLLOWING FORM:
;
;       JSR     R5,$OVRH        ;CALL TO COMMON CODE
;       .WORD   <OVERLAY #*6>   ;# OF DESIRED SEGMENT
;       .WORD   <ENTRY ADDR>    ;ACTUAL CORE ADDR (VIRTUAL ADDR)
;
; ONE DUMMY ROUTINE OF THE ABOVE FORM IS STORED IN THE RESIDENT PORTION
; OF THE USER'S PROGRAM FOR EACH ENTRY POINT TO AN OVERLAY SEGMENT.
; ALL REFERENCES TO THE ENTRY POINT ARE MODIFIED BY THE LINKER TO INSTEAD
; BE REFERENCES TO THE APPROPRIATE DUMMY ROUTINE.  EACH OVERLAY SEGMENT
; IS CALLED INTO CORE AS A UNIT AND MUST BE CONTIGUOUS IN CORE.  AN
; OVERLAY SEGMENT MAY HAVE ANY NUMBER OF ENTRY POINTS, TO THE LIMITS
; OF CORE MEMORY.  ONLY ONE SEGMENT AT A TIME MAY OCCUPY AN OVERLAY REGION.
;-



.SBTTL  DEFINITIONS, AND MISC.

;+
; UNDEFINED GLOBALS IN THE OVERLAY HANDLER MUST BE NAMED "$OVDF1" TO
; "$OVDFn" SUCH THAT A RANGE CHECK MAY BE DONE BY LINK TO DETERMINE IF
; THE UNDEFINED GLOBAL NAME IS FROM THE OVERLAY HANDLER. A CHECK IS
; DONE ON THE .RAD50 CHARACTERS "$OV", AND THEN A RANGE CHECK IS DONE ON
; THE .RAD50 CHARATERS "DF1" TO "DFn". THESE GLOBAL SYMBOLS DO NOT APPEAR
; ON LINK MAPS, SINCE THEIR VALUE IS NOT KNOWN UNTILL AFTER THE MAP HAS BEEN
; PRINTED. CURRENTLY $OVDF1 TO $OVDF5 ARE IN USE.
;
;
; GLOBAL SYMBOLS O$READ, AND O$DONE ARE USEFULL WHEN DEBUGGING
; OVERLAID PROGRAMS.
;
; O$READ:: WILL APPEAR IN THE LINK MAP, AND LOCATES THE .READ STATEMENT
; IN THE OVERLAY HANDLER.
;
; O$DONE:: WILL APPEAR IN THE LINK MAP, AND LOCATES THE FIRST INSTRUCTION
; AFTER A .READ IS COMPLEATED IN THE OVERLAY HANDLER.
;-


.MCALL  .READW,..V1..
        ..V1..



.SBTTL  OVERLAY HANDLER CODE

.PSECT  $OHAND,GBL

.ENABL  GBL
.ENABL  LSB

; $OVRH IS THE ENTRY POINT TO THE OVERLAY HANDLER

$OVRH:: MOV     R0,-(SP)        ;MUST SAVE SINCE READ ETC USE IT
        MOV     R1,-(SP)        ;/0 OVERLAY ENTRY POINT
        MOV     R2,-(SP)
```

(continued on next page)

Figure 3–5: The Run-Time Overlay Handler (Cont.)

```
2$:
;       MOV     @R5,R1          ;PICK UP OVERLAY NUMBER
        BR      7$              ;FIRST CALL ONLY * * *
        ADD     #$OVTAB-6,R1    ;CALC TABLE ADDR
        MOV     (R1)+,R2        ;GET FIRST ARG. OF OVERLAY SEG. ENTRY

3$:     CMP     (R5)+,@R2       ;IS OVERLAY ALREADY RESIDENT?
        BEQ     4$              ;YES, BRANCH TO IT

;+
; THE .READ USES INFORMATION AS FOLLOWS:
; CHANNEL NUMBER, CORE ADDRESS, LENGTH TO READ, RELATIVE BLOCK ON DISK.
; THESE ARE PICKED UP IN REVERSE ORDER OF THAT SPECIFIED IN THE CALL.
;-

O$READ::.READW  17,R2,@R1,(R1)+ ;READ FROM OVERLAY FILE
O$DONE::BCS     5$
4$:     MOV     (SP)+,R2        ;RESTORE USERS REGS
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        MOV     @R5,R5          ;GET ENTRY ADDRESS
        RTS     R5              ;ENTER OVERLAY ROUTINE AND RESTORE USER'S R5

5$:     EMT     376             ;SYSTEM ERROR 10 (OVERLAY I/O)
        .BYTE   0,373

7$:     MOV     #11501,2$       ;RESTORE SWITCH INSTR (MOV @R5,R1)
        MOV     (PC)+,R1        ;START ADDR FOR CLEAR OPERATION
$ODF1:: .WORD   $OVDF1          ;HIGH ADDR OF ROOT SEGMENT              ;MAS01
8$:     CLR     (R1)+           ;CLEAR ALL OVERLAY REGIONS
        CMP     R1,$ODF2        ;DONE?
        BLO     8$              ;LO -> NO, REPEAT
        BR      2$              ;AND RETURN TO CALL IN PROGRESS

$ODF2:: .WORD   $OVDF2          ;HIGH ADDRESS OF /O OVERLAYS            ;MAS01

.DSABL  LSB



.SBTTL  $OVTAB  OVERLAY TABLE

;+
; OVERLAY SEGMENT TABLE FOLLOWS:
; $OVTAB:       .WORD   <CORE ADDR>,<RELATIVE BLK>,<WORD COUNT>   /O OVERLAYS
;
; 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 TABLE.
;-

.PSECT  $OTABL,D,GBL,OVR

$OVTAB::

.END
```

There is no special formula for creating an overlay structure. You do not need a special code or function call. Some general guidelines must be followed, however. For example, a FORTRAN main program must always be

placed in the root segment. This is true also for a global program section (such as a named COMMON block) that is referenced by more than one overlay segment.

The assignment of region numbers to overlay segments is crucial. Segments that overlay each other (have the same region number) must be logically independent; that is, the components of one segment cannot reference the components of another segment assigned to the same region. Segments that need to be memory resident simultaneously must be assigned to different regions.

When you make calls to routines or subprograms that are in overlay segments, the entire return path must be in memory. This means that from an overlay segment you cannot call a routine that is in a different segment of the same region. If this is done, the called routine overlays the segment making the call and destroys the return path.

Figure 3–6 illustrates a sample set of subroutine calls and return paths. In the example, solid lines represent legal subroutine calls and dotted lines represent invalid calls.

Suppose the following subroutine calls were made:

1. The root calls segment 8
2. Segment 8 calls segment 4
3. Segment 4 calls segment 3

Segment 3 can now call any of the following, in any order:

itself segment 8

segment 4 the root

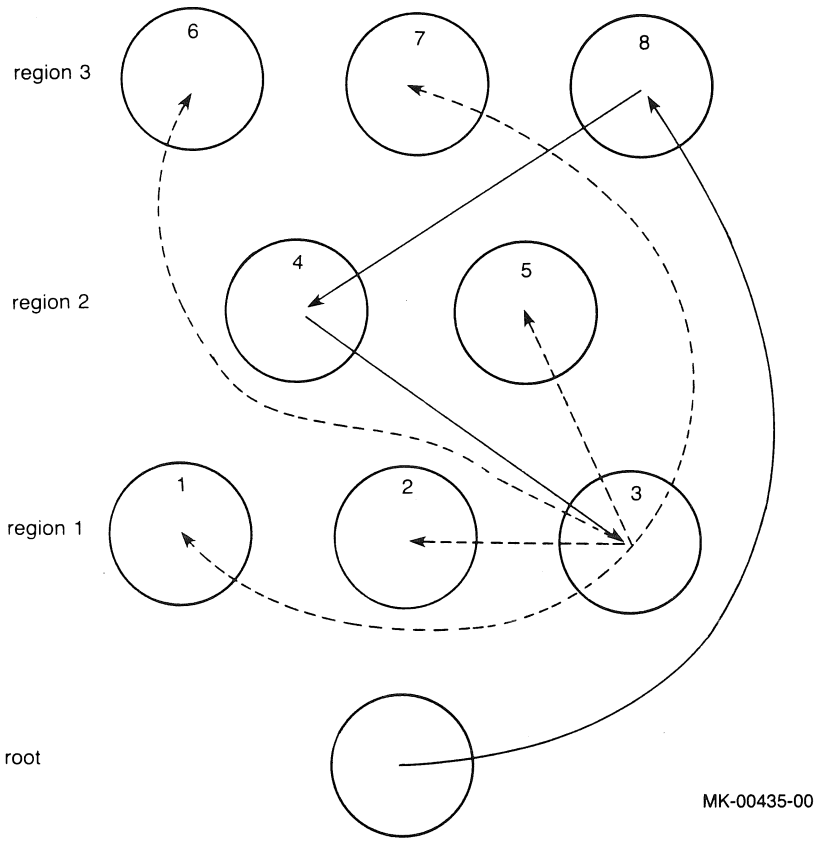
These segments and the root, of course, are all currently in memory.

Segment 3 cannot call any of the following segments because this would destroy its return path:

segments 2 and 1
segment 5
segments 6 and 7

Look at what might happen if one of these invalid calls is made. Suppose segment 4 calls segment 3 and segment 3 in turn calls segment 5. Segment 5 is not resident in region 2, so an overlay read-in occurs: segment 5 is read into memory, thus destroying the memory-resident copy of segment 4. The subroutine in segment 5 executes and returns control to segment 3. Segment 3 finishes its task and tries to return control to segment 4. Segment 4, however, has been replaced in memory by segment 5. Segment 4 cannot regain control and the program loops indefinitely, or traps, or random results occur.

Figure 3–6: Sample Subroutine Calls and Return Paths



The guidelines already mentioned and some additional rules for creating overlay structures are summarized as follows:

1. $SYSLIB must be present to create an overlay structure because it contains the overlay handler.
2. Overlay segments assigned to the same region must be logically independent; that is, the components of one segment cannot reference the components of another segment assigned to the same region.
3. The root segment contains the transfer address, stack space, impure variables, data, and variables needed by many different segments. The FORTRAN main program unit must be placed in the root segment.
4. The absolute section (. ABS.) never takes part in overlaying in any way. It is part of the root and is always resident.
5. A global program section (such as a named COMMON block or a .PSECT with the GBL attribute) that is referenced in more than one segment is placed in the root segment by the linker. This permits common access across the different segments.

6. Object modules that are automatically acquired from a library file are placed automatically in an overlay segment, as long as that library module is referenced only by that segment. If a library module is referenced by more than one segment, LINK places that library module in the root. You can, however, extract modules from a library file using the librarian utility program as explained in Chapter 4. Extracted object modules can be placed in overlay segments.

   Do not specify a library file on the same command line as an overlay segment. You must specify all library modules before specifying any overlay modules. LINK places in the root any modules from a multiple definition library and any modules included with the /I switch.

7. All COMMON blocks that are initialized with DATA statements must be similarly initialized in the segment in which they are placed.

8. When you make calls to overlay segments, the entire return path to the calling routine must be in memory. This means you should take the following points into account:

   a. You can make calls with expected return (as from a FORTRAN main program to a FORTRAN or MACRO subroutine) from an overlay segment to entries in the same segment, the root segment, or to any other segment, as long as the called segment does not overlay in memory part of your return path to the main program.

   b. You can make jumps with no expected return (as in a MACRO program) from an overlay segment to any entry in the program, with one exception: you cannot make such a jump to a segment if the called segment will overlay an active routine in that region (that is, a routine whose execution has begun, but not finished, and that will be returned to).

   c. Calls you make to entries in the same region as the calling routine must be entirely within the same segment, not within another segment in the same region.

9. You must make calls or jumps to overlay segments directly to global symbols defined in an instruction p-sect (entry points). For example, if ENTER is a global symbol in an overlay segment, the first of the following two commands is valid, but the second is not:

```
JMP ENTER      ;VALID
JMP ENTER+6    ;INVALID
```

10. You can use globals defined in an instruction p-sect (entry points) of an overlay segment only for transfer of control and not for referencing data within an overlay segment. The assembler and linker cannot detect a violation of this rule so they issue no error. However, such a violation can cause the program to use incorrect data. If you reference these global symbols outside of their defining segment, the linker resolves them by using dummy subroutines of four words each in the overlay handler. Such a reference is indicated on the load map by an at sign character (@) following the symbol.

11. The linker directly resolves symbols that you define in a data p-sect. It is your responsibility to load the data into memory before referencing a global symbol defined in a data section.

12. In the linker command string, specify overlay regions in ascending order.

13. Overlay regions are read-only. The overlay handler does not save the segment it is overlaying. Any tables, variables, or instructions that are modified within a given overlay segment are reinitialized to their original values in the .SAV file if that segment has been overlaid by another segment. You should place any variables or tables whose values must be maintained across overlays in the root segment.

14. Your program cannot use channel 17 (octal) because overlays are read on that channel.

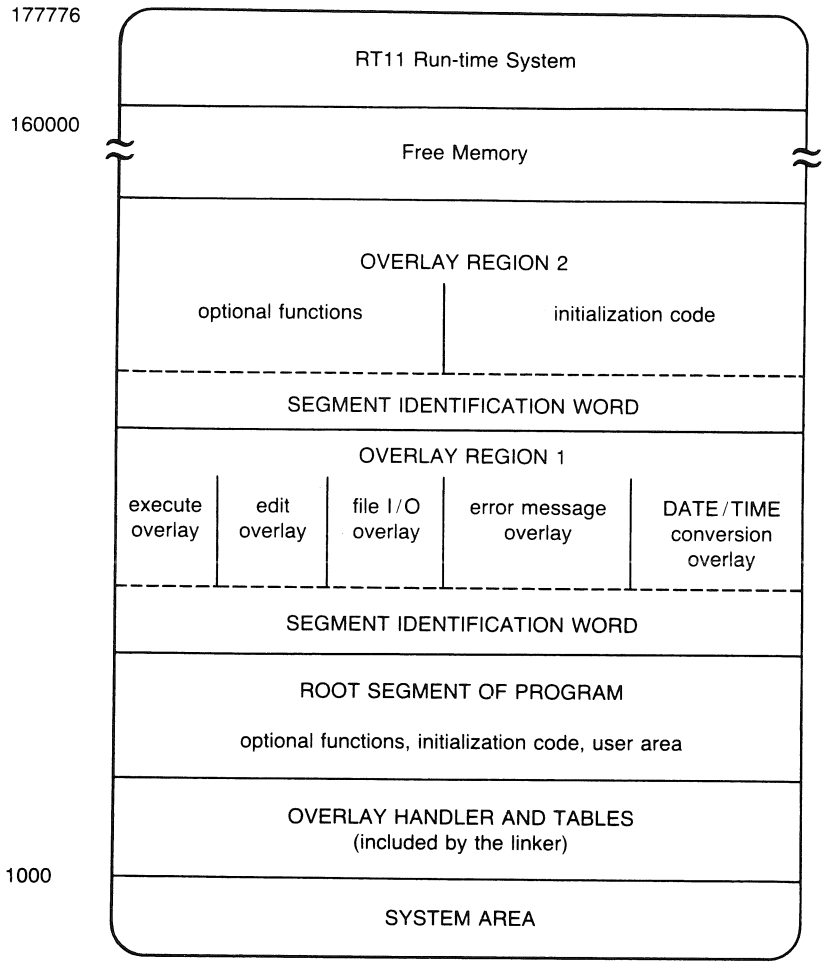
15. Note that the condition codes set by your program are not preserved across overlay segment boundaries.

16. MACRO and FORTRAN directly resolve all global symbols that are defined in a module. If LINK moves the p-sect where they are defined from an overlay segment to the root, LINK will not generate an overlay table entry for those symbols.

This set of rules applies only to communications among the various modules that make up a program. Internally, each module must only observe standard programming rules for the PDP–11 (as described in the *PDP–11 Processor Handbook* and in the FORTRAN and MACRO–11 language reference manuals).

The linker provides overlay services by including a small resident overlay handler in the same file with your program to be used at program run time. (Refer to Figure 3–5.) The linker inserts this overlay handler plus some tables into your program beginning at the bottom address. The linker then moves your program up in memory to make room for the overlay handler and tables, if necessary. The handler is stored in $SYSLIB. This scheme is diagrammed in Figure 3–7.

Figure 3–7: Memory Diagram Showing Sample Link with Overlay Regions



MK-00436-00

3.5 Switch Descriptions

Full descriptions of the switches summarized in Table 3–7 follow in alphabetical order.

3.5.1 Alphabetical Switch (/A)

The /A switch lists global symbols within program sections in alphabetical order on the load map.

3.5.2 Bottom Address Switch (/B:n)

The /B:n switch supplies the lowest address to be used by the relocatable code in the load module. The argument n is an unsigned octal number that defines the bottom address of the program being linked. If you do not supply a value for n, the linker prints:

```
?LINK-F-/B No value
```

Retype the command line, supplying an even octal value.

When you do not specify /B, the linker positions the load module so that the lowest address is location 1000 (octal). If the ASECT size is greater than 1000, the size of ASECT is used.

If you supply more than one /B switch during the creation of a load module, the linker uses the first /B switch specification. The /B switch is illegal when you are linking to a high address (/H).

The bottom value must be an unsigned, even, octal number. If the value is odd, the linker prints the message:

```
?LINK-F-/B odd-value
```

Reenter the command string specifying an unsigned, even, octal number as the argument to the /B switch.

3.5.3 Continue Switch (/C) or (//)

The continue switch (/C) lets you type additional lines of command string input. Use the /C switch at the end of the current line and repeat it on subsequent command lines as often as necessary to specify all the input modules in your program. Do not enter a /C switch on the last line of input.

The following command indicates that input is to be continued on the next line:

```
*OUTPUT,LP:=INPUT/C
*
```

An alternate way to enter additional lines of input is to use the / / switch on the first line. The linker continues to accept lines of input until it encounters another / / switch, which can be either on a line with input file specifications, or on a line by itself. The advantage of using the / / switch instead of the /C switch is that you do not have to type the / / switch on each continuation line. This example shows how the linker itself is linked:

```
*LINK,LINK=LINKO/W//
*LINK1/O:1
*LINK2/O:1
*LINK3/O:1
*LINK4/O:1
*LINK5/O:1
*LINK6/O:1
*LINK7/O:1
*LINKEM/O:1//
```

You cannot use the /C switch and the / / switch together in a link command sequence. That is, if you use / / on the first line, you must use / / to terminate input on the last line. If you use /C on the first line, use /C on all lines but the last.

3.5.4 Extend Program Section Switch (/E:n)

The /E:n switch allows you to extend a program section in the root to a specific value. Type the /E:n switch at the end of the first command line. After you have typed all input command lines, the linker prompts with:

```
Extend section?
```

Enter the name of the program section to be extended, and then press the RETURN key. The resultant program section size (in bytes) is equal to or greater than the value you specify, depending on the space the object code requires. The value you specify must be an even value. Note that you can extend only one section.

The following example extends section CODE to 100 (octal) bytes:

```
*X,KB:=LK001/E:100
Extend section? CODE
```

3.5.5 Default FORTRAN Library Switch (/F)

By indicating the /F switch in the command line, you can link the FORTRAN library ($FORLIB.OBJ on device DK:) with the other object modules you specify. You do not need to specify FORLIB explicitly. For example:

```
*FILE,LP:=AB/F
```

The object module AB.OBJ from DK: and the required routines from the FORTRAN library DK:$FORLIB.OBJ are linked together to form a load module called FILE.SAV.

The linker automatically searches the DK:$SYSLIB.OBJ default system library. The library normally includes the modules that compose FORLIB. You should not have to use /F.

3.5.6 Directory Buffer Size Switch (/G)

When you are using modules for your program that are from a multiple definition library, LINK has to store that library's directory in an internal buffer. Occasionally, this buffer area is too small to contain an entire directory, in which case LINK is unable to process those modules. The /G switch causes LINK to adjust the size of its directory buffer to accommodate the largest directory size of the multiple definition libraries you are using. Because the /G switch slows the linking process, you should use it only when it is necessary. In particular, use it only after an attempt to link your program failed because the buffer was too small. LINK prints the following message when a failure of this type occurs:

```
?LINK-F-Directory buffer too small
```

3.5.7 Highest Address Switch (/H:n)

The /H:n switch allows you to specify the top (highest) address to be used by the relocatable code in the load module. The argument n represents an unsigned, even, octal number. If you do not specify n, the linker prints:

```
?LINK-F-/H no value
```

Retype the command, supplying an even octal number to be used as the value.

If you specify an odd value, the linker responds with:

```
?LINK-F-/H odd value
```

Retype the command, supplying an even octal number.

If the value is not large enough to accommodate the relocatable code, the linker prints:

```
?LINK-F-/H value too low
```

Relink the program with a larger value.

You cannot use the /H switch with the /Y or /B switch.

NOTE

Be careful when you use the /H switch. Most FORTRAN programs use the free memory above the relocatable code as a dynamic working area for I/O buffers, device handlers, symbol tables, and so forth. The size of this area differs according to the memory configuration. Programs linked to a specific high address might fail to run for users with a lower maximum memory size because there is less free memory.

3.5.8 Include Switch (/I)

The /I switch lets you take global symbols from any library and include them in the linking process even when they are not needed to resolve globals. This provides a method for forcing modules that are not called by other modules to be loaded from the library. All modules that you specify with /I go into the root. When you specify the /I switch, the linker prints:

```
Library search?
```

Reply with the list of global symbols to be included in the load module; press the RETURN key to enter each symbol in the list. Pressing only the RETURN key terminates the list of symbols.

The following example includes the global $SHORT in the load module:

```
*SCCA=RK1:SCCA/I
Library search? $SHORT (RET)
Library search? (RET)
```

3.5.9 Memory Size Switch (/K:n)

The /K:n switch lets you insert a value into word 56 of block 0 of the image file. The argument n represents the number of 1K blocks of memory required by the program; n is an integer in the range 1–28. Note that the value for n is interpreted as an octal number unless you place a period (.) after it. For example, when LINK sees a switch such as /K:28., it interprets the value 28. as a decimal number.

3.5.10 Modify Stack Address Switch (/M[:n])

The stack address, location 42, is the address that contains the initial value for the stack pointer. The /M switch lets you specify the stack address. The argument n (if present) is an even, unsigned, octal number that defines the stack address. After all input lines have been typed, the linker prints the following message if you have not specified a value for n:

```
Stack symbol?
```

In this case, specify the global symbol whose value is the stack address, and press the RETURN key. You must not specify a number. If you specify a nonexistent symbol, the linker prints an error message and sets the stack address to the system default (1000) or to the bottom address if you used /B. If the program's absolute section extends beyond location 1000, the default stack space starts after the largest .ASECT allocation of memory.

Direct assignment (with .ASECT) of the stack address within the program takes precedence over assignment with the /M switch. The statements to do this in a MACRO program are as follows:

```
.ASECT
.=42
.WORD  INITSP   ;INITIAL STACK SYMBOL VALUE
.PSECT          ;RETURN TO PREVIOUS SECTION
```

The following example modifies the stack address:

```
*OUTPUT=INPUT/M
Stack symbol? BEG
```

3.5.11 Overlay Switch (/O:n)

The /O switch segments the load module so that the entire program is not memory-resident at one time. This lets you execute programs that are larger than the available memory. The argument n is an unsigned octal number (up to six digits) 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 you can specify only one overlay region on a command line. Overlay regions cannot be specified on the first command line; that is reserved for the root segment. You must use /C or // for continuation.

You specify co-resident overlay routines (a group of subroutines that occupy the overlay region and segment at the same time) as follows:

```
*OBJA,OBJB,OBJC/O:1/C
*OBJD,OBJE/O:1/C
.
.
.
```

All modules that the linker encounters until the next /O switch are co-resident overlay routines; that is, they all go into the same segment. If you specify, at a later time, the /O switch with the same value you used previously (same overlay region), then the linker opens up the corresponding overlay area for a new group of subroutines. This group occupies the same locations in memory as the first group, but it is never needed at the same time as the previous group. 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 following example establishes two overlay regions:

```
*OUTPUT,LP:=INPUT/ /
*OBJA/O:1
*OBJB/O:1
*OBJC/O:2
*OBJD/O:2
*/ /
```

You must specify overlays in ascending order by region number. For example:

```
*A=A/C
*B/O:1/C
*C/O:1/C
*D/O:1/C
*G/O:2
```

The following overlay specification is invalid because the overlay regions are not given in ascending numerical order. LINK prints an error message in each case, and ignores the overlay switch immediately preceding the message:

```
*X=LIBRO/ /
*LIBR1/O:1
*LIBR2/O:0
?LINK-W-/O or /V option error, re-enter line
*
```

In this example, LINK ignores the overlay line immediately preceding the error message and should be reentered with an overlay region number greater than or equal to one.

3.5.12 Library List Size Switch (/P:n)

The /P:n switch lets you change the amount of space allocated for the library routine list. Normally, the default value allows enough space for your needs. It reserves space for approximately 170 unique library routines, which is the equivalent of specifying /P:170. (decimal) or /P:252 (octal).

The following error message indicates that you need to allocate more space for the library routine list:

```
?LINK-F-Library list overflow, increase size with /P
```

You must relink the program that makes use of the library routines, and use the /P:n switch. Make sure you specify a value for n that is greater than 170.

You can use the /P:n switch to correct for symbol table overflow. Specify a value for n that is less than 170. This reduces the space used by the library routine list and increases the space allocated for the symbol table. If the value you choose is too small, LINK prints the message:

```
?LINK-F-Library list overflow, increase size with /P
```

In the following command, the amount of space for the library routine list is increased to 300 (decimal):

```
*SCCA=DM1:SCCA/P:300.
```

3.5.13 Absolute Base Address Switch (/Q)

The /Q switch lets you specify the absolute base addresses of up to eight p-sects in your program. This switch is particularly handy if you are preparing your program sections for placement in ROM storage. When you use this switch in the first command line, the linker prompts you for the p-sect names and load addresses. The p-sect name must be six characters or less, and the load address must be an even octal number. Press the RETURN key to terminate each line. If you press only the RETURN key in response to any of the prompts, LINK stops issuing a prompt.

If you use /E, /Y, or /U with /Q, LINK processes those switches before it processes /Q.

When you use the /Q switch, observe the following restrictions:

- Enter only even addresses. If you enter an odd address, no address, or invalid characters, LINK prints an error message and then prompts you again for the p-sect and load address.
- Do not use /Q with /H. These switches are mutually exclusive.
- LINK moves your p-sects up to the specified address; moving down might destroy code. If your address requires code to be moved down, LINK prints an error message, ignores the p-sect for which you have specified a load address, and continues.

The following example specifies the load addresses for three p-sects:

```
*FILE,TT:=FILE,FILE1/Q
Load Section:Address? PSECT1:1000
Load Section:Address? PSECT3:4000
Load Section:Address? PSECT2:2500
Load Section:Address? (RET)
```

3.5.14 Symbol Table Switch (/S)

The /S switch instructs the linker to allow the largest possible memory area for its symbol table at the expense of input and output buffer space. Because this makes the linking process slower, you should use the /S switch only if an attempt to link a program failed because of symbol table overflow. When you use /S, you cannot specify a symbol table file and a map in the command string.

3.5.15 Transfer Address Switch (/T[:n])

The transfer address is the address at which a program starts when you begin execution with the RUN command. It prints on the last line of the load map. The /T switch lets you specify the start address of the load module. The argument n is an unsigned octal number that defines the transfer address. If you do not specify n, LINK prints the following message:

```
Transfer symbol?
```

Specify the global symbol whose value is the transfer address of the load module. Terminate your response by pressing the RETURN key. You cannot specify a number in response to this message. If you specify a nonexistent symbol, LINK prints an error message and sets the transfer address to 1 so that the program is not executable. If the transfer address you specify is odd, the program does not start after loading. (A RUN command produces the message: ?Bad start address.)

Direct assignment (with .ASECT) of the transfer address within the program takes precedence over assignment with the /T switch. The transfer address assigned with a /T switch has precedence over that assigned with an .END assembly directive. To assign the transfer address within a MACRO program, use statements similar to these:

```
          .ASECT
          .=40
          .WORD     START1   ;SYMBOL VALUE FOR TRANSFER ADDRESS
          .PSECT             ;RETURN TO PREVIOUS SECTION

START1:     .
            .
            .

          or

START2:     .                ;SECONDARY STARTING ADDRESS
            .
            .
          .END      START2
```

The following example links the files LIBR0.OBJ and ODT.OBJ and starts execution at ODT's transfer address, O.ODT:

```
*LBRODT,LBRODT=LIBR0,ODT/T/W//
*LIBR1/O:1
*LIBR2/O:1
*LIBR3/O:1
*LIBR4/O:1
*LIBR5/O:1
*LIBR6/O:1
*LBREM/O:1//
Transfer symbol? O.ODT
*
```

3.5.16 Round Up Switch (/U:n)

The /U:n switch rounds up the section you specify in the root so that the size of the root segment is a whole number multiple of the value. The argument n must be a power of 2. When you specify the /U:n switch, the linker prompts:

```
Round section?
```

Reply with the name of the program section to be rounded, and then press the RETURN key. The program section must be in the root segment. Note that you can round only one program section. The following example rounds up section CHAR:

```
*LK007,KB:=LK007/U:200
Round section? CHAR
```

If the program section you specify cannot be found, LINK prints the following message and then continues the linking process with no rounding:

```
?LINK-W-Round section not found
```

3.5.17 Map Width Switch (/W)

The /W switch directs the linker to produce a wide load map listing. If you do not specify the /W switch, the listing is wide enough for three Global Value columns (normal for paper with 80 columns). If you use /W, the listing is six columns wide, which is suitable for a 132–column page.

3.5.18 Bitmap Inhibit Switch (/X)

The /X switch instructs the linker not to generate the bitmap if code is located between 360 and 377 inclusive. You use this switch to link the RSTS/E monitor. The bitmap is stored in locations 360 and 377 in block 0 of the load module, and the linker normally stores the program memory usage bits in these eight words. Each bit represents one 256–word block of memory. This information is required by the RUN command when loading the program; therefore, be careful when you use this switch.

3.5.19 Boundary Switch (/Y:n)

The /Y:n switch starts a specific program section in the root on a particular address boundary. Do not use this switch with /H. The linker generates a whole number multiple of n, the argument you specify for the starting address of the program section. The argument must be a power of 2. The

linker extends the size of the previous program section to accommodate the new starting address. When you have entered all input lines, the linker prompts:

```
Boundary section?
```

Respond with the name of the program section whose starting address you are modifying. Press the RETURN key to terminate your response. Note that you can specify only one program section for this switch. If the program section you specify cannot be found, the linker prints the following message and then continues:

```
?LINK-W-Boundary section not found
```

3.5.20 Zero Switch (/Z:n)

The /Z:n switch fills unused locations in the load module and places a specific value in these locations. The argument n represents that value. You can use this switch to eliminate random results that occur when the program references uninitialized memory by mistake. The linker automatically zeros unused locations. Use the /Z:n switch only when you want to store a value other than zero in unused locations. The /Z switch without an argument is equivalent to /Z:0. Thus, when n is equal to zero, you need to specify only /Z.

3.6 LINKER Prompts

Some of the linker operations prompt for more information, such as the names of specific global symbols or sections. The linker issues the prompt after you have entered all the input specifications, but before the actual linking begins. Table 3–10 shows the sequence in which the prompts occur.

Table 3–10: Linker Prompting Sequence

Prompt	Switch
Transfer symbol?	/T
Stack symbol?	/M
Extend section?	/E:n
Boundary section?	/Y:n
Round section?	/U:n
Load section:address?	/Q
Library search?	/I

The library search and the load section prompts can accept more than one symbol and are terminated by pressing the RETURN key in response to the prompt.

The following example shows how the linker prompts for information when you combine switches:

```
*LK001=LK001/T/M/E:100/Y:400/U:20/I/Q
Transfer symbol? 0.ODT
Stack symbol? ST3
Extend section? CHAR
Boundary section? CODE
Round section? STKSP
Load section:address? MAIN:100000
Load section:address? RET
Library search? $SHORT
Library search? RET
*
```

Chapter 4
Librarian (LIBR)

The librarian utility program (LIBR) lets you create, update, modify, list, and maintain object library files. It also allows you to create macro library files for use with the V03 and later versions of the MACRO–11 assembler.

4.1 The Librarian

A library file is a direct access file (a file that has a directory) that contains one or more modules of the same module type. The librarian organizes the library files so that the linker and MACRO–11 assembler can access them rapidly. Each library contains a library header, library directory (or global symbol table, or macro name table) and one or more object modules or macro definitions. The object modules in a library file can be routines that are:

- Repeatedly used in a program
- Used by more than one program
- Related and simply gathered together for convenience

Your needs determine the contents of the library file. An example of a typical object library file is the default system library SYSLIB.OBJ that the linker uses. An example of a macro library file is SYSMAC.SML, which MACRO uses to process .MCALL (macro call) directives.

You access object modules in a library file from another program by making calls or references to their global symbols. You then link the object modules with the program that uses them, producing a single load module (see Chapter 3).

The following sections describe how to:

- Run the librarian (Section 4.2)
- Use the librarian to create and maintain object libraries (Section 4.3)
- Create macro libraries (Section 4.4)

4.2 Running and Using LIBR

To run the librarian, type the following command in response to your keyboard monitor prompt:

```
RUN $LIBR (RET)
*
```

If your system manager has added a Concise Command Language (CCL) command, such as LIBR, type:

```
LIBR (RET)
*
```

You can also use the LIBR command to run the librarian whenever DCL is your keyboard monitor. In any case, when LIBR is ready to accept a command line, it prints an asterisk prompt (*) on your terminal. You then can enter a command string or type CTRL/Z to exit the program. Once you type a command line and press the RETURN key to begin execution, you must type CTRL/C to stop the librarian and return control to your keyboard monitor. Typing a CTRL/Z during the execution of a command has no effect. Use CTRL/Z only to exit LIBR at the asterisk prompt.

Specify the LIBR command string in the following general format:

library-filespec,list-filespec = input-filespec[/switch(es)],...

The definition of each file specification follows:

library-filespec	Represents the library file specification to be created or updated.
list-filespec	Represents a listing file for the library's contents.
input-filespec	Represents the input object modules (you can specify up to six input files); it can also represent a library file to be updated.
switch	Represents a switch from Table 4–1.

You specify devices and file names in the standard RSTS/E command string syntax (see Chapter 1), with default file types for object libraries assigned as follows:

Object File	Default File Type
List file	.LST
Library output file	.OBJ
Input file (library or module)	.OBJ

If you do not specify a device, DK: is assumed, which is normally equivalent to SY: (the public structure), unless you use the ASSIGN command to change it.

Each input file consists of one or more object modules and is stored on a given device under a specific file name and file type. Once you insert an object module into a library file, LIBR no longer references the module by the name of the file of which it was a part; instead you reference it by its individual module name. (But when referencing from other modules in LINK, for example, the global symbols in the modules are important, not the module name used in operations like LIBR deletes.) Use the assembler to assign this module name with either a .TITLE statement in the assembly source program (the default name is .MAIN. in the absence of a .TITLE statement) or the subprogram name for FORTRAN routines. Thus, for example, the input file FORT.OBJ can exist on DM2: and can contain an object module called ABC. Once you insert the module into a library file, reference only ABC (not FORT.OBJ).

The input files normally do not contain main programs but rather subprograms, functions, and subroutines. The library file must never contain a FORTRAN "BLOCK DATA" subprogram because there is no global symbol to cause the linker to load it automatically.

4.3 Switches and Functions for Object Libraries

You maintain object library files by using switches. The functions you can perform include object module deletion, insertion and replacement, library file creation, and listing of an object library file's contents.

Table 4–1 summarizes the switches used with LIBR for object libraries. The following sections, which are arranged alphabetically by switch, describe the switches in greater detail.

There is no switch to indicate module insertion. If you do not specify a switch, the librarian automatically inserts modules into the library file.

4.3.1 Include All Global and Absolute Global Symbols Switch (/A)

Normally, the librarian includes in the directory only global entry points (labels) and not absolute global symbols. Use the /A switch when you want all the global symbols to appear in the library file's directory. When you use /A, the librarian includes in the directory all absolute global symbols, including those that have a value of 0.

The following example places all the global symbols from module MOD1 and MOD2 in the library directory for ALIB.OBJ:

```
*ALIB=MOD1,MOD2/A
```

Table 4–1: LIBR Object Switches

Switch	Command Line	Section	Meaning
/A	First	4.3.1	Puts all globals in the directory, including all absolute global symbols.
/C	Any but last	4.3.2	Allows you to type the input specification on more than one line.
/D	First	4.3.5	Deletes modules (from a library file) that you specify.
/E	First	4.3.6	Extracts a module from a library and stores it in an .OBJ file.
/G	First	4.3.7	Deletes global symbols (from the library directory) that you specify. (The module containing the global being deleted is not itself deleted from the library.)
/N	First	4.3.8	Includes the module names in the directory.
/P	First	4.3.9	Includes the program section names (p-sect names) in the directory.
/R	First	4.3.10	Replaces modules in a library file. This switch must follow the file specification to which it applies.
/U	First	4.3.11	Inserts and replaces (updates) modules in a library file. This switch must follow the file specification to which it applies.
/W	First	4.3.12	Indicates wide format for the listing file.
/X	First	4.3.13	Allows multiple definitions of global entry points to appear in the library entry point table.
//	First and last	4.3.2	Allows you to type the input specification on more than one line.

4.3.2 Command Continuation Switches (/C and //)

You must use a continuation switch whenever there is not enough room to enter a command string on one line. The maximum number of input files that you can enter on one line is six; you can use the /C or // switch to enter more. Type the /C switch at the end of the current line, and repeat it at the end of subsequent command lines as often as necessary, so long as memory is available; if you exceed memory, LINK prints an error message. Each continuation line after the first command line can contain only input file specifications (and no other switches). Do not specify a /C switch on the last line of input. If you use the // switch, type it at the end of the first input line and again at the end of the last input line.

The following example creates a library file on DK: under the file name ALIB.OBJ. It also creates a listing of the library file's contents as LIBLST.LST (also on DK:). The file names of the input files (all from DM1:) are MAIN.OBJ, TEST.OBJ, FXN.OBJ, and TRACK.OBJ.

```
*ALIB,LIBLST=DM1:MAIN,TEST,FXN/C
*DM1:TRACK
```

The next example creates a library file on DK: under the name BLIB.OBJ. It does not produce a listing. Input files are MAIN.OBJ from the system device, TEST.OBJ from DM1:, FXN.OBJ from DM0:, and TRACK.OBJ from DB1:.

```
*BLIB=MAIN//
*DM1:TEST
*DM0:FXN
*DB1:TRACK//
```

Another way of writing this command line is:

```
*BLIB=MAIN,DM1:TEST,DM0:FXN//
*DB1:TRACK
*//
```

4.3.3 Creating a Library File

To create a library file, specify a file name on the output side of a command line. The following example creates a new library (on DK:) called NEWLIB.OBJ. The modules that make up this library file are in the files FIRST.OBJ and SECOND.OBJ, both on the system device.

```
*NEWLIB=FIRST,SECOND
```

Assume you then enter this command line:

```
*NEWLIB,LIST=THIRD,FOURTH
```

The existing library file NEWLIB.OBJ is lost when the new library file is created. A listing of the library file's contents is created under the file name LIST.LST. The object modules in the files THIRD.OBJ and FOURTH.OBJ are inserted into the library file NEWLIB.OBJ.

4.3.4 Inserting Modules into a Library

Whenever you specify an input file without specifying an associated switch, the librarian inserts the input file's modules into the library file you name on the output side of the command string. You can specify any number of input files. If you include section names (by using /P) in the global symbol table and if you attempt to insert a file that contains a global symbol or PSECT (or CSECT) having the same name as a global symbol or PSECT already existing in the library file, the librarian prints a warning message (see Section 4.3.13 for multiple definition library creation). The librarian does, however, update the library file, ignore the global symbol or section name in error, and then prompt you with an asterisk for another command line.

Although you can insert object modules even if the module name (as assigned by the .TITLE statement or SUBROUTINE name statement in FORTRAN) conflicts with that of a module already in the library, this

practice is not recommended because of possible confusion when you need to update these modules. (Sections 4.3.10 and 4.3.11 describe replacing and updating.)

NOTE

You must indicate the library file to which the operation is directed on both the input and output sides of the command line when making changes to an existing library; in effect, the librarian creates a "new" output library file each time it performs one of these operations. You must specify the library file first in the input field.

The following command line inserts the modules included in the files FA.OBJ, FB.OBJ, and FC.OBJ on DB1: into a library file named DXYNEW.OBJ on the system device. The resulting library also includes the contents of library DXY.OBJ.

```
*DXYNEW=DXY,DB1:FA,FB,FC
```

The next command line inserts the modules contained in files THIRD.OBJ and FOURTH.OBJ into the library NEWLIB.OBJ.

```
*NEWLIB,LIST=NEWLIB,THIRD,FOURTH
```

Note that the resulting library (1) contains the original library plus some new modules and (2) replaces the original library because the same name was used in this example for the input and output library.

4.3.5 Delete Switch (/D)

The /D switch deletes modules and all their associated global symbols from the library.

When you use the /D switch, the librarian prompts:

```
Module name?
```

Respond with the name of the module to be deleted, and then press the RETURN key. Continue until you have entered all modules to be deleted. Press RETURN immediately after the Module name? message to terminate input and to begin execution of the command line.

The following example deletes the modules SGN and TAN (on DM3:) from the library file TRAP.OBJ:

```
*DM3:TRAP=DB3:TRAP/D
Module name? SGN
Module name? TAN
Module name?
```

The next example deletes the module FIRST from the library LIBFIL.OBJ. All modules in the file ABC.OBJ replace old modules of the same name in the library. The example also inserts the modules in the file DEF.OBJ into the library:

```
*LIBFIL=LIBFIL/D,ABC/R,DEF
Module name? FIRST
Module name?
```

In the following example, the librarian deletes two modules of the same name from the library file LIBFIL.OBJ:

```
*LIBFIL=LIBFIL/D
Module name? X
Module name? X
Module name?
```

4.3.6 Extract Switch (/E)

The /E switch allows you to extract an object module from a library file and place it in an .OBJ file.

When you specify the /E switch, the librarian prints:

```
Global?
```

Respond with the name of a global symbol defined in the module you want to extract. If you specify a global name, the librarian extracts the entire module of which that global is a part. LIBR stops printing the Global? prompt if you press the RETURN key.

The following example extracts the ATAN routine from the FORTRAN library $SYSLIB.OBJ and stores it on DM1: in a file called ATAN.OBJ:

```
*DM1:ATAN=$SYSLIB/E
Global? ATAN
Global?
```

The next example extracts the $PRINT routine from $SYSLIB.OBJ and stores it on DM1: as PRINT.OBJ:

```
*DM1:PRINT=$SYSLIB/E
Global? $PRINT
Global?
```

You cannot use the /E switch in the same command line as another switch.

4.3.7 Delete Global Switch (/G)

The /G switch lets you delete a specific global symbol from a library file's directory. When you use the /G switch, the librarian prints:

```
Global?
```

Respond with the name of the global symbol you want to delete, and then press the RETURN key; continue until you have entered all globals to be deleted. Press the RETURN key immediately after the Global? prompt to end input and begin execution of the command line.

For example, the following command causes LIBR to delete the global symbols NAMEA and NAMEB from the directory found in the library file ROLL.OBJ on the system device:

```
*ROLL=ROLL/G
Global? NAMEA
Global? NAMEB
Global?
```

The librarian deletes globals from the directory only (and not from the library itself). The module containing the global symbol being deleted is not itself deleted. Whenever you update a library file, all globals that you previously deleted are restored, unless you use the /G switch again to delete them. This feature lets you recover if you delete the wrong global.

4.3.8 Include Module Names Switch (/N)

When you use the /N switch on the first line of the command, the librarian includes module names in the directory. The linker loads modules from libraries based on the fact that those modules define needed global symbols that were undefined in the linker's previous input files, not on the basis of module names. Normally, then, it is a waste of space and a performance compromise to include module names in the directory.

If you do not include module names in the directory, the MODULE column of the directory listing is blank, unless the module requires a continuation line to print all its globals. A plus sign (+) in the MODULE column indicates continued lines. The /N switch is most useful when you create a temporary library in order to obtain a directory listing.

If the library does not have module names in its directory, you must create a new library to include the module names. The following example illustrates how to do this. The library directory is listed on the terminal, and because the library output is to the null device (NL:), no output library file is actually generated. The current library OLDLIB remains unchanged.

```
*NL:,KB:=OLDLIB/N
RT-11 LIBRARIAN V04.00   TUE 10-NOV-81 20:36:41
NL:TEMP.OBJ              TUE 10-NOV-81 20:36:40

MODULE          GLOBALS          GLOBALS         GLOBALS

IRAD50          IRAD50           RAD50
JMUL            JMUL
LEN             LEN
SUBSTR          SUBSTR
JADD            JADD
JCMP            JCMP
```

4.3.9 Include P-section Names Switch (/P)

The librarian does not include program section names in the directory unless you use the /P switch on the first line of the command. The linker does not use section names to load routines from libraries. In fact, including the names can decrease linker performance. Including program section names also causes a conflict in the library directory and subsequent searches, because the librarian treats section names and global symbols identically.

This switch is provided for compatibility with RT–11 V2C. DIGITAL recommends that you avoid using it with RSTS/E.

4.3.10 Replace Switch (/R)

Use the /R switch to replace modules in a library file. The /R switch replaces existing modules in the library file you specify as output with the modules of the same names contained in the file(s) you specify as input. In the command string, enter the input library file before the files used in the replacement operation.

If an old module does not exist under the same name as an input module or if you specify the /R switch on a library file, the librarian prints an error message followed by the module name and ignores the replace command. The /R switch must follow each input file name containing modules for replacement. (An error results if any of the modules in the replacement file is absent from the library; other modules are still replaced. Thus, you may want to use the /U switch, which is less restricting.)

The following command line indicates that the modules in the file INB.OBJ are to replace existing modules of the same names in the library file TFIL.OBJ. The object modules in the files INA.OBJ and INC.OBJ (all files are stored on DK:) are to be added to TFIL.

```
*TFIL=TFIL,INA,INB/R,INC
```

The same operation occurs in the next command as in the preceding example, except that this updated library file is assigned the new name XFIL.

```
*XFIL=TFIL,INA,INB/R,INC
```

4.3.11 Update Switch (/U)

The /U switch lets you update a library file by combining the insert and replace functions. If the object modules that compose an input file in the command line already exist in the library file, the librarian replaces the old modules in the library file with the new modules in the input file. If the object modules do not already exist in the library file, the librarian inserts those modules into the library. (Note that some of the error messages that

might occur with separate insert and replace functions are not printed when you use the update function.) The /U switch must follow each input file that contains modules to be updated. Specify the input library file before the input files in the command line.

If the input file contains some modules that already exist in the library and some that do not, /U picks up all modules. The /R switch picks up only those that already exist, reporting "Illegal replacement of xxx" for those that do not. In contrast, insert (no switch) picks up only modules that do not yet exist in the library, causing "Illegal insert of xxx" for those that do. Thus, by choice of switches, you can perform the operation you want.

The following command line instructs the librarian to update the library file BALIB.OBJ on the system device. First the modules in FOLT.OBJ and BART.OBJ replace old modules of the same names in the library file, or if none already exist under the same names, the modules are inserted. The modules from the file TAL.OBJ are then inserted; the librarian prints an error message if the name of the module in TAL.OBJ already exists.

```
*BALIB=BALIB,FOLT/U,TAL,BART/U
```

In the next example, there are two object modules of the same name, X, in both Z and XLIB; these are first deleted from XLIB so that both the modules called X in file Z are correctly placed in the library. Globals SEC1 and SEC2 are also deleted from the directory but automatically return the next time the library XLIB.OBJ is updated.

```
*XLIB=XLIB/D,Z/U/G
Module name? X
Module name? X
Module name?
Global? SEC1
Global? SEC2
Global?
```

4.3.12 Wide Switch (/W)

The /W switch gives you a wider listing if you request a listing file. The wider listing has six Global columns instead of three, as in the normal listing. This is useful if you list the directory on a line printer or a terminal that has 132 columns.

4.3.13 Creating Multiple Definition Libraries Switch (/X)

The /X switch lets you create libraries that can have more than one definition for a global entry point. These libraries are called multiple definition libraries. They are processed differently from libraries that contain only one definition for each global entry point name that appears in the library's directory. For more information on processing multiple definition libraries, see Section 3.3.2.

In multiple definition libraries, two library modules may contain the same global symbol name, and both definitions will appear in the entry point table (EPT). At least one entry point name should be unique in each module so that you can easily identify it.

When you use the /X switch, the librarian does not issue the following message when it encounters a duplicate global symbol name. The global name appears in the directory for each module that defines it.

?LIBR–W–Illegal insert of AAAAAA

In addition, the /X switch causes the librarian to turn on the /N switch (see Section 4.3.8).

The following example creates the multiple definition library MLTLIB from modules MOD1, MOD2, and MOD3 and lists the library on the terminal:

```
*MLTLIB,KB:=MOD1,MOD2,MOD3/X

RT-11 LIBRARIAN V04.00   THU 12-NOV-81 09:45:31
DK:MLTLIB.OBJ            THU 12-NOV-81 09:45:31

MODULE          GLOBALS          GLOBALS          GLOBALS

MOD1            OMA$R            SWP$             ATP$
MOD2            ATP$             OMA$R            MER$CR
                LBM
MOD3            ATP$             OMA$R            MER$CR
                ENTZ
```

4.3.14 Listing the Directory of a Library File

You can request a listing of the contents of a library file (the global symbol table) by indicating both the library file and a list file in the command line. Because a library file is not being created or updated, you do not need to indicate the file name on the output side of the command line; however, you must use a comma to designate a null output library file.

The command syntax can be either of the following:

*,KB:=library-filespec
*,list-filespec=library-filespec

The definition of these file specifications is:

library-filespec	Represents the file specification for the existing library file
KB:	Indicates that the listing is to be sent directly to a terminal
list-filespec	Represents the file specification for the list file of the library file's contents

The following command stores a listing of all modules in the library file LIBFIL.OBJ (on the system device) in the file LIST.LST (on DM2:):

```
*,DM2:LIST=LIBFIL
```

The next command sends to a terminal a listing of all modules in the library file FLIB.OBJ, which is stored on the system device:

```
*,KB:=FLIB
```

Here is a sample section of a large directory listing:

```
*,KB:=SYSLIB
RT-11 LIBRARIAN V04.00   FRI 20-NOV-81 21:01:01
DK:SYSLIB.OBJ            FRI 20-NOV-81 20:59:47

MODULE          GLOBALS          GLOBALS          GLOBALS

                DCO$             ECO$             FCO$
+               GCO$             RCI$
                DIC$IS           DIC$MS           DIC$PS
+               DIC$SS           $DIVC            $DVC
                ADD$IS           ADD$MS           ADD$PS
+               ADD$SS           SUD$IS           SUD$MS
+               SUD$PS           SUD$SS           $ADD
```

The first line of the listing file shows the version of the librarian that was used and the current date and time. The second line prints the library file name and the date and time the library was created. Each line in the rest of the listing shows only the globals that appear in a particular module. If a module contains more global symbol names than can print on one line, a new line will be started with a plus sign (+) in column 1 to indicate continuation.

If you request a listing of a library file that was created with the /X or /N switch, the listing includes module names under the MODULE heading.

4.3.15 Merging Library Files

You can merge two or more library files under one file name by specifying in a single command line all the library files to be merged. The librarian does not delete the individual library files following the merge unless the output file name is identical to one of the input file names.

The command syntax is:

library-filespec = input-filespec,...

These file specifications have the following definitions:

library-filespec	Represents the library file that will contain all the merged files. (If a library file already exists under this name, you must also specify it in the input side of the command line so that it is included in the merge.)
input-filespec	Represents a library file to be merged.

The following command combines library files MAIN.OBJ, TRIG.OBJ, STP.OBJ, and BAC.OBJ (all files are on DK:) under the existing library file name MAIN.OBJ, replacing the old contents of MAIN.OBJ:

```
*MAIN=MAIN,TRIG,STP,BAC
```

The next command creates a library file named FORT.OBJ and merges existing library files A.OBJ, B.OBJ, and C.OBJ under the file name FORT.OBJ:

```
*FORT=A,B,C
```

NOTE

Library files should only be combined using the previous procedure; in particular, do not use the PIP program for this purpose. The resulting output is unacceptable to both LINK and LIBR.

4.3.16 Combining Library Switch Functions

You can request two or more library functions in the same command line, with the exception of the /E and /M switches, which cannot be specified on the same command line with any other switch. The librarian performs functions (and issues appropriate prompts) in the following order:

1. /C or //
2. /D
3. /G
4. /U
5. /R
6. Insertions
7. Listing

For example:

```
*FILE,LP:=FILE/D,MODX,MODY/R
Module name? XYZ
Module name? A
Module name?
```

The librarian performs the functions in this example in the following order:

1. Deletes modules XYZ and A from the library file FILE.OBJ
2. Replaces any duplicate of the modules in the file MODY.OBJ
3. Inserts the modules in the file MODX.OBJ
4. Lists the directory of FILE.OBJ on the line printer

4.4 Switch Commands and Functions for MACRO Libraries

The librarian lets you create macro libraries. A macro library works with the V03 and later MACRO–11 assembler.

The .MACRO directive produces the entries in the library directory (macro names). LIBR does not maintain a directory listing file for macro libraries; to list the macros in the library, print the ASCII input file.

The default input and output file type for macro library files is .MAC (using the /M switch).

If you give the library file the same name as one of the input files, the librarian prints the error message:

```
?LIBR-F-Output and input filenames the same
```

This prevents the deletion of an input file when the library is created.

The librarian removes all comments from your source input file except for those within a macro (that is, between a .MACRO and .ENDM pair of directives). Comments take up space during the assembly and in the library. If saving space and shortening assembly time are important to you, remove them from the macros wherever possible before creating a macro library. (This may make the macro expansions less clear, however.)

Table 4–2 summarizes the switches you can use with macro libraries. The switches are explained in detail in the following two sections.

Table 4–2: LIBR Macro Switches

Switch	Command Line	Section	Meaning
/C	Any but last	4.4.1	Command continuation; allows you to type the input specification on more than one line.
/M[:n]	First	4.4.2	Macro; creates a macro library from the ASCII input file containing .MACRO directives.
//	First and last	4.4.1	Command continuation; allows you to type the input specification on more than one line.

4.4.1 Command Continuation Switches (/C or //)

These switches for macro libraries are the same as for object libraries. See Section 4.3.2.

4.4.2 Macro Switch (/M[:n])

The /M[:n] switch creates a macro library file from an ASCII input file that contains .MACRO directives. The optional argument n determines the amount of space to allocate for the macro name directory by representing

the number of macros you want the directory to hold. Remember that n is interpreted as an octal number; you must follow n by a decimal point (n.) to indicate a decimal number. Each 64 macros occupies one block of library directory space. The default value for n is 128, enough space for 128 macros, which use 2 blocks for the macro name table.

The command syntax is:

library-filespec = input-filespec /M[:n]

Definitions for these file specifications follow:

library-filespec	Represents the macro library to be created
input-filespec	Represents the ASCII input file that contains .MACRO definitions
/M[:n]	Is the macro switch

The continuation switches (/C or //) are the only switches you can use with the macro switch.

The following example creates the macro library SYSMAC.SML from the ASCII input file SYSMAC.MAC. Both files are on the system device.

```
*SYSMAC.SML=SYSMAC/M
```

Chapter 5
Object Module Patch Utility (PAT)

This chapter describes how to use the Object Module Patch Utility (PAT).

5.1 Introduction to the PAT Utility

The PAT utility program allows you to update code in a relocatable binary object module (.OBJ). Unlike other programs such as ODT, PAT does not act like an editor, which usually allows you to inspect a module's octal contents. Instead, the program performs a merge of (1) the original input file and (2) a correction file containing the corrections and additions to the original file. The original input file consists of one or more concatenated object modules, only one of which can be corrected with a single execution of the PAT utility. The correction file consists of object code that, when linked by the linker, either replaces or appends to the original object module. Output from PAT is the updated input file. You then may need to use the linker to create an executable program, run the librarian to update the library, or do nothing if the corrected module typically exists as an object module.

Prior to using PAT, you must have created the correction file with a text editor and compiled or assembled it to create the correction file object module. Figure 5–2 illustrates the entire procedure, which results in an updated executable file. Note that it is always good practice to create a backup version of the file you want to patch before using PAT to make changes.

5.2 Running and Using PAT

To run PAT, type the following command in response to your keyboard monitor prompt:

```
RUN $PAT (RET)
*
```

If your system manager has installed PAT as a Concise Command Language command (for example, PAT), run the PAT program by typing:

```
PAT (RET)
*
```

The PAT command also works (if the CCL PAT is installed) when your default (or job) keyboard monitor is DCL. In any case, the PAT program prints an asterisk prompt (*) on your terminal indicating its readiness to accept command input. Chapter 1 describes the RSTS/E file specification format you use to construct command lines for the PAT utility program.

You specify a PAT command string in the form:

[output-filespec] = input-filespec[/C[:n]],correct-filespec[/C[:n]]

Parameters in the PAT command string have the following definitions:

output-filespec	Is the file specification for the output file. If you do not specify an output file, PAT does not generate one.
input-filespec	Is the file specification for the input file. This file can contain one or more concatenated object modules.
correct-filespec	Is the file specification for the correction file. This file contains the updates being made to a single module in the input file.
/C	Specifies the checksum switch for the associated file. This causes PAT to generate an octal value for the sum of all the binary data composing the module in that file.
number	Specifies an octal value. PAT compares the checksum value it computes for a module with the octal value you specify. The use of the checksum option is optional. If you include it in a file specification, you can specify /C alone without an argument (number).

Type CTRL/C to stop PAT at any time or CTRL/Z in response to the asterisk prompt to return control to your keyboard monitor.

Figure 5–1 shows how you use PAT to update a file (FILE1) consisting of three object modules (MOD1, MOD2, and MOD3) by appending a correction file to MOD2. After running PAT, you use the linker to relink the updated module with the rest of the file and to produce a corrected executable program.

Figure 5–1: Updating a Module Using PAT

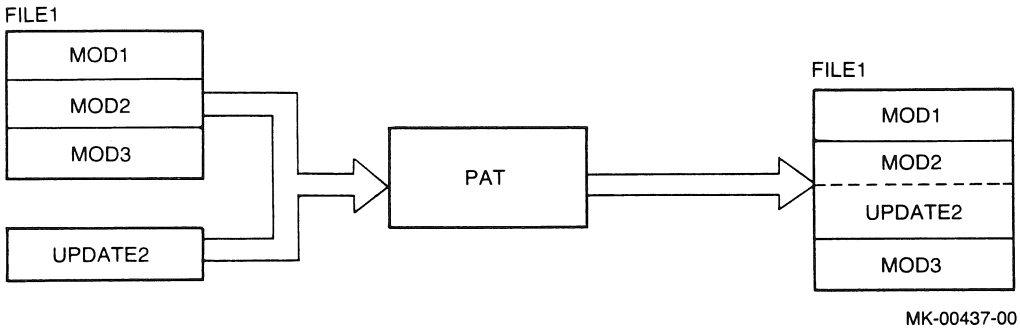
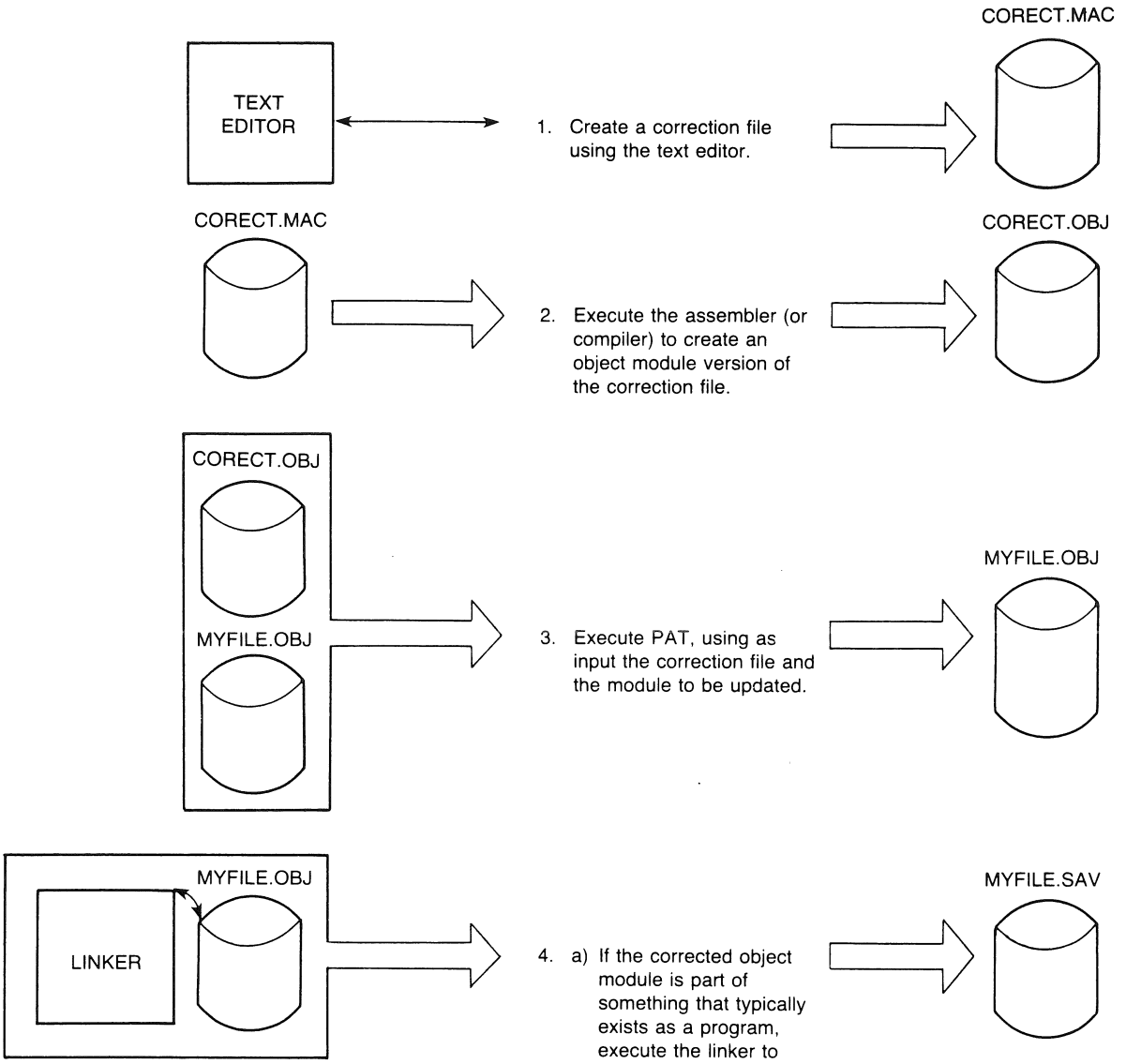


Figure 5–2: Processing Steps Required to Update a Module Using PAT



MK-00438-00

There are several steps you must follow when using PAT to update a file:

1. Use a text editor to create the correction file.
2. Assemble the correction file to produce an object correction module.
3. Submit the input file and the correction file in object module form to PAT for processing.
4. Link the updated object module, along with the object modules that make up the rest of the program, to create an executable program.

Figure 5–2 shows the processing steps involved in generating an updated executable file using PAT.

5.3 How PAT Updates a Module

PAT updates a base input module by using additions and corrections you supply in a correction file. This section describes the PAT input and correction files and gives information on how to create the correction file.

5.3.1 Input File

The input file is the file to be updated; it is the base for the output file and must be in object module format. When PAT executes, the module in the correction file is applied to this file.

5.3.2 Correction File

The correction file must be in object module format, and it is usually created from a MACRO–11 source file in the following format:

```
.TITLE inputname
[.IDENT updatenum]
[section name]
inputline
inputline
    *
    *
    *
```

Definitions of these parameters follow:

inputname	Is the name of the module to be corrected by the PAT update. That is, inputname must be the same name as the name on the input file .TITLE directive for a module in the input file that is to be corrected.

updatenum	Is any value acceptable to the MACRO–11 assembler. Generally, this value reflects the update version of the file being processed by PAT, as shown in the examples to follow.
section name	Is the ASECT, CSECT, or PSECT included in the correction file.
inputline	Are lines of input for PAT's use in correcting and updating the input file.

During execution, PAT adds any new global symbols that are defined in the correction file to the module's symbol table. Duplicate global symbols in the correction file supersede their counterparts in the input file, provided that both definitions are relocatable or both are absolute.

A duplicate PSECT or CSECT supersedes the previous PSECT or CSECT, provided that both have the same relocatability attribute (ABS or REL) — the relocatability attribute of a CSECT is REL. If PAT encounters duplicate PSECT names, it sets the length for the PSECT to the length of the longer PSECT and appends a new PSECT to the module. (Duplicate PSECT means a PSECT name in the correction file that matches the name of some PSECT in the original input module.)

If you specify a transfer address, it supersedes that of the module you are patching.

5.4 Updating Object Modules

The following examples show the source code for an input file and a correction file to be processed by PAT and the linker. The examples show as output a single source file that, if assembled and linked, would produce a binary module equivalent to the file generated by PAT and LINK. Two techniques are described: one is for overlaying lines in a module, and the other is for appending a subroutine to a module.

5.4.1 Overlaying Lines in a Module

In the following example, PAT first appends the correction file to the input file. The linker is then executed to replace code within the input file.

The input file for this example is:

```
        .TITLE  ABC
        .IDENT  /01/
        .ENABL  GBL
ABC::
        MOV     A,C
        JSR     PC,XYZ
        RTS     PC
        .END
```

To add the instruction ADD A,B after the JSR instruction, the following patch source file is included:

```
          .TITLE   ABC
          .IDENT   /01.01/
          .ENABL   GBL
.=.+12
          ADD      A,B
          RTS      PC
          .END
```

Note that both the original and the patch files use the "blank" PSECT by default (because no .PSECT or .CSECT directive is present). The patch source is assembled using MACRO–11 and the resulting object file becomes the input to PAT along with the original object file. The following source code represents the result of PAT processing:

```
          .TITLE   ABC
          .IDENT   /01.01/
          .ENABL   GBL
ABC::
          MOV      A,C
          JSR      PC,XYZ
          RTS      PC
.=ABC
.=.+12
          ADD      A,B
          RTS      PC
          .END
```

After the linker processes these files, the load image appears, as this source code representation shows:

```
          .TITLE   ABC
          .IDENT   /01.01/
          .ENABL   GBL
ABC::
          MOV      A,C
          JSR      PC,XYZ
          ADD      A,B
          RTS      PC
          .END
```

The linker uses the .=.+12 in the program counter field to determine where to begin overlaying instructions in the program and, finally, overlays the RTS instruction with the patch code:

```
ADD       A,B
RTS       PC
```

5.4.2 Adding a Subroutine to a Module

In many cases, a patch requires that more than a few lines be added to patch the file. A convenient technique for adding new code involves appending it to the end of the module in the form of a subroutine. This way, you can insert a JSR instruction to the subroutine at an appropriate location. The JSR directs the program to branch to the new code, execute that code, and then return to in-line processing.

The source code for the input file for the example is:

```
        .TITLE  ABC
        .IDENT  /01/
        .ENABL  GBL
ABC::
        MOV     A,B
        JSR     PC,XYZ
        MOV     C,R0
        RTS     PC
        .END
```

Suppose you wish to add the instructions:

```
MOV     D,R0
ASL     R0
```

between

```
MOV     A,B
```

and

```
JSR     PC,XYZ
```

The correction file to accomplish this is:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
        JSR     PC,PATCH
        NOP
        .PSECT  PATCH
PATCH:
        MOV     A,B
        MOV     D,R0
        ASL     R0
        RTS     PC
        .END
```

PAT appends the correction file to the input file, and the linker then processes the file, generating the following output file:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
ABC::
        JSR     PC,PATCH
        NOP
        JSR     PC,XYZ
        MOV     C,R0
        RTS     PC
        .PSECT  PATCH
PATCH:
        MOV     A,B
        MOV     D,R0
        ASL     R0
        RTS     PC
        .END
```

In this example, the JSR PC,PATCH and NOP instructions overlay the three-word MOV A,B instruction. (The NOP is included because this is a case where a two-word instruction replaces a three-word instruction. NOP is required to maintain alignment.) The linker allocates additional storage for .PSECT PATCH, writes the specified code into this program section, and binds the JSR instruction to the first address in this section. Note that the MOV A,B instruction, replaced by the JSR PC,PATCH, is the first instruction the PATCH subroutine executes.

5.5 Determining and Validating the Contents of a File

Use the checksum switch (/C) to determine or validate the contents of a module. The checksum switch directs PAT to compute the sum of all binary data composing a file. If you specify the command in the form /C:n, PAT computes the checksum and compares that checksum to the value you specify as n.

To determine the checksum of a file, enter the PAT command line with the /C switch applied to the file whose checksum you want to determine. For example, PAT responds to the command =INFILE/C,INFILE.PAT with the message:

```
?PAT-W-Input module checksum is nnnnnn
```

PAT generates a similar message when you request the checksum for the correction file.

To validate the changes made to a file, enter the checksum switch in the form /C:n. PAT compares the value it computes for the checksum with the value you specify as n. If the two values do not match, PAT enters the changes but displays one of the following two messages reporting the checksum error:

1. ?PAT-W-Input file checksum error
2. ?PAT-W-Correction file checksum error

Checksum processing always results in a nonzero value.

Appendix A
Switch and Argument Summary

A.1 MACRO Switches

At assembly time you may need to override certain MACRO directives appearing in the source programs. You may also need to direct MACRO–11 on the handling of certain files during assembly. You can satisfy these needs by including special switches in the MACRO–11 command string in addition to the file specifications. A table of the switches and a description of each follows.

Table A–1: File Specification Switches

Option	Usage
/L[:arg]*	Listing control, overrides source program directives .LIST and .NLIST
/N[:arg]*	Listing control, overrides source program directives .LIST and .NLIST
/E:arg**	Object file function enabling, overrides source program directives .ENABL and .DSABL
/D:arg**	Object file function disabling, overrides source program directives .ENABL and .DSABL
/M	Indicates input file is a MACRO library file
/C[:arg]	Requests or controls the contents of cross-reference listing
/P:arg	Specifies whether input source file is to be assembled in pass 1 or pass 2 only, rather than in both passes

* Both /L and /N disable .LIST and .NLIST for the argument(s) specified; however, /L turns it on, and /N turns it off.

** Both /E and /D disable .ENABL and .DSABL for the argument(s) specified; however, /E turns it on, and /D turns it off.

Refer to the text in Section 2.3 for a complete description of these switches.

A.1.1 Arguments for Listing Control Switches

Two switches, /L:arg and /N:arg, affect listing control. By specifying these switches with a set of selected arguments, you can control the content and format of assembly listings. You can override the arguments of .LIST and .NLIST directives in the MACRO source program. Table A–2 lists the arguments you use with the /L and /N switches.

Table A–2: Arguments for /L and /N Switches

Argument	Default	Listing Control
SEQ	List	Source line sequence number
LOC	List	Address location counter
BIN	List	Generated binary code (includes BEX)
BEX*	List	Binary extensions
SRC	List	Source code
COM	List	Comments
MD	List	Macro definitions, repeat range definitions
MC	List	Macro calls, repeat range expansion
ME	No list	Macro expansions (includes MEB)
MEB	No list	Macro expansion binary code
CND	List	Unsatisfied conditionals, .IF and .ENDC statements
LD	No list	List control directives with no arguments
TOC	List	Table of Contents
TTM	No list	132–column line printer format when not specified, terminal mode (80–column mode) when specified
SYM	List	Symbol table

* This option applies to the listing of assembled binary code. There is room on a listing line to display three octal words (one if TTM is set) of assembled code. If you assemble a source statement that assembles to more than three words, only the first three are listed if .NLIST BEX is in effect. If .LIST BEX is in effect, MACRO uses additional lines to list all assembled words.

Read more about the listing control switches in Section 2.3.1.

A.1.2 Arguments for Function Control Switches

Two switches, /E:arg and /D:arg, allow you to enable or disable functions at assembly time, and thus influence the form and content of the binary object file. These functions can override .ENABL and .DSABL directives in the source program. The following table summarizes the acceptable /E and /D function arguments, their normal default status, and the functions they control.

Table A–3: Arguments for /E and /D Switches

Argument	Default Mode	Function
ABS	Disable	Produces output in paper tape absolute binary format instead of a standard object file.
AMA	Disable	Assembles all relative addresses as absolute addresses. Replaces all uses of relative addressing mode (mode 67) by absolute addressing (mode 37).
CDR	Disable	Ignores all source information beyond column 72.
CRF	Enable	Allows cross-reference listing. Disabling this function inhibits CREF output even if switch /C is active.
FPT	Disable	Truncates floating point values (instead of rounding).
GBL	Disable	Treats undefined symbols as globals.
LC	Disable	Allows lowercase ASCII source input.
LSB	Disable	Allows local symbol block (not recommended in /E:arg or /D:arg).
PNC	Enable	Allows binary output.
REG	Enable	Automatically defines register mnemonics if enabled. You should set or clear the REG argument at the beginning of the source module.

See Section 2.3.2 for more information about these arguments.

A.1.3 Arguments for the Cross-Reference Switch (/C)

A complete cross reference contains six sections: (1) program symbols, (2) register symbols (if the REG switch has been disabled), (3) MACRO symbols, (4) permanent symbols, (5) program sections, or (6) errors. You can include any or all of these six sections on the cross-reference listing by specifying the appropriate arguments with the /C:arg switch. Table A–4 summarizes these arguments.

Table A–4: /C Switch Arguments

Argument	CREF Section
S	User-defined symbols
R	Register symbols
M	MACRO symbolic names
P	Permanent symbols including instructions and directives
C	Control and program sections
E	Error code grouping

NOTE

Specifying /C with no arguments is equivalent to specifying /C:S:M:E:. Except for that special case, you must explicitly request each CREF section by including its arguments. The /C switch must be used to produce a cross-reference file even if the command string includes a CREF file specification.

Refer to Section 2.3.4 for more information about obtaining a complete CREF listing.

A.2 LINK Switches

The table of switches that follows is associated with the linker. You must precede the letter representing each switch by the slash character. Switches must appear on the line indicated if you continue the input on more than one line, but you can position them anywhere on the line. The column titled Command Line lists on which line in the command string the switch can appear.

Table A–5: Linker Switches

Switch Name	Command Line	Section	Explanation
/A	First	3.5.1	Lists global symbols in program sections in alphabetical order in the load map.
/B:n	First	3.5.2	Changes the bottom address of a program to n (invalid with /H).
/C	Any but last	3.5.3	Continues input specification on another command line. (You can also use /C with /O; however, do not use /C with the // switch.)
/E:n	First	3.5.4	Extends a particular program section in the root to a specific value.
/F	First	3.5.5	Instructs the linker to use the default FORTRAN library, $FORLIB.OBJ, to resolve any undefined global references. Do not specify this switch in the command line when $FORLIB has been incorporated into $SYSLIB.
/G	First	3.5.6	Adjusts the size of the linker's library directory buffer to accommodate the largest multiple definition library directory.
/H:n	First	3.5.7	Specifies the top (highest) address to be used by the relocatable code in the load module. Invalid with /B, /Y, or /Q.
/I	First	3.5.8	Allows you to specify additional external global symbols to be satisfied (typically from the libraries). In general, this is used to explicitly request the inclusion of additional library modules.

(continued on next page)

Table A–5: Linker Switches (Cont.)

Switch Name	Command Line	Section	Explanation
/K:n	First	3.5.9	Inserts the value you specify (the valid range for n is from 1 to 28) into word 56 of block 0 of the image file. This switch informs the RT11 run-time system that the program requires nK words of memory.
/M[:n]	First	3.5.10	Causes the linker to prompt you for a global symbol that represents the stack address (if n is omitted) or that sets the initial stack address to the value n (if n is specified).
/O:n	Any but first	3.5.11	Indicates that the program is an overlay structure; n specifies the overlay region to which the module is assigned.
/P:n	First	3.5.12	Changes the default amount of space the linker uses for a library routines list.
/Q	First	3.5.13	Lets you specify the base addresses of up to eight root program sections. Invalid with /H.
/S	First	3.5.14	Makes the maximum amount of space in memory available for the linker's symbol table. (Use this switch only when a particular link stream causes a symbol table overflow.)
/T[:n]	First	3.5.15	Causes the linker to prompt you for a global symbol that represents the transfer address (if n is omitted) or that sets the transfer address to the value n (if n is specified).
/U:n	First	3.5.16	Rounds up the root program section you specify so that the size of the root segment is an integer multiple of the value you supply (n must be a power of 2).
/W	First	3.5.17	Directs the linker to produce a wide load map listing.
/X	First	3.5.18	Does not output the bitmap if the area normally used by the bitmap (location 360–377) is used by code.
/Y:n	First	3.5.19	Starts a specific program section in the root on a particular address boundary. Invalid with /H.
/Z:n	First	3.5.20	Sets unused locations in the load module to the value n (if n is omitted, the linker uses zero as the default).
//	First and last	3.5.3	Allows you to specify command string input on additional lines. Do not use this switch with /C.

A.3 LIBR Switches

You maintain object library files by using switch commands. Functions you can perform include object module deletion, insertion and replacement, library file creation, and listing of an object library file's contents. The following table summarizes the switches available for you to use with LIBR.

Table A–6: LIBR Object Switches

Switch	Command Line	Section	Meaning
/A	First	4.3.1	Puts all globals in the directory, including all absolute global symbols.
/C	Any but last	4.3.2	Allows you to type the input specification on more than one line.
/D	First	4.3.5	Deletes modules (from a library file) that you specify.
/E	First	4.3.6	Extract modules from a library and stores it in an .OBJ file.
/G	First	4.3.7	Deletes global symbols (from the library directory) that you specify. (The module containing the global being deleted is not itself deleted from the library.)
/N	First	4.3.8	Includes the module names in the directory.
/P	First	4.3.9	Includes the program section names (p-sect names) in the directory.
/R	First	4.3.10	Replaces modules in a library file. This switch must follow the file specification to which it applies.
/U	First	4.3.11	Inserts and replaces (updates) modules in a library file. This switch must follow the file specification to which it applies.
/W	First	4.3.12	Indicates wide format for the listing file.
/X	First	4.3.13	Allows multiple definitions of global entry points to appear in the library entry point table.
//	First and last	4.3.2	Allows you to type the input specification on more than one line.

Appendix B
Error Message Summary

The utilities print error messages in the format: ?UTILITY–n–message. The first character of each error message is a question mark (?) followed by the name of the utility in uppercase letters. The single-character code n indicates whether the error was a fatal (F) or a warning (W) message:

FATAL — messages cause the current command or statement to be ignored; you can usually correct the error by entering another command.

WARNING — messages indicate an error condition that may affect execution at a later time. A message of this type may require some attention.

A message includes: (1) a description of the conditions that may have caused the error and (2) methods to recover. The error messages are listed in alphabetical order.

Before trying to interpret and then correct the error conditions generated by the utilities in this manual, you should be aware of the following comments:

COMMENT #1

There may be times when error messages returned by the utilities do not help you identify the condition causing the error. This results from having a large set of RSTS/E error messages map into a smaller set provided by RT11. RSTS/E chooses the most logical RT11 error when the emulator encounters an error condition, but the RT11 error message that is chosen may not retain the "flavor" provided by the original. For example, many RSTS/E error conditions are mapped into the RT–11 "device full" error. The intent is to convey "cannot write output file." But the actual cause of the error may have nothing to do with "device full," which is a relatively rare occurrence on RSTS/E. When situations like this occur and you are

using RT11 as your primary run-time system, type CTRL/C to return to the dot prompt (.) generated by RT11. Typing ERR at that point and pressing the RETURN key causes the system to print the original RSTS/E error message. An example of this follows:

```
.RUN $MACRO
*SY:=KB:
?MACRO-F-Device full
*^C

.ERR (RET)
?Illegal file name
.
```

The resulting error message should, as in this case, more accurately reflect the condition that caused the error.

COMMENT #2

There are a number of errors in LINK, LIBR, and PAT that indicate "bad input." You may be able to correct the error condition if you follow these procedures:

1. Verify that you are using the correct file(s).
2. Assemble or compile the file(s) again.
3. Retry the operation.
4. Submit an SPR if the problem persists. The error might be an compiler or assembler bug or a LINK/LIBR/PAT bug. Be sure when you submit an SPR to include a copy of the dialogue used and machine-readable copies of both the source and the object files.

COMMENT #3

A few of the error messages refer to the word "option" which in this context has the same meaning as the term "switch" used in RSTS/E documentation. Switch refers to the combination of a slash character (/) and a word which cause the MACRO, LINK, LIBR, or PAT utility to act in a prescribed way. The word "switch" has been used here to comply with other RSTS/E documentation.

B.1 MACRO Error Messages

A list of MACRO assembler error messages follows:

?MACRO–F–Bad option

The specified switch was not recognized by the program. Check for a typing error in the command line. Use only a valid listing control or functional control (or CREF) switch.

?MACRO–F–Device full DEV:

The output device does not have enough room for an output file specified in the command string. Increase storage space or specify another device. Refer to COMMENT #1 at the beginning of this appendix.

?MACRO–F–File not found DEV:FILENAME.TYPE

The input file in the command line does not exist on the specified device. Correct any file specification errors in the command line and retype.

?MACRO–F–Illegal command

The command line contains a syntax error or specifies more than six input files. Correct the command line and retype.

?MACRO–F–Illegal device DEV:

The device specified in the command line does not exist on the system. Specify a different device name.

?MACRO–F–Insufficient memory

There were too many symbols, macro, or nested repeat blocks in the program being assembled. Increase memory space. Try to reduce the complexity of nested macro calls.

?MACRO–F–I/O error on DEV:FILENAME.TYPE

A hardware error occurred during a read from or write to the specified file.

?MACRO–F–I/O error on work file

MACRO failed to read, write, or open its work file, WRK.TMP. Free up some space on the public structure or specify a different device for the work file.

?MACRO–F–Invalid macro library

The library file has been corrupted, or it was not produced by the librarian LIBR. Use LIBR to generate a new copy of the library.

?MACRO–F–Output device full on DEV:FILENAME.TYPE

There was no room to continue writing the output file. Increase storage area. Refer to COMMENT #1 at the beginning of this appendix.

?MACRO–F–Read error on MACRO library

MACRO detected a bad record in the MACRO library. This error can occur when the library is bad. Rebuild the MACRO library.

?MACRO–F–Storage limit exceeded (64K)

MACRO's Virtual Symbol Table can store symbols and macros up to 64K words in any combination. The program contains more than 64K total of these elements. Check for a condition that leads to excessive size, such as a macro expansion that recursively calls itself without a terminating condition. If necessary, reduce the requirements of the source program by segmenting it into separate modules, and assemble each of them separately.

?MACRO–W–I/O error on CREF file: CREF aborted

Either there is not enough space to perform the operation, or an I/O error occurred while the CREF work file was being written. CREF processing is terminated but the assembly will continue. Increase storage space, specify a different device for the CREF file, or correct the cause of the I/O error.

B.2 LINK Error Messages

A list of LINK error messages follows:

?LINK–F–Address space exceeded

The high limit of all program sections exceeded 32K words when all sections were concatenated. Reduce the size of the program by using overlays or by reducing the size of the largest segment within each overlay region.

?LINK–F–ASECT too big

An absolute section overlaps into an occupied area of memory or an overlay region. Locate a segment of available memory large enough to contain the absolute section, and substitute the appropriate starting address.

?LINK–F–/B No value

No argument was specified to the /B switch. Reenter the command string, specifying an unsigned, even, octal number as the argument to the /B switch.

?LINK–F–/B Odd value

The argument to the /B switch was not an even number. Reenter the command string, specifying an even number as the argument. This error indicates that the object module was bad (or perhaps not a legal object module).

?LINK–F–Bad complex relocation in DEV:FILENAME.TYPE

During pass 2 of the linker, a complex relocation string in the input file was found to be invalid. Check for a typing error in the command line; verify that the correct file names were specified as input. Refer to COMMENT #2 at the beginning of this appendix.

?LINK–F–Bad GSD in DEV:FILENAME.TYPE

There was an error in the global symbol directory (GSD). The file is probably not an legal object module. Verify that the correct file names were specified as input; check for a typing error in the command line. Refer to COMMENT #2 at the beginning of this appendix.

?LINK–F–Bad RLD in DEV:FILENAME.TYPE

An invalid relocation directory (RLD) command exists in the input file. The file is probably not a legal input module. Check for a typing error in the command line; verify that correct file names were specified as input. Refer to COMMENT #2 at the beginning of this appendix. (In addition to the remarks in COMMENT #2, check the source code to make sure that all modules contributing to a data p-sect are word-aligned.)

?LINK–F–Bad RLD symbol in DEV:FILENAME.TYPE

An error occurred in the language processor because a global symbol named in a relocatable record was not defined in the global symbol definition record. The object file is bad. Refer to COMMENT #2 at the beginning of this appendix.

?LINK–F–/H Value too low

The value specified as the high address for linking was too small to accommodate the code. Obtain map output without using the /H switch to determine the space required, and then retry the operation.

?LINK–F–Illegal character

The character specified was not used in the proper context. Examine the command string for errors in syntax, making sure that the characters for symbols are legal Radix–50 characters. Correct and retype.

?LINK–F–Illegal device

The device indicated was not available. Verify that the device name is valid for the system in use.

?LINK–F–Illegal error

An internal error occurred while the linker was recovering from a previous system or user error. Refer to COMMENT #2 at the beginning of this appendix.

?LINK–F–Illegal record type in DEV:FILENAME.TYPE

A formatted binary record had a type not in the range 1–10 (octal). Refer to COMMENT #2 at the beginning of this appendix.

?LINK–F–Insufficient memory

There was not enough memory to accommodate the symbol table, or other buffers used by LINK. Try linking without the /G, /P, or /S switch (if used), reduce the number of globals used, or use fewer libraries.

?LINK–F–Library EPT too big, increase buffer with /G

The /G switch was not specified in RT11 and a /X library with too large an Entry Point Table was encountered. Relink, and issue /G switch on first input line.

?LINK–F–Library list overflow, increase size with /P

The linker's library routing list was exceeded. Relink the program that uses the library routines. The /P:n switch default is 170 (decimal). Increase the size of the list by specifying a size greater than the default.

?LINK–F–/M Odd value

An odd value was specified for the stack address. Check for a typing error in the command line. Reenter the command, specifying an even value to the /M switch.

?LINK–F–Map device full DEV:FILENAME.TYPE

There was no room in the directory for the file name, or there was no room on the output device for the map file. Increase storage space or use another device. Refer to COMMENT #1 at the beginning of this appendix.

?LINK–F–Old library format in DEV:FILENAME.TYPE

The format of the library file is outdated (previous to Version 2C). Rebuild the library file using the current librarian.

?LINK–F–Protected file already exists DEV:FILENAME.TYPE

An attempt was made to open a file using a name already associated with an existing protected file. Use a different name to open a new file, rename the file with PIP, change the protection code, or use another file.

?LINK–F–Read error in DEV:FILENAME.TYPE

A hardware error occurred while the indicated input file was being read. Check for read-locked or off-line devices.

?LINK–F–SAV device full DEV:FILENAME.TYPE

There was no room in the directory for the file name, or there was no room on the output device for the SAV image. Increase storage space. Refer to COMMENT #1 at the beginning of this appendix.

?LINK–F–SAV read error

A hardware error occurred while LINK was reading the output SAV file. Check for read-locked or off-line devices.

?LINK–F–SAV write error

A hardware error occurred while LINK was writing the SAV image file. The device may be full or you protected the file against yourself (that is, *FOO<63>=BAR). Check for write-locked or off-line devices, use another device, free up space, or fix the command line.

?LINK–F–Size overflow of section AAAAAA

The program section in question increased program size to more than 32K words. Reduce the size of the program, either in this section or elsewhere in the program.

?LINK–F–STB device full DEV:FILENAME.TYPE

There was no room in the directory for the file name, or there was no room on the output device for the symbol table (STB) file. Increase storage space or use another device. Refer to COMMENT #1 at the beginning of this appendix.

?LINK–F–STB not allowed with /S and a map

An attempt was made to produce STB and MAP in the same linking operation, which is prohibited with the /S switch. Produce STB and MAP files in separate linking operations.

?LINK–F–STB write error

A hardware error occurred while LINK was writing the symbol table (STB) file. The device may be full, or you protected the file against yourself (that is, *FOO<63>=BAR). Check for write-locked or off-line devices.

?LINK–F–Storing text beyond high limit

An input object module may have caused the linker to store information in the image file beyond the high limit of the program; there is an error condition in the object module. Reassemble or recompile the program. Submit an SPR if the condition persists.

The amount of space allocated for the output file was insufficient, or there was not enough room on the output device for the output file. Specify a larger output file size, or increase storage space.

?LINK–F–Symbol table overflow

Too many global symbols were used in the program. Retry the link, using the /S switch. If the error still occurs, reduce the size of the library list using the /S and /P:n switches, with a value less than the default (170). If the error continues, the link cannot take place in the available memory. Reduce the number of globals used.

?LINK–F–/T Odd value

An odd value was specified for the transfer address. Check for a typing error in the command line. Reenter the command, specifying an even value to the /T switch.

?LINK–F–Too many program segments

More than 1023 program segments were specified. Restructure overlays to reduce the number.

?LINK–F–/U or /Y value not a power of 2

The value specified with the /U or the /Y switch is not a power of 2. Reenter the command with a value that is a power of 2.

?LINK–F–Word relocation error in FILENAME

During concatenation of data p-sects, a word reference was moved to an odd byte. Place the .EVEN assembler directive at the end of data p-sects to make sure that all word references in data p-sects are on a word boundary when relocated by LINK.

?LINK–W–Additive reference of NNNNNN a segment # MMMMMM

A call or a jump to an overlay segment was not made directly to an entry point in the segment. NNNNNN represents the entry point; MMMMMM represents the segment number. Make sure that calls or jumps to overlay segments are made directly to entry points in the segment. See Section 3.4 for more information about using overlays.

?LINK–W–Bad option: /a

The linker did not recognize the /a switch (/a represents the unrecognized switch) specified in the command line, or an illegal combination of switches was used. If the bad switch occurred in the first command line, the entire command line is ignored and LINK prompts for a new command; enter another command. If the bad switch occurred on a subsequent command line, the switch is ignored and processing continues. In a continued command line, make sure that the only switches used are /O, /V, /C, and //. See Section 3.2.3 for a list of valid switches. Reexamine the command line and check for a typing error.

?LINK–W–Boundary section not found

The program section name specified as a boundary section with the /Y switch was not found in the modules that were linked; or the program section does not exist in the root segment. The linker continues after the warning, without performing the /Y operation. Check the responses to the Boundary section? prompt, and use the correct section name the next time you link.

?LINK–W–Byte relocation error at NNNNNN

The linker attempted to relocate and link byte quantities but failed because the high byte of the relocated value (or the linked value) was not all zeros. NNNNNN represents the address at which the error occurred.

The relocated value is truncated to eight bits and the linker continues processing. Correct the source program so that no overflow occurs in relocated byte quantities. Reassemble and relink.

?LINK–W–Complex relocation divide by 0 in DEV:FILENAME.TYPE

An attempt was made to divide by zero in a complex relocation string in the file indicated. A result of zero is returned and linking continues. Check uses of division in complex relocation string expressions to keep the result of the division from equaling zero.

?LINK–W–Conflicting section attributes AAAAAA

The program section symbol was defined with different attributes. The attributes of the first definition are used and the linking process continues. Check the source program, and use the desired section attributes for that program section.

?LINK–W–Default system library not found SYSLIB.OBJ

The linker did not find $SYSLIB.OBJ on the public structure when undefined globals existed or when overlays were being used. Obtain a copy of $SYSLIB.OBJ from backup and relink the program, or correct the source files by removing the undefined globals listed on the terminal. SYSLIB in the system library account [1,2] contains the overlay handlers, which are required when overlays are specified.

?LINK–W–Extend section not found

The extend section name given with the /E switch was not found in the modules that were linked; or the extend section did not exist in the root segment. The linker continues after the warning, without performing the extend operation. Check the response to the Extend section? prompt, and use the correct section name the next time you link.

?LINK–W–File not found DEV:FILENAME.TYPE

The input file indicated was not found. Check for a typing error in the command line. Verify that the file name exists as entered in the command line, and retry the operation.

?LINK–W–Load address odd

An odd load address was specified with the /Q switch. Reenter the line with an even address.

?LINK–W–Load address too low AAAAAA

The load address specified for the p-sect was too low. The p-sect was ignored to avoid overlaying code in a previous section.

Link continues execution without loading the p-sect at the specified address. Relink and specify a higher load address for the p-sect.

?LINK–W–Load section not found AAAAAA

The load section specified was not found in the root or did not exist in the root segment.

LINK continues execution, ignoring the placement request for the p-sect in question. Reorder the modules to place the p-sect containing the load section in the root or specify the correct name, and then relink.

?LINK–W–Map write error

A hardware error occurred while the map output file was being written. The map output is terminated and the linking process continues.

?LINK–W–Multiple definition of symbol

The symbol indicated was defined more than once. Extra definitions are ignored. Make sure each symbol is defined only once.

?LINK–W–No load address

No address was specified with the /Q switch. Reenter the command line, and specify a load address.

?LINK–W–/O or /V option error, re-enter line

An error was made in the use of the /O switch. There are two probable errors: (1) no value was given with the /O switch, or (2) a value was given but it is incorrect. Check the context and reenter the line.

?LINK–W–Round section not found AAAAAA

The symbol representing the program section specified with the /U switch was not found in the symbol table. Linking continues with no round-up action. Check the source to make sure the p-sect exists in the root.

?LINK–W–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 default 1000. Check for a typing error in the command line. Verify that the stack address or global symbol is not defined in an overlay segment.

?LINK–W–Transfer address undefined or in overlay

The transfer address was not defined or was in an overlay. Check for a typing error in the command line. Respond to the /T switch with either a colon followed by an unsigned six-digit octal number or with a carriage return followed by the global symbol whose value is the transfer address of the load module.

?LINK–W–Undefined globals:

The globals listed were undefined (possibly because $SYSLIB was not present and $SYSLIB modules were referenced or overlays were used). Check for a typing error in the command line. The undefined globals are listed on the terminal and also in the link map when requested. Correct the source program. Verify that all necessary object modules are indicated in the command line or are present in the libraries specified or in $SYSLIB.

B.3 LIBR Error Messages

A list of LIBR error messages follows:

?LIBR–F–Bad GSD in DEV:FILENAME.TYPE

There was an error in the global symbol directory (GSD). The file is probably not a legal object module. Refer to COMMENT #2 at the beginning of this appendix.

?LIBR–F–Bad library for listing or extract

The input file specified for extraction or to produce a directory listing was not a valid object library file. It may be necessary to rebuild the input file. Refer to COMMENT #2 at the beginning of this appendix.

?LIBR–F–Bad option: /a

The librarian did not recognize the given switch; /a represents the unrecognized switch. The librarian restarts and prompts with an asterisk. Check for a typing error in the command line. Verify that the switch is legal for the librarian, and retry the operation.

?LIBR–F–EOF during extract

The end of the input file was reached before the end of the module being extracted. The object module format is probably incorrect. Rebuild the library file. If the error condition persists, reassemble the object module(s) belonging to that file. Also, refer to COMMENT #2 at the beginning of this appendix.

?LIBR–F–File not found DEV:FILENAME.TYPE

One of the input files indicated in the command line was not found. LIBR prints an asterisk; the command may be reentered. Check for a typing error in the command line. Verify that the file name exists as entered in the command line, and retry the operation.

?LIBR–F–Illegal device

The device indicated was not available. Verify that the device is valid for the system in use.

?LIBR–F–Illegal error

An internal error occurred while the librarian was recovering from a previous system or user error. Refer to COMMENT #2 at the beginning of this appendix.

?LIBR–F–Illegal input file DEV:FILENAME.TYPE

A file other than a form library file or a form descriptor file was given as input when a form (/F) library was being created. Make sure that you entered the input file name correctly and that the file is a valid one.

?LIBR–F–Illegal option combination

Switches have been specified that request conflicting functions to be performed. For example, if /E is specified, no other switch may be used. If /M is specified, only continuation switches (/C and //) may follow. Examine the logic of the command line and correct it if necessary. Check for typing errors, and retry the operation.

?LIBR–F–Illegal record type in DEV:FILENAME.TYPE

A formatted binary record had a type not in the range 1–10 (octal). Verify that the correct file names were specified as input; check for a typing error in the command line. Refer to COMMENT #2 at the beginning of this appendix.

?LIBR–F–Insufficient memory

Available memory was used up. The current command is aborted. Increase memory space.

?LIBR–F–Macro name table full, use /M:n

The number of macros to be placed in the macro name table was greater than the number allowed. Increase the size of the macro name table by supplying a value (n) to the switch /M:n. The default is 128 names.

?LIBR–F–No value allowed: /a

The specified switch /a does not take a value; /a represents the switch that you used. The librarian restarts and prompts with an asterisk. Check for typing errors; verify that the correct switch has been specified in the command line, and retry the operation.

?LIBR–F–Output and input filenames the same

The same file name was specified for both input and output files when the command string to build the macro library was specified. Use different file names for the input and output files specified to build a macro library. The default input and output file type is .MAC.

?LIBR–F–Output device full DEV:FILENAME.TYPE

The device was full; LIBR was unable to create or update the indicated library file. Increase storage space, or use another device. Refer to COMMENT #1 at the beginning of this appendix.

?LIBR–F–Read error in DEV:FILENAME.TYPE

An unrecoverable error occurred during the processing of an input file. LIBR prints an asterisk and waits for another command to be entered. A hardware problem may have caused this error.

?LIBR–F–/R or /U given on library file DEV:FILENAME.TYPE

A /R or /U switch incorrectly followed the specified library file in the command string. Use the /R or /U switch only after input file names containing modules for replacement or updating. Correct and reenter the command string.

?LIBR–F–/U given on library file DEV:FILENAME.TYPE

This message occurs if the /U illegally modified a forms library file. Use the /U switch only after input file names containing modules for replacement or updating.

?LIBR–F–Write error

The LIBR program detected an unrecoverable error while processing an output file. This may indicate that there was not enough space left on a device to create a file. Increase storage space.

?LIBR–W–Duplicate form name of FORMNM

Two forms of the same name were specified as input and a /U switch was not given on the second form. The first form encountered was put in the output file. All duplicates are ignored. Use the /U switch to update a form of the same name as a previously specified file.

?LIBR–W–Duplicate module name of AAAAAA

A new module was inserted in a library, but its name is the same as a module that is already in the library. The librarian does not reenter the name in the directory. The old module is not updated or replaced. For the librarian program, insertion is the default operation and no command switch is needed; the switch for update is /U, and the switch for replacement is /R.

?LIBR–W–Illegal character

The symbol name entered contained an illegal character. Retype the command line, using Radix–50 characters only, and retry the operation.

?LIBR–W–Illegal delete of AAAAAA

An attempt was made to delete from the library's directory a module or an entry point that does not exist; AAAAAA represents the module or entry point name. The entry point name or module name is ignored, and processing continues. Check for a typing error in the command line.

?LIBR–W–Illegal extract of AAAAAA

An extraction of the identified global symbol was attempted, but the symbol was not found in the library. Check the command string and the contents of the library file for the correct library file and global symbol specifications.

?LIBR–W–Illegal insert of AAAAAA

An attempt was made to insert into a library a module that contains the same entry point as an existing module; AAAAAA represents the entry point name. The entry point is ignored, but the module is still inserted into the library. No user action is necessary.

?LIBR–W–Illegal replacement of AAAAAA

An attempt was made to replace in the library file a module that does not already exist; AAAAAA represents the module name. The module is ignored and the library is built without it. Review the module names in the library file. Make sure the correct module was specified.

?LIBR–W–Null library

An attempt was made to build a library file containing no directory entries. Verify that the correct file names were specified as input; check for a typing error in the command line. Verify that the input to the library has at least one directory entry.

?LIBR–W–Only continuation allowed

An attempt was made to enter a command string beyond the end of the current line without the use of a continuation character. Enter a /C switch or / / at the end of the current line.

B.4 PAT Error Messages

A list of PAT error messages follows:

?PAT–F–Command line error

There is a syntax error in the PAT command line. Check for typing errors, and reenter the command line.

?PAT–F–Correction file has bad GSD

There was an error in the global symbol directory (GSD). The file is probably not a legal object module. Refer to COMMENT #2 at the beginning of this appendix.

?PAT–F–Correction file has bad RLD

A global symbol named in a relocatable record was not defined in the global symbol definition record. This error condition indicates a bad object file. Refer to COMMENT #2 at the beginning of this appendix.

?PAT–F–Correction file has illegal record

The correction file does not appear to be a proper object file. The standard language processors should produce the required format. Verify that the correction file has the proper format, and retype the command line. Refer to COMMENT #2 at the beginning of this appendix.

?PAT–F–Correction file missing

The command line does not have a correction file specification. PAT requires both an input file and a correction input file in every command. Enter a complete command.

?PAT–F–Correction file missing RLD record

The file is missing an RLD 7 command before the first TXT record. This is the p-sect definition command. PAT cannot process the file. This could simply mean a bad input file. Reassemble the correction file. Refer to COMMENT #2 at the beginning of this appendix.

?PAT–F–Correction file read error

PAT detected an error while reading the correction file. Input hardware can cause this error. Retry the command. Check for off-line devices.

?PAT–F–Illegal error

PAT has detected an internal software error condition. Refer to COMMENT #2 at the beginning of this appendix.

?PAT–F–Incompatible reference to global AAAAAA

The correction file contains a global symbol with improper attributes. Modify the attributes of the global symbol. Choose definition or reference, and choose relocatable or absolute. Reassemble the correction file, and retype the command line.

?PAT–F–Incompatible reference to section AAAAAA

The correction file contains a section name with improper attributes. Modify the section attributes or section type. Make sure the attributes match. Reassemble the correction file, and retype the command line.

?PAT–F–Input file has bad GSD

There was an error in the global symbol directory (GSD). The file is probably not a legal object module. Verify that the input file name is correct; check for a typing error in the command line. Refer to COMMENT #2 at the beginning of this appendix.

?PAT–F–Input file has bad RLD

An error occurred in the language processor because a global symbol named in a relocatable record was not defined in the global symbol definition record. Refer to COMMENT #2 at the beginning of this appendix.

?PAT–F–Input file has illegal record

The format of the input file is not compatible with the object file format PAT requires. The standard language processors should produce the required format. Verify that the input file has the proper format, and retype the command line. Refer to COMMENT #2 at the beginning of this appendix.

?PAT–F–Input file missing

The command line does not have an explicit input file specification. PAT requires both an input file and a correction file in every command. Enter a complete command.

?PAT–F–Input file read error

PAT detected an error while reading the input file. Hardware errors on input can cause this error. Correct the problem, and retry the command.

?PAT–F–Insufficient memory

PAT ran out of memory. Allow the job to use more memory, or apply the patch in sections.

?PAT–F–Only /C allowed

The input module or correction file specification contain an illegal switch. Enter a command line with the appropriate switches.

?PAT–F–Output file full

There was not enough free space on the output volume for the corrected object file. Increase storage space, or use another device. Refer to COMMENT #1 at the beginning of this appendix.

?PAT–F–Output write error

PAT encountered an error while writing the output file. This error occurs when the output device is write-locked or when there is a hardware error. Correct the problem and retry.

?PAT–F–Unable to locate module AAAAAA

The correction file has a module name that does not exist in the input file; AAAAAA represents the name of the nonexistent module. Update the input file to include the missing module, or correct an improper module name in the correction file. Retype the command line.

?PAT–W–Additional input files ignored

The command line specified more than two input files. PAT processed the first as the input module to be corrected and the second as the correction file. PAT ignores all other files. Only one correction file was processed. Merge the corrections into one file and reissue the command.

?PAT–W–Additional output files ignored

The command line has more than one output file specification. PAT cannot create more than one file for each command line and ignores all other output files specified, except the first. Enter a correct command. PAT's output file must be in the "out1" position for the general command line format:

```
out1,out2,out3 = input,correct
```

The command was in fact executed and thus further action may not be needed.

?PAT–W–Correction file checksum error

PAT found a checksum value that was different from the value for the /C correction file switch. Mistyping the /C switch value or specifying an invalid version of the correction file causes this error. Check for typing errors, and check both the checksum value and the correction file name used. Enter a correct command line.

?PAT–W–Correction file checksum is NNNNNN

PAT responds to the /C switch on the correction file with this message; NNNNNN is the octal value of the sum of all binary data composing the

file. This message is for your information. The number printed is the checksum you would use with the /C:n switch if you plan to apply this patch again.

?PAT–W–Input file checksum error

PAT found a checksum value that was different from the value for the /C input file switch. Mistyping the /C switch value or specifying an invalid version of the input file causes this warning. Check for typing errors, and check both the checksum value and the input file name used. Enter a correct command line.

?PAT–W–Input module checksum is NNNNNN

PAT responds to the /C switch on the input module with this message. The octal value NNNNNN is the sum of all binary data in the file. This message is for your information. The number printed is the checksum you would use with the /C:n switch if you apply this patch again.

Glossary

Absolute Address

The binary number that is assigned as the address of a physical memory storage location.

Absolute Section

The portion of a program in which the programmer has specified physical memory locations of data items.

Address

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

Argument

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

Assembler

A program that translates symbolic source code into machine instructions by replacing symbolic operation codes with binary operation codes, and symbolic addresses with absolute or relocatable addresses.

Assembly Language

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

Assembly Listing

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

CCL (Concise Command Language)

A shorthand way to run a RSTS/E system program, a DIGITAL–supplied program such as MACRO, or a user program. The CCL syntax allows you to run a program (MACRO, LINK, LIBR, PAT, for example) without the RUN command and unlike the RUN command allows you to place the entire command string on one line. After the program finishes executing, control returns to your keyboard monitor.

Command

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

Command Language

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

Command String

A line of input to a computer system that generally includes a command, one or more file specifications, and optional switches. Command string may be used interchangeably with the term command line.

Compiler

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

Cross-Reference Listing

A printed listing that identifies all references in a program to each specific symbol in a program and the statements where they are defined or used.

DCL (DIGITAL Command Language)

A set of commands available on many different DIGITAL systems. These perform basic tasks like copying files, printing files, and running programs. On RSTS/E, the DCL command environment is managed by the DCL run-time system, which has a keyboard monitor like RT11.

Device Name

A unique name that identifies each device unit on a system. It usually consists of a two-character device mnemonic followed by an optional device unit number and a colon.

Directives

Mnemonics in an assembly language source program that are recognized by the assembler as commands to control a specific assembly process (as opposed to instructions).

DK: (Public Structure)

See Public Structure and System Disk.

Emulator

Code that allows software written for a specific operating system to be run on a different type of computer system or on a different operating system.

FORTRAN (FORmula TRANslation)

A problem-oriented language designed to permit programmers to express mathematical operations in a form resembling conventional notation. It is used in a variety of applications, including process control, information retrieval, and commercial and scientific data processing.

Global

A value defined in one program module and used in others. Globals are often referred to as entry points in the module in which they are defined and as externals in the other modules that use them.

Global Symbol

A global value or global label.

Instruction

A coded command that tells the computer what to do and where to find the values with which it is to work. A symbolic instruction is a mnemonic chosen to represent the operation being performed. Symbolic instructions must, however, be changed into machine instructions (by the assembler) before they can be executed by the computer.

Job Keyboard Monitor

The keyboard monitor that manages a job. Your job keyboard monitor is the same as the default keyboard monitor unless you change it. This you can do with the SWITCH program. After you change your job keyboard monitor, you remain under its control until you log out or use SWITCH again to change your keyboard monitor.

Keyboard Monitor

The part of a run-time system with which you communicate. When you work in the DCL environment, for example, you type commands that the DCL keyboard monitor receives and then interprets. Each RSTS/E keyboard monitor has an identifying "prompt" that it displays to indicate when it expects command input. Common keyboard monitor prompts on RSTS/E are: dollar sign ($) for DCL, "Ready" for BASIC–PLUS, angle bracket (>) for RSX, and dot (.) for RT11.

Library

A file containing one or more macro definitions or one or more object modules that are routines that can be incorporated into other programs.

Library Module

A module from a library.

Linker

A program that combines many object modules into an executable module. It satisfies global references and combines program sections.

Load Module

A program in a format ready for loading and executing.

Machine Language

The actual language used by the computer when performing operations.

Main Program

The module of a program that contains the instructions at which program execution begins. Normally, the main program exercises primary control over the operations performed and calls subroutines or subprograms to perform specific functions.

Monitor

The master control program that observes, supervises, controls, or verifies the operation of a computer system. The collection of routines that controls the operation of user and system programs, schedules operations, allocates resources, and performs I/O.

Object Module

The primary output of an assembler or compiler, which can be linked with other modules and loaded into memory as an executable program. The object module is composed of the machine language code, relocation information, and the global symbol table specifying entry points and external symbols used within the program. It is also known as a module.

Object Time System (OTS)

The collection of modules that is called by compiled code in order to perform various utility or supervisory operations (for example, FORTRAN Object Time System).

OP–Code (Operation Code)

The part of a machine language instruction that identifies the operation the CPU is to perform.

Operand

The data that an instruction operates upon. An operand is usually identified by an address part of an instruction.

Operating System

The collection of programs, including a monitor or executive and system programs, that organizes a central processor and peripheral devices into a working unit for the development and execution of application programs.

Overlay Segment

A section of code treated as a unit that can overlay code already in memory and be overlaid by other overlay segments when called from the root segment or another resident overlay segment. It is also known as an overlay.

Program

A set of machine instructions or symbolic statements combined to perform some task.

Program Development

The process of writing, entering, translating, and debugging source programs.

Program Section

A named, contiguous unit of code (instructions or data) that is considered an entity and that can be relocated as a unit without destroying the logic of the program.

Public Structure

The set of all disks that are public. When you do not include a device name in your file specification, the system by default accesses one of the disks on the public structure. Each of logicals SY: and DK: represents the name for all disks in the public structure. Thus, if you do not have any public disks other than the system disk, then SY0: and SY: are equivalent. If you have more than the system disk in the public structure, then SY: or DK: refers to the aggregate of all public disks. (See System Disk.)

Relocate

To move a routine from one portion of storage to another and to adjust the necessary address references so that the routine, in its new location, can be executed.

Root Segment

The segment of an overlay structure that, when loaded, remains resident in memory during the execution of a program. It is also known as the root.

Source Code

Text, usually in the form of an ASCII format file, that represents a program. Such a file can be processed by a compiler or assembler.

Source Language

The language in which a source program is written. It is a system of symbols and syntax that is used to describe a procedure that a computer can execute.

Subprogram

A program or a sequence of instructions that can be called to perform the same task (though perhaps on different data) at different points in a program, or even in different programs.

System Disk

The disk that is required by the RSTS/E monitor to get the system started and thereafter to allow the system to run properly under timesharing. The system-wide logical SY0: is assigned to the system disk. (See Public Structure.)

Utility Program

Any general-purpose program included in an operating system to perform common functions. On RSTS/E, a utility program is called a CUSP (Commonly Used System Program).

Index

Page numbers marked in **bold** indicate the main entry for an indexed item. A page number followed by the letter "f" means the entry is in a figure and a "t" means the entry is in a table. Page references that begin with "Gl" mark Glossary entries.

S

HOW TO ORDER ADDITIONAL DOCUMENTATION

In Continental USA and Puerto Rico
call 800–258–1710

In New Hampshire, Alaska or
Hawaii call 603–884–6660

DIRECT MAIL ORDERS (U.S. and Puerto Rico)

Purchase orders should be mailed directly to:

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

In Canada
call 800–267–6146

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: A&SG Business Manager

INTERNATIONAL

When placing orders outside the U.S.A., please send orders to:

DIGITAL EQUIPMENT CORPORATION
A&SG Business Manager
c/o Digital's Local Subsidiary
or Approved Distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

Reader's Comments

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

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

_____

_____

_____

_____

_____

_____

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

_____

_____

_____

_____

_____

_____

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

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

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

—Do Not Tear - Fold Here and Tape —

digital

No Postage Necessary if Mailed in the United States

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

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN: Commercial Engineering Publications MK1-2/ H3
DIGITAL EQUIPMENT CORPORATION
CONTINENTAL BOULEVARD
MERRIMACK N.H. 03054

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line

Printed in U.S.A.