December 1977

This manual describes six utility programs supplied with the FORTRAN IV compiler under the RSTS/E operating system. These programs are required for full utilization and maintenance of the compiler.

RSTS/E FORTRAN IV
Utilities Manual

Order No. AA-2140B-TC

SUPERSESSION/UPDATE INFORMATION:	This document completely supersedes the document of the same name, Order No. DEC-11-LRUMA-A-D, for Version 2 of PDP-11 FORTRAN IV. DEC-11-LRUMA-A-D is still valid for Version 1C of FORTRAN IV.
OPERATING SYSTEM AND VERSION:	RSTS/E V06C
SOFTWARE VERSION:	FORTRAN IV V02

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation • maynard, massachusetts

First Printing, December 1977

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 only be used or copied 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-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

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

CONTENTS

CONTENTS (Cont.)

FIGURES

TABLES

Page

TABLES (Cont.)

PREFACE

This manual describes the RSTS/E FORTRAN IV Utilities, a set of six programs supplied with the FORTRAN IV compiler. These utilities will enable you to use the FORTRAN IV compiler and other system resources more efficiently when running FORTRAN programs. Two of these utilities are needed to maintain the compiler and the other utilities.

0.1 MANUAL OVERVIEW

This manual presents an overview of all the utilities in Chapter 1 (INTRODUCTION). It then describes the utilities for assembling MACRO-11 subprograms (Chapter 2, MACRO), linking object modules (Chapter 3, LINK), debugging assembly language subprograms (Chapter 4, ODT), creating and maintaining libraries (Chapter 5, LIBR), patching the utilities (Chapter 6, PATCH), and making code modifications to object modules (Chapter 7, PAT). The appendices include the ASCII and RADIX-50 character sets, RSTS/E filename extensions, an option summary and an error message summary.

0.2 READER ASSUMPTIONS

This manual assumes that you are familiar with the RSTS/E operating system and the FORTRAN IV language in general, as implemented on the PDP-11. If you are not, you should read the RSTS/E System User's Guide and the PDP-11 FORTRAN Language Reference Manual. Then use this utilities manual in conjunction with the RT-11/RSTS/E FORTRAN IV User's Guide to efficiently write and execute FORTRAN programs under the RSTS/E system.

Additionally you should have had some exposure to assembly language subprogramming to use this utilities manual. Chapter 2, MACRO, which describes usage of the MACRO assembler, contains no MACRO language reference material. Therefore use in conjunction with Chapter 2 the PDP-11 MACRO-11 Language Reference Manual.

NOTE

The MACRO assembler as supplied with the RSTS/E FORTRAN IV compiler is suitable for subprogram processing only. Digital does not support I/O or monitor service requests with MACRO programming under RSTS/E.

0.3 ASSOCIATED DOCUMENTS

RSTS/E System User's Guide

RT-11/RSTS/E FORTRAN IV User's Guide

PDP-11 FORTRAN Language Reference Manual

PDP-11 MACRO-11 Language Reference Manual

RSTS/E Documentation Directory

0.4 DOCUMENTATION CONVENTIONS

The documentation conventions used throughout this manual include the following items:

1. Actual computer output is used in examples wherever possible. Where necessary, computer output is underlined to differentiate it from user responses.

2. Unless the manual indicates otherwise, terminate all commands and command strings with a carriage return. Where necessary, this manual uses the symbol RET to represent a carriage return, LF to represent a line feed, SP for a space, and TAB to represent a tab.

3. Terminal, console terminal, and teleprinter are general terms used throughout this manual to represent any one of the following: LA30 or LA36 DECwriter, VT05 or VT50 Display, LT33 or LT35 Teletype*.

4. To produce several characters in system commands you must type a combination of keys concurrently. For example, hold down the CTRL key and type O at the same time to produce the CTRL/O character. Key combinations such as this one are documented as CTRL/O, CTRL/C, SHIFT/N, etc. Note that you do not type the slash. It is included for documentation purposes only.

5. In descriptions of command syntax, capital letters represent the command name, which you must type as shown. Lower case letters represent a variable, for which you must supply an appropriate value.

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

* Teletype is a registered trademark of the Teletype Corporation.

CHAPTER 1

INTRODUCTION

1.1 RSTS/E FORTRAN IV UTILITY PROGRAMS

The utility programs provided with the RSTS/E FORTRAN IV compiler are:

- Macro Assembler (MACRO)
  MACRO processes assembly language subprograms, producing object modules for later combination with FORTRAN programs using the LINK utility. You can obtain from MACRO formatted listings of your source (input) code as well as cross-reference, symbol table, and table of contents listings.

- Linker (LINK)
  LINK combines FORTRAN object programs (output from the RSTS/E FORTRAN IV compiler) with any necessary library and assembly language subroutines to create a runnable program.

- Online Debugging Technique (ODT)
  ODT is a tool for debugging assembly language subprograms under RSTS/E. It provides the capability of stopping program execution at various points to examine or modify the contents of variables. You combine ODT with the subprogram to be debugged using the LINK utility.

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

- Patch Program (PATCH)
  PATCH is primarily used to incorporate any system patches published by DIGITAL into the utilities. However, you can use PATCH to modify runnable program (.SAV) files.

- Object Module Patch (PAT)
  PAT lets you modify code in a relocatable binary object (.OBJ) module. This means you can change previously assembled code for which the source file is no longer available. PAT can also be used to update library files (as can LIBR) and to patch the compiler and FORTRAN Object Time System (OTS).

1.2 CALLING THE UTILITY PROGRAMS

To call a RSTS/E FORTRAN IV utility program you respond to the monitor's READY prompt by typing a command of the form:

```
RUN $prog    RET
```

where prog represents a utility name. When the utility you called is ready to accept a command string, it prompts you with an asterisk (*).

Note that ODT is not invoked in this way. Chapter 4 explains procedures for calling and using ODT.

1.3 COMMAND STRING SPECIFICATIONS

When a utility is ready to accept a command string, it prompts you with an asterisk. The first command string you can enter in response has the general format:

```
output = input
```

where	output	represents the output filename specifications. Zero to three filenames are allowed describing the output files to be produced.
	input	represents the input filename specifications. Zero to six filenames are allowed describing the input files to be used.

(PATCH requires you to enter this information slightly differently. Complete instructions are provided in Chapter 6.)

Each filename specification has the form:

```
dev:filnam.ext[p,pn]<prot>/fop/option
```

where	dev:	represents an optional 2- to 3-character logical or physical device name. Explicit keyboard specifications cannot be used (i.e., KB: is legal, but KB13: is not). Table 1-3 in the RT-11/RSTS/E FORTRAN IV User's Guide lists acceptable physical device names.
	filnam	is any 1- to 6-character alphanumeric filename.
	.ext	is any 0- to 3-character alphanumeric extension.
	[p,pn]	is any RSTS/E project (p), programmer number (pn) and is used to identify the owner of a file. If the number is specified, then the file being sought must exist under that number. If no [p,pn] is given, the file will first be sought in the current user's

directory. If the file cannot be found in the directory, a search will be made of the system library ([p,pn] = 1,2).

<prot> is a protection code that uses decimal values to restrict access to a file. The degree of restriction is determined by a code or combination of codes as shown in Table 1-1. Protection codes have effect only when given to output files; they are ignored on input files.

/fop represents an optional RSTS/E file specification option. Refer to the RSTS/E System User's Guide or the RSTS/E Programming Manual for more information.

/option represents one or more utility program options whose functions vary according to the utility you are using. Valid options for a particular utility are listed in tabular form in the appropriate chapter. (Do not confuse utility program options with file specification options mentioned above. They are two different items.)

The device name you specify for the first file in a list of input or output files applies to all the files in that input or output list until you supply a different device name. If you do not supply a device name, the system uses the default device DK:. For example:

```
*DT1:FIRST.OBJ,LP:=TASK.1,DK1:TASK.2,TASK.3
```

This command is interpreted as follows:

```
*DT1:FIRST.OBJ,LP:=DK:TASK.1,DK1:TASK.2,DK1:TASK.3
```

File FIRST.OBJ is stored on device DT1:. File TASK.1 is stored on default device DK:. Files TASK.2 and TASK.3 are stored on device DK1:. Notice that file TASK.1 is on device DK:. It is the first file in the input file list and the system uses the default device DK:. Device DT1: applies only to the file on the output side of the command.

Some options for certain utilities are of the form:

```
/o:arg
```

where o represents the option name and arg represents an argument or value. Ranges for the values of option arguments are given in the appropriate chapters. Generally, options and their associated values, if any, should follow the device and filename to which they apply.

If the same option is to be repeated several times with different values, you can abbreviate the command string. For example,

```
/L:MEB/L:TTM/L:CND
```

can be abbreviated as

```
/L:MEB:TTM:CND.
```

The equal sign (=) used to separate the output and input fields is optional. You can use the < sign in place of the = sign. You can omit the separator entirely if there are no output files.

Table 1-1
Protection Codes

Code	Meaning
1	read protect against owner
2	write protect against owner
4	read protect against owner's project number
8	write protect against owner's project number
16	read protect against all others who do not have owner's project number
32	write protect against all others who do not have owner's project number
64	compile, run-only files
128	privileged program

These codes can be used singularly or combined to provide greater degrees of protection. For example, a protection code of <60> is a combination of codes <32>, <16>, <8>, and <4>. This combination of codes allows only the user to read and write the file.

CHAPTER 2

MACRO

MACRO is a two pass assembler used to process subprograms written in MACRO-11, the PDP-11 assembly language. MACRO is provided as a RSTS/E FORTRAN IV utility for the programmer who wishes to combine FORTRAN routines and assembly language routines. Note that the MACRO assembler as supplied with the FORTRAN IV compiler is suitable for subprogram processing only. Digital Equipment Corporation does not support I/O or monitor service requests with MACRO programming under the RSTS/E operating system.

Routine calculations or operations that you use often are prime candidates for coding in MACRO-11. If you do elect to write MACRO-11 subprograms use the programming rules defined in the PDP-11 MACRO-11 Language Reference Manual. Then, using the commands described in this chapter, you can submit your subprogram to MACRO for processing.

The Online Debugging Technique, ODT, can be used to debug MACRO-11 subprograms. ODT is provided as a RSTS/E FORTRAN IV utility and is described in Chapter 4.

To combine MACRO subprograms with a FORTRAN main program you use the LINK utility. From your subprograms, main program, and any other FORTRAN or library routines that are needed, the linker creates a load module which is ready for execution under a RSTS/E system.

LINK is also provided as a RSTS/E FORTRAN IV utility. Chapter 3 describes LINK.

To ensure that your subprograms are effectively called, certain programming considerations must be made. These are described in Section 2.3 of the RT-11/RSTS/E FORTRAN IV User's Guide.

2.1 INPUT AND OUTPUT

Input to the MACRO assembler is up to six ASCII source files containing MACRO-11 statements. These are the files to be processed by MACRO. Source files can be input from any legal RSTS/E file structured device.

During an assembly process MACRO can create the following files:

1. object
2. listing
3. cross-reference

The object file contains the machine language code generated by the

MACRO assembler as a result of processing your source (input) files. The object file is in relocatable binary object format. This file can be output to any legal RSTS/E device except TT: or LP:.

The listing file can contain a source (input) program listing, a symbol table, and a table of contents. When output to a printing device this file provides listings that contain useful reference and debugging information. The listing file can be output to any legal RSTS/E device.

The cross-reference file is a temporary file that MACRO automatically deletes following an assembly process. It is created on the default device DK: unless you specify otherwise. The cross-reference file is a set of tables that, like the listing file, can contain information useful for debugging and reference. You can obtain from MACRO a listing of the cross-reference file.

You determine which of these output files will be created during an assembly process by including specifications for the desired files in the MACRO command string, as described in Section 2.2.1. A set of MACRO options let you control the exact form and content of each output file. The MACRO options are described in Section 2.3.

2.2 CALLING MACRO

To call MACRO, respond to the system's READY message by typing:

```
RUN $MACRO  RET
```

MACRO prints an asterisk (*) when it is ready to accept command string input.

Typing CTRL/C at this point will return control to the monitor, which prompts you with a READY message. If an assembly is in process, you must type two CTRL/Cs to return control to the monitor.

2.2.1 Command String Specification

In response to the * printed by MACRO, you can enter a command line consisting of the output file specifications followed by an equal sign or left angle bracket, and then by the input file specifications. Format this command line as follows:

```
*dev:obj,dev:list,dev:cref/o:arg=dev:sourcei,...,dev:sourcen/o:arg
```

where	dev:	represents the 2- or 3-character code for a physical or logical device name
	obj	represents the file specification for the relocatable binary object (output) module
	list	represents the file specification for the listing file
	cref	represents the file specification for the temporary cross-reference file

sourcei, ...,sourcen	represents the file specifications for the ASCII source (input) files that are to be assembled. A maximum of six source files is allowed.
/o:arg	represents an option and argument as explained in Section 2.3.

In the command string above, dev: must be a legal RSTS/E file-structured device for input files. For output files dev: can be any legal RSTS/E device with one exception; the device for the object file cannot be a terminal or line printer.

The format for an output file specification is as follows:

```
dev:filename.ext[p,pn]<prot>
```

However, the file specification for a listing file can be abbreviated to include only the device code if the listing file is to be output directly to a printing device such as TTn: or LPn:.

The format for an input (source) file specification is:

```
dev:filename.ext[p,pn]
```

Here the protection codes can be omitted since MACRO ignores them on input files.

Object and listing files are output by MACRO only when a file specification for them is included in the command string.

Cross-reference file listings are output by MACRO when an option (/C:arg) is included in the command string. The cross-reference file specification is needed only when you want a device other than the default device DK: to contain the file. Section 2.4 explains procedures for handling the temporary cross-reference file.

When omitting an output file specification from the command string you must include leading commas. Trailing commas need not be included.

For example, the following command produces a listing file and a cross-reference file listing but no object file:

```
*,LP:/C=DK1:DEMO.MAC[240,129]
```

The next command produces an object file but no listing file or cross-reference listings. Input files are on the default device.

```
*DK1:BINF.OBJ[240,129]<40>=SRC1.MAC, SRC2.MAC
```

MACRO assumes certain default values when you do not specify devices and file extensions in the command string. These default values are listed in Table 2-1.

Table 2-1
Default File Specification Values

File	Default Device	Default Filename	Default File Extension
object	DK:	must specify	.OBJ
listing	same as for object file	must specify	.LST
cref	DK:	CREF	.TMP
source1	DK:	must specify	.MAC
source2 . . . sourcen	same as for preceding source file	must specify	.MAC
User MACRO Library	DK: if first file, otherwise same as preceding source file	must specify	.MAC

The CCL (Concise Command Language) option provides an alternative procedure for invoking MACRO. For information on this option, see Chapter 5 of the RT-11/RSTS/E FORTRAN IV User's Guide.

2.3 OPTIONS

Seven MACRO options are provided to give you some degree of control over the assembly process.

Four of the options let you override certain MACRO directives appearing in the source code. These are the listing control options (/L:arg and /N:arg) and the function control options (/E:arg and /D:arg). These options and their possible arguments are described in Section 2.3.2 and Section 2.3.3, respectively.

The CREF Table Generation Option, /C:arg, and its possible arguments allow you to obtain from MACRO detailed cross-reference file listings. This option is described in Section 2.3.4.

Two options (/P:arg and /M) direct MACRO on the handling of certain input files. These options are described in Sections 2.3.5 and 2.3.6, respectively.

Table 2-2 lists the options and describes generally the effect of each.

2.3.1 Command String Specification

With some options you may want to specify more than one argument, or value. Where legal, multiple arguments may be specified for a particular option by separating each argument from the next by a colon. For example:

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

The /M and /P options affect only the particular source file specification to which they are directly appended in the command string. The other options are unaffected by their placement in the command string.

Table 2-2
MACRO Options

Option	Usage
/L:arg	Listing control, overrides source program directive .LIST
/N:arg	Listing control, overrides source program directive .NLIST
/E:arg	Object file function enabling, overrides source program directive .ENABL
/D:arg	Object file function disabling, overrides source program directive .DSABL
/M	Indicates input file is MACRO library file
/C:arg	Control contents of cross-reference listing
/P:arg	Specifies whether input source file is to be assembled during pass 1 or pass 2

2.3.2 Listing Control Options

There are two listing control options, /L:arg (list) and /N:arg (no list). By specifying these options with a set of selected arguments you override at assembly time the arguments of .LIST and .NLIST directives in the source code. In doing so you can control the content and format of assembly listings.

Table 2-3 lists and explains the listing control options.

An assembly listing of a small program is shown in Figure 2-1. This listing shows the more important listing features. The features are labeled with the argument from Table 2-3 that caused their appearance on the listing.

```
DIST-- COMPUTE DISTANCE MACRO V03.01 14-NOV-77 15:34:29 PAGE 1
 SEQ                                  SRC
  1                              .TITLE DIST-- COMPUTE DISTANCE
  2                     ; THIS ROUTINE COMPUTES THE DISTANCE BETWEEN TWO POINTS
  3                     ; ON A PLANE.  IT REQUIRES THE FP11-B FLOATING POINT
  4                     ; PROCESSOR.
  5                     ;
  6                     ; THE FORTRAN CALL IS OF THE FORM:
  7                     ;
  8                     ; A = DIST( X1, Y1, X2, Y2 )
  9                     ;
 10                     ; WHERE: X1, Y1 ARE THE COORDINATES OF THE 1ST POINT
 11              BIN    ;        X2, Y2 ARE THE COORDINATES OF THE 2ND POINT
 12                     ;        A IS THE RETURNED DISTANCE
 13                              .GLOBL  DIST,   SQRT
 14             000000           R0=     %0             ;DEFINE THE REGISTERS
 15             000001           R1=     %1
 16             000002           R2=     %2
 17             000003           R3=     %3
 18             000004           R4=     %4
 19             000005           R5=     %5
 20             000006           SP=     %6
 21             000007           PC=     %7
 22             000000           F0=     %0             ;DEFINE FP11-B REGISTERS
 23   LOC       000001           F1=     %1
 24             000002           F2=     %2
 25             000003           F3=     %3
 26 000000      005725   DIST:   TST     (R5)+          ;SKIP # OF ARGUMENTS
 27 000002      172435           LDF     @(R5)+,F0      ;F0 = X1
 28 000004      172535           LDF     @(R5)+,F1      ;F1 = Y1
 29 000006      173035           SUBF    @(R5)+,F0      ;F0 = X1-X2
 30 000010      173135           SUBF    @(R5)+,F1      ;F1 = Y1-Y2
 31 000012      171000           MULF    F0,F0          ;F0 = (X1-X2)**2
 32 000014      171101           MULF    F1,F1          ;F1 = (Y1-Y2)**2
 33 000016      172001           ADDF    F1,F0          ;F0 = (X1-X2)**2 + (Y1-Y2)**2
 34 000020      174046           STF     F0,-(SP)       ;STORE THE VALUE FOR THE CALL
 35 000022      010600           MOV     SP,R0          ;GET THE ADDRESS OF VALUE
 36 000024      010046           MOV     R0,-(SP)       ;AND BUILD ARG LIST
 37 000026      012746           MOV     #1,-(SP)       ;COUNT OF ARGS (=1)
                000001
 38 000032      010605           MOV     SP,R5          ;R5=> ARG LIST FOR SQRT
 39 000034      004767           JSR     PC,SQRT        ;CALL SQRT
                000000G
 40 000040      062706           ADD     #10,SP         ;POP ARG LIST AND VALUE
                000010
 41 000044      000207           RTS     PC             ;AND RETURN (SQRT LEFT
 42                                                     ;VALUE IN R0,R1)
 43             000001           .END


     DIST-- COMPUTE DISTANCE MACRO V03.01 14-NOV-77 15:34:29 PAGE 1-1
     SYMBOL TABLE

     DIST    000000RG        F1    =%000001        F3    =%000003
SYM  F0    =%000000          F2    =%000002        SQRT  = ****** G

     . ABS.  000000     000
             000046     001
     ERRORS DETECTED:  0

     VIRTUAL MEMORY USED:  283 WORDS  ( 2 PAGES)
     DYNAMIC MEMORY AVAILABLE FOR  70 PAGES
     SY:DIST.OBJ,TT:=DIST.MAC/L:MEB:TTM <---- COPY OF MACRO COMMAND STRING

     ERRORS DETECTED:  0
```

Figure 2-1 Example of an Assembly Listing

The /N option with no argument causes MACRO to list only the symbol table, table of contents and error messages.

The /L option with no arguments causes MACRO to ignore .LIST and .NLIST directives in the source code that have no arguments.

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

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

Table 2-3
Valid Arguments for /L and /N Options

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

2.3.3 Function Control Options

The function control options (/D:arg and /E:arg) allow you to override ENABL and DSABL directives in the source code.

A summary of the arguments which are valid for use with the function control options is provided in Table 2-4.

Table 2-4
Valid Arguments for /E and /D Options

Argument	Default Mode	Enables or Disables
ABS	Disable	Absolute binary output
AMA	Disable	Assembly of all absolute addresses as relative addresses
CDR	Disable	Source columns 73 and greater to be treated as comments

(continued on next page)

Table 2-4 (Cont.)
Valid Arguments for /E and /D Options

Argument	Default Mode	Enables or Disables
CRF	Enable	Cross-reference listing. /D:CRF inhibits CREF output even if /C is specified.
FPT	Disable	Floating point truncation
GBL	Disable	Undefined symbols treated as globals
LC	Disable	Accepts lower case ASCII input
LSB	Disable	Local symbol block
PNC	Enable	Binary output
REG	Enable	Mnemonic definitions of registers

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

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

In this example, you disable the listing of macro expansion binary code (MEB) for some portion of the subprogram and subsequently resume MEB listing. However, if you indicate /L:MEB in the assembly command string, the system ignores both the .NLIST MEB and the .LIST MEB directives. This enables MEB listing throughout the program.

Sections 6.1.1 and 6.2 of the PDP-11 MACRO-11 Language Reference Manual contain more detailed information on the arguments for both the listing control and function control options. The material is presented in the context of arguments for the assembler directives .LIST and .NLIST, ENABLE and DSABLE.

2.3.4 Cross-Reference Table Generation Option

MACRO outputs a cross-reference file (CREF) listing when you include in your command string the /C:arg option. CREF is a temporary file that lists various types of symbols used in your source code and identifies the statements that referenced the symbols. CREF listings can be very useful when debugging a subprogram.

A CREF listing can contain up to six sections. Each section cross-references a different type of symbol. The arguments you specify with the /C option determine which sections are produced. Valid arguments for /C are described in Section 2.3.4.2.

2.3.4.1 **Restrictions** - When /C:arg is specified to MACRO you must also include a listing file specification in the command string. (The cross-reference listing you request from MACRO is generated at the end of the MACRO assembly listing.) The /C:arg option is usually appended to the listing file specification, but it is just as effective anywhere in the command string.

Unless you specify otherwise CREF is created on the default device. If you wish to have CREF.TMP created on some other device (because of space considerations, for example), you must include the CREF file specification (dev:name.TMP) in the input command string. Note that you still must specify the /C:arg option to get a CREF file listing.

Another way to assign an alternative device for the CREF file is by the monitor ASSIGN command. Enter this command before calling MACRO (via RUN $MACRO) as follows:

```
READY
ASSIGN dev:CF
```

If you later include a CREF file specification in a command string after entering the ASSIGN command, the CREF file specification will prevail for that particular assembly only.

Note that regardless of where a CREF file is created it is always deleted by MACRO following the assembly process.

2.3.4.2 **The CREF File Listing** - A complete CREF file listing contains the following six sections:

1. A cross reference of program symbols; that is, labels used in the program and symbols followed by the "=" operator.

2. A cross reference of register equate symbols. These are symbols defined in the source code by the construct:

   symbol=%n

   with $0 \leq n \geq 7$.

   %n represents the eight general registers of the PDP-11 processor. MACRO-11 assigns the following default symbols to %0-%7 (register 0 through register 7):

   R0, R1, R2, R3, R4, R5, SP, PC

3. A cross reference of MACRO symbols; that is, those symbols defined by .MACRO and .MCALL directives.

4. A cross reference of permanent symbols. These include operation mnemonics and assembler directives.

5. A cross reference of program sections. These symbols include the names specified as the operand of a .CSECT or .PSECT directive. The blank .CSECT and the absolute section . ABS. are cross-referenced also.

6. A cross-reference of errors. Certain types of programming and syntax errors in your source code are detected by MACRO and flagged 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 on the assembly listings.

   Error codes and error messages are explained in Section 2.4.

Any or all of the above sections may be included in the cross reference listing by specifying the appropriate arguments with the /C option. These arguments are described in Table 2-5.

Table 2-5
/C Option Arguments

Option Argument	CREF Section
S	User-defined symbols
R	Register symbols
M	MACRO symbolic names
P	Permanent symbols including instructions and directives
C	Program sections
E	Error codes
<no arg>	Equivalent to /C:S:M:E

Figure 2-2 contains a cross-reference listing produced by the following command line:

```
*SY:DIST.OBJ,TT:=DIST.MAC/L:MEB:TTM/C:S:R:P:C
```

An explanation of the listing follows the figure.

```
DIST-- COMPUTE DISTANCE MACRO V03.01 14-NOV-77 15:29:56 PAGE S-1
CROSS REFERENCE TABLE (CREF V01-05 )


DIST      1-13      1-26#
F0        1-22#     1-27*     1-29*     1-31      1-31*     1-33*     1-34
F1        1-23#     1-28*     1-30*     1-32      1-32*     1-33
F2        1-24#
F3        1-25#
SQRT      1-13      1-39


DIST-- COMPUTE DISTANCE MACRO V03.01 14-NOV-77 15:29:56 PAGE R-1
CROSS REFERENCE TABLE (CREF V01-05 )


PC        1-21#     1-39*     1-41*
R0        1-14#     1-35*     1-36
R1        1-15#
R2        1-16#
R3        1-17#
R4        1-18#
R5        1-19#     1-26      1-27      1-28      1-29      1-30      1-38*
SP        1-20#     1-34*     1-35      1-36*     1-37*     1-38      1-40*


DIST-- COMPUTE DISTANCE MACRO V03.01 14-NOV-77 15:29:56 PAGE P-1
CROSS REFERENCE TABLE (CREF V01-05 )


.END      1-43
.GLOBL    1-13
.TITLE    1-1
ADD       1-40
ADDF      1-33
JSR       1-39
LDF       1-27      1-28
MOV       1-35      1-36      1-37      1-38
MULF      1-31      1-32
RTS       1-41
STF       1-34
SUBF      1-29      1-30
TST       1-26


DIST-- COMPUTE DISTANCE MACRO V03.01 14-NOV-77 15:29:56 PAGE C-1
CROSS REFERENCE TABLE (CREF V01-05 )


          0-0
. ABS.    0-0
```

Figure 2-2 Cross-Reference Listing

Cross reference tables you request from MACRO are generated at the end of the MACRO assembly listing. Each table begins on a new page (the tables in Figure 2-2 have been consolidated to save space).

Symbols, program section names, and error codes are listed at the left margin of the page. References to each symbol are listed on the same line across the page from left to right. A reference is of the form p-n, where p is the page on which the symbol appears, and n is the line number within the page.

A number sign (#) next to a reference indicates a symbol definition.

An asterisk (*) next to a reference indicates an operation was made that altered the contents of an addressed location (called a destructive reference).

2.3.5 Assembly Pass Option

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

The specification /P:1 calls for assembly of the file during pass 1 only. Some files consist entirely of code that is completely assembled at the end of pass 1. By specifying /P:1 for these files, you can cause MACRO 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. Situations where the /P:2 option can be meaningfully employed are unusual.

2.3.6 Macro Library File Designation Option

The /M option is meaningful only if appended to a source file specification. It has no arguments, and it designates its associated source file as a macro library.

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

Consider the following command string:

```
*(output file specification)=ALIB.MAC/M,BLIB.MAC/M,XIZ
```

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

2.4 ERROR CODES AND MESSAGES

MACRO can detect errors on two levels; programming level and input-output level.

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

When output to a line printer programming level error codes appear on the left margin of the assembly listing, preceding the source line sequence numbers.

When output to a terminal (via /L:TTM or .LIST TTM) these error codes appear on the assembly listing following a field of six asterisk characters and precede the source line containing the error. For example:

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

Programming level error codes also appear on the cross-reference listing if you specify /C:E in the MACRO command string.

Section 2.4.1 describes programming level errors detected by MACRO and their corresponding error codes.

Input-Output (I/O) level error messages appear when incorrect command strings are specified to MACRO or when problems arise with I/O devices. I/O level error messages are of one of the following formats:

```
?MACRO-F-message
     or
?CREF-F-message
```

These messages are output to the terminal.

Section 2.4.2 describes I/O level error messages produced by MACRO.

2.4.1 Programming Level Error Codes

Error Code	Meaning
A	Addressing or relocation error. This occurs when an instruction operand has an invalid address, or when the definition of a local symbol occurs more than 128 words from the beginning of a local symbol block.
B	Boundary error. The current setting of the location counter would cause the assembly of instruction or word data at an odd memory address. The system increments the location counter by 1 to correct this.
D	Reference to multiple-definition symbol. The program refers to a non-local label that is defined more than once.
E	No END directive. The assembler has reached the end of a source file and found no END directive. The system generates .END and continues.

Error Code	Meaning
I	Illegal character detected. The assembler has encountered in the source file a character that is not included in the language character set. The system replaces each illegal character with a ? on the assembly listing and proceeds as if the illegal character were not present.
L	Link buffer overflow. The assembler has encountered an input line greater than 132 characters. In terminal mode the system ignores additional characters.
M	Multiple definition of a label. The source program is attempting to define a label equivalent in the first six characters to a label defined previously.
N	Decimal point missing from decimal number. A number containing the digit 8 or 9 lacks a decimal point.
O	Op-code error. A directive appears in an inappropriate context.
P	Phase error. The definition or value of a label differs from one pass to another, or a local symbol occurs more than once in a local symbol block.
Q	Questionable syntax. This can have any of several causes, as follows: 1. There are missing arguments. 2. The instruction scan is not complete. 3. A line feed or form feed does not immediately follow a carriage return.
R	Register-type error. The source program attempts an invalid reference to a register.
T	Truncation error. A number generates more than 16 significant bits, or an expression generates more than 8 significant bits while a .BYTE directive is active.
U	Undefined symbol. A symbol not defined elsewhere in the program appears as a factor in an expression. The assembler assigns the undefined symbol a constant zero value.
Z	Incompatible instruction (warning). The instruction is not defined for all PDP-11 hardware configurations.

2.4.2 I/O Level Error Messages

Message	Explanation
?CREF-F-Chain-only CUSP	Programs must chain to CREF in order to use it. Attempts to use RUN $CREF can cause this error. Use a language processor to invoke CREF.

Message	Explanation
?CREF-F-CRF file error	An input error occurred while reading DK:CREF.TMP, the temporary input file passed to CREF. Run the language processor again to create a good CREF input file.
?CREF-F-Device	The language processor chaining to CREF has specified an invalid device. This may be a system error. However, writing a CREF listing to magtape or cassette before manually loading the magtape or cassette handler causes this error. The error also occurs when the input file to CREF, CREF.TMP, is not on a random access device. If the error persists, submit a Software Performance Report with a program listing and a machine readable source program, if possible.
?CREF-F-List file error	An output error occurred while attempting to write the cross-reference table to the listing file. The output volume may not have enough free space remaining for the listing file.
?MACRO-F-Bad option	The specified option was not recognized by the program. Check for a typing error in your command line.
?MACRO-F-Device full	The output volume does not have sufficient room for an output file specified in the command string. Delete unnecessary files or use another device.
?MACRO-F-File not found	An input file specified in the command line does not exist on the specified device, or was protected against the current user. 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 6 input files. Correct the command line and retype.
?MACRO-F-Illegal device	A device specified in the command line does not exist on the system.

Message	Explanation
?MACRO-F-Input-output error on channel N	A hardware error occurred while attempting to read from or write to the device on the channel specified in the message. Channels correspond to files in the command string as follows: Channel — File 0 — .OBJ output file 1 — .LST output file 2 — CREF temporary file 3 — Input (source) file #1 4 — Input (source) file #2 5 — Input (source) file #3 6 — Input (source) file #4 7 — Input (source) file #5 8 — Input (source) file #6
?MACRO-F-Input-output error on MACRO library	MACRO detected a bad record in the MACRO library. For example, this error occurs when the library area is bad. Rebuild the MACRO library.
?MACRO-F-Input-output error on workfile	MACRO failed to read or write to its workfile, WRK.TMP. Check for hard error conditions such as read or write-locked, or offline devices.
?MACRO-F-Insufficient memory	There were too many symbols in the program being assembled.
?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 file.
?MACRO-F-Output device full	There was no room to continue writing the output file.

CHAPTER 3

LINKER (LINK)

The RSTS/E linker (LINK) converts object modules produced by the FORTRAN IV compiler or the RSTS/E MACRO assembler into a format suitable for loading and execution. The linker processes the object modules of the main program and subroutines to:

- relocate each object module and assign absolute addresses;
- link the modules by correlating global symbols that are defined in one module and referenced in another;
- create the initial control block for the linked program;
- create an overlay structure if specified and include the necessary run-time overlay handlers and tables;
- search libraries you specify to locate unresolved globals;
- automatically search a default system library to locate any remaining unresolved globals;
- produce a load map showing the layout of the load module;
- produce a symbol definition file.

The RSTS/E linker requires two passes over the input modules. During the first pass it constructs the global symbol table, including all program section names and symbols in the input modules. After it processes all non-library files, the linker scans the library files to resolve undefined globals. It links only those modules that are required into the root segment (that part of the program that is never overlaid). During the final pass, the linker reads the object modules, performs most of the functions listed above, and produces a load module (which is in memory image format for execution under a RSTS/E system or formatted binary for use with the Absolute Loader).

3.1 CALLING AND USING THE LINKER

To call the RSTS/E linker from the system device, respond to the monitor prompt printed by the keyboard monitor by typing:

```
RUN $LINK RET
```

LINK prints an asterisk at the left margin on the terminal when it is ready to accept a command line. If you enter only a carriage return at this point, the linker prints its current version number.

The CCL (Concise Command Language) option provides an alternative procedure for invoking LINK. For information on this option, see Chapter 5 of the RT-11/RSTS/E FORTRAN IV User's Guide.

Type two CTRL/Cs to halt the linker at any time (or a single CTRL/C to halt the linker when it is waiting for terminal input) and return control to the monitor.

The first command string you enter in response to the linker's prompt has this syntax:

```
dev:bin,dev:map,dev:sym=dev:obj's/option(s)
```

where

dev:bin	represents the file specification to be assigned to the linker's load module output file.
dev:map	represents the file specification of the load map file.
dev:sym	represents the file specification of the symbol definition file.
dev:obj's	represents the file specifications for the one or more object modules (that can be a library file) to be linked.
/option(s)	represents one or more of the options from Table 3-2.

Section 1.2 lists the correct format for a RSTS/E file specification.

In each filespec above, the device should be a random access device, with these exceptions: the output device for the load map file can be any RSTS/E device, as can the output device for an .LDA file if you use the /L option. If you do not specify a device, the linker uses the default device.

If you do not specify an output file, the linker assumes that you do not desire the associated output. For example, if you do not specify the load module and load map (by using a comma in place of each file specification) the linker prints only error messages, if any occur. Ordinarily, though, you would want at least a load module output.

Table 3-1 shows the default values for each specification.

If you make a syntax error or if you specify a non-existent file, the system prints an error message. LINK prints an asterisk and you can then enter a new command string.

Table 3-1
Linker Defaults

	Device	File Name	File Type
Load Module	DK:	must assign	SAV, LDA(/L)
Map Output	Same as load module	must assign	MAP
Symbol Definition Output	DK: or same as previous output device	must assign	STB
Object Module	DK: or same as previous object module	must assign	OBJ

3.2 OPTIONS

Table 3-2 lists the options associated with the linker. You must precede the letter representing each option by the slash character. Options must appear on the line indicated if you continue the input on more than one line, but you can position them anywhere on the line. (Section 3.8 provides a more detailed explanation of each option.)

Table 3-2
Linker Options

Option Name	Command Line	Section	Explanation
/A	first	3.8.1	Alphabetizes the entries in the load map.
/B:n	first	3.8.2	Changes the bottom address of a program to n.
/C	any but last	3.8.3	Continues input specification on another command line (you can use /C also with /O; do not use /C with the // option).
/E:n	first	3.8.4	Extends a particular program section to a specific size in blocks.

(continued on next page)

Table 3-2 (Cont.)
Linker Options

Option Name	Command Line	Section	Explanation
/F	first	3.8.5	Instructs the linker to use the default FORTRAN library, FORLIB.OBJ, to resolve any undefined global references. Note that this option should not be specified in the command line when FORLIB has been incorporated into SYSLIB.
/H:n	first	3.8.6	Specifies the top (highest) address to be used by the relocatable code in the load module.
/I	first	3.8.7	Extracts the object modules which define the global symbols you specify from the library and links them into the load module.
/K:n	first	3.8.8	Inserts the value you specify (the valid range for n is from 1 to 28) into word 56 of block 0 of the image file. This option indicates that the program requires nk words of memory.
/L	first	3.8.9	Produces a formatted binary output file (.LDA format).
/M or /M:n	first	3.8.10	Cause the linker to prompt you for a global symbol that represents the stack address, or sets the stack address to the value n.
/O:n	any, but the first	3.8.11	Indicates that the program is an overlay structure; n specifies the overlay region to which the module is assigned.
/P:n	first	3.8.12	Changes the default amount of space the linker uses for a library routines list.
/S	first	3.8.13	Makes the maximum amount of space in memory available for the linker's symbol table. (Use this option only when a particular link stream causes a symbol table overflow.)

(continued on next page)

Table 3-2 (Cont.)
Linker Options

Option Name	Command Line	Section	Explanation
/T or /T:n	first	3.8.14	Causes the linker to prompt you for a global symbol that represents the transfer address, or sets the transfer address to the value n.
/U:n	first	3.8.15	Rounds up the section you specify so that the size of the root segment is a whole number multiple of the value you supply (n must be a power of 2).
/W	first	3.8.16	Directs the linker to produce a wide load map listing.
/X		3.8.17	Does not output the bitmap if the code is below 400.
/Z:n	first	3.8.18	Sets unused locations in the load module to the value n.
//	first and last	3.8.3	Allows you to specify command string input on additional lines. Do not use this option with /C.

3.3 MEMORY ALLOCATION

The linker allocates the physical memory and address space that the load module requires. The area of memory that the linker allocates for a load module contains the following elements:

- a system communication area
- hardware vectors
- a stack
- a set of named areas called program sections (p-sections).

Section 3.5.2 describes the system communication area.

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

The system communication area, the hardware vectors, and the stack areas are all part of the load module area called the absolute section. The absolute section is often called the ASECT because it is the assembler directive .ASECT that allows information to be stored there. This section appears in the load map with the name . ABS. and is always the first section in the listing. The absolute section (ASECT) normally ends at address 1000 (octal).

A program section is an area of the load module that contains code or data; you can reference it by name. The set of attributes associated with each p-section controls the allocation and placement of the section within the load module.

A p-section is the basic unit of memory for a program. It is composed of the following elements:

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

You create p-sections by using the COMMON statement in FORTRAN, or the .PSECT (or .CSECT) directive in MACRO. You can use the .PSECT (or .CSECT) directive to attach attributes to the section. Note that the attributes that follow the p-section name are not part of the name; only the name itself distinguishes one p-section from another. You should make sure, then, that p-sections of the same name that you want to link together also have the same attribute list. Do this because the linker uses the first appearance of the .PSECT and its attributes throughout the operation. If the linker encounters p-sections with the same name that have different attributes, it prints a warning message.

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

Table 3-3
P-section Attributes

Attribute	Value	Explanation
access-code*	RW	Read/Write - data can be read from, and written into, the p-section.
	RO	Read Only - data can be read from, but cannot be written into, the p-section.
type-code	D	Data - the p-section contains data.
	I	Instruction - the p-section contains either instructions, or data and instructions.
scope-code	GBL	Global - the p-section name is recognized across overlay segment boundaries. The linker allocates storage for the p-section from references outside the defining overlay segment.

* Not used by the linker

(continued on next page)

Table 3-3 (Cont.)
P-section Attributes

Attribute	Value	Explanation
scope-code (cont.)	LCL	Local - the p-section name is recognized only within the defining overlay segment. The linker allocates storage for the p-section from references within the defining overlay segment only.
reloc-code	REL	Relocatable - the base address of the p-section is relocated relative to the virtual base address of the program.
	ABS	Absolute - the base address of the p-section is not relocated. It is always 0.
alloc-code	CON	Concatenate - all references to a given p-section name are concatenated. The total allocation is the sum of the individual allocations.
	OVR	Overlay - all references to a given p-section name overlay each other. The total allocation is the length of the longest individual allocation.

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

The alloc-code determines the starting address and length of memory allocated by modules that reference a common p-section. If the alloc-code indicates that such a p-section is to be overlaid, the linker places the allocations from each module at the same location in memory. It determines the total size from the length of the longest reference to the p-section. The last input module that stores information in a particular location determines which values the linker stores in the indicated locations of the load module. If the alloc-code indicates that a p-section is to be concatenated, the linker places the allocations from the modules one after the other in the load module; it determines the total allocation from the sum of the lengths of the references.

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

The .CSECT directive of MACRO is converted internally by both MACRO and the linker to an equivalent .PSECT with fixed attributes. An unnamed CSECT (blank section) is the same as a blank PSECT with the following attributes: RW, I, LCL, REL, and CON.

A named CSECT is equivalent to a named PSECT with these attributes: RW, I, GBL, REL, and OVR. Table 3-4 shows these sections and their attributes.

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

```
MOV     #PNAME,R0
```

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

The linker determines the memory allocation of p-sections by the order of occurrence of the p-sections in the input modules. The absolute section (. ABS.) always comes first, followed by the blank section of the input file (if one exists) and the named section. If there is more than one named section, the named sections appear in the same order in which they occur in the input files. For example, the FORTRAN compiler arranges the p-sections in the main program module so that the USR can swap over pure code in low memory rather than over data required by the function making the USR call.

Table 3-4
Section Attributes

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

3.4 GLOBAL SYMBOLS

Global symbols provide the link, or communication, between object modules. You create global symbols with the .GLOBL or .ENABL GBL assembler directive (or with double colon, ::, or double equal sign, ==). If the global symbol is defined in an object module (as a label using :: or by direct assignment using ==), other object modules can reference it. If the global symbol is not defined in the object

module, it is an external symbol and is assumed to be defined in some other object module. If a global symbol is used as a label in a routine, it is often called an entry point. That is, it is an entry point to that subroutine.

As the linker reads the object modules it keeps track of all global symbol definitions and references. It then modifies the instructions and data that reference the global symbols. The linker prints undefined globals on the console terminal after pass-1, if you do not request a load map on the terminal. They also appear at the end of the load map.

Table 3-5 shows how the linker resolves global references when it creates the load module.

Table 3-5
Global Reference Resolution

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

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

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

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

3.5 INPUT AND OUTPUT

Linker input and output is in the form of modules; the linker uses one or more input modules to produce a single output (load) module.

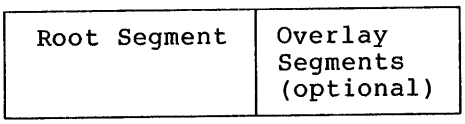
3.5.1 Object Modules

Object files, consisting of one or more object modules, are the input to the linker (the linker ignores files that are not object modules). Object modules are created by the FORTRAN IV compiler or the MACRO-11 assembler. 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 and final pass, the linker reads the object modules, links and relocates the modules and outputs the load module.

3.5.2 Load Module

The primary output of the linker is a load module that you can run under RSTS/E. The load module is output as a save image file (SAV). The linker can produce an absolute load module (LDA) if you need to load the module with the Absolute Loader on a stand-alone PDP-11 system.

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



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 the load module uses; it is located in locations 360 through 377.

The control parameters are located in locations 40 through 50. They contain the following information when the module is loaded:

Address	Information
40	Start address of program
42	Initial setting of SP (stack pointer)
44	Job status word (overlay bit set by LINK)
46	USR swap address (0 implies normal location)
50	Highest memory address in program

The linker stores default values in locations 40, 42, and 50, unless you use options to specify otherwise. The /T option affects location 40, for example, and /M affects location 42. You can also use the .ASECT directive to change the defaults. The overlay bit is located in the job status word. LINK automatically sets this bit if the program is overlaid. Otherwise, the linker initially sets location 44 to 0. Location 46 also contains zero unless you specify another value by using the .ASECT directive.

You can assign initial values to memory locations 0-476 (which include the interrupt vectors and system communication area) by using an .ASECT assembler directive. They appear in block 0 of the load module, but there are restrictions on the use of ASECTs in this region. You should not perform ASECTs of location 54 or of locations 360-377 because the memory usage map is passed in those locations.

You can set with an .ASECT any location that is not restricted, but be careful if you change the system communication area. The program itself must initialize restricted areas, such as the region 360-377. There are no restrictions on ASECTs if the output format is LDA.

3.5.3 Load Map

If you request, the linker produces a load map following the completion of the initial pass. This map, shown in Figure 3-1, diagrams the layout of memory for the load module.

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

The map begins with the version of the linker, followed by the date and time the program was linked. The second line lists the file name of the program, its title (which is determined by the first module name record in the input file), and the first identification record found. The absolute section is always shown first, followed by any non-relocatable symbols. The modules located in the root segment of the load module list next, followed by those modules that were assigned to overlays in order by their region number (see Section 3.6). Any undefined global symbols then list. The map ends with the transfer address (start address) and high limit of relocatable code in both octal bytes and decimal words.

3.5.4 Library Files

The RSTS/E linker can automatically search libraries. Libraries are composed of library files. These are specially formatted files produced by the librarian program (described in Chapter 5) that contain one or more object modules. The object modules provide routines and functions to aid you in meeting specific programming needs. (For example, FORTRAN has a set of modules containing all necessary computational functions--SQRT, SIN, COS, etc.) You can use the librarian to create and update libraries. Then you can easily access routines that you use repeatedly or routines that different programs use. Selected modules from the appropriate library file are linked as needed with your program to produce one load module. Libraries are further described in Section 3.7 and in Chapter 5.

NOTE

Library files that you combine with the PIP /U or /B option are illegal as input to both the linker and the librarian.

```
RT-11 LINK  V05.02      Load Map        Mon 14-Nov-77 15:45:18
AVG   .SAV      Title:  .MAIN.  Ident:  FORY02

Section  Addr    Size    Global  Value   Global  Value   Global  Value

. ABS.   000000  001000    (RW,I,GBL,ABS,OVR)
                         $USRSW  000000  $RF2A1  000000  .VIR    000000
                         $NLCHN  000006  $HRDWR  000010  $WASIZ  000131
                         $LRECL  000210  $TRACE  004737
OTS$I    001000  014642    (RW,I,LCL,REL,CON)
                         $$OTSI  001000  $OTI    001026  $$OTI   001030
                         $$SET   002750  CMI$SS  003244  CMI$SI  003250
                         CMI$SM  003254  CMI$IS  003260  CMI$II  003264
                         CMI$IM  003270  CMI$MS  003274  CMI$MI  003300
                         CMI$MM  003304  NMI$1M  003310  NMI$1I  003322
                         BLE$    003332  BEQ$    003334  BGT$    003342
                         BGE$    003344  BRA$    003346  BNE$    003352
                         BLT$    003354  RET$L   003364  RET$F   003370
                         RET$I   003376  RET$    003400  MOI$SS  003434
                         MOL$SS  003434  MOI$SM  003440  MOI$SA  003444
                         MOI$IS  003450  MOL$IS  003450  REL$    003450
                         MOI$IM  003454  MOI$IA  003460  MOI$MS  003464
                         MOI$MM  003470  MOI$MA  003474  MOI$0S  003500
                         MOI$0M  003504  MOI$0A  003510  MOI$1S  003514
                         MOI$1M  003522  MOI$1A  003530  MOI$RS  003536
                         MOL$RS  003536  MOI$RM  003542  MOI$RP  003546
                         MOI$RA  003550  OCI$    003554  ICI$    003562
                         $ECI    003576  OCO$    003756  ICO$    003764
                         IFW$    004162  $IFW    004166  IFW$$   004230
                         IFR$    004300  $IFR    004304  IFR$$   004342
                         TVL$    004364  $TVL    004364  TVF$    004372
                         $TVF    004372  TVD$    004400  $TVD    004400
                         TVQ$    004406  $TVQ    004406  TVP$    004414
                         $TVP    004414  TVI$    004422  $TVI    004422
                         $CHKER  004556  $IOEXI  004602  $EOL    004630
                         EOL$    004632  CAI$    004746  CAL$    004754
                         ISN$    005004  $ISNTR  005010  LSN$    005024
                         $LSNTR  005030  SAVRG$  005164  THRD$   005342
                         $WAIT   005344  $PUTRE  005406  $PUTBL  005714
                         $GETBL  006124  $EOFIL  006310  $EOF2   006324
                         $STPS   006344  STP$    006352  $STP    006352
                         FOO$    006356  $EXIT   006376  $FCHNL  006522
                         $INITI  006620  $CLOSE  006732  $GETRE  007576
                         $TTYIN  007652  $FIO    010526  $$FIO   010532
                         $ERRTB  011662  $ERRS   011770  $VRINT  015512
                         $DUMPL  015514
OTS$P    015642  000050    (RW,D,GBL,REL,OVR)
SYS$I    015712  000000    (RW,I,LCL,REL,CON)
USER$I   015712  000000    (RW,I,LCL,REL,CON)
$CODE    015712  000154    (RW,I,LCL,REL,CON)
                         $$OTSC  015712
OTS$0    016066  000750    (RW,I,LCL,REL,CON)
                         $$OTSO  016066  $OPEN   016066
SYS$0    017036  000000    (RW,I,LCL,REL,CON)
$DATAP   017036  000172    (RW,D,LCL,REL,CON)
OTS$D    017230  000016    (RW,D,LCL,REL,CON)
OTS$S    017246  000052    (RW,D,LCL,REL,CON)
                         $AOTS   017316
SYS$S    017320  000000    (RW,D,LCL,REL,CON)
$DATA    017320  000004    (RW,D,LCL,REL,CON)
USER$D   017324  000000    (RW,D,LCL,REL,CON)
.$$$$.   017324  000000    (RW,D,GBL,REL,OVR)
         017324  000034    (RW,I,LCL,REL,CON)
                         ISUM    017324

Transfer address = 015712, High limit = 017360 =  3960. words
```

Figure 3-1 Linker Load Map

3.6 USING OVERLAYS

The ability of RSTS/E to handle overlays gives you virtually unlimited space for a FORTRAN program. A program using overlays can be much larger than would normally fit in the available memory space, since portions of the program reside on a backup storage device such as disk or DECtape. To utilize this capability however, 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. However, each region of memory 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 or DECtape. They are brought into memory when called, replacing (or overlaying) the segment previously stored in that region. The root segment, on the other hand, contains those parts of the program which must always be memory resident. Therefore the root is never overlaid.

Figure 3-2 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 DECtape 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 on the DECtape in Figure 3-2, is created by the linker when you specify an overlay structure. The overlay file contains at all times a copy of the root segment and each overlay segment, including those overlay segments currently in memory.

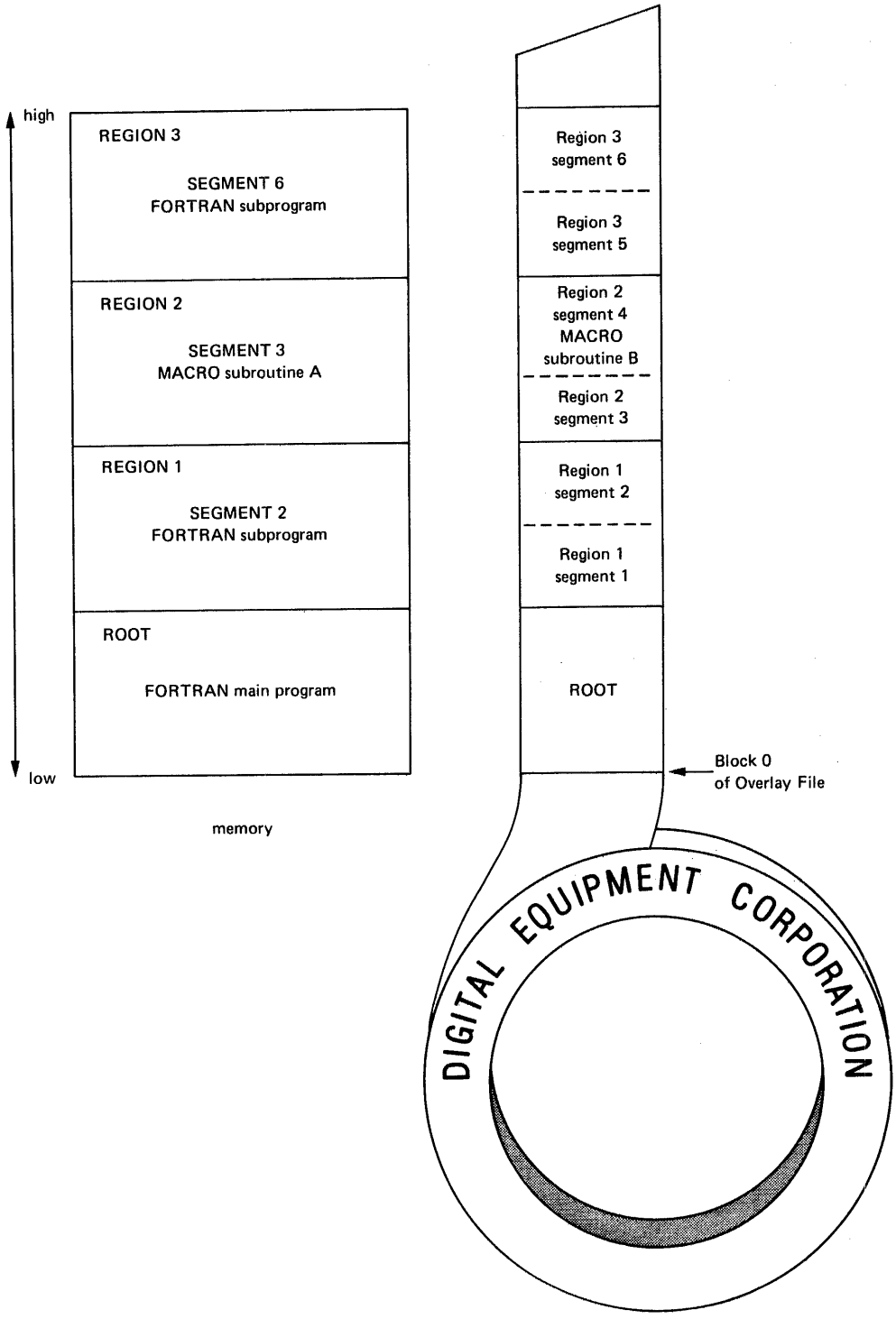


Figure 3-2 An Overlay Structure for a FORTRAN Program

You specify an overlay structure to the linker using the /O option. This option is described fully in Section 3.8.11. Figure 3-3 is an example of using the /O option to specify an overlay structure.

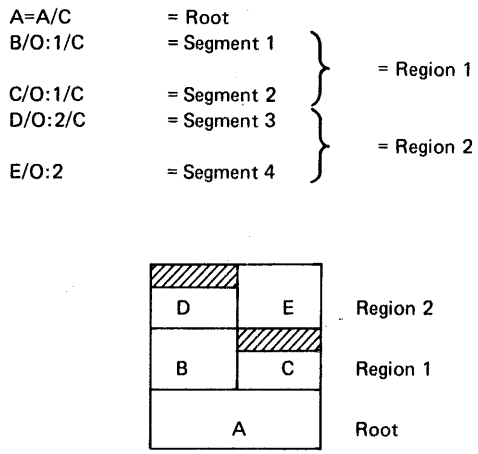


Figure 3-3 Specifying An Overlay Structure With /O

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, inserts in your program the code shown in Figure 3-4, and edits the program to produce the desired overlays at runtime.

```
        .SBTTL  $OVRH   THE RUN-TIME OVERLAY HANDLER
;THE FOLLOWING CODE IS INCLUDED IN THE USER'S PROGRAM BY THE
;LINKER WHENEVER OVERLAYS ARE REQUESTED BY THE USER.
;THE RUN-TIME OVERLAY HANDLER IS CALLED BY A DUMMY
;SUBROUTINE OF THE FOLLOWING FORM:

;       JSR     R5,$OVRH        ;CALL TO COMMON CODE
;       .WORD   <OVERLAY #>     ;# OF DESIRED SEGMENT
;       .WORD   <ENTRY ADDR>    ;ACTUAL CORE ADDR

;ONE DUMMY ROUTINE OF THE ABOVE FORM IS STORED IN THE RESIDENT PORTION
;OF THE USER'S PROGRAM FOR EACH ENTRY POINT TO AN OVERLAY SEGMENT.
;ALL REFERENCES TO THE ENTRY POINT ARE MODIFIED BY THE LINKER TO INSTEAD
;BE REFERENCES TO THE APPROPRIATE DUMMY ROUTINE.  EACH OVERLAY SEGMENT
;IS CALLED INTO CORE AS A UNIT AND MUST BE CONTIGUOUS IN CORE.  AN
;OVERLAY SEGMENT MAY HAVE ANY NUMBER OF ENTRY POINTS, TO THE LIMITS
;OF CORE MEMORY.  ONLY ONE SEGMENT AT A TIME MAY OCCUPY AN OVERLAY REGION.

.ENABL  LSB
        $OVTAB=1000+$OVRHE-$OVRH
$OVRH:  MOV     R0,-(SP)
        MOV     R1,-(SP)
        MOV     R2,-(SP)
1$:
;       MOV     (R5)+,R0        ;PICK UP OVERLAY NUMBER
        BR      3$              ;FIRST CALL ONLY * * *
        MOV     R0,R1
$OVRHA: ADD     #$OVTAB-6,R1    ;CALC TABLE ADDR
        MOV     (R1)+,R2        ;GET CORE ADDR OF OVERLAY REGION
        CMP     R0,@R2          ;IS OVERLAY ALREADY RESIDENT?
        BEQ     2$              ;YES, BRANCH TO IT
        .READW  17,R2,(R1)+,(R1)+ ;READ FROM OVERLAY FILE
        BCS     5$
```

Figure 3-4 The Run-Time Overlay Handler

```
2$:     MOV     (SP)+,R2        ;RESTORE USER'S REGS
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        MOV     @R5,R5          ;GET ENTRY ADDRESS
        RTS     R5              ;ENTER OVERLAY ROUTINE AND
                                ;RESTORE USER'S R5

3$:     MOV     #12500,1$       ;RESTORE SWITCH INSTR (MOV (R5)+,R0)
        MOV     (PC)+,R1        ;START ADDR FOR CLEAR OPERATION
$HROOT: .WORD   0               ;HIGH ADDR OF ROOT SEGMENT
        MOV     (PC)+,R2        ;COUNT
$HOVLY: .WORD   0               ;HIGH LIMIT OF OVERLAYS
4$:     CLR     (R1)+           ;CLEAR ALL OVERLAY REGIONS
        CMP     R1,R2
        BLO     4$
        BR      1$              ;AND RETURN TO CALL IN PROGRESS

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

;OVERLAY SEGMENT TABLE FOLLOWS:
; $OVTAB:       .WORD   <CORE ADDR>,<RELATIVE BLK>,<WORD COUNT>
;THREE WORDS PER ENTRY, ONE ENTRY PER OVERLAY SEGMENT.

;ALSO, THERE IS ONE WORD PREFIXED TO EACH OVERLAY REGION
;THAT IDENTIFIES THE SEGMENT CURRENTLY RESIDENT IN THAT REGION.
```

Figure 3-4 The Run-Time Overlay Handler (Cont.)

There is no magic formula for creating an overlay structure. You do not need a special code or function call. However, some general guidelines must be followed. 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 which need to be memory-resident simultaneously must be assigned to different regions.

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

Figure 3-5 illustrates a sample set of subroutine calls and return paths. In the example, solid lines represent legal subroutine calls and dotted lines represent illegal 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 segments, in any order:

1. itself
2. Segment 4
3. Segment 8
4. The root

Segment 3 cannot call any of the following segments since doing so wipes out its return path to the root:

1. Segments 2 and 1
2. Segment 5
3. Segments 6 and 7

Look at what might happen if one of these illegal calls is made. Assume that segments 3, 4 and 5 all contain MACRO subroutines. Suppose segment 4 calls segment 3 and segment 3 in turn calls segment 5. 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 either loops indefinitely, traps, or random results occur.

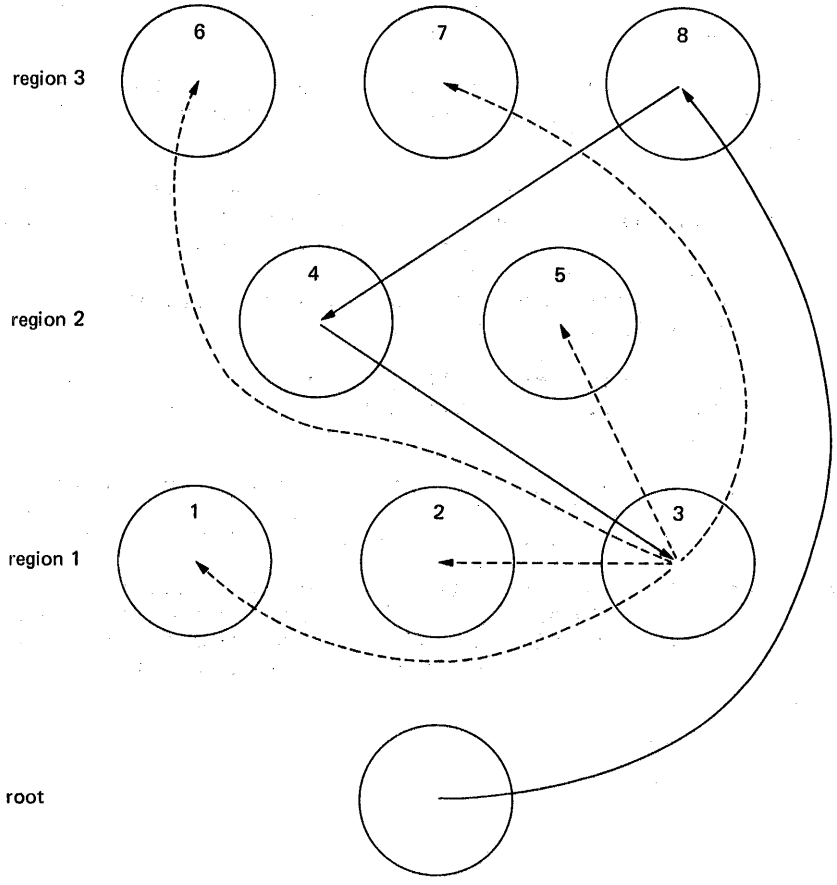


Figure 3-5 Sample Subroutine Calls and Return Paths

The guidelines already mentioned and some additional rules for creating overlay structures are summarizeed below.

1. 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.

2. 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.

3. 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 must be placed in the root segment. This permits common access across the different segments.

4. Object modules that are automatically acquired from a library file cannot be placed in an overlay segment. (This means you cannot specify a library file on the same command line as an overlay segment.) The linker always places library object modules in the root segment. However, you can extract modules from a library file using the librarian utility program as explained in Chapter 5. Extracted object modules can be placed in overlay segments.

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

6. When you make calls to overlay segments, the entire return path to the calling routine must be in memory. Observing the following rules will ensure this:

   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, so 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.

   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.

7. You must make calls or jumps to overlay segments directly to global symbols defined in an instruction p-section (entry points). For example, if ENTER is a global tag in an overlay segment, the first command is valid, but the second is illegal:

```
JMP ENTER
JMP ENTER+6
```

8. You can use globals defined in an instruction p-section (entry points) of an overlay segment only for transfer of control and not for referencing data within an overlay 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. If such a reference occurs, it is indicated on the load map by a "@" following the symbol.

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

10. You cannot use a .CSECT name to pass control to an overlay. It does not load the appropriate segment into memory. For example, JSR PC,OVSEC is illegal if you use OVSEC as a .CSECT name in an overlay. You must use a global symbol to pass control from one segment to the next.

11. In the linker command string, overlay regions are specified in ascending order.

12. Overlay regions are read-only. Unlike USR swapping, an overlay handler does not save the segment it is overlaying. Any tables, variables, or instructions that are modified within a given overlay segment are re-initialized 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.

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

Refer to Chapter 1, Section 1.4.1 of the RT-11/RSTS/E FORTRAN IV User's Guide for additional information.

The ASECT never takes part in overlaying in any way. It is part of the root and is always resident.

The aforementioned sets of rules apply only to communications among the various modules that make up a program. Internally, each module must only observe standard programming rules for the PDP-11 (as described in the PDP-11 Processor Handbook and in the FORTRAN and MACRO-11 Language Reference Manuals).

Note that the condition codes set by your program are not preserved across overlay segment boundaries. You can still use the C-bit for error returns.

The linker provides overlay services by including a small resident overlay handler in the same file with your program to be used at program run-time. The linker inserts this overlay handler plus some tables into your program beginning at the bottom address. The linker moves your program up in memory by an appropriate amount to make room for the overlay handler and tables, if necessary. This scheme is diagrammed in Figure 3-6.

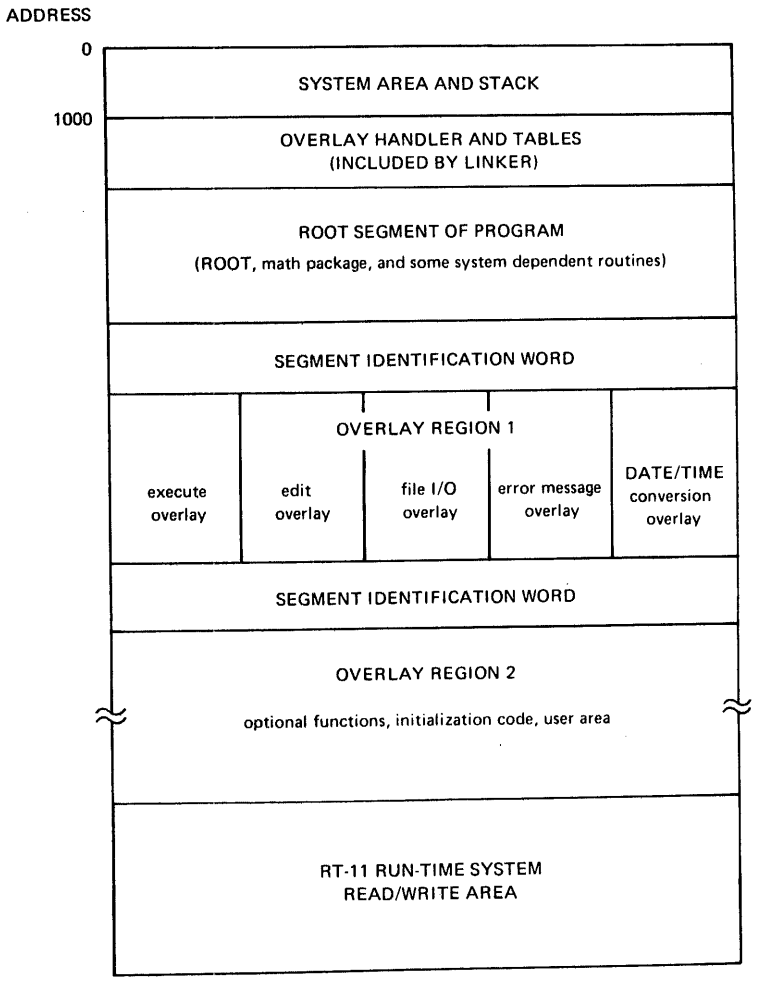


Figure 3-6 Memory With Overlays

3.7 USING LIBRARIES

You specify libraries in a command string in the same fashion as normal modules; you can include them anywhere in the command string, except in overlay lines. If a global symbol is undefined at the time the linker encounters the library in the input stream, and if a module is included in the library that contains that global definition, then the linker pulls that module from the library and links it into the load image. Only the modules needed to resolve references are pulled from the library; unreferenced modules are not linked.

NOTE

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

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

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

The linker selectively relocates and links object modules from specific user libraries that were built by the librarian. Figure 3-7 diagrams this general process. During pass-1 the linker processes the input files in the order in which they appear in the input command line. If the linker encounters a library file during pass-1, it makes note of the library in an internal save status block, and then proceeds to the next file. The linker processes only non-library files during the initial phase of pass-1. In the final phase of pass-1 the linker processes only library files. This is when it resolves the undefined globals that were referenced by the non-library files.

The linker processes library files in the order in which they appear in the input command line. The processing steps are as follows:

1. If there are any undefined globals, the linker proceeds to step 2. Otherwise, it skips to step 5.

2. The linker reads as much of the library directory as the buffer can hold.

3. The linker then searches the entire list of undefined globals for a match with the library directory. It places any globals that match in an internal library module list. If more of the library directory remains to be read, the linker proceeds to step 2.

4. The linker now processes the modules from the library that are associated with the matching undefined globals. If this processing results in new undefined globals that can be resolved by the current library, the linker goes back to step 2.

5. The linker closes the current library and processes the next library file, starting with step 2.

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

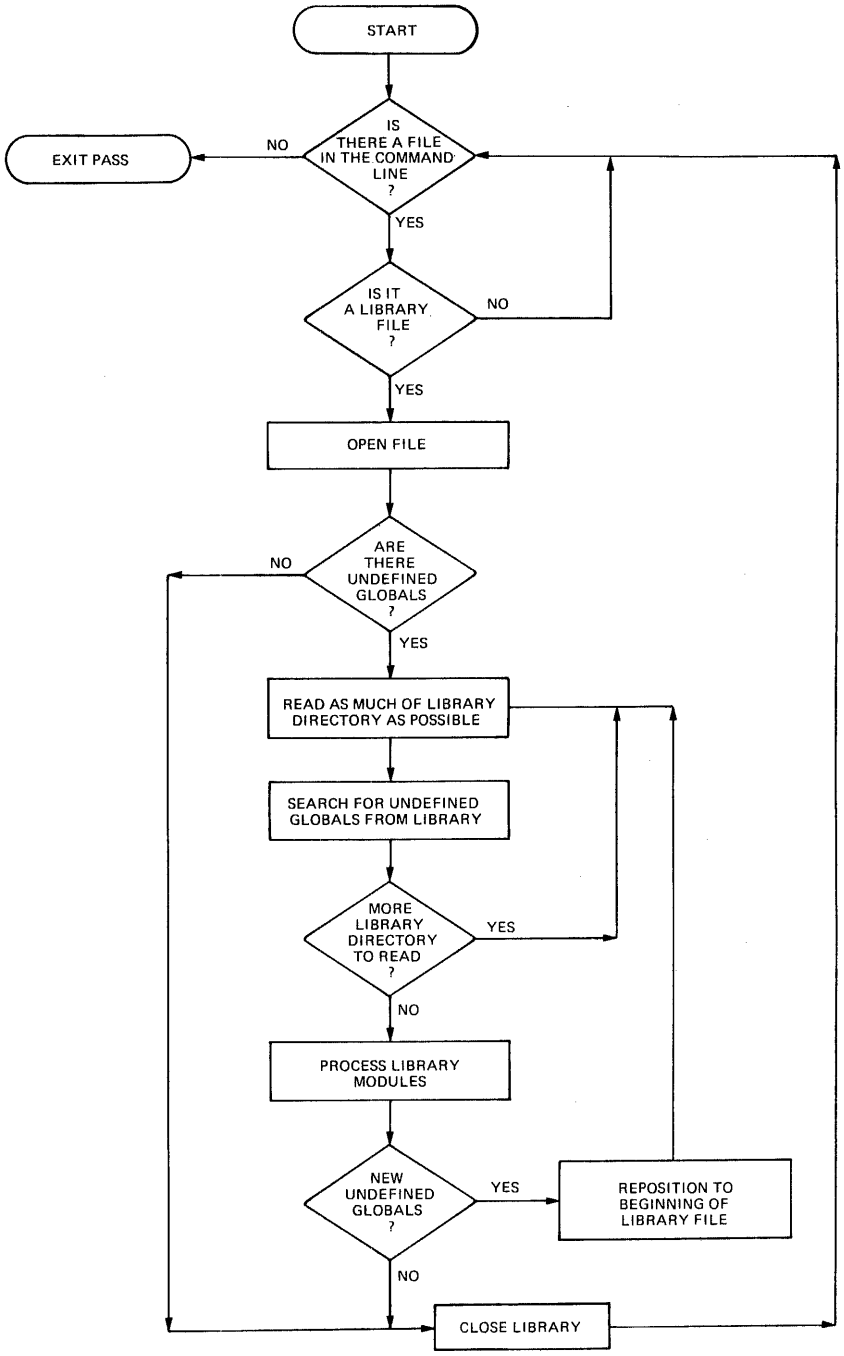


Figure 3-7 Library Searches

Libraries are input to the linker the same way that other input files are. Here is a sample LINK command string:

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

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

3.8 OPTION DESCRIPTION

The options summarized in Table 3-2 are described in detail below.

3.8.1 Alphabetize Option (/A)

Specifying this option causes the linker to alphabetize the entries in the load map.

3.8.2 Bottom Address Option (/B:n)

The /B:n option supplies the lowest address to be used by the relocatable code in the load module. The argument, n, is a 6-digit unsigned octal number that defines the bottom address of the program being linked. If you do not supply a value for n, the linker prints

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

Retype the command, supplying an even octal value.

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

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

NOTE

The bottom value must be an unsigned even octal number. If the value is odd, the ?LINK-F-/B odd value error message prints. Reenter the command string specifying an unsigned even octal number as the argument to the /B option.

The following command causes the input file to be linked starting at location 500 (octal).

```
*OUTPUT,LP:=INPUT/B:500
```

The bottom address determines the amount of stack (SP) space available to the program being linked. The default bottom address of 1000 (octal) provides approximately 80 words of stack.

3.8.3 Continue Option (/C) or (//)

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

The following command indicates that input is to be continued on the next line; the linker prints an asterisk.

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

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

```
*LBR,LBR=LIBRO/W//
*LIBR1/O:1
*LIBR2/O:1
*LIBR3/O:1
*LIBR4/O:1
*LIBR5/O:1
*LBREM/O:1
*//
```

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

3.8.4 Extend Program Section Option (/E:n)

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

```
Extend section?
```

Respond with the name of the program section to be extended. The resultant program section size will be equal to or greater than the value you specify depending upon the space the object code requires. Note that you can extend only one section.

The following example extends section CODE to 100 (octal) blocks.

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

3.8.5 Default FORTRAN Library Option (/F)

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

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

The object module AB.OBJ from the default device and the FORTRAN library SY:FORLIB.OBJ are linked together to form a load module called FILE.SAV.

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

3.8.6 Highest Address Option (/H:n)

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

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

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

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

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

Retype the command, supplying an even octal number.

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

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

Relink the program with a larger value.

The /H option cannot be used with the /B option.

NOTE

Be careful when you use the /H option. Most FORTRAN programs use the free core above the relocatable code as a dynamic working area for I/O buffers, device handlers, symbol tables, etc. The size of this area differs on different memory configurations. Programs linked to a specific high address might not run in a system with less physical memory.

3.8.7 Include Option (/I)

The /I option lets you include in the linking process modules from any library which define the global symbols you specify even when the modules are not needed to resolve undefined globals. This provides a method for forcing modules (which are not called by other modules) to be loaded from the library. When you specify the /I option, the linker prints:

```
Library search?
```

Reply with the list of global symbols to be included in the load module; type a carriage return to enter each symbol in the list. A carriage return alone terminates the list of symbols.

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

```
*SCCA=DK1:SCCA/I
Library search?  $SHORT
Library search?
*^C
```

3.8.8 Memory Size Option (/K:n)

The /K:n option lets you insert a value into word 56 of block 0 of the image file. The argument, n, represents the number of 1K blocks of memory required by the program; n is an integer in the range 1-28.

3.8.9 LDA Format Option (/L)

The /L option produces an output file in LDA format instead of memory image format. The LDA format file can be output to any device including those that are not block-replaceable, such as paper tape or cassette. It is useful for files that are to be loaded with the Absolute Loader. The default file type .LDA is assigned when you use the /L option. You cannot use the /L option with the overlay option (/O). The following example links files IN and IN2 on device DK: and outputs an LDA format file OUT.LDA to the paper tape punch and a load map to the line printer.

```
*PP:OUT,LP:=IN,IN2/L
```

3.8.10 Modify Stack Address Option (/M[:n])

The stack address, location 42, is the address that contains the initial value for the stack pointer. The /M option lets you specify the stack address. The argument, n, is an even, unsigned 6-digit octal number that defines the stack address. After all input lines have been typed, the linker prints the following message if you have not specified a value for n:

```
Stack symbol?
```

In this case, specify the global symbol whose value is the stack address. You may not specify a number. If you specify a nonexistent symbol, an error message prints and the stack address is set to the system default (1000 for SAV files) or to the bottom address if you used /B.

Direct assignment (with .ASECT) of the stack address within the MACRO source code takes precedence over assignment with the /M option. The statements to do this in a MACRO subprogram 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
```

3.8.11 Overlay Option (/O:n)

The /O option segments the load module so that the entire program is not memory resident at one time. This lets you execute programs that are larger than the available memory. The argument n is an unsigned octal number (up to six digits in length) specifying the overlay region to which the module is assigned. The /O option must follow (on the same line) the specification of the object modules to which it applies, and only one overlay region can be specified on a command line. Overlay regions cannot be specified on the first command line; that is reserved for the root segment. You must use /C or // for continuation.

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

```
*OBJA,OBJB,OBJC/O:1/C
*OBJD,OBJE/O:2/C
 .
 .
 .
```

All modules that the linker encounters until the next /O option will be co-resident overlay routines. If you specify, at a later time, the /O option with the same value you used previously, (same overlay region), then the linker opens up the corresponding overlay area for a new group of subroutines. The new group of subroutines will occupy the same locations in memory as the first group, but not at the same time. For example, if subroutines in object modules R and S are to be in memory together, but are never needed at the same time as T, then the following commands to the linker make R and S occupy the same memory as T (but at different times):

```
*MAIN,LP:=ROOT/C
*R,S/O:1/C
*T/O:1
```

The example shown above can also be written as follows:

```
*MAIN,LP:=ROOT/C
*R/O:1/C
*S/C
*T/O:1
```

The following example establishes two overlay regions.

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

You must specify overlays in ascending order (numbers need not be sequential) by region number. For example:

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

The following overlay specification is illegal since the overlay regions are not given in ascending numerical order (an error message prints in each case):

```
*X=LIBRO//
*LIBR1/O:1
*LIBR2/O:0
?LINK-W-/O ignored
*//
```

In the above example, the overlay option immediately preceding the error message is ignored.

3.8.12 Library List Size Option (/P:n)

The /P:n option lets you change the amount of space allocated for the library routine list. Normally, the default value allows enough space for your needs. It reserves space for approximately 256 unique library routines, which is the equivalent of specifying /P:256. (decimal) or /P:400 (octal).

The error message ?LINK-F-Library list overflow, increase size with /P indicates that you need to allocate more space for the library routine list. You must relink the program that makes use of the library routines. Use the /P:n option and supply a value for n that is greater than 256.

You can use the /P:n option to correct for symbol table overflow. Specify a value for n that is less than 256. This reduces the space used for the library routine list and increases the space allocated for the symbol table. If the value you choose is too small, the ?LINK-F-Library list overflow, increase size with /P message prints. In the following command, the amount of space for the library routine list is increased to 300 (decimal).

```
*SCCA=RK1:SCCA/P:300.
```

3.8.13 Symbol Table Option (/S)

The /S option instructs the linker to allow the largest possible memory area for its symbol table at the expense of input and output buffer space, which makes the linking process slower. You should use the /S option only if an attempt to link a program failed because of symbol table overflow. Use of /S in this case often will allow the program to link.

3.8.14 Transfer Address Option (/T[:n])

The transfer address is the address at which a program starts when you initiate execution with a RUN command. The /T option lets you specify the start address of the load module. The argument, n, is a six-digit unsigned octal number that defines the transfer address. If you do not specify n, the following message prints:

```
Transfer symbol?
```

In this case, specify the global symbol whose value is the transfer address of the load module. Terminate your response with a carriage return. You cannot specify a number in answer to this message. If you specify a nonexistent symbol, an error message prints and the transfer address is set to 1 so that the program traps immediately if you attempt to execute it. If the transfer address you specify is odd, the program does not start after loading and control returns to the monitor.

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

```
            .ASECT
            .=40
            .WORD     START2   ;SYMBOL VALUE FOR TRANSFER ADDRESS
            .PSECT             ;RETURN TO PREVIOUS SECTION

START1:     .
            .
            .

START2:     .                  ;SECONDARY STARTING ADDRESS
            .
            .
            .END      START1
```

The following example links the files LIBR0.OBJ and ODT.OBJ together and starts execution at ODT's transfer address, 0.ODT.

```
*LBRODT,LBRODT=LIBR0,ODT/T/W//
*LIBR1/O:1
*LIBR2/O:1
*LIBR3/O:1
*LIBR4/O:1
*LIBR5/O:1
*LBREM/O:1//
Transfer symbol? 0.ODT
*
```

3.8.15 Round Up Option (/U:n)

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

```
Round section?
```

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

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

If the program section you specify cannot be found, the linker prints ?LINK-W-Round section not found. The linking process continues with no rounding.

3.8.16 Map Width Option (/W)

The /W option directs the linker to produce a wide load map listing. If you do not specify the /W option, the listing is wide enough for three GLOBAL VALUE columns (normal for paper with 80 columns). If you use the /W command, the listing is six columns wide, which is ideal for a 132 column page.

3.8.17 Bitmap Inhibit Option (/X)

The /X option instructs the linker not to output the bitmap if code is below 400. The bitmap is stored in locations 360-377 in block 0 of the load module. The linker normally stores the program memory usage bits in these eight words. Each bit represents one 256-word block of memory. This information is used by the RUN command when loading the program; therefore, use care when you use this option.

3.8.18 Zero Option (/Z:n)

The /Z:n option fills unused locations (e.g., those locations specified by .BLKW or .BLKB) in the load module and places a specific value in these locations. The argument, n, represents the value to be placed in the unused locations. This option can be useful in eliminating random results that occur when the program references uninitialized memory by mistake. The system automatically zeroes unused locations. However, if an entire disk block of the load module has no code or data stored on it, the block is not zeroed. Use the /Z:n option only when you want to store a value other than zero in unused locations.

3.9 LINKER PROMPTS

Some of the linker operations prompt for more information, such as the names of specific global symbols or sections. The linker issues the prompt after you have entered all the input specifications, but before the actual linking begins. Table 3-6 shows the sequence in which the prompts occur.

Table 3-6
Linker Prompting Sequence

Prompt	Option
Transfer symbol?	/T
Stack symbol?	/M
Extend section?	/E:n
Round section?	/U:n
Library search?	/I

The library search prompt is last because it can accept more than one symbol and is terminated by a carriage return on a line by itself.

The following example shows how the linker prompts for information when you combine options.

```
RUN $LINK
*LK001=LK001/T/M/E:100/U:20/I
Transfer symbol? O.ODT
Stack symbol? ST3
Extend section? CHAR
Round section? STKSP
Library search? $SHORT
Library search?
*
```

3.10 LINKER ERROR MESSAGES

The following error messages can be output by the linker. Messages appear on the load map if you requested a load map. Otherwise, error messages are output to the terminal.

All messages are of the form:

```
?LINK-n-message
```

where n represents the severity code of the error. Severity codes can be F (Fatal) or W (Warning). Fatal errors cause the current command or statement to be ignored. You must enter another command. A warning message indicates an error condition that may affect execution at a later time. The condition causing the message may require some attention.

Message	Explanation
?LINK-F-/B No value	No argument was specified to the /B option. Reenter the command string specifying an unsigned even octal number as the argument to the /B option.
?LINK-F-/B Odd value	The argument to the /B option was not an unsigned even octal number. Reenter the command string specifying an unsigned even octal number as the argument to /B.
?LINK-F-/H Value too low	The value specified as the high address for linking was actually too small to accommodate the code. Obtain map output without using /H to determine the space required and then retry the operation.
?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 option.
?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 option.
?LINK-F-/U or /Y value not a power of 2	The value specified with /U is not a power of 2. Reenter the command with a value that is a power of 2.
?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-Bad complex relocation in FILNAM	A complex relocation string in the input file was found to be invalid. The message occurs during pass 2 of the linker. Check for a typing error in the command line; verify that the correct filenames were specified as input. Reassemble or recompile to obtain a good object module and retry the operation. If the error persists, verify that the source code is correct.

Message	Explanation
?LINK-F-Bad GSD in FILNAM	There was an error in the global symbol directory (GSD). The file is probably not a legal object module. Verify that the correct filenames were specified as input; check for a typing error in the command line. Reassemble or recompile the source to obtain a good object module and retry the operation.
?LINK-F-Bad RLD in FILNAM	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 filenames were specified as input. Reassemble or recompile and retry the operation. If the error persists, verify that the source code is correct.
?LINK-F-Bad RLD symbol in DEV:FILNAM.TYP	A global symbol named in a relocatable record was not defined in the global symbol definition record. Reassemble the indicated file. If the condition persists, submit a Software Performance Report (SPR).
?LINK-F-Default system library not found SYSLIB.OBJ	The linker did not find SYSLIB.OBJ on the system device when undefined globals existed. Obtain a copy from your backup system volume and relink your program, or correct the source files by removing the undefined globals listed on the terminal.
?LINK-F-File not found DEV:FILNAM.TYP	The input file indicated was not found. Check for a typing error in the command line. Verify that the filename exists as entered in the command line and retry the operation.
?LINK-F-Illegal character	The character specified was not used in proper context. Characters for symbols must be legal Radix-50 characters. Examine the command string for errors in syntax. Correct and retype.

Message	Explanation
?LINK-F-Illegal device	The device/volume indicated was not available. Verify that the device is valid for the system in use.
?LINK-F-Illegal error	An internal error occurred while the linker was in the process of recovering from a previous system or user error. Retry the operations that produced this error; if it recurs, report the error to DIGITAL using an SPR (Software Performance Report); include a program listing and a machine-readable source program, if possible.
?LINK-F-Illegal record type in DEV:FILNAM.TYP	A formatted binary record had a type not in the range 1-10 (octal). Verify that the correct filenames were specified as input; check for a typing error in the command line. Reassemble or recompile and retry the operation.
?LINK-F-Insufficient memory	There was not enough memory to accommodate the command, the symbol table or the resultant load module.
?LINK-F-Map device full	There was no room in the directory for the filename or there was no room on the output device for the map file.
?LINK-F-Old library format in DEV:FILNAM.TYP	The indicated library file is formatted from an old LIBR version. Rebuild the library file using the current librarian.
?LINK-F-Read error in DEV:FILNAM.TYP	A hardware error occurred while reading the indicated input file. Check for read-locked or off-line devices.
?LINK-F-SAV device full	There was no room in the directory for the filename or there was no room on the output device for the image file.

Message	Explanation
?LINK-F-SAV read error	A hardware error occurred while reading the image file (SAV, LDA). Check for read-locked or off-line devices.
?LINK-F-SAV write error	A hardware error occurred while writing the image file (LDA). Check for write-locked or off-line devices.
?LINK-F-STB device full	There was no room in the directory for the filename or there was no room on the output device for the symbol table (STB) file.
?LINK-F-STB not allowed with /S and a MAP	Production of STB and MAP in the same linking operation is prohibited in order to maximize space in the symbol table with /S. Produce STB and MAP in separate linking operations.
?LINK-F-STB write error	A hardware error occurred while writing the symbol table (STB) file. Check for write-locked or off-line devices.
?LINK-F-Storing text beyond high limit	An input object module has 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 and/or recompile the program.
?LINK-F-Symbol table overflow	Too many global symbols were used in the program.
?LINK-W-/O Ignored	Overlays were specified in the wrong order. Check for a typing error in the command line. The overlay option is ignored. Consult the overlay restrictions in this chapter.
?LINK-W-Additive reference of NNNNNN at segment # MMMMMM	A call or a branch 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.

Message	Explanation
?LINK-W-Bad option: /a	The linker did not recognize the option (/a) specified in the command line, or an illegal combination of options was used. If the bad option occurred in the first command line, control returns to LINK; enter another command. If the bad option occurred on a subsequent command line, the option is ignored and processing continues. In a continued command line, only /O, /C, and // are legal options. Reexamine the command line and check for a typing error.
?LINK-W-Bad overlay at segment # NNNNNN	An overlay tried to store text outside its region; NNNNNN represents the segment number. Check for an .ASECT in the overlay.
?LINK-W-Byte relocation error at NNNNNN	The linker attempted to relocate and link byte quantities, but failed. NNNNNN represents the address at which the error occurred. Failure is defined as the high byte of the relocated value (or the linked value) not being all zeroes. The relocated value is truncated to 8 bits and the linker continues processing. Correct the source program so that there are no relocated byte quantities, reassemble, and relink.
?LINK-W-Complex relocation divide by 0 in DEV.FILNAM.TYP	A divide by 0 was attempted in a complex relocation string in the file indicated. A result of 0 is returned and linking continues.
?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. The source program should be checked to use the desired section attributes for that program section.

Message	Explanation
?LINK-W-Extend section not found	The extend section name given with /E was not found in the modules that were linked; or the extend section does not exist in the root segment. The linker continues after the warning, without extending the section. Check the response to the "Extend section?" prompt, and use the correct section name the next time you link.
?LINK-W-Map write error	A hardware error occurred while writing the map output file. 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.
?LINK-W-Round section not found AAAAAA	The round program section was not found in the symbol table to match the symbol entered (following use of the /U option). Linking continues with no round-up action.
?LINK-W-Stack address undefined or in overlay	The stack address specified by the /M option was either undefined or in an overlay. For SAV files, 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. The response to the /T option must be either a colon followed by an unsigned 6-digit octal number, or a carriage return followed by the global symbol whose value is the transfer address of the load module.

Message	Explanation
?LINK-W-Undefined globals:	The globals listed were undefined. 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 present in the libraries specified.

CHAPTER 4

ON-LINE DEBUGGING TECHNIQUE (ODT)

RSTS/E on-line debugging technique (ODT) is a program supplied with the FORTRAN system that aids in debugging assembly language subprograms and FORTRAN in line code. From your terminal you direct the execution of your program with ODT. ODT can:

- print the contents of any location for examination or alteration;
- run all or any portion of an object program using the breakpoint feature;
- search the object program for specific bit patterns;
- search the object program for words that reference a specific word;
- calculate offsets for relative addresses;
- fill a single word, block of words, byte or block of bytes with a designated value.

Make sure you have an assembly listing and a link map available for the subprogram you want to debug with ODT. You can make minor corrections to the program on line during the debugging session, and you can then execute the program under the control of ODT to verify the corrections. If you need to make major changes, such as adding a missing section of code, note them on the assembly listing and incorporate them in a new assembly.

4.1 CALLING ODT

ODT is supplied as a relocatable object module. It is "called" when you link it with the subprogram to be debugged. You can link ODT with your program (using the linker) for an absolute area in memory and load it with your program. When you link ODT with your program, it is a good idea to link ODT low in memory relative to the program. If you link ODT high in memory, you must be sure that the buffer space for your program is contained within program bounds. Otherwise, if your program uses dynamic buffering, program execution may destroy ODT in memory.

To link ODT correctly with your object module, you must use the /T option in the LINK command string. When LINK prompts you for a transfer address, type O.ODT. This is the global symbol representing the normal entry address for O.ODT. The system uses as an absolute address the address of the entry point O.ODT shown in the linker load map.

NOTE

If you link ODT with an overlay structured file, ODT should reside in the root segment so that it will always be in memory. A breakpoint inserted in an overlay will be destroyed if it is overlaid during program execution.

The following example shows how to link and load ODT.

```
RUN $LINK
*MYPROG,LP:=MYPROG,ODT,MACSUB/T
Transfer address? 0.ODT
```

If ODT is awaiting a command, a CTRL/C from the keyboard will call the keyboard monitor. The monitor responds with a READY message on the terminal and awaits a command. If you type CTRL/U during a search printout, the search terminates and ODT prints an asterisk.

4.2 RELOCATION

When the assembler produces a relocatable object module, the base address of the module is assumed to be location 000000. The addresses of all program locations as shown in the assembly listing are relative to this base address. After you link the module, many of the values and all of the addresses in the program will be incremented by a constant whose value is the actual absolute base address of the module after it has been relocated. This constant is called the relocation bias for the module. Since a linked program may contain several relocated modules, each with its own relocation bias, and since, in the process of debugging, these biases will have to be subtracted from absolute addresses continually in order to relate relocated code to assembly listings, ODT provides automatic relocation.

The basis of automatic relocation is the eight relocation registers, numbered 0 through 7. You may set them to the values of the relocation biases at different times during debugging. Obtain relocation biases by consulting the link map. Once you set a relocation register, ODT uses it to relate relative addresses to absolute addresses. For more information on the exact nature of the relocation process, consult Chapter 3, the RSTS/E linker.

ODT evaluates a relocatable expression as a 16-bit (6-digit octal) number. You may type an expression in any one of the three forms presented in Table 4-1. In this table, the symbol n stands for an integer in the range 0 to 7 inclusive, and the symbol k stands for an octal number up to six digits long, with a maximum value of 177777. If you type more than six digits, ODT takes the last six digits typed, truncated to the low-order 16 bits. k may be preceded by a minus sign, in which case its value is the two's complement of the number typed. For example:

k (number typed)	Values
1	000001
-1	177777
400	000400
-177730	000050
1234567	034567

Table 4-1
Forms of Relocatable Expressions (r)

Form	r	Value of r
A)	k	The value of k
B)	n,k	The value of k plus the contents of relocation register n. (If the n part of this expression is greater than 7, ODT uses only the last octal digit of n.)
C)	C or C,k or n,C or C,C	Whenever you type the letter C, ODT replaces C with the contents of a special register called the constant register. (This value has the same role as the k or n that it replaces. The constant register is designated by the symbol $C and may be set to any value, as indicated below.)

Section 4.3.13 describes the relocation register commands in greater detail.

4.3 COMMANDS AND FUNCTIONS

When ODT starts (as explained in Section 4.1) it indicates readiness to accept commands by printing an asterisk on the left margin of the terminal page. You can issue most of the ODT commands in response to the asterisk. You can examine a word and change it; you can run the object program in its entirety or in segments; you can search memory for specific words or references to them. The discussion below explains these features. In the following examples, characters printed by ODT are underlined.

4.3.1 Printout Formats

Normally, when ODT prints addresses it attempts to print them in relative form (Form B in Table 4-1). ODT looks for the relocation register whose value is closest to, but less than or equal to the address to be printed. It then represents the address relative to the contents of the relocation register. However, if no relocation register fits the requirement, the address prints in absolute form. Since the relocation registers are initialized to -1 (the highest number), the addresses initially print in absolute form. If you change the contents of any relocation register, it may then, depending on the command, qualify for relative form.

For example, suppose relocation registers 1 and 2 contain 1000 and 1004 respectively, and all other relocation registers contain numbers much higher. In this case, the following sequence might occur (the slash command causes the contents of the location to be printed; the line feed command (LF) accesses the next sequential location):

```
*1000;1R                    sets relocation register 1 to 1000
*1,4;2R                     sets relocation register 2 to 1004
*774/000000 LF              opens location 774
000776 /007665 LF           opens location 776
1,000000 /000000 LF         opens absolute location 1000
1,000002 /000000 LF         opens absolute location 1002
2,000000 /000000            opens absolute location 1004
```

The printout format is controlled by the format register, $F. Normally this register contains 0, in which case ODT prints addresses relatively whenever possible. You can open $F and change its contents to a non-zero value, however. In that case all addresses will print in absolute form (see Section 4.3.4, Accessing Internal Registers).

4.3.2 Opening, Changing, and Closing Locations

An open location is one whose contents ODT prints for examination, making those contents available for change. In a closed location, the contents are no longer available for change. Several commands are used for opening and closing locations.

Any command that opens a location when another location is already open causes the currently open location to be closed. You may change the contents of an open location by typing the new contents followed by a single character command which requires no argument (i.e., LF, ^, RET, _, @, >, <).

4.3.2.1 **The Slash (/)** - One way to open a location is to type its address followed by a slash. For example:

```
*1000/012746
```

This command opens location 1000 for examination and makes it ready to be changed.

If you do not want to change the contents of an open location, press the RETURN key to close the location. ODT prints an asterisk and waits for another command. However, to change the word, simply type the new contents before giving a command to close the location. For example:

```
*1000/012746 012345 RET
*
```

This command inserts the new value, 012345, in location 1000 and closes the location. ODT prints another asterisk indicating its readiness to accept another command.

Used alone, the slash reopens the last location opened. For example:

```
*1000/012345 2340 RET
*/002340
```

This command opens location 1000, changes its contents to 002340, and then closes the location. ODT prints an asterisk indicating its readiness to accept another command. The / character reopens the last location opened and verifies its value.

Note again that opening a location while another is open automatically closes the currently open location before opening the new location.

Also note that if you specify an odd numbered address with a slash, ODT opens the location as a byte, and subsequently behaves as if you had typed a backslash (see the following paragraph).

4.3.2.2 **The Backslash (\)** - In addition to operating on words, ODT operates on bytes. Typing the address of the byte followed by a backslash character opens the byte. (On the LT33 or LT35 terminal type \ by pressing the SHIFT key while typing the L key.) This causes ODT to print the byte value at the specified address, to interpret the value as ASCII code, and to print the corresponding character, if possible, on the terminal. (ODT prints a ? when it cannot interpret the ASCII value as a printable character.)

```
*1001\101 =A
```

A backslash typed alone reopens the last open byte. If a word was previously open, the backslash reopens its even byte:

```
*1002/000004 \004 =?
```

4.3.2.3 **The LINE FEED Key (LF)** - If you type the LINE FEED key when a location is open, ODT closes the open location and opens the next sequential location:

```
*1000/002340 LF
001002 /012740
```

In this example, the LINE FEED key caused ODT to print the address of the next location along with its contents, and to wait for further instructions. After the above operation, location 1000 is closed and 1002 is open. You may modify the open location by typing the new contents.

If a byte location was open, typing a line feed opens the next byte location.

4.3.2.4 **The Circumflex or Up-Arrow (^)** - If you type the circumflex (or up-arrow) when a location is open (circumflex is produced on an LT33 or LT35 by typing SHIFT/N) ODT closes the open location and opens the previous location. To continue from the example above:

```
*001002/012740 ^
001000 /002340
```

This command closes location 1002 and opens location 1000. You may modify the open location by typing the new contents.

If the opened location was a byte, then the circumflex opens the previous byte.

4.3.2.5 **The Underline or Back-Arrow (_)** - If you type the underline, or back-arrow (by using SHIFT/O on an LT33 or LT35 terminal) to an open word, ODT interprets the contents of the currently open word as an address indexed by the program counter (PC) and opens the addressed location:

```
*1006/000006
001016 /000405
```

Notice in this example that the open location, 1006, was indexed by the PC as if it were the operand of an instruction with addressing mode 67 (PC relative mode).

You can make a modification to the opened location before you type a line feed, circumflex, or underline. Also, the new contents of the location will be used for address calculations using the underline command. For example:

```
*100/000222 4  LF    modifies to 4 and opens next location
000102 /000111 6^    modifies to 6 and opens previous location
000100 /000004 200_  changes to 200 and opens location indexed
000302 /123456       by PC
```

4.3.2.6 **Open the Addressed Location (@)** - You can use the at (@) symbol (SHIFT/P on the LT33 or LT35 terminal) to optionally modify a location, close it, and then use its contents as the address of the location to open next. For example:

```
*1006/001044 @          open location 1044 next
001044 /000500

*1006/001044 2100@      modifies to 2100 and opens location
002100 /000167          2100
```

4.3.2.7 **Relative Branch Offset (>)** - The right-angle bracket, >, optionally modifies a location, closes it, and then uses its low-order byte as a relative branch offset to the next word to be opened. For example:

```
*1032/000407  301>  modifies to 301 and interprets as a
000636 /000010      relative branch
```

Note that 301 is a negative offset (-77). ODT doubles the offset before it adds it to the PC; therefore, 1034+(-176)=636.

4.3.2.8 **Return to Previous Sequence (<)** - The left-angle bracket, <, lets you optionally modify a location, close it, and then open the next location of the previous sequence that was interrupted by an underline, @, or right-angle bracket command. Note that underline, @, or right-angle bracket causes a sequence change to the open word. If a sequence change has not occurred, the left-angle bracket simply opens the next location as a LINE FEED does. This command operates on both words and bytes.

```
*1032/000407  301>       > causes a sequence change
000636 /000010 <         return to original sequence
001034 /001040 @         @ causes a sequence change
001040 /000405 \005 = <  < now operates on byte
001035 \002 =?  <        < acts like LF
001036 \004 =?
```

4.3.3 Accessing General Registers 0-7

Open the program's general registers 0-7 with a command in the following format:

```
*$n/
```

The symbol n is an integer in the range 0-7 that represents the desired register. When you open these registers, you can examine them or change their contents by typing in new data as with any addressable location. For example:

```
*$0/000033 RET      examines register 0 then closes it
*

*$4/000474 464 RET  opens register 4, changes its contents
*                    to 000464, then closes the register
```

The example above can be verified by typing a slash in response to ODT's asterisk:

```
*/000464
```

You may use the LINE FEED, circumflex, underline or @ command when a register is open.

4.3.4 Accessing Internal Registers

The program's status register contains the condition codes of the most recent operational results. Open it by typing $S. For example:

```
*$S/000311
```

$S represents the address of the status register. In response to $S in the example above, ODT printed the 16-bit word, of which only the low-order eight bits are meaningful. Bits 0-3 indicate whether a carry, overflow, zero, or negative (in that order) has resulted.

You can also use the $ to open certain other internal locations listed in Table 4-2.

Table 4-2
Internal Registers

Register	Section	Function
$B	4.3.6	Location of the first word of the breakpoint table
$M	4.3.9	Mask location for specifying which bits are to be examined during a bit pattern search
$S	4.3.4	Location containing the condition codes (bits 0-3)
$C	4.3.10	Location of the constant register
$R	4.3.13	Location of relocation register 0, the base of the relocation register table
$F	4.3.1	Location of format register

4.3.5 Radix-50 Mode (X)

Many PDP-11 system programs employ the Radix-50 mode of packing certain ASCII characters three to a word. You can use Radix-50 mode by specifying the MACRO .RAD50 directive. ODT provides a method for examining and changing memory words packed in this way with the X command.

When you open a word and type the X command, ODT converts the contents of the opened word to its 3-character Radix-50 equivalent and prints these characters on the terminal. You can then type one of the responses from Table 4-3:

Table 4-3
Radix-50 Terminators

Response	Effect
RETURN key (RET)	Closes the currently open location
LINE FEED key (LF)	Closes the currently open location and opens the next one in sequence
Circumflex (^)	Closes the currently open location and opens the previous one in sequence
Any three characters whose octal code is 040 (space) or greater	Converts the three characters into packed Radix-50 format. Legal Radix-50 characters for this response are: . $ Space 0 through 9 A through Z

If you type any other characters, the resulting binary number is unspecified (that is, no error message prints and the result is unpredictable). You must type exactly three characters before ODT resumes its normal mode of operation. After you type the third character, the resulting binary number is available to be stored in the opened location. Do this by closing the location in any one of the ways listed in Table 4-3. For example:

```
*1000/042431 X=KBI CBA RET
*1000/011421 X=CBA
```

NOTE

After ODT converts the three characters to binary, the binary number can be interpreted in one of many different ways, depending on the command that follows. For example:

```
*1234/063337 X=PRO XIT/013704
```

Since the Radix-50 equivalent of XIT is 113574, the final slash in the example will cause ODT to open location 113574 if it is a legal address.

4.3.6 Breakpoints

The breakpoint feature helps you monitor the progress of program execution. You can set a breakpoint at any instruction that is not referenced by the program for data. When a breakpoint is set, ODT replaces the contents of the breakpoint location with a BPT trap instruction so that program execution will be suspended when a breakpoint is encountered. Then the original contents of the breakpoint location are restored, and ODT regains control.

With ODT you can set up to eight breakpoints, numbered 0 through 7, at any one time. Set a breakpoint by typing the address of the desired location of the breakpoint followed by ;B. Thus, r;B sets the next available breakpoint at location r. (If all 8 breakpoints have been set, ODT ignores the r;B command.) You can set or change specific breakpoints with the r;nB command, where n is the number of the breakpoint. For example:

```
*1020;B   sets breakpoint 0
*1030;B   sets breakpoint 1
*1040;B   sets breakpoint 2
*1032;1B  resets breakpoint 1
*
```

The ;B command removes all breakpoints. Use the ;nB command to remove only one of the breakpoints, where n is the number of the breakpoint. For example:

```
*;2B      (removes the third breakpoint)
*
```

ODT keeps a table of breakpoints; you can access that table. The $B/ command opens the location containing the address of breakpoint 0. The next seven locations contain the addresses of the other breakpoints in order. You can sequentially open them by using the LINE FEED key. For example:

```
*$B/001020 LF
001136/001032 LF
001140/007070 LF
001142/007070 LF
001144/007070 LF
001146/001046 LF
001150/001066 LF
001152/007070
```

In this example, breakpoint 0 is set to 1020, breakpoint 5 is set to 1046, and breakpoint 6 is set to 1066. The other breakpoints are not set.

4.3.7 Running the Program (r;G and r;P)

ODT controls program execution. There are two commands for running the program: r;G and r;P. The r;G command starts execution (go) and r;P continues (proceed) execution after halting at a breakpoint. For example:

```
*1000;G
```

This command starts execution at location 1000. The program runs until it encounters a breakpoint or until it completes. If it gets caught in an infinite loop, it must be restarted as explained in Section 4.1.

Upon execution of either the r;G or r;P command, the general registers 0-6 are set to the values in the locations specified as $0-$6 (as explained in Section 4.3.3). The processor status register is set to the value in the location specified as $S.

When ODT encounters a breakpoint, execution stops and ODT prints Bn; (where n is the breakpoint number), followed by the address of the breakpoint. You can then examine locations for expected data. For example:

```
*1010;3B        sets breakpoint 3 at location 1010
*1000;G         starts execution at location 1000
B3;001010       stops execution at location 1010
*
```

To continue program execution from the breakpoint, type ;P in response to ODT's last prompt (*).

When you set a breakpoint in a loop, you may want to allow the program to execute a certain number of times through the loop before ODT recognizes the breakpoint. Set a proceed count by using the k;P command. This command specifies the number of times the breakpoint is to be encountered before ODT suspends program execution (on the kth encounter). The count, k, refers only to the numbered breakpoint that most recently occurred. (Execution of other breakpoints is determined by their own repeat counts.) You may specify a different proceed count for the breakpoint when it is encountered. Thus:

```
B3;001010       halts execution at breakpoint 3
*1026;3B        resets breakpoint 3 at location 1026
*4;P            sets proceed count to 4 and
B3;001026       continues execution; the program loops
*               through the breakpoint three times and halts on
                the fourth occurrence of the breakpoint
```

Following the table of breakpoints (as explained in Section 4.3.6) is a table of proceed command repeat counts for each breakpoint. You can inspect these repeat counts by typing $B/ and nine line feeds. The repeat count for breakpoint 0 prints (the first seven line feeds cause the table of breakpoints to be printed; the eighth types the single instruction mode, explained in the next section, and the ninth line feed begins the table of proceed command repeat counts). The repeat counts for breakpoints 1 through 7 and the repeat count for the single-instruction trap follow in sequence. ODT initializes a proceed count to 0 before you assign it a value. After the command has been executed, it is set to -1. Opening any one of these provides an alternative way of changing the count. Once the location is open, you can modify its contents in the usual manner by typing the new contents followed by the RETURN key. For example:

```
.
.
.
nnnnnn /001036 LF      address of breakpoint 7
nnnnnn /006630 LF      single instruction address
nnnnnn /000000 15 LF   count for breakpoint 0; change to 15
nnnnnn /000000 LF      count for breakpoint 1
.
.
.
nnnnnn /000000 LF      count for breakpoint 7
nnnnnn /nnnnnn         repeat count for single instruction mode
```

Both the address indicated as the single instruction address and the repeat count for single instruction mode are explained in the following section.

4.3.8 Single Instruction Mode

With this mode you specify the number of instructions to be executed before ODT suspends the program run. The proceed command, instead of specifying a repeat count for a breakpoint encounter, specifies the number of succeeding instructions to be executed. Note that breakpoints are disabled in single instruction mode.

Table 4-4 lists the single instruction mode commands.

Table 4-4
Single Instruction Mode Commands

Command	Explanation
;nS	Enables single instruction mode. (n can be any digit and serves only to distinguish this form from the form ;S). Breakpoints are disabled.
n;P	Proceeds with program run for next n instructions before reentering ODT. (If n is missing, it is assumed to be 1.) Trap instructions and associated handlers can affect the proceed repeat count (see Section 4.4.2).
;S	Disables single instruction mode.

When the repeat count for single instruction mode is exhausted and the program suspends execution, ODT prints:

```
B8;nnnnnn
```

where nnnnnn is the address of the next instruction to be executed. The $B breakpoint table contains this address following that of breakpoint 7. However, unlike the table entries for breakpoints 0-7, direct modification has no effect.

Similarly, following the repeat count for breakpoint 7 is the repeat count for single instruction mode. You may modify this table entry directly. This is an alternative way of setting the single-instruction mode repeat count. In such a case, ;P implies the argument set in the $B repeat count table rather than an assumed 1.

4.3.9 Searches

With ODT you can search all or any specific portion of memory for any bit pattern or for references to a particular location.

4.3.9.1 **Word Search (r;W)** - Before initiating a word search, you must specify the mask and search limits. The location represented by $M specifies the mask of the search. $M/ opens the mask register. The next two sequential locations (opened by line feeds) contain the lower and upper limits of the search. ODT examines in the search all bits set to 1 in the mask; it ignores other bits.

You must then give the search object and the initiating command using the r;W command, where r is the search object. When ODT finds a match, (i.e., each bit set to 1 in the search object is set to 1 in the word ODT searches over the mask range) the matching word prints. For example:

```
*$M/000000 177400 LF          tests high-order eight bits
nnnnnn /000000  1000 LF       sets low address limit
nnnnnn /000000  1040 RET      sets high address limit
*400;W                        initiates word search
001010 /000770
001034 /000404
*
```

In the above example, nnnnnn is an address internal to ODT; this location varies and is meaningful only for reference purposes. In the first line above, the slash was used to open $M, which now contains 177400; the line feeds opened the next two sequential locations, which now contain the upper and lower limits of the search.

Typing CTRL/U during a search printout terminates the search.

4.3.9.2 **Effective Address Search (r;E)** - ODT provides a search for words that reference a specific location. Open the mask register only to gain access to the low and high limit registers. After specifying the search limits (as explained for the word search), type the command r;E (where r is the effective address) to initiate the search.

Words that are an absolute address (argument r itself), a relative address offset, or a relative branch to the effective address, print after their addresses. For example:

```
*$M/177400 LF                 opens mask register only to gain
nnnnnn /001000 1010 LF        access to search limits
nnnnnn /001040 1060  RET
*1034;E                       initiates search
001016 /001006                relative branch
001054 /002767                relative branch
*1020;E                       initiates a new search
001022 /177774                relative address offset
001030 /001020                absolute address
```

Give particular attention to the reported effective address references. A word may have the specified bit pattern of an effective address without actually being used as one. ODT reports all possible references whether they are actually used or not.

Typing CTRL/U during a search printout terminates the search.

4.3.10 The Constant Register (rC)

It is often desirable to convert a relocatable address into its value after relocation or to convert a number into its two's complement, and then to store the converted value into one or more places in a program. Use the constant register to perform this and other useful functions.

Typing r:C evaluates the relocatable expression to its 6-digit octal value, prints the value on the terminal, and stores it in the constant register. Invoke the contents of the constant register in subsequent relocatable expressions by typing the letter C. Examples follow:

*-4432;C=173346	places the two's complement of 4432 in the constant register
*6632/062701 C RET	stores the contents of the constant register in location 6632
*1000;1R	sets relocation register 1 to 1000
*1,4272;C=005272	reprints relative location 4272 as an absolute location and stores it in the constant register

4.3.11 Memory Block Initialization (;F and ;I)

Use the constant register with the commands ;F and ;I to set a block of memory to a specific value. While the most common value required is zero, other possibilities are plus one, minus one, ASCII space, etc.

When you type the command ;F, ODT stores the contents of the constant register in successive memory words starting at the memory word address you specify in the lower search limit, and ending with the address you specify in the upper search limit.

Typing the command ;I stores the low-order 8 bits in the constant register in successive bytes of memory starting at the byte address you specify in the lower search limit and ending with the byte address you specify in the upper search limit.

For example, assume relocation register 1 contains 7000, 2 contains 10000, and 3 contains 15000. The following sequence sets word locations 7000-7776 to zero, and byte locations 10000-14777 to ASCII spaces:

```
*$M/000000 LF                  opens the mask register to gain
                               access to search limits
nnnnnn /000000 1,0 LF          sets the lower limit to 7000
nnnnnn /000000 2,-2 LF         sets the upper limit to 7776
*0;C=000000                    sets the constant register to zero
*;F                            sets locations 7000-7776 to zero

*$M/000000 LF
nnnnnn/007000 2,0 LF           sets the lower limit to 10000
nnnnnn/007776 3,-1 RET         sets the upper limit to 14777
*40;C=000040                   sets the constant register to 40
                               (space)
*;I                            sets the byte locations 10000-14777
*                              to the value in the low-order 8
                               bits of the constant register
```

4.3.12 Calculating Offsets (r;O)

Relative addressing and branching involve the use of an offset. An offset is the number of words or bytes forward or backward from the current location to the effective address. During the debugging session it may be necessary to change a relative address or branch reference by replacing one instruction offset with another. ODT calculates the offsets in response to the r;O command.

The command r;O causes ODT to print the 16-bit and 8-bit offsets from the currently open location to address r. For example:

```
*346/000034 414;O 000044 022 22  RET
*/000022
```

This command opens location 346, calculates and prints the offsets from location 346 to location 414, changes the contents of location 346 to 22 (the 8-bit offset) and verifies the contents of location 346.

The 8-bit offset prints only if it is in the range -128(decimal) to 127(decimal) and the 16-bit offset is even, as was the case above. In the next example, the offset of a relative branch is calculated and modified:

```
*1034/103421 1034;O 177776 377 \021 377  RET
*/103777
```

Note that the modified low-order byte 377 must be combined with the unmodified high-order byte.

4.3.13 Relocation Register Commands

The use of the relocation registers is described briefly in Section 4.2. At the beginning of a debugging session it is desirable to preset the registers to the relocation biases of those relocatable modules that will be receiving the most attention.

Do this by typing the relocation bias, followed by a semicolon and the specification of relocation registers, as follows:

```
r;nR
```

where r may be any relocatable expression and n is an integer in the range 0 - 7. If you omit n, it is assumed to be 0. For example:

```
*1000;5R        puts 1000 into relocation register 5
*5,100;5R       adds 100 to the contents
*               of relocation register 5
```

Once a relocation register is defined, you can use it to reference relocatable values. For example,

```
*2000;1R        puts 2000 into relocation register 1
*1,2176/002466  examines the contents of location 4176
*1,3712;0B      sets a breakpoint at location 5712
```

Sometimes programs may be relocated to an address below the one at which they were assembled. This could occur with PIC code (position independent code), which is moved without using the linker. In this case, the appropriate relocation bias would be the two's complement of the actual downward displacement. One method for easily evaluating

the bias and putting it in the relocation register is illustrated in the following example.

Assume a program was assembled at location 5000 and was moved to location 1000. Then the following sequence enters the two's complement of 4000 in relocation register 1.

```
*1000;1R
*1,-5000;1R
*
```

Relocation registers are initialized to -1 so that unwanted relocation registers never enter into the selection process when ODT searches for the most appropriate register.

To set a relocation register to -1, type ;nR. To set all relocation registers to -1, type ;R.

ODT maintains a table of relocation registers, beginning at the address specified by $R. Opening $R ($R/) opens relocation register 0. Successively typing a line feed opens the other relocation registers in sequence. When a relocation register is opened in this way, you can modify it as you would any other memory location.

4.3.14 The Relocation Calculators, nR and n!

When a location has been opened, it is often desirable to relate the relocated address and the contents of the location back to their relocatable values. To calculate the relocatable address of the opened location relative to a particular relocation bias, type:

```
n!
```

The symbol n specifies the relocation register. This calculator works with opened bytes and words. If you omit n, the relocation register whose contents are closest to, but less than or equal to the opened location is selected automatically by ODT. In the following example, assume that these conditions are fulfilled by relocation register 3, which contains 2000. Use the following command to find the most likely module that a given opened byte is in:

```
*2500\011 = !=3,000500
```

To calculate the difference between the contents of the opened location and a relocation register, type nR, where n represents the relocation register. If you omit n, ODT selects the relocation register whose contents are closest to but less than or equal to the contents of the opened location. For example, assume the relocation bias stored in relocation register 1 is 7000:

```
*1,500/11032 1R=1,2032
```

The value 2032 is the content of 1,500, relative to the base 7000. The next example shows the use of both relocation calculators.

If relocation register 1 contains 1000, and relocation register 2 contains 2000, use the following command to calculate the relocatable addresses of location 3000 and its contents, relative to 1000 and 2000:

```
*3000/006410 1!=1,002000 2!=2,001000 1R=1,5410 2R=2,4410
```

4.3.15 ODT Priority Level, $P

$P is used under other PDP-11 operating systems to represent the processor priority level at which ODT is to operate. $P has no function under RSTS/E and is present for compatibility purposes only.

4.3.16 ASCII Input and Output (r;nA)

Inspect and change ASCII text by the command:

```
r;nA
```

where r represents a relocatable expression, and n is a character count. If you omit n, it is assumed to be 1. ODT prints n characters starting at location r, followed by a carriage return/line feed combination. Table 4-5 lists responses and their effect.

Table 4-5
ASCII Terminators

Response	Effect
RET	ODT outputs a carriage return/line feed combination followed by an asterisk, and waits for another command.
LF	ODT opens the byte following the last byte output.
Up to n characters of text	ODT inserts the text into memory, starting at location r. If you type fewer than n characters, terminate the command by typing CTRL/U. This causes a carriage return/line feed/asterisk combination to print. However, if you type exactly n characters, ODT responds with a carriage return/line feed combination, the address of the next available byte, and then a carriage return/line feed/asterisk combination.

4.4 PROGRAMMING CONSIDERATIONS

Information in this section is not necessary for the efficient use of ODT. However, it does provide a better understanding of how ODT performs some of its functions. In certain difficult debugging situations, this understanding is necessary.

4.4.1 Functional Organization

The internal organization of ODT is almost totally modularized into independent subroutines. The internal structure consists of three major functions: command decoding, command execution, and utility routines.

The command decoder interprets the individual commands, checks for command errors, saves input parameters for use in command execution, and sends control to the appropriate command execution routine.

The command execution routines take parameters saved by the command decoder and use the utility routines to execute the specified command. Command execution routines either return to the command decoder or transfer control to your program.

The utility routines are common routines such as SAVE-RESTORE and I/O. They are used by both the command decoder and the command executers.

4.4.2 Breakpoints

The function of a breakpoint is to give control to ODT whenever a program tries to execute the instruction at the selected address. Upon encountering a breakpoint, you can use all of the ODT commands to examine and modify the program.

When a breakpoint is executed, ODT removes all the breakpoint instructions from the code so that you can examine and alter the locations. ODT then types a message on the terminal in the form Bn;k, where k is the breakpoint address and n is the breakpoint number. ODT automatically restores the breakpoints when execution resumes.

There is a major restriction in the use of breakpoints: the program must not reference the word where a breakpoint was set since ODT altered the word. You should also avoid setting a breakpoint at the location of any instruction that causes or returns from traps (e.g., EMT, RTI). These instructions are likely to clear the T-bit, since a new word from the trap vector or the stack is loaded into the status register.

A breakpoint occurs when a trace trap instruction (placed in your program by ODT) is executed. When a breakpoint occurs, ODT operates according to the following algorithm:

1. Sets processor priority to 7 (automatically set by trap instruction). See Section 4.3.15.
2. Saves registers and sets up stack.
3. If internal T-bit trap flag is set, goes to step 13.
4. Removes breakpoints.
5. Resets processor priority to ODT's priority or user's priority. See Section 4.3.15.
6. Makes sure a breakpoint or single-instruction mode caused the interrupt.
7. If the breakpoint did not cause the interrupt, goes to step 15.
8. Decrements repeat count.
9. Goes to step 18 if non-zero; otherwise resets count to 1.
10. Saves terminal status.

11. Types message about the breakpoint or single-instruction mode interrupt.

12. Goes to command decoder.

13. Clears T-bit in stack and internal T-bit flag.

14. Jumps to the go processor.

15. Saves terminal status.

16. Types BE (bad entry) followed by the address.

17. Clears the T-bit, if set, in the user status and proceeds to the command decoder.

18. Goes to the proceed processor, bypassing the TT restore routine.

ODT processes a proceed (;P) command according to the following algorithm:

1. Checks the proceed for legality.

2. Sets the processor priority to 7. See Section 4.3.15.

3. Sets the T-bit flags (internal and user status).

4. Restores the user registers, status, and program counter.

5. Returns control to the user.

6. When the T-bit trap occurs, executes steps 1, 2, 3, 13, and 14 of the breakpoint sequence, restores breakpoints, and resumes normal program execution.

When a breakpoint is placed on an IOT, EMT, TRAP, or any instruction causing a trap, ODT follows this algorithm:

1. When the breakpoint occurs as described above, enters ODT.

2. When ;P is typed, sets the T-bit and executes the IOT, EMT, TRAP, or other trapping instruction.

3. Pushes the current PC and status (with the T-bit included) on the stack.

4. Obtains the new PC and status (no T-bit set) from the respective trap vector.

5. Executes the whole trap service routine without any breakpoints.

6. When an RTI is executed, restores the saved PC and PS (including the T-bit). Executes the instruction following the trap-causing instruction. If this instruction is not another trap-causing instruction, the T-bit trap occurs; reinserts the breakpoints in the user program, or decrements the single-instruction mode repeat count. If the following instruction is a trap-causing instruction, repeats this sequence starting at step 3.

NOTE

Exit from the trap handler must be via the RTI instruction. Otherwise, the T-bit is lost. ODT cannot regain control since the breakpoints have not yet been reinserted.

Note that the ;P command is illegal if a breakpoint has not occurred (ODT responds with ?). ;P is legal, however, after any trace trap entry.

The internal breakpoint status words have the following format:

1. The first eight words contain the breakpoint addresses for breakpoints 0-7. (The ninth word contains the address of the next instruction to be executed in single-instruction mode.)

2. The next eight words contain the respective repeat counts. (The following word contains the repeat count for single-instruction mode.)

You may change these words at will, either by using the breakpoint commands or by directly manipulating $B.

If program runaway occurs (that is, when the program is no longer under ODT control, perhaps executing an unexpected part of the program where you did not place a breakpoint) type CTRL/C if you wish to stop execution. To restart ODT you must relink your program with ODT, as described in Section 4.1.

4.4.3 Searches

The word search lets you search for bit patterns in specified sections of memory. Using the $M/ command, specify a mask, a lower search limit ($M+2), and an upper search limit ($M+4). Specify the search object in the search command itself.

The word search compares selected bits (where 1's appear in the mask) in the word and search object. If all of the selected bits are equal, the unmasked word prints.

The search algorithm is:

1. Fetches a word at the current address.

2. XORs (exclusive OR) the word and search object.

3. ANDs the result of step 2 with the mask.

4. If the result of step 3 is zero, types the address of the unmasked word and its contents; otherwise, proceeds to step 5.

5. Adds two to the current address. If the current address is greater than the upper limit, types * and returns to the command decoder; otherwise, goes to step 1.

Note that if the mask is zero, ODT prints every word between the limits, since a match occurs every time (i.e., the result of step 3 is always zero).

In the effective address search, ODT interprets every word in the search range as an instruction that is interrogated for a possible direct relationship to the search object. The mask register is opened only to gain access to the search limit registers.

The algorithm for the effective address search follows. ((X) denotes contents of X, and K denotes the search object):

1. Fetches a word at the current address X.
2. If (X)=K [direct reference], prints contents and goes to step 5.
3. If (X)+X+2=K [indexed by PC], prints contents and goes to step 5.
4. If (X) is a relative branch to K, prints contents.
5. Adds two to the current address. If the current address is greater than the upper limit, performs a carriage return/line feed combination and returns to the command decoder; otherwise, goes to step 1.

4.5 ERROR DETECTION

ODT detects two types of error: illegal or unrecognizable command, and bad breakpoint entry. ODT does not check for the legality of an address when you command it to open a location for examination or modification. Thus the command:

```
177774/
?MON-F-Trap to 4 003362
```

references nonexistent memory, thereby causing a trap through the vector at location 4. If the program you are debugging with ODT has requested traps through location 4 with the .TRPSET EMT, the program will receive control at its TRPSET address.

Typing something other than a legal command causes ODT to ignore the command and to print:

```
(echoes illegal command)?
*
```

and to wait for another command. Therefore, to cause ODT to ignore a command just typed, type any illegal character (such as 9 or RUBOUT) and the command will be treated as an error and ignored.

ODT suspends program execution whenever it encounters a breakpoint (that is, traps to its breakpoint routine). If the breakpoint routine is entered and no known breakpoint caused the entry, ODT prints:

```
BEnnnnnn
*
```

and waits for another command. BEnnnnnn denotes bad entry from location nnnnnn. A bad entry may be caused by an illegal trace trap instruction, by a T-bit set in the status register, or by a jump to some random location within ODT.

CHAPTER 5

LIBRARIAN

The librarian utility program (LIBR) lets you create, update, modify, list, and maintain object library files. It also lets you create macro library files to use with the MACRO-11 assembler.

A library file is a direct access file that contains one or more modules of the same module type. The librarian organizes the library files so that the linker and MACRO-11 assembler can access them rapidly. Each library contains a library header, library directory (or global symbol or macro name table), and one or more object modules or macro definitions. The object modules in a library file can be routines that are repeatedly used in a program, routines that are used by more than one program, or routines that are related and simply gathered together for convenience. The contents of the library file are determined by your needs. An example of a typical object library file is the default system library that the linker uses, SYSLIB.OBJ.

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

5.1 CALLING AND USING LIBR

To call the RSTS/E librarian from the system device, respond to the prompt printed by the keyboard monitor by typing:

```
RUN $LIBR RET
```

LIBR prints an asterisk at the left margin on the terminal when it is ready to accept a command line.

Type two CTRL/C's to halt the librarian at any time (or a single CTRL/C to halt the librarian when it is waiting for console terminal input) and return control to the monitor.

Section 5.2 explains how to use the librarian to create and modify object libraries; Section 5.3 describes how to create macro libraries.

LIBR accepts command strings in the following general format:

```
*dev:lib,dev:list=dev:obj's/option
```

where

dev:lib	represents the file specification for the library file to be created or updated.

dev:list	represents the file specification for a listing file of the library's contents.
dev:obj's	represents the input object modules (you can specify up to six input files); it can also represent a library file to update.
option	represents an option from Table 5-1.

All file specifications are of the standard RSTS/E command string format, as described in Section 1.3. Default file extensions are assigned as follows:

File	File Extension
list file:	.LST
library file:	.OBJ
input files:	.OBJ

If you specify no device, the default device is assumed.

Each input file consists of one or more object modules, and is stored on a given device under a specific filename and extension. Once an object module is inserted into a library file, the module is no longer referenced by the name of the file of which it was a part, but by its individual module name. This module name is either the subprogram name you assign to FORTRAN routines or the name you specify in a .TITLE statement of an assembly language routine. In the latter case the default name .MAIN. is assigned by the assembler when no .TITLE statement is included.

Thus, for example, the input file FORT.OBJ can exist on DT2: and can contain an object module called ABC. Once the module is inserted into a library file, reference only ABC (not FORT.OBJ).

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

5.2 OPTION COMMANDS AND FUNCTIONS FOR OBJECT LIBRARIES

You maintain object library files by using option commands. Functions that you can perform include object module deletion, insertion and replacement, library file creation, and listing of an object library file's contents.

Table 5-1 summarizes the options available for you to use with RSTS/E LIBR for object libraries. The following sections, which are arranged alphabetically by option, describe the options in greater detail.

Table 5-1
LIBR Object Options

Option	Command Line	Section	Meaning
/C	any but last	5.2.1	Command continuation; allows you to type the input specification on more than one line.
/D	first	5.2.4	Delete; deletes modules that you specify from a library file.
/E	first	5.2.5	Extract; extracts a module from a library and stores it in an OBJ file.
/G	first	5.2.6	Global deletion; deletes global symbols that you specify from the library directory.
/N	first	5.2.7	Names; includes the module names in the directory.
/P	first	5.2.8	P-section names; includes the program section names in the directory.
/R	first	5.2.9	Replace; replaces modules in a library file.
/U	first	5.2.10	Update; inserts and replaces modules in a library file.
/W	first	5.2.11	Indicates wide format for the listing file.
//	first and last	5.2.1	Command continuation; allows you to type the input specification on more than one line.

There is no option to indicate module insertion. Modules are automatically inserted into the library file by the librarian if you do not specify an option.

5.2.1 Command Continuation Options (/C and //)

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

In the following example, a library file is created on the default device under the file name ALIB.OBJ; a listing of the library file's contents is created as LIBLST.LST (also on the default device). The file names of the input modules are MAIN.OBJ, TEST.OBJ, FXN.OBJ, and TRACK.OBJ, all from DK1:.

```
*ALIB,LIBLST=DK1:MAIN,TEST,FXN/C
*DK1:TRACK
```

In the next example, a library file is created on the default device under the name BLIB.OBJ. No listing is produced. Input files are MAIN.OBJ from the default device, TEST.OBJ from DK1:, FXN.OBJ from DK0:, and TRACK.OBJ from DT1:.

```
*BLIB=MAIN//
*DK1:TEST
*DK0:FXN
*DT1:TRACK//
```

Another way of writing this command line is:

```
*BLIB=MAIN,DK1:TEST,DK0:FXN//
*DT1:TRACK
*//
```

5.2.2 Creating a Library File

To create a library file, specify a filename on the output side of a command line.

The following example creates a new library called NEWLIB.OBJ on the default device. The modules that make up this library file are in the files FIRST.OBJ and SECOND.OBJ, both on the default device.

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

Assume this command line is next entered:

```
*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, and the object modules in the files THIRD.OBJ and FOURTH.OBJ are inserted into the library file NEWLIB.OBJ.

5.2.3 Inserting Modules into a Library

The insert function is assumed whenever an input file does not have an associated option; the modules in the file are inserted into the library file you name on the output side of the command string. Any number of input files are allowed. 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. However, the librarian updates the library file, ignoring the global symbol or PSECT in error. You can then enter another command string.

Although you can insert object modules that exist under the same name (as assigned by the .TITLE statement or SUBROUTINE name statement in FORTRAN) this practice is not recommended because of the difficulty involved when replacing or updating these modules (Sections 5.2.9 and 5.2.10 describe replacing and updating).

NOTE

The library operations of module insertion, replacement, deletion, merge, and update are actually performed along with the library file creation operation. Therefore, you must indicate the library file to which the operation is directed on both the input and output sides of the command line, since effectively a "new" output library file is created each time the operation is performed. You must specify the library file first in the input field.

The following command line inserts the modules included in the files FA.OBJ, FB.OBJ, and FC.OBJ on DT1: into a library file named DXYNEW.OBJ on the default device. The resulting library also includes the contents of library DXY.OBJ.

```
*DXYNEW=DXY,DT1:FA,FB,FC
```

5.2.4 Delete Option (/D)

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

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

```
Module name?
```

Respond with the name of the module to be deleted followed by a carriage return; continue until all modules to be deleted have been entered. Type a carriage return immediately after the Module name? message to terminate input and initiate execution of the command line.

The following example deletes the modules SGN and TAN from the library file TRAP.OBJ on DK3:.

```
*DK3:TRAP=DK3: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, and the modules in the file DEF.OBJ are inserted into the library.

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

In the following example, two modules of the same name are deleted from the library file LIBFIL.OBJ.

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

5.2.5 Extract Option (/E)

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

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

```
Global?
```

Respond with the global symbol (entry point) which is defined in the object module to be extracted.

The /E option cannot be used on the same command line with any other option.

The following example extracts the ATAN routine from the FORTRAN library, SYSLIB.OBJ and stores it in a file called ATAN.OBJ on DK1:.

```
*DK1: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?
```

5.2.6 Delete Global Option (/G)

The (/G) option lets you delete a specific global symbol from a library file's directory.

When you use the /G option, the librarian prints:

```
Global?
```

Respond with the name of the global symbol to be deleted followed by a carriage return; continue until all globals to be deleted have been entered. Type a carriage return immediately after the Global? message to terminate input and initiate execution of the command line.

The following command instructs LIBR to delete the global symbols NAMEA and NAMEB from the directory found in the library file ROLL.OBJ on the default device.

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

Globals are only deleted from the directory (and not from the library itself). Whenever a library file is updated, all globals that were previously deleted are restored unless you use the /G option again to delete them. This feature lets you recover if you have inadvertently deleted the wrong global.

5.2.7 Include Module Names Option (/N)

The librarian does not include module names in the directory unless you use the /N option on the first line of the command. The linker loads modules from libraries based on undefined globals, not on module names. It also provides equivalent functions by using global symbols and not module names. Normally, then, it is a waste of space and a performance compromise to include module names in the directory.

If you do not include module names in the directory, the MODULE column of the directory listing is blank, unless the module requires a continuation line to print all its globals. Continued lines are indicated by a plus (+) sign in the MODULE column.

If the library does not have module names in its directory, you must create a new library to include the module names. The following example illustrates how to do this. It creates a new library from the current library, and lists its directory on the terminal.

```
*NEWLIB,TT:=OLDLIB/N
RT-11 LIBRARIAN Y03.00  TUE 03-MAY-77 20:36:41
NEWLIB                  TUE 03-MAY-77 20:36:40

MODULE          GLOBALS         GLOBALS         GLOBALS

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

5.2.8 Include P-section Names Option (/P)

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

This option is provided for compatibility with the RT-11 operating system. Digital recommends that you avoid using it with RSTS/E.

5.2.9 Replace Option (/R)

Use the /R option to replace modules in a library file. The /R option replaces existing modules in the library file you specify as output with the modules of the same names contained in the file(s) you specify as input. The input library file must precede the files used in the replacement operation.

If an old module does not exist under the same name as an input module, or if you specify the /R option on a library file, the librarian prints an error message preceded by the module name, and ignores the replace command. /R must follow each input filename containing modules for replacement.

The following command line indicates that the modules in the file INB.OBJ are to replace existing modules of the same names in the library file TFIL.OBJ. The object modules in the files INA.OBJ and INC.OBJ are to be added. All files are stored on the default device.

```
*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
```

5.2.10 Update Option (/U)

The /U option lets you update a library file by combining the insert and replace functions. If the object modules included in an input file in the command line already exist in the library file, they are replaced; if not, they are inserted. (Some of the error messages that might occur under the insert and replace functions are not printed when you use the update function.) /U must follow each input file containing modules to be updated. The input library file must precede the input files used in the update operation.

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

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

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

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

5.2.11 Wide Option (/W)

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

5.2.12 Listing the Directory of a Library File

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

The command syntax is as follows:

```
*,LP:=dev:lib
```

or

```
*,dev:list=dev:lib
```

where

dev:lib	represents the file specification for the existing library file.
LP:	indicates that the listing is to be sent directly to the line printer (or terminal, if you use TT:).
dev:list	represents the file specification for the list file of the library file's contents.

The following command outputs to DT2: as LIST.LST a listing of the contents of the library file LIBFIL.OBJ on the default device.

```
*,DT2:LIST=LIBFIL
```

The next command sends to the line printer a listing of all modules in the library file FLIB.OBJ, which is stored on the default device.

```
*,LP:=FLIB
```

Here is a sample directory listing:

```
*,TT:=SYSLIB
RT-11 LIBRARIAN Y03.00  TUE 03-MAY-77 21:01:01
SYSLIB                  TUE 03-MAY-77 20:59:47

MODULE          GLOBALS         GLOBALS         GLOBALS

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

The first line of the listing file shows the version of the librarian that was used, and the current date and time. The second line prints the library filename and the date and time the library was created. Module names are not included in this example. Each line in the rest of the listing shows only the globals that appear in a particular module. If a module contains more global symbol names than can print on one line, a new line will be started with a plus (+) sign in column one to indicate continuation.

5.2.13 Merging Library Files

You can merge two or more library files under one filename 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 filename is identical to one of the input filenames.

The command syntax is as follows:

```
*dev:newlib=dev:oldlib1,dev:oldlib2,...,dev:oldlibn
```

where

dev:newlib	represents the library file that will contain all the merged files. (If a library file already exists under this filename, it must also be indicated in the input side of the command line in order to be included in the merge).
dev:oldlibn	represents a library file to be merged.

Thus, the following command combines library files MAIN.OBJ, TRIG.OBJ, STP.OBJ, and BAC.OBJ under the existing library file name MAIN.OBJ; all files are on the default device. Note that this replaces the old contents of MAIN.OBJ.

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

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

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

NOTE

Library files that have been combined under PIP are illegal as input to both the librarian and the linker.

5.2.14 Combining Library Option Functions

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

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

Here is an example that combines options:

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

The librarian performs the functions in this example in order, as follows:

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

5.3 OPTION COMMANDS AND FUNCTIONS FOR MACRO LIBRARIES

The librarian lets you create macro libraries. A macro library works with the MACRO assembler to reduce macro search time.

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

The default input and output file extension for macro files is .MAC.

Be careful not to give the library file the same name as one of the input files. This deletes the input file when the library is created.

Table 5-2 summarizes the options you can use with macro libraries. The options are explained in detail in the following two sections.

Table 5-2
LIBR Macro Options

Option	Command Line	Section	Explanation
/C	any but last	5.3.1	Command continuation; allows you to type the input specification on more than one line.
/M[:n]	first	5.3.2	Macro; creates a macro library from the ASCII input file containing .MACRO directives.
//	first and last	5.3.1	Command continuation; allows you to type the input specification on more than one line.

5.3.1 Command Continuation Options (/C or //)

These options are the same for macro libraries as for object libraries. They are described in Section 5.2.1.

5.3.2 Macro Option (/M[:n])

The /M[:n] option creates a macro library file from an ASCII input file that contains .MACRO directives. The optional argument, n, determines the amount of space to allocate for the macro name directory. Remember that n is interpreted as an octal number; you must follow n by a decimal point (n.) to indicate a decimal number. Each 64 macros occupy one block of library directory space. The default value for n is 128, enough space for 128 macros, which will use 2 blocks for the macro name table.

The command syntax is as follows:

```
*dev:lib=dev:source/M[:n]
```

where

dev:lib	represents the macro library to be created.
dev:source	represents the ASCII input file that contains .MACRO definitions.
/M[:n]	is the macro option.

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

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

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

5.4 LIBR ERROR MESSAGES

The following error messages are output on the terminal by the librarian program. All messages are of the form:

```
?LIBR-n-message
```

where n represents the severity code of the error. Severity codes can be F (Fatal) or W (Warning). Fatal errors cause the current command or statement to be ignored. You must enter another command. A warning message indicates an error condition that may affect execution at a later time. The condition causing the message may require some attention.

Message	Explanation
?LIBR-F-Bad GSD in FILNAM	There was an error in the global symbol directory (GSD). The file is probably not a legal object module.
?LIBR-F-Bad library for listing or extract	The input file specified for extraction or to produce a directory listing was not an object library file. Verify the filename in the command line and check for typing errors. A valid object library file is required for extraction or to produce a directory listing.
?LIBR-F-Bad option: /x	The librarian did not recognize the given option (/x). The librarian restarts and prompts with an asterisk.
?LIBR-F-EOF during extract	The end of the input file was reached before the end of the module being extracted. This is an unusual internal consistency-check error. 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.
?LIBR-F-File not found FILNAM	One of the input files indicated in the command line was not found. LIBR prints an asterisk; the command may be reentered.
?LIBR-F-Illegal error	An internal error occurred while the librarian was in the process of recovering from a previous system or user error. Retry the operations that produced this error; if it recurs, report the error to DIGITAL using an SPR (Software Performance Report); include a program listing and a machine-readable source program, if possible.
?LIBR-F-Illegal extract of AAAAAA	An extraction of the identified global symbol was attempted but the symbol was not found in the library.

Message	Explanation
?LIBR-F-Illegal option combination	Options have been specified that request conflicting functions to be performed. For example, if /E is specified, no other option may be used. If /M is specified, only continuation options (/C, //) may follow.
?LIBR-F-Illegal record type in FILNAM	A formatted binary record had a type not in the range 1-10 (octal). Verify that the correct filenames were specified as input; check for a typing error in the command line. Reassemble or recompile the source and retry the operation.
?LIBR-F-Illegal replace of library file FILNAM	The command line specified that a library file be replaced by another library file. Check for a typing error in the command line. Only object modules can be replaced in a library file. Enter another command.
?LIBR-F-Insufficient memory	Available free memory has been used up. The current command is aborted.
?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 option /M: The default is 128 names.
?LIBR-F-No value allowed: /a	The specified option (/a) does not take a value. The librarian restarts and prompts with an asterisk.
?LIBR-F-Output and input filnam the same	The same filename was specified for both input and output files when the command string to build the macro library was specified. Use different filenames for the input and output files specified to build a macro library.

Message	Explanation
?LIBR-F-Output device full	The device was full; LIBR was unable to create or update the indicated library file.
?LIBR-F-Output file full	The output file was not large enough to hold the library file or list file.
?LIBR-F-Output write error	An unrecoverable error occurred while processing an output file. This may indicate that there was not enough space left on a device to create a file, although there may have been enough directory entries left.
?LIBR-F-Read error in FILNAM	An unrecoverable error has occurred while processing an input file. LIBR prints an asterisk and waits for another command to be entered.
?LIBR-W-Duplicate module name of AAAAAA	A new module has been 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 option is needed; the option for update is /U and the option for replacement is /R.
?LIBR-W-Illegal character	The symbol name entered contained a non-Radix-50 character. Retype the command line 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. Check for a typing error in the command line. The entry point name or module name isnd processing continues.
?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.

Message	Explanation
?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.
?LIBR-W-Null library	An attempt was made to build a library file containing no directory entries. Verify that the correct filenames 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.

CHAPTER 6

PATCH

You can use the PATCH utility program to make code modifications to memory image (.SAV) files, including overlay-structured files. You use PATCH to interrogate, and then to change, words or bytes in the file.

Additionally, PATCH is needed for maintenance of the RSTS/E FORTRAN IV utilities. System patches published by DEC are incorporated into the appropriate system program using the PATCH utility.

It is always a good idea to create a backup version of the file you want to patch, because PATCH makes changes directly to the file as you work.

NOTE

The PATCH program should not be used to modify code in a (user-written) .SAV file that originated as FORTRAN source code. If such a modification becomes necessary, the source file should be edited and recompiled.

6.1 CALLING AND USING PATCH

To call PATCH from the system device, respond to the READY message printed by the keyboard monitor by typing:

```
RUN $PATCH RET
```

PATCH then prints:

```
FILE NAME --
*
```

You should enter the name of the file you want to patch according to this general syntax:

```
dev:filnam.ext/options
```

where

dev represents an optional device specification; if not specified, the default device is assumed.

filnam.ext represents the name of the file which is to be patched; if an extension is not indicated, .SAV is assumed.

/options is one or more of the options listed in Table 6-1.

6.1.1 PATCH Options

Table 6-1 summarizes the options that are valid for PATCH at this point in the opening command.

Table 6-1
PATCH Options

Option	Meaning
/O	Use if the file is an overlay-structured file.
/C	Requires you to enter a checksum. If you make no modifications, PATCH ignores the /C option.
/D	Use if you do not know the checksum for a particular patch. PATCH prints the checksum for that patch. If you make no modifications, PATCH ignores the /D option.

Note that you must enter the complete file specification and accompanying options at this point; they are not legal at any other time. If you enter a carriage return instead of a file specification, however, PATCH prints its current running version number. It then repeats the prompt for a file specification.

After you enter the file specification, PATCH prints another asterisk and waits for a command. Table 6-2 lists valid PATCH commands.

6.1.2 Checksum

PATCH can maintain a running total of the value of each command, argument, and character you enter. This total is called the checksum for the patch.

The checksum option helps you verify your work. It lets you compare the patch that you make to another patch that is known to be correct. The checksum does not tell you specifically where your error is, but it does tell you that an inconsistency exists.

For example, if you receive from DIGITAL a patch to improve your system's performance, the patch contains a checksum value. You should use the /C option in the first PATCH command line, then make the modifications to your file exactly as shown in the DIGITAL patch. When you exit, PATCH asks you for a checksum. Enter the value from the DIGITAL patch. If the checksum you enter and the checksum that PATCH generated when you made your modifications do not match, PATCH prints the ?PATCH-W-CHECKSUM ERROR message. You then know that you made an error in patching your file, and that you need to try again.

6.2 PATCH COMMANDS

Table 6-2 summarizes the PATCH commands. Upper case characters represent PATCH commands; lower case characters represent octal values or ASCII characters. The following sections describe the commands in detail. Section 6.3 provides examples that use PATCH.

Table 6-2
PATCH Commands

Command	Section	Explanation
v;nR	6.2.8	Sets relocation register n to value v.
x;B	6.2.7	Sets the bottom address of the overlay file to the value x.
r,o/	6.2.3	Opens the word location indicated by the contents of relocation register r + offset o.
r,o\	6.2.3	Opens the byte location indicated by the contents of relocation register r + offset o.
s:r,o/	6.2.3	Opens the word location indicated by the contents of relocation register r + offset o in overlay segment s.
s:r,o\	6.2.3	Opens the byte location indicated by the contents of relocation register r + offset o in overlay segment s.
RET	6.2.3	Closes the currently open word or byte.
LF	6.2.3	Closes the currently open word or byte and opens the next sequential word or byte.
^	6.2.3	Closes the currently open word or byte and opens the previous word or byte.
@	6.2.3	Closes the currently open word and opens the word it addresses.
F	6.2.1	Closes the file currently open and requests a new file specification.
E	6.2.2	Closes the file currently open and returns control to the monitor.
x;O	6.2.5	Indicates that a value in the overlay handler or its tables is being modified to the value x and that the overlay structure must be re-initialized. A value of zero (0) is illegal and generates an error message.

(continued on next page)

Table 6-2 (Cont.)
PATCH Commands

Command	Section	Explanation
&	6.2.6	Indicates that PATCH should add the contents of all subsequently opened locations to the checksum, until it encounters another & symbol.
A	6.2.4	Prints the contents of the opened word or byte as ASCII characters (if a byte is open, one character prints; if a word is open, two characters print).
X	6.2.4	Prints the contents of the opened word as an unpacked Radix-50 word.
C(x[x])	6.2.4	Resets the contents of the opened word or byte to the ASCII value you type (if a byte is open, you must type one character; if a word is open, you must type two characters).
P(xxx)	6.2.4	Resets the contents of the currently opened word to the packed Radix-50 value of the three ASCII characters you type (you must type three characters).

6.2.1 Patching a New File (F)

The F command causes PATCH to request you to enter a checksum, or it prints the required checksum (depending upon the options you specify). It also causes PATCH to close the currently open file, and to print an asterisk indicating its readiness to accept another command string. No checksum dialogue is invoked if you have not previously specified checksum options (with /D or /C).

6.2.2 Exiting from PATCH (E)

The E command causes PATCH to close the currently open file after printing the checksum dialogue according to the options you specify and return control to the monitor. As with the F command, the checksum dialogue is by-passed if you have not specified checksum options.

6.2.3 Examining and Changing Locations in the File

For a non-overlay file, you can open a word address (as with ODT) by typing:

```
[relocation register,]offset/
```

PATCH types the contents of the location and waits for you to enter either a new location contents or another command.

For an overlay file, the format is:

```
[segment number:][relocation register,]offset/
```

where segment number is the overlay segment number as it is printed on the link map for the file. If you omit the segment number, PATCH assumes the root segment. If you make an error in a command string while patching an overlaid program, you can use CTRL/U to cancel the command. However, PATCH assumes the entire line is incorrect and preserves only the previously set relocation registers. PATCH preserves the segment number only across the ^ and LF commands.

Similarly, you can open a byte address in a file. The format for non-overlay files is:

```
[relocation register,]offset\
```

The format for overlay files is:

```
[segment number:][relocation register,]offset\
```

Once a location has been opened, you can optionally type in the new contents in the format:

```
[relocation register,]octal value
```

Follow this line by one of the control characters from Table 6-3.

Table 6-3
PATCH Control Characters

Character	Function
RET	Closes the current location by changing its contents to the new contents (if any), and awaits additional control input.
LF	Closes the current location by changing its content to the new contents (if any), and opens the next sequential word or byte.
^	Closes the current location by changing its contents to the new contents (if any), and opens the previous word or byte.
@	Closes the current word location, and opens the word it addresses (in the same segment if it is an overlay file).

6.2.4 Translating and Indirectly Modifying Locations with a File

After opening a location within a file, you can translate the contents into ASCII characters or into the equivalent of a Radix-50 packed word.

To obtain the ASCII equivalent of the opened location, type the following command after PATCH prints the contents in octal.

```
A
```

PATCH then translates the word or byte into two (or one, if a byte is opened) ASCII characters. In this example, a byte is opened:

```
*1,100\ 102     A = B LF
```

PATCH prints only the printable ASCII characters in the opened word or byte (all non-printing characters, such as ASCII codes 0-37, are represented by the ? character). In this example, a word is opened:

```
*1, 100/ 302     A = B?  LF
```

In the next example, a word is opened, and both ASCII characters are printable:

```
*1, 100/ 33502     A = B7 RET
```

In these examples, one or both of the characters cannot be printed:

```
*0, 400/ 466     A = 6?  LF

*1, 202/ 55001     A = ?Z LF

*616/ 401     A = ??  RET
```

To unpack a Radix-50 word as three ASCII characters, type the following command after PATCH prints the contents of the opened word.

```
X
```

PATCH then unpacks the opened word and prints three ASCII characters.

Note that you must open a word and not a byte.

If the word you open contains an illegal Radix-50 word, PATCH prints ???. If the translated character is not printable, PATCH prints ? in place of it.

Neither the A command nor the X command alters the contents of the open location; however, PATCH updates the checksum to reflect the fact that you have entered a new command.

You can specify the A and X commands in any order on the same command line without altering the contents of the open location. For example,

```
*1, 15022/    50553  X = MAC A = kQ
```

After examining the location with the A or X command, you can change the location if you wish. For example,

```
*45660/10146  A = F?  X = BX8 12122 RET or LF
```

If the same location is reopened, the following change appears:

```
*45660/12122
```

You can change the contents of a location to the ASCII code of the value you specify by using the C command. You can use the P command to change a word to the packed Radix-50 word of the three characters you specify. This example changes an open byte to the ASCII code for the letter Z:

```
*1, 115\ 101    C (Z) RET
```

Note that PATCH prints the parentheses itself; you type only the character Z.

When reopened, that byte contains the ASCII code for Z:

```
*1, 115\ 132
```

Similarly, PATCH inserts the ASCII code for two ASCII characters into the low order and high order bytes, respectively, of one word. This example changes an open word to the ASCII code for AZ:

```
*0,10116/ 103523  C (AZ) ^
```

If reopened, the location contains the ASCII code for AZ:

```
*0,10116/ 55101   A = AZ
```

You can examine the same location in more conventional ways, as this example shows:

```
*0,10116\ 101 LF
0,10117\ 132
```

Similarly, you can use the P command to change the contents of an open word to the Radix-50 packed word equivalent of the three ASCII characters you specify. This example changes the Radix-50 word equivalent of SAV to REL:

```
*2:1,400/ 73376  x = SAV  P (REL)RET
```

6.2.5 Setting Values in the Overlay Handler Tables of a Program

Use the ;O command to effect any changes to the overlay handler tables in an overlaid program. For example,

```
*616/   1043    1100;O
*
```

This command line increases the size of the referenced overlay region by 35(8) words or 58(10) bytes, to allow room for a patch. The value being modified is a value associated with the overlay handler tables, or a value required by the overlay handler for proper overlay structure initialization. The overlay structure is re-initialized and you can enter commands to modify the new region on the next line. A value of zero is not permitted with the ;O command. If you omit the preceding argument, or use 0, an error message prints on the terminal.

6.2.6 Including the Old Contents Into the Checksum

Sometimes it is important that the present contents of the locations being changed have known specific values. This is the case when DIGITAL publishes system patches. The & command is designed to aid in implementing system patches.

It automatically includes the old contents of an open location into the checksum. This command is a simple switch. The first occurrence of the & turns the switch on, the second turns it off. While the switch is on, the old contents of every location you open and close properly become part of the checksum. To use the & command, type:

```
&
```

PATCH then prints a carriage return-line feed sequence and another * indicating its readiness to accept another command. This switch is then enabled.

If you type the command on a line where a location is currently open, PATCH closes the location and resets the switch. PATCH then prompts with an asterisk indicating that it is ready to accept additional commands.

6.2.7 Setting the Bottom Address

To patch an overlay file, PATCH must know the bottom address at which the program was linked if it is different from the initial stack pointer. This is the case if the program sets location 42 in an .ASECT. To set the bottom address, type:

```
bottom address;B
```

You must issue the B command before you open any locations in an overlay for modification.

6.2.8 Setting Relocation Registers

You set the relocation registers 0-7 (as with ODT) with the R command. The R command has the syntax:

```
relocation value;relocation registerR
```

Be careful when you type this command string. If you inadvertently substitute a comma (,) for the semi colon (;) in the R command, PATCH does not generate an error message. However, it does not set the value you specify in the relocation register.

Once you set one of the eight relocation registers, the expression:

```
relocation register,octal number
```

in a command string will have the value:

```
relocation value + octal number
```

6.3 PATCH EXAMPLES

This section contains two examples of patching a file. The program used in both examples is a FORTRAN main program that calls one MACRO subroutine. In the first example the program is in a non-overlaid file. In the second example the MACRO subroutine has been placed in an overlay segment. In each case the steps necessary to assemble the subroutine, link, and patch the files are shown.

6.3.1 Patching a Non-Overlaid File

The following commands assemble the MACRO subroutine file E2.MAC, producing the binary object file SUB.OBJ.

```
RUN $MACRO
*SUB.OBJ, TT:LIST=E2.MAC
```

These commands also cause an assembly listing to be sent directly to the terminal. The assembly listing is shown below.

```
.MAIN.  MACRO V03.01 14-NOV-77 15:42:38 PAGE 1


      1                                                 .GLOBL  ISUM
      2 000000  067567  000002  000022  ISUM:   ADD     @2(R5),SUM
      3 000006  005267  000020                  INC     CNT
      4 000012  016701  000012                  MOV     SUM,R1
      5 000016  005000                          CLR     R0
      6 000020  071067  000006                  DIV     CNT,R0
      7 000024  010100                          MOV     R1,R0
      8 000026  000207                          RTS     PC
      9 000030  000000                  SUM:    .WORD   0
     10 000032  000000                  CNT:    .WORD   0
     11         000001                          .END


.MAIN.  MACRO V03.01 14-NOV-77 15:42:38 PAGE 1-1
SYMBOL TABLE

CNT     000032R          ISUM    000000RG        SUM     000030R

. ABS.  000000     000
        000034     001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  288 WORDS  ( 2 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  70 PAGES
SUB.OBJ,TT:LIST=E2.MAC

ERRORS DETECTED:  0
```

The next commands link the MACRO subroutine SUB.OBJ with the previously compiled FORTRAN main program file called AVG.OBJ. The linking process produces the memory image load module AVG.SAV.

```
Ready
RUN $LINK
*AVG.SAV,TT:MAP=AVG.OBJ,SUB.OBJ, SYSLIB
```

This linker command string sends a load map directly to the terminal. The load map is shown on the next page.

```
RT-11 LINK  V05.02      Load Map        Mon 14-Nov-77 15:45:18
AVG   .SAV      Title:  .MAIN.  Ident:  FORY02

Section  Addr    Size    Global  Value   Global  Value   Global  Value

. ABS.   000000  001000   (RW,I,GBL,ABS,OVR)
                         $USRSW  000000  $RF2A1  000000  .VIR    000000
                         $NLCHN  000006  $HRDWR  000010  $WASIZ  000131
                         $LRECL  000210  $TRACE  004737
OTS$I    001000  014642   (RW,I,LCL,REL,CON)
                         $$OTSI  001000  $OTI    001026  $$OTI   001030
                         $$SET   002750  CMI$SS  003244  CMI$SI  003250
                         CMI$SM  003254  CMI$IS  003260  CMI$II  003264
                          .                                       .
                          .                                       .
                          .                                       .
                         $TTYIN  007652  $FIO    010526  $$FIO   010532
                         $ERRTB  011662  $ERRS   011770  $VRINT  015512
                         $DUMPL  015514
OTS$P    015642  000050   (RW,D,GBL,REL,OVR)
SYS$I    015712  000000   (RW,I,LCL,REL,CON)
USER$I   015712  000000   (RW,I,LCL,REL,CON)
$CODE    015712  000154   (RW,I,LCL,REL,CON)
                         $$OTSC  015712
OTS$O    016066  000750   (RW,I,LCL,REL,CON)
                         $$OTSO  016066  $OPEN   016066
SYS$O    017036  000000   (RW,I,LCL,REL,CON)
$DATAP   017036  000172   (RW,D,LCL,REL,CON)
OTS$D    017230  000016   (RW,D,LCL,REL,CON)
OTS$S    017246  000052   (RW,D,LCL,REL,CON)
                         $AOTS   017316
SYS$S    017320  000000   (RW,D,LCL,REL,CON)
$DATA    017320  000004   (RW,D,LCL,REL,CON)
USER$D   017324  000000   (RW,D,LCL,REL,CON)
.$$$$.   017324  000000   (RW,D,GBL,REL,OVR)
         017324  000034   (RW,I,LCL,REL,CON)
                         ISUM    017324

Transfer address = 015712, High limit = 017360 =  3960. words
```

The load module AVG.SAV is ready for execution. This is the result of running AVG.SAV.

```
RUN AVG.SAV
ENTER NUMBERS TO FIND AVERAGE OF, 1 PER LINE.
ENTER '32767' TO TERMINATE INPUT.
1
2
3
4
5
32767
THE AVERAGE IS 0
STOP--

Ready
```

Apparently AVG.SAV contains an error which causes the wrong result to be returned. By re-examining the logic of the subroutine it is determined that the error occurs in source statement 7. The following example uses PATCH to correct the error.

```
RUN $PATCH

FILE NAME--
*AVG
*017324;1R
*1,24/  10100    240
*E

Ready
```

In this example the first PATCH command specifies the name of the .SAV file to be patched. The second command sets relocation register 1 to the transfer address of the subroutine. The transfer address is shown on the link map to be 17324. The next command opens relative location 24 which, as the assembly listing shows, contains the code of (subroutine) source statement 7. The code stored in relative location 24 is 010100, which is a MOV instruction. To correct the error this code is changed to 240, a no-op instruction. Finally PATCH is terminated with an E (Exit) command and control returns to the monitor.

To verify the changes just made relative location 24 can be reopened for examination, as follows:

```
RUN $PATCH
FILE NAME--
*AVG
*017324;1R
*1,24/240
1,26/207
1,30/0
*E

Ready
```

As before relocation register 1 is set to 17324. Next relative location 24 is opened. Now it contains the correct value, 240. This command line and the next one are terminated with a line feed. This closes the open location and opens the next one. The command line that opens relative location 30 is terminated with a RET. PATCH then prompts for a new command with an asterisk. The E (Exit) command is given and control returns to the monitor.

Execution of the load module AVG.SAV should now produce the desired results.

6.3.2 Patching an Overlaid File

The following commands link the previously compiled FORTRAN main program AVG.OBJ with the previously assembled MACRO subroutine SUB.OBJ. An overlay structure is created which places the main program in the root and the subroutine in overlay segment 1.

```
RUN $LINK
*MYPROG,TT:MAP = AVG.OBJ, SYSLIB//
*SUB.OBJ/O:1//
```

The linking process creates the load module called MYPROG.SAV and sends the linker load map shown below directly to the terminal.

```
RT-11 LINK  V05.02      Load Map        Mon 14-Nov-77 15:52:06
MYPROG.SAV      Title:  .MAIN.  Ident:  FORY02

Section  Addr    Size    Global  Value   Global  Value   Global  Value

. ABS.   000000  001122    (RW,I,GBL,ABS,OVR)
                          $USRSW  000000  $RF2A1  000000  .VIR    000000
                          $NLCHN  000006  $HRDWR  000010  $WASIZ  000131
                          $LRECL  000210  $TRACE  004737
OTS$I    001122  014642    (RW,I,LCL,REL,CON)
                          $$OTSI  001122  $OTI    001150  $$OTI   001152
                          $$SET   003072  CMI$SS  003366  CMI$SI  003372
                          CMI$SM  003376  CMI$IS  003402  CMI$II  003406
                           .               .               .
                           .               .               .
                           .               .               .
                          $INITI  006742  $CLOSE  007054  $GETRE  007720
                          $TTYIN  007774  $FIO    010650  $$FIO   010654
                          $ERRTB  012004  $ERRS   012112  $VRINT  015634
                          $DUMPL  015636
OTS$P    015764  000050    (RW,D,GBL,REL,OVR)
SYS$I    016034  000000    (RW,I,LCL,REL,CON)
USER$I   016034  000000    (RW,I,LCL,REL,CON)
$CODE    016034  000154    (RW,I,LCL,REL,CON)
                          $$OTSC  016034
OTS$O    016210  000750    (RW,I,LCL,REL,CON)
                          $$OTSO  016210  $OPEN   016210
SYS$O    017160  000000    (RW,I,LCL,REL,CON)
$DATAP   017160  000172    (RW,D,LCL,REL,CON)
OTS$D    017352  000016    (RW,D,LCL,REL,CON)
OTS$S    017370  000052    (RW,D,LCL,REL,CON)
                          $AOTS   017440
SYS$S    017442  000000    (RW,D,LCL,REL,CON)
$DATA    017442  000004    (RW,D,LCL,REL,CON)
USER$D   017446  000000    (RW,D,LCL,REL,CON)
.$$$$.   017446  000000    (RW,D,GBL,REL,OVR)
 Segment size = 017446 =  3987. words

 Overlay region 000001  Segment 000001
        017450  000034    (RW,I,LCL,REL,CON)
                          ISUM  @ 017450
 Segment size = 000034 =    14. words


Transfer address = 016034, High limit = 017504 =  4002. words
```

When the load module MYPROG.SAV is run the wrong result (0) is again returned. MYPROG.SAV contains an error in (subroutine) source statement 7 also. The following example uses PATCH to correct the error.

```
Ready

RUN $PATCH

FILE NAME--
*MYPROG/O
*17450;0R
*1:0,24/        10100   240
*E

Ready
```

The option /O is used in the file specification to indicate that an overlaid file is to be patched. The next command line sets relocation register 0 to the start of the overlay segment to be patched. The load map shows that overlay segment 1 begins at location 17450. The next command opens relative location 24, which is in overlay segment 1. (It was determined from an assembly listing that the code of subroutine source statement 7, which contains the error, is in relative memory location 24.) The contents of relative memory location 24 are changed from 10100 (a MOV instruction) to 240 (a no-op instruction). The last command, E (Exit), terminates PATCH and control is returned to the system monitor.

The load module MYPROG.SAV now produces the desired results when executed.

```
RUN MYPROG.SAV
ENTER NUMBERS TO FIND AVERAGE OF, 1 PER LINE.
ENTER '32767' TO TERMINATE INPUT.
1
2
3
4
5
32767
THE AVERAGE IS 3
STOP--

Ready
```

6.4 PATCH ERROR MESSAGES

The following error messages can be output by the PATCH program. All messages are of the form:

```
?PATCH-n-message
```

where n represents the severity code of the error. Severity codes can be F (Fatal), I (Information) or W (Warning). Fatal errors cause the current command or statement to be ignored. You must enter another command. An informational message requires no further action. It is there for your benefit only. A warning message indicates an error condition that may affect execution at a later time. The condition causing the message may require some attention.

Message	Explanation
?PATCH-F-Insufficient memory	There was not enough free core to contain the device handler and the internal "overlay tables." This message should not occur under normal circumstances.
?PATCH-F-Read error	PATCH detected an input error in reading from the file. Check for read-locked or off-line devices.

Message	Explanation
?PATCH-F-Write error	PATCH detected an input error in writing to the file. Check for write-locked or off-line devices.
?PATCH-I-[+2K core]	PATCH needs more memory for overlay handling. PATCH continues executing normally. This message is for your information.
?PATCH-I-CHECKSUM=NNNNNN	PATCH prints out the checksum in response to the /D option after an "E" or "F" command has been issued. This message is for your information only.
?PATCH-W-Address not in segment	The specified address exceeds the limits of the particular overlay. Recheck the linker load map for the address and proper overlay segment.
?PATCH-W-Bottom address wrong	The contents of the address specified does not correspond to the first word in the standard RSTS/E overlay handler. Correct the line in error; specify the correct address using the x;B command.
?PATCH-W-CHECKSUM error	PATCH responds to an incorrectly entered checksum three times. A failure to enter the correct checksum on the third attempt will cause an automatic exit to the monitor. The file being patched has been changed. The incorrectly patched file should be deleted and the backup procedures repeated before attempting to patch the file a second time.
?PATCH-W-Illegal command	The response to the message "FILE NAME --" was incorrect. Check for a typing error in the command line. The file specification must be of the form: dev:filnam.ext/options
?PATCH-W-Illegal option	One of the options encountered in the entered file specification was not a recognized legal option.
?PATCH-W-Invalid overlay handler modification	An attempt was made to insert a zero value into the overlay handler tables for an overlaid program. A non-zero value must be given in conjunction with the ";O" command.

Message	Explanation
?PATCH-W-Invalid relocation register	An attempt was made to reference a relocation register outside the range 0-7. Relocation registers must be set within the range 0-7.
?PATCH-W-Invalid segment number	The specified segment number did not exist in the file being patched. Recheck the linker load map and command string to determine the overlay structure.
?PATCH-W-Must open word	The "@", "P," or "X" command was typed when no address was open.
?PATCH-W-Must specify segment number	The specified address exceeds the limits of the root segment.
?PATCH-W-No address open	The "LF", "^", "@", "X", "P", "C", or "A" command character was typed when no address was open. Check for a typing error in the command line.
?PATCH-W-Not in program bounds	An attempt was made to reference a location outside the limit defined by location 50 in block zero of the file. The value of the initial stack pointer for the program may also be beyond the last location of the program. Check for a typing error in the command line. Check the linker load map to determine where the program was loaded. Check the initial value of the stack pointer.
?PATCH-W-Odd address	An attempt was made to open an odd address as a word with the "/" command. Word addresses must be even numbers. Use "\" to open an odd address.
?PATCH-W-Odd bottom address	The bottom address specified or contained in location 42 of an overlay file was odd. The overlay handler must start on an even word boundary.
?PATCH-W-Program has no segments	An attempt was made to reference an overlay region in a program which was not identified as an overlaid program in the file specification, or an attempt was made to reference an overlay region in a program which has none.

CHAPTER 7

OBJECT MODULE PATCH UTILITY (PAT)

PAT, the RSTS/E object module patch utility, allows you to patch, or update, code in a relocatable binary object module. PAT is primarily used to incorporate system patches published by Digital Equipment Corporation into the compiler and FORTRAN Object Time System. It is also useful when making corrections to routines that are in .OBJ format for which the MACRO source files are not available. (The PAT program should not be used to modify user FORTRAN .OBJ files. It is more efficient to edit and recompile the source file.)

PAT makes the patch to the object module by means of the procedure outlined in Figure 7-2. PAT accepts a file containing corrections or additional instructions and applies these corrections and additions to the original object module. Prepare correction input in MACRO-11 source form and assemble it with the MACRO-11 assembler.

Input to PAT is two files: 1)the original input file and 2)a correction file containing the corrections and additions to the input file. The input file consists of one or more concatenated object modules. You may correct only one of these object modules with a single execution of the PAT utility. The correction file consists of object code that, when linked by the linker, either replaces or appends to the original object module.

Output from PAT is the updated input file.

7.1 CALLING AND USING PAT

To call PAT from the system device, respond to the monitor prompt "READY" by typing:

```
RUN $PAT  RET
```

PAT prints an asterisk at the left margin on the console terminal when it is ready to accept a command line.

Type two CTRL/C's to halt PAT at any time (or a single CTRL/C to halt PAT when it is waiting for terminal input) and return control to the monitor.

Figure 7-1 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.

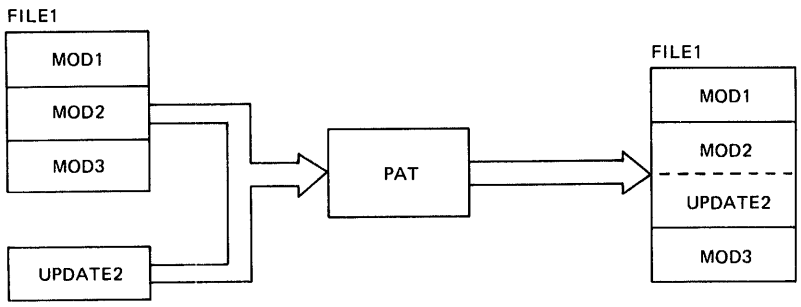


Figure 7-1 Updating a Module Using PAT

There are several steps you must perform to use PAT to update a file. First, create the correction file using a text editor. Then, assemble the correction file to produce an object module. Submit the input file and the correction file in object module form to PAT for processing. Finally, link the updated object module, along with the object modules that make up the rest of the file, to resolve global symbols and create an executable program. Figure 7-2 shows the processing steps involved in generating an updated executable file using PAT.

Specify the PAT command string in the following form:

```
dev:newfile=dev:oldfile/C:number,dev:correct/C:number
```

where

dev:newfile	is the file specification for the output file. If you do not specify an output file, none is generated.
dev:oldfile	is the file specification for the input file. This file can contain one or more concatenated object modules.
dev:correct	is the file specification for the correction file. This file contains the updates being applied to a single module in the input file.
/C	specifies the checksum option, which directs PAT to generate an octal value for the sum of all the binary data composing the module in the file to which the option is applied.
number	specifies an octal value that directs PAT to compare the checksum value it computes for a module with the octal value you specify as number.

Note that inclusion of /C:number in the command string is optional. Also, you can specify /C alone, with no argument (number).

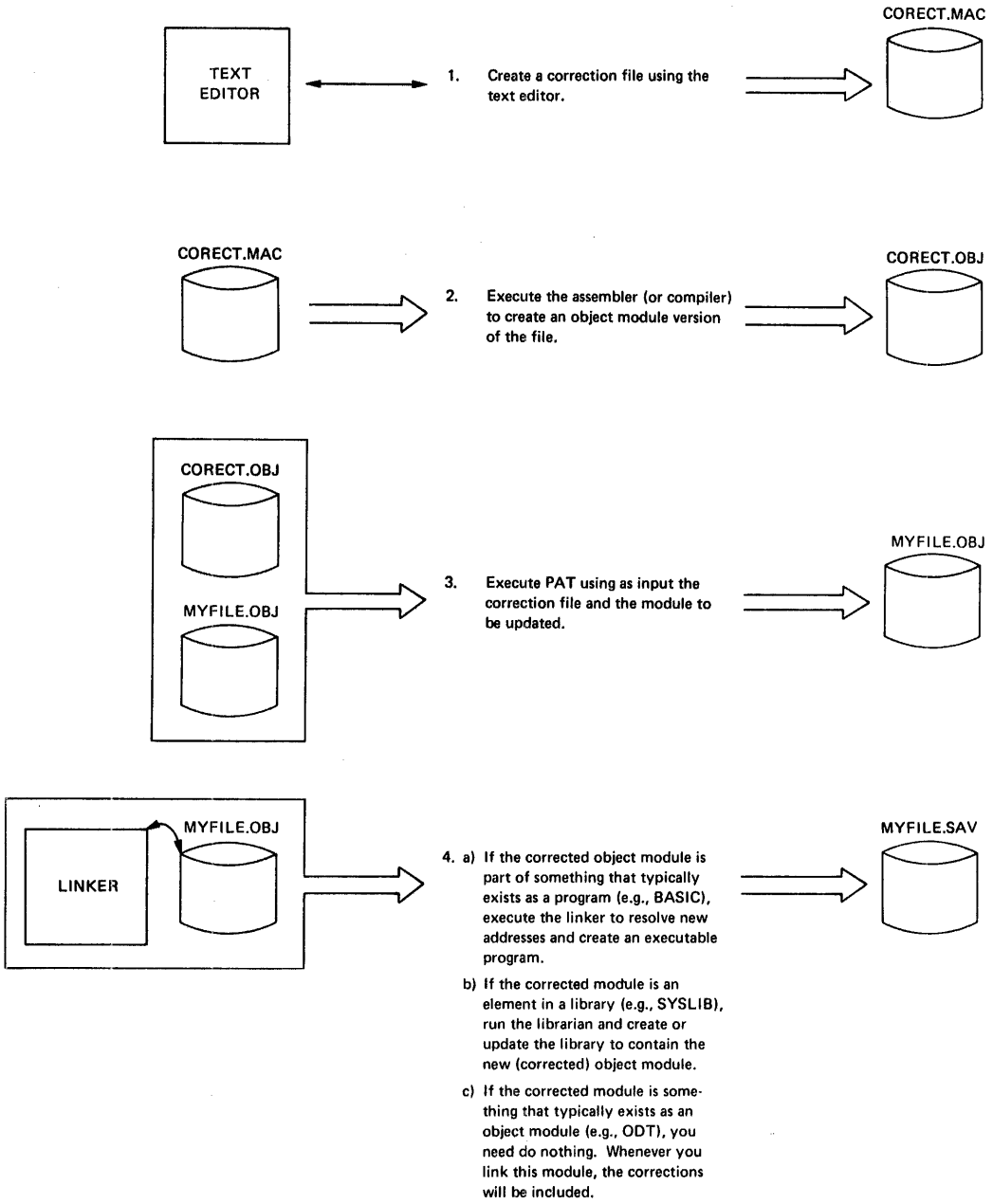


Figure 7-2 Processing Steps Required to Update a Module Using PAT

7.2 HOW PAT APPLIES UPDATES

PAT applies updates to a base input module using additions and corrections supplied in a correction file. This section describes the PAT input and correction files, gives information on how to create the correction file, and gives examples of how to use PAT.

7.2.1 The Input File

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

7.2.2 The Correction File

The correction file must also be in object module format. It is usually created from a MACRO source file in the following format:

```
.TITLE inputname

.IDENT updatenum

inputline

inputline

    *

    *

    *
```

where:

inputname	is the name of the module to be corrected by the PAT update. That is, inputname must be the same name as the name specified on the input file .TITLE directive for a single module in the input file.
updatenum	is any value acceptable to the MACRO assembler. Generally, this value reflects the update version of the file being processed by PAT, as shown in the examples below.
inputline	are lines of input to be used to correct and update the input file.

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

A duplicate PSECT or CSECT supercedes the previous PSECT or CSECT, provided:

- both have the same relocatability attribute (ABS or REL);
- both are defined with the same directive (.PSECT or .CSECT).

If PAT encounters duplicate PSECT names, it sets the length attribute for the PSECT to the length of the longer PSECT and appends a new PSECT to the module.

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

7.2.3 Creating the Correction File

As shown in Figure 7-2, the first step in using PAT to update an object file is to generate a MACRO-11 source correction file. Use the editor to generate these additions and corrections to your file. The correction file must be in object module format before PAT can process it.

Once you have created the source version of the correction file, assemble it to produce an object module that PAT can process.

7.2.4 How PAT and the Linker Update Object Modules

The following examples show an input file and a correction file (called a patch file) to be processed by PAT and the linker, along with a source-like representation of what the output file would look like once PAT and the linker complete processing. Two techniques are described: one is for overlaying lines in a module and the other is for appending a subroutine to a module.

7.2.4.1 Overlaying Lines in a Module - The first example illustrates a technique for overlaying lines in a module by using a patch file. First, PAT appends the correction file to the input file. The linker then executes to replace code within the input file.

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

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

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

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

The patch source is assembled using MACRO-11 and the resulting object file is input to PAT along with the original object file. The following source code represents the result of PAT processing.

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

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

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

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

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

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

The input file for the example is:

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

Suppose you wish to add the instructions:

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

between

```
        MOV     A,B
```

and

```
        JSR     PC,XYZ
```

The correction file to accomplish this goal is as follows:

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

PAT appends the correction file to the input file, as in the previous example. The linker then processes the file and generates the following output file:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL


ABC::
        JSR     PC,PATCH
        NOP
        JSR     PC,XYZ
        MOV     C,R0
        RTS     PC
        .PSECT  PATCH
PATCH:
        MOV     A,B
        MOV     D,R0
        ASL     R0
        RTS     PC
        .END
```

In this example, the JSR PC,PATCH and NOP instructions overlay the three-word MOV A,B instruction. (The NOP is included because this is a case where a 2-word instruction replaces a 3-word instruction. NOP is required to maintain alignment.) The linker allocates additional storage for .PSECT PATCH, writes the specified code into this program section, and binds the JSR instruction to the first address in this section. Note that the MOV A,B instruction replaced by the JSR PC,PATCH is the first instruction the PATCH subroutine executes.

7.2.5 Determining and Validating the Contents of a File

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

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

```
=INFILE/C,INFILE.PAT
```

PAT responds to this command with the message:

```
INPUT MODULE CHECKSUM IS <checksum>
```

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

To validate the changes made to a file, enter the checksum option in the form /C:number. PAT compares the value it computes for the checksum with the value you specify as number. If the two values do not match, PAT displays a message reporting the checksum error:

```
INPUT FILE CHECKSUM ERROR
```

or

```
CORRECTION FILE CHECKSUM ERROR
```

Checksum processing always results in a nonzero value.

Do not confuse this checksum with the record checksum byte.

7.3 PAT ERROR MESSAGES

The following messages can be output by PAT. All messages are of the form:

```
?PAT-n-message
```

where n represents the severity code of the error causing the message. Possible severity codes are F (Fatal), I (Information) or W (Warning). Fatal errors cause the current command or statement to be ignored. You must enter another command. Information messages are for your benefit only. They require no further action. A warning message indicates an error condition that may affect execution at a later time. The condition causing the message may require some attention.

Message	Explanation
?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. Verify that the input filename is correct; check for a typing error in the command line. Reassemble or recompile the source to obtain a good object module and retry the operation.
?PAT-F-Correction file has bad RLD	A global symbol named in a relocatable record was not defined in the global symbol definition record. Reassemble the indicated file. If the condition persists, submit a Software Performance Report.
?PAT-F-Correction file has illegal record	The format of the correction file is not compatible with the object file format PAT requires. The format is not what the standard language processors should produce. Verify that the correction file has the proper format, and retype the command line.
?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 to PAT.
?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 read-locked or off-line devices.
?PAT-F-Illegal error	PAT has generated an illegal error message call. This is an internal software error condition. If the error persists, submit a Software Performance Report with the related console dialogue and any other pertinent information.

Message	Explanation
?PAT-F-Incompatible reference of 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. Choose RELOCATABLE or ABSOLUTE; and specify .PSECT or .CSECT. 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 filename is correct; check for a typing error in the command line. Reassemble or recompile the source and retry the operation.
?PAT-F-Input file has bad RLD	A global symbol named in a relocatable record was not defined in the global symbol definition record. Reassemble the indicated file. If the error persists, submit a Software Performance Report.
?PAT-F-Input file has illegal record	The format of the input file is not compatible with the object file format PAT requires. The format is not what the standard DIGITAL language processors should produce. Verify that the input file has the proper format, and retype the command line.
?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 to PAT.

Message	Explanation
?PAT-F-Input file read error	PAT detected an error while reading the input file. Input hardware can cause this error. Retry the command. If the error persists, submit a Software Performance Report with a copy of the console dialogue and any other pertinent information.
?PAT-F-Insufficient memory	There is not sufficient contiguous memory for PAT to use for the corrected output file.
?PAT-F-Only /C allowed	The input module or correction file specifications contain an illegal option. /C and /C:n are the only option forms PAT accepts.
?PAT-F-Output file full	There is not enough free space on the output volume for the corrected object file.
?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.
?PAT-F-Unable to locate module AAAAAA	The correction file has a module name that does not exist in the input file. PAT shows the name of the module in this message. 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 specifies more than two input files. PAT processes the first as the input module to be corrected and the second as the correction file. PAT ignores all other files.
?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. For the general command line format out1,out2,out3=input,correct PAT's output file must be in the "out1" position. PAT ignores all other output files.

Message	Explanation
?PAT-W-Correction file checksum error	PAT finds a checksum value that is different from the value for the /C correction file option. Mistyping the /C option value or specifying an invalid version of the correction file causes this error.
?PAT-W-Correction file checksum is NNNNNN	PAT responds to the /C option on the correction file with this message. NNNNNN is the octal value of the sum of all binary data composing the file. The message is for your information.
?PAT-W-Input file checksum error	PAT finds a checksum value that is different from the value for the /C input file option. Mistyping the /C option value or specifying an invalid version of the input file causes this warning.
?PAT-W-Input module checksum is NNNNNN	PAT responds to the /C option on the input module with this message. NNNNNN is the octal value of the sum of all binary data composing the file. The message is for your information.

APPENDIX A

CHARACTER CODES

A.1 ASCII CHARACTER SET

EVEN PARITY BIT	7-BIT OCTAL CODE	CHARACTER	REMARKS
0	000	NUL	NULL,TAPE FEED CONTROL/SHIFT/P.
1	001	SOH	START OF HEADING: ALSO SOM, START OF MESSAGE, CONTROL/A.
1	002	STX	START OF TEXT: ALSO EOA, END OF ADDRESS, CONTROL/B.
0	003	ETX	END OF TEXT: ALSO EOM, END OF MESSAGE, CONTROL/C.
1	004	EOT	END OF TRANSMISSION (END): SHUTS OFF TWX MACHINES, CONTROL/D.
0	005	ENQ	ENQUIRY (ENQRY); ALSO WRU, CONTROL/E.
0	006	ACK	ACKNOWLEDGE; ALSO RU, CONTROL/F.
1	007	BEL	RINGS THE BELL. CONTROL/G.
1	010	BS	BACKSPACE; ALSO FEO, FORMAT EFFECTOR. BACKSPACES SOME MACHINES, CONTROL/H.
0	011	HT	HORIZONTAL TAB. CONTROL/I.
0	012	LF	LINE FEED OR LINE SPACE (NEW LINE); ADVANCES PAPER TO NEXT LINE, DUPLICATED BY CONTROL/J.
1	013	VT	VERTICAL TAB (VTAB). CONTROL/K.
0	014	FF	FORM FEED TO TOP OF NEXT PAGE (PAGE) CONTROL/L.
1	015	CR	CARRIAGE RETURN TO BEGINNING OF LINE. DUPLICATED BY CONTROL/M.
1	016	SO	SHIFT OUT; CHANGES RIBBON COLOR TO RED. CONTROL/N.
0	017	SI	SHIFT IN; CHANGES RIBBON COLOR TO BLACK. CONTROL/O.
1	020	DLE	DATA LINK ESCAPE. CONTROL/P (DC0).
0	021	DC1	DEVICE CONTROL 1, TURNS TRANSMITTER (READER) ON, CONTROL/Q (X ON).
0	022	DC2	DEVICE CONTROL 2, TURNS PUNCH OR AUX ON. CONTROL/R (TAPE, AUX ON).
1	023	DC3	DEVICE CONTROL 3, TURNS TRANSMITTER (READER) OFF, CONTROL/S (X OFF).
0	024	DC4	DEVICE CONTROL 4, TURNS PUNCH OR AUX OFF. CONTROL/T (AUX OFF).

EVEN PARITY BIT	7-BIT OCTAL CODE	CHARACTER	REMARKS
1	025	NAK	NEGATIVE ACKNOWLEDGE; ALSO ERR, ERROR. CONTROL/U.
1	026	SYN	SYNCHRONOUS FILE (SYNC). CONTROL/V.
0	027	ETR	END OF TRANSMISSION BLOCK; ALSO LEM, LOGICAL END OF MEDIUM. CONTROL/W.
0	030	CAN	CANCEL (CANCL). CONTROL/X.
1	031	EM	END OF MEDIUM. CONTROL/Y.
1	032	SUB	SUBSTITUTE. CONTROL/Z.
0	033	ESC	ESCAPE. PREFIX. CONTROL/SHIFT/K.
1	034	FS	FILE SEPARATOR. CONTROL/SHIFT/L.
0	035	GS	GROUP SEPARATOR. CONTROL/SHIFT/M.
0	036	RS	RECORD SEPARATOR. CONTROL/SHIFT/N.
1	037	US	UNIT SEPARATOR. CONTROL/SHIFT/O.
1	040	SP	SPACE.
0	041	!	
0	042	"	
1	043	#	
0	044	$	
1	045	%	
1	046	&	
0	047	'	APOSTROPHE.
0	050	(	
1	051	)	
1	052	*	
0	053	+	
1	054	,	
0	055	-	
0	056	.	
1	057	/	
0	060	0	
1	061	1	
1	062	2	
0	063	3	
1	064	4	
0	065	5	
0	066	6	
1	067	7	
1	070	8	
0	071	9	
0	072	:	
1	073	;	
0	074	<	
1	075	=	
1	076	>	
0	077	?	
1	100	@	
0	101	A	
0	102	B	
1	103	C	
0	104	D	
1	105	E	
1	106	F	
0	107	G	
0	110	H	
1	111	I	

EVEN PARITY BIT	7-BIT OCTAL CODE	CHARACTER	REMARKS
1	112	J	
0	113	K	
1	114	L	
0	115	M	
0	116	N	
1	117	O	
0	120	P	
1	121	Q	
1	122	R	
0	123	S	
1	124	T	
0	125	U	
0	126	V	
1	127	W	
1	130	X	
0	131	Y	
0	132	Z	
1	133	[	SHIFT/K.
0	134	\	SHIFT/L.
1	135	]	SHIFT/M.
1	136	^	appears as ^ on some machines.
0	137	←	appears as _ on some machines.
0	140	`	ACCENT, GRAVE
1	141	a	
1	142	b	
0	143	c	
1	144	d	
0	145	e	
0	146	f	
1	147	g	
1	150	h	
0	151	i	
0	152	j	
1	153	k	
0	154	l	
1	155	m	
1	156	n	
0	157	o	
1	160	p	
0	161	q	
0	162	r	
1	163	s	
0	164	t	
1	165	u	
1	166	v	
0	167	w	
0	170	x	
1	171	y	
1	172	z	
0	173	{	
1	174	|	
0	175	}	THIS CODE GENERATED BY ALTMODE
0	176	~	THIS CODE GENERATED BY ESC KEY (IF PRESENT)
1	177	DEL	DELETE, RUB OUT

NOTES

1. Teleprinters manufactured by Teletype Corporation, Skokie, Illinois, have used codes 175 (ALT) and 176 for ESC. Programs may forgo the use of (175) and (176) in order to use these codes as ESC on older teleprinters.

2. ASCII is a 7 bit character code with an optional odd parity bit (200) added for many devices. Programs normally use just seven bits internally; the 200 bit is either stripped or added so the program will operate with either parity or non-parity generating devices.

   ISO Recommendation R646 and CCITT Recommendation V.3 (International Alphabet No. 5) is identical to ASCII except that number sign (043) is represented as £ instead of # and certain characters are reserved for national use.

A.2 RADIX-50 CHARACTER SET

Character	ASCII Octal Equivalent	Radix-50 Octal Equivalent
space	40	0
A-Z	101-132	1-32
$	44	33
:	56	34
unused		35
0-9	60-71	36-47

The maximum Radix-50 Octal value:

47*50 +47*50+47=174777

Table A-1 provides a convenient means of translating between the ASCII character set and its Radix-50 equivalents. For example, given the ASCII string X2B, the Radix-50 equivalent follows (arithmetic is performed in octal).

```
X=113000
2=002400
B=000002
X2B=115402
```

Table A-1
Radix-50 Character Set

Single Char. or First Char.		Second Character		Third Character	
A	003100	A	000050	A	000001
B	006200	B	000120	B	000002
C	011300	C	000170	C	000003
D	014400	D	000240	D	000004
E	017500	E	000310	E	000005
F	022600	F	000360	F	000006
G	025700	G	000430	G	000007
H	031000	H	000500	H	000010
I	034100	I	000550	I	000011
J	037200	J	000620	J	000012
K	042300	K	000670	K	000013
L	045400	L	000740	L	000014
M	050500	M	001010	M	000015
N	053600	N	001060	N	000016
O	056700	O	001130	O	000017
P	062000	P	001200	P	000020
Q	065100	Q	001250	Q	000021
R	070200	R	001320	R	000022
S	073300	S	001370	S	000023
T	076400	T	001440	T	000024
U	101500	U	001510	U	000025
V	104600	V	001560	V	000026
W	107700	W	001630	W	000027
X	113000	X	001700	X	000030
Y	116100	Y	001750	Y	000031
Z	121200	Z	002020	Z	000032
$	124300	$	002070	$	000033
.	127400	.	002140	.	000034
unused	132500	unused	002210	unused	000035
0	135600	0	002260	0	000036
1	140700	1	002330	1	000037
2	144000	2	002400	2	000040
3	147100	3	002450	3	000041
4	152200	4	002520	4	000042
5	155300	5	002570	5	000043
6	160400	6	002640	6	000044
7	163500	7	002710	7	000045
8	166600	8	002760	8	000046
9	171700	9	003030	9	000047

APPENDIX B

FILENAME EXTENSIONS

Extension	Attribute
.DAT	FORTRAN data file
.FOR	FORTRAN source file
.LDA	Absolute binary file
.LLD	Library listing file
.LST	FORTRAN or MACRO listing file
.MAC	MACRO source file
.MAP	Link load map file
.OBJ	Relocatable binary file; library file
.SAV	Executable FORTRAN program file
.STB	Symbol Definition file
.TMP	CREF temporary cross-reference file

APPENDIX C

OPTION SUMMARY

C.1 MACRO OPTIONS

The options recognized by MACRO are listed below. The /P:n option has two possible arguments; 1 and 2. The /M option has no arguments. Sections C.1.1, C.1.2 and C.1.3 list the arguments for the remaining MACRO options.

Option	Usage
/L:arg	Listing control, overrides source program directive .LIST
/N:arg	Listing control, overrides source program directive .NLIST
/E:arg	Object file function enabling, overrides source program directive .ENABL
/D:arg	Object file function disabling, overrides source program directive .DSABL
/M	Indicates input file is MACRO library file
/C:arg	Control contents of cross-reference listing
/P:arg	Specifies whether input source file is to be assembled during pass 1 or pass 2

C.1.1 Arguments for Listing Control Options

Valid arguments for the listing control options (/L:arg or /N:arg) are given below.

Argument	Default	Controls Listing of
SEQ	list	Source line sequence numbers
LOC	list	Location counter
BIN	list	Generated binary code
BEX	list	Binary extensions
SRC	list	Source code
COM	list	Comments
MD	list	Macro expansions and repeat range expansions
MC	list	Macro calls and repeat range expansions
ME	nolist	Macro expansions

Argument	Default	Controls Listing of
MEB	nolist	Macro expansion binary code
CND	list	Unsatisfied conditions and all .IF and .ENDC statements.
LD	nolist	Listing directives having no arguments
TOC	list	Table of contents
TTM	nolist	132 column line printer format when not specified; terminal mode when specified
SYM	list	Symbol table

The /N option with no argument causes MACRO to list only the symbol table, table of contents and error messages. The /L option with no arguments causes MACRO to ignore .LIST and .NLIST directives in the source code that have no arguments.

C.1.2 Arguments for Function Control Options

This section lists valid arguments for the MACRO function control options, /E:arg (enable) or /D:arg (disable).

Argument	Default Mode	Enables or Disables
ABS	Disable	Absolute binary output
AMA	Disable	Assembly of all absolute addresses as relative addresses
CDR	Disable	Source columns 73 and greater to be treated as comments
CRF	Enable	Cross-reference listing. /D:CRF inhibits CREF output even if /C is specified.
FPT	Disable	Floating point truncation
GBL	Disable	Undefined symbols treated as globals
LC	Disable	Accepts lower case ASCII input
LSB	Disable	Local symbol block
PNC	Enable	Binary output
REG	Enable	Mnemonic definitions of registers

C.1.3 Arguments for the Cross-Reference Option

This section lists valid arguments for the cross-reference option, /C:arg. These arguments control the contents of cross-reference listings.

Argument	Produces Cross-Reference of
S	User-defined symbols
R	Register symbols
M	MACRO symbolic names
P	Permanent symbols
C	Control sections
E	Error codes
<no arg>	Equivalent to /C:S:M:E

C.2 LINK OPTIONS

This section describes the options recognized by the LINK utility. The command line on which each option must appear is listed.

Option Name	Command Line	Explanation
/A	first	Alphabetizes the entries in the load map.
/B:n	first	Changes the bottom address of a program to n.
/C	any but last	Continues input specification on another command line (you can use /C also with /O; do not use /C with the // option).
/E:n	first	Extends a particular program section to a specific value.
/F	first	Instructs the linker to use the default FORTRAN library, FORLIB.OBJ, to resolve any undefined global references. Note that this option should not be specified in the command line when FORLIB has been incorporated into SYSLIB.
/H:n	first	Specifies the top (highest) address to be used by the relocatable code in the load module.
/I	first	Extracts the global symbols you specify from the library and links them into the load module.
/K:n	first	Inserts the value you specify (the valid range for n is from 1 to 28) into word 56 of block 0 of the image file. This option indicates that the program requires nk words of memory.
/L	first	Produces a formatted binary output file (.LDA format).
/M or /M:n	first	Cause the linker to prompt you for a global symbol that represents the stack address, or sets the stack address to the value n.
/O:n	any, but the first	Indicates that the program is an overlay structure; n specifies the overlay region to which the module is assigned.
/P:n	first	Changes the default amount of space the linker uses for a library routines list.

Option Name	Command Line	Explanation
/S	first	Makes the maximum amount of space in memory available for the linker's symbol table. (Use this option only when a particular link stream causes a symbol table overflow)
/T or /T:n	first	Causes the linker to prompt you for a global symbol that represents the transfer address, or sets the transfer address to the value n.
/U:n	first	Rounds up the section you specify so that the size of the root segment is a whole number multiple of the value you supply (n must be a power of 2).
/W	first	Directs the linker to produce a wide load map listing.
/X		Does not output the bitmap if the code is below 400.
/Z:n	first	Sets unused locations in the load module to the value n.
//	first and last	Allows you to specify command string input on additional lines. Do not use this option with /C.

C.3 LIBR OPTIONS

This section lists options recognized by the librarian utility program, LIBR. Note that there is no option to indicate module insertion. Modules are automatically inserted into the library file if you do not specify an option.

Option	Command Line	Meaning
/C	any but last	Command continuation; allows you to type the input specification on more than one line.
/D	first	Delete; deletes modules that you specify from a library file.
/E	first	Extract; extracts a module from a library and stores it in an OBJ file.
/G	first	Global deletion; deletes global symbols that you specify from the library directory.
/N	first	Names; includes the module names in the directory.
/P	first	P-section names; includes the program section names in the directory.

Option	Command Line	Meaning
/R	first	Replace; replaces modules in a library file.
/U	first	Update; inserts and replaces modules in a library file.
/W	first	Indicates wide format for the listing file.
//	first and last	Command continuation; allows you to type the input specification on more than one line.

C.4 PATCH OPTIONS

The following options are recognized by the PATCH utility program:

Option	Meaning
/O	Use if the file is an overlay-structured file.
/C	Requires you to enter a checksum. If you make no modifications, PATCH ignores the /C option.
/D	Use if you do not know the checksum for a particular patch. PATCH prints the checksum for that patch. If you make no modifications, PATCH ignores the /D option.

APPENDIX D

ERROR MESSAGE SUMMARY

D.1 MACRO ERROR CODES AND ERROR MESSAGES

MACRO outputs two types of error indications; a single-letter error code and a descriptive error message.

The single-letter error codes automatically appear on assembly listings. They also appear on the cross-reference listing if you specify /C:E in the MACRO command string.

Descriptive error messages are output to the terminal. They are of one of the following formats:

```
?MACRO-F-message
     or
?CREF-F-message
```

Following the appearance of a message of either format, control returns to MACRO, which prompts you with an asterisk. You can then enter a new command string.

D.1.1 MACRO Error Codes

Error Code	Meaning
A	Addressing or relocation error. This occurs when an instruction operand has an invalid address, or when the definition of a local symbol occurs more than 128 words from the beginning of a local symbol block.
B	Boundary error. The current setting of the location counter would cause the assembly of instruction or word data at an odd memory address. The system increments the location counter by 1 to correct this.
D	Reference to multiple-definition symbol. The program refers to a non-local label that is defined more than once.
E	No END directive. The assembler has reached the end of a source file and found no END directive. The system generates .END and continues.
I	Illegal character detected. The assembler has encountered in the source file a character that is not included in the language character set. The system replaces each illegal character with a ? on the assembly listing and proceeds as if the illegal character were not present.

Error Code	Meaning
L	Link buffer overflow. The assembler has encountered an input line greater than 132 characters. In terminal mode the system ignores additional characters.
M	Multiple definition of a label. The source program is attempting to define a label equivalent in the first six characters to a label defined previously.
N	Decimal point missing from decimal number. A number containing the digit 8 or 9 lacks a decimal point.
O	Op-code error. A directive appears in an inappropriate context.
P	Phase error. The definition or value of a label differs from one pass to another, or a local symbol occurs more than once in a local symbol block.
Q	Questionable syntax. This can have any of several causes, as follows: 1. There are missing arguments. 2. The instruction scan is not complete. 3. A line feed or form feed does not immediately follow a carriage return.
R	Register-type error. The source program attempts an invalid reference to a register.
T	Truncation error. A number generates more than 16 significant bits, or an expression generates more than 8 significant bits while a .BYTE directive is active.
U	Undefined symbol. A symbol not defined elsewhere in the program appears as a factor in an expression. The assembler assigns the undefined symbol a constant zero value.
Z	Incompatible instruction (warning). The instruction is not defined for all PDP-11 hardware configurations.

D.1.2 MACRO Error Messages

Message	Explanation
?CREF-F-Chain-only CUSP	Programs must chain to CREF in order to use it. Attempts to use RUN $CREF can cause this error. Use a language processor to invoke CREF.
?CREF-F-CRF file error	An input error occurred while reading DK:CREF.TMP, the temporary input file passed to CREF. Run the language processor again to create a good CREF input file.

Message	Explanation
?CREF-F-Device	The language processor chaining to CREF has specified an invalid device. This may be a system error. However, writing a CREF listing to magtape or cassette before manually loading the magtape or cassette handler causes this error. The error also occurss when the input file to CREF, CREF.TMP, is not on a random access device. If the error persists, submit a Software Performance Report with a program listing and a machine readable source program, if possible.
?CREF-F-List file error	An output error occurred while attempting to write the cross-reference table to the listing file. The output volume may not have enough free space remaining for the listing file.
?MACRO-F-Bad option	The specified option was not recognized by the program. Check for a typing error in your command line.
?MACRO-F-Device full	The output volume does not have sufficient room for an output file specified in the command string. Delete unnecessary files or use another device.
?MACRO-F-File not found	An input file specified in the command line does not exist on the specified device, or was protected against the current user. 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 6 input files. Correct the command line and retype.
?MACRO-F-Illegal device	A device specified in the command line does not exist on the system. Either install the device or substitute another.
?MACRO-F-Input-output error on channel N	A hardware error occurred while attempting to read from or write to the device on the channel specified in the message. Channels correspond to files in the command string as follows:

Message	Explanation
?MACRO-F-Input-output error on channel N (cont.)	

Channel	File
0	.OBJ output file
1	.LST output file
2	CREF temporary file
3	Input (source) file #1
4	Input (source) file #2
5	Input (source) file #3
6	Input (source) file #4
7	Input (source) file #5
8	Input (source) file #6

?MACRO-F-Input-output error on MACRO library	MACRO detected a bad record in the MACRO library. For example, this error occurs when the library area is bad. Rebuild the MACRO library.
?MACRO-F-Input-output error on workfile	MACRO failed to read or write to its workfile, WRK.TMP. Check for hard error conditions such as read or write-locked, or offline devices.
?MACRO-F-Insufficient memory	There were too many symbols in the program being assembled.
?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 file.
?MACRO-F-Output device full	There was no room to continue writing the output file.

D.2 LINK ERROR MESSAGES

The following error messages can be output by the linker. Messages appear on the load map if you requested a load map. Otherwise, error messages are output to the terminal.

All messages are of the form:

```
?LINK-n-message
```

where n represents the severity code of the error. Severity codes can be F (Fatal) or W (Warning). Fatal errors cause the current command or statement to be ignored. You must enter another command. A warning message indicates an error condition that may affect execution at a later time. The condition causing the message may require some attention.

Message	Explanation
?LINK-F-/B No value	No argument was specified to the /B option. Reenter the command string specifying an unsigned even octal number as the argument to the /B option.
?LINK-F-/B Odd value	The argument to the /B option was not an unsigned even octal number. Reenter the command string specifying an unsigned even octal number as the argument to /B.
?LINK-F-/H Value too low	The value specified as the high address for linking was actually too small to accommodate the code. Obtain map output without using /H to determine the space required and then retry the operation.
?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 option.
?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 option.
?LINK-F-/U or /Y value not a power of 2	The value specified with /U is not a power of 2. Reenter the command with a value that is a power of 2.
?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.

Message	Explanation
?LINK-F-Bad complex relocation in FILNAM	A complex relocation string in the input file was found to be invalid. The message occurs during pass 2 of the linker. Check for a typing error in the command line; verify that the correct filenames were specified as input. Reassemble or recompile to obtain a good object module and retry the operation. If the error persists, verify that the source code is correct.
?LINK-F-Bad GSD in FILNAM	There was an error in the global symbol directory (GSD). The file is probably not a legal object module. Verify that the correct filenames were specified as input; check for a typing error in the command line. Reassemble or recompile the source to obtain a good object module and retry the operation.
?LINK-F-Bad RLD in FILNAM	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 filenames were specified as input. Reassemble or recompile and retry the operation. If the error persists, verify that the source code is correct.
?LINK-F-Bad RLD symbol in DEV:FILNAM.TYP	A global symbol named in a relocatable record was not defined in the global symbol definition record. Reassemble the indicated file. If the condition persists, submit a Software Performance Report (SPR).
?LINK-F-Default system library not found SYSLIB.OBJ	The linker did not find SYSLIB.OBJ on the system device when undefined globals existed. Obtain a copy from your backup system volume and relink your program, or correct the source files by removing the undefined globals listed on the terminal.

Message	Explanation
?LINK-F-File not found DEV:FILNAM.TYP	The input file indicated was not found. Check for a typing error in the command line. Verify that the filename exists as entered in the command line and retry the operation.
?LINK-F-Illegal character	The character specified was not used in proper context. Characters for symbols must be legal Radix-50 characters. Examine the command string for errors in syntax. Correct and retype.
?LINK-F-Illegal device	The device/volume indicated was not available. Verify that the device is valid for the system in use.
?LINK-F-Illegal error	An internal error occurred while the linker was in the process of recovering from a previous system or user error. Retry the operations that produced this error; if it recurs, report the error to DIGITAL using an SPR (Software Performance Report); include a program listing and a machine-readable source program, if possible.
?LINK-F-Illegal record type in DEV:FILNAM.TYP	A formatted binary record had a type not in the range 1-10 (octal). Verify that the correct filenames were specified as input; check for a typing error in the command line. Reassemble or recompile and retry the operation.
?LINK-F-Insufficient memory	There was not enough memory to accommodate the command, the symbol table or the resultant load module.
?LINK-F-Map device full	There was no room in the directory for the filename or there was no room on the output device for the map file.
?LINK-F-Old library format in DEV:FILNAM.TYP	The indicated library file is formatted from an old LIBR version. Rebuild the library file using the current librarian.

Message	Explanation
?LINK-F-Read error in DEV:FILNAM.TYP	A hardware error occurred while reading the indicated input file. Check for read-locked or off-line devices.
?LINK-F-SAV device full	There was no room in the directory for the filename or there was no room on the output device for the image file.
?LINK-F-SAV read error	A hardware error occurred while reading the image file (SAV, LDA). Check for read-locked or off-line devices.
?LINK-F-SAV write error	A hardware error occurred while writing the image file (LDA). Check for write-locked or off-line devices.
?LINK-F-STB device full	There was no room in the directory for the filename or there was no room on the output device for the symbol table (STB) file.
?LINK-F-STB not allowed with /S and a MAP	Production of STB and MAP in the same linking operation is prohibited in order to maximize space in the symbol table with /S. Produce STB and MAP in separate linking operations.
?LINK-F-STB write error	A hardware error occurred while writing the symbol table (STB) file. Check for write-locked or off-line devices.
?LINK-F-Storing text beyond high limit	An input object module has 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 and/or recompile the program.
?LINK-F-Symbol table overflow	Too many global symbols were used in the program.
?LINK-W-/O Ignored	Overlays were specified in the wrong order. Check for a typing error in the command line. The overlay option is ignored. Consult the overlay restrictions in Chapter 3.

Message	Explanation
?LINK-W-Additive reference of NNNNNN at segment # MMMMMM	A call or a branch 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.
?LINK-W-Bad option: /a	The linker did not recognize the option (/a) specified in the command line, or an illegal combination of options was used. If the bad option occurred in the first command line, control returns to LINK; enter another command. If the bad option occurred on a subsequent command line, the option is ignored and processing continues. In a continued command line, only /O, /C, and // are legal options. Reexamine the command line and check for a typing error.
?LINK-W-Bad overlay at segment # NNNNNN	An overlay tried to store text outside its region; NNNNNN represents the segment number. Check for an .ASECT in the overlay.
?LINK-W-Byte relocation error at NNNNNN	The linker attempted to relocate and link byte quantities, but failed. NNNNNN represents the address at which the error occurred. Failure is defined as the high byte of the relocated value (or the linked value) not being all zeroes. The relocated value is truncated to 8 bits and the linker continues processing. Correct the source program so that there are no relocated byte quantities, reassemble, and relink.
?LINK-W-Complex relocation divide by 0 in DEV.FILNAM.TYP	A divide by 0 was attempted in a complex relocation string in the file indicated. A result of 0 is returned and linking continues.

Message	Explanation
?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. The source program should be checked to use the desired section attributes for that program section.
?LINK-W-Extend section not found	The extend section name given with /E was not found in the modules that were linked; or the extend section does not exist in the root segment. The linker continues after the warning, without extending the section. Check the response to the "Extend section?" prompt, and use the correct section name the next time you link.
?LINK-W-Map write error	A hardware error occurred while writing the map output file. 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.
?LINK-W-Round section not found AAAAAA	The round program section was not found in the symbol table to match the symbol entered (following use of the /U option). Linking continues with no round-up action.
?LINK-W-Stack address undefined or in overlay	The stack address specified by the /M option was either undefined or in an overlay. For SAV files, 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.

Message	Explanation
?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. The response to the /T option must be either a colon followed by an unsigned 6-digit octal number, or 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. 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 present in the libraries specified.

D.3 LIBRARY ERROR MESSAGES

The following error messages are output on the terminal by the librarian program. All messages are of the form:

?LIBR-n-message

where n represents the severity code of the error. Severity codes can be F (Fatal) or W (Warning). Fatal errors cause the current command or statement to be ignored. You must enter another command. A warning message indicates an error condition that may affect execution at a later time. The condition causing the message may require some attention.

Message	Explanation
?LIBR-F-Bad GSD in FILNAM	There was an error in the global symbol directory (GSD). The file is probably not a legal object module.
?LIBR-F-Bad library for listing or extract	The input file specified for extraction or to produce a directory listing was not an object library file. Verify the filename in the command line and check for typing errors. A valid object library file is required for extraction or to produce a directory listing.

Message	Explanation
?LIBR-F-Bad option: /x	The librarian did not recognize the given option (/x). The librarian restarts and prompts with an asterisk.
?LIBR-F-EOF during extract	The end of the input file was reached before the end of the module being extracted. This is an unusual internal consistency-check error. 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.
?LIBR-F-File not found FILNAM	One of the input files indicated in the command line was not found. LIBR prints an asterisk; the command may be reentered.
?LIBR-F-Illegal error	An internal error occurred while the librarian was in the process of recovering from a previous system or user error. Retry the operations that produced this error; if it recurs, report the error to DIGITAL using an SPR (Software Performance Report); include a program listing and a machine-readable source program, if possible.
?LIBR-F-Illegal extract of AAAAAA	An extraction of the identified global symbol was attempted but the symbol was not found in the library.
?LIBR-F-Illegal option combination	Options 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 options (/C, //) may follow.
?LIBR-F-Illegal record type in FILNAM	A formatted binary record had a type not in the range 1-10 (octal). Verify that the correct filenames were specified as input; check for a typing error in the command line. Reassemble or recompile the source and retry the operation.

Message	Explanation
?LIBR-F-Illegal replace of library file FILNAM	The command line specified that a library file be replaced by another library file. Check for a typing error in the command line. Only object modules can be replaced in a library file. Enter another command.
?LIBR-F-Insufficient memory	Available free memory has been used up. The current command is aborted.
?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 option /M: The default is 128 names.
?LIBR-F-No value allowed: /a	The specified option (/a) does not take a value. The librarian restarts and prompts with an asterisk.
?LIBR-F-Output and input filnam the same	The same filename was specified for both input and output files when the command string to build the macro library was specified. Use different filenames for the input and output files specified to build a macro library.
?LIBR-F-Output device full	The device was full; LIBR was unable to create or update the indicated library file.
?LIBR-F-Output file full	The output file was not large enough to hold the library file or list file.
?LIBR-F-Output write error	An unrecoverable error occurred while processing an output file. This may indicate that there was not enough space left on a device to create a file, although there may have been enough directory entries left.
?LIBR-F-Read error in FILNAM	An unrecoverable error has occurred while processing an input file. LIBR prints an asterisk and waits for another command to be entered.

Message	Explanation
?LIBR-W-Duplicate module name of AAAAAA	A new module has been 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 option is needed; the option for update is /U and the option for replacement is /R.
?LIBR-W-Illegal character	The symbol name entered contained a non-Radix-50 character. Retype the command line 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. Check for a typing error in the command line. The entry point name or module name is ignored and processing continues.
?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.
?LIBR-W-Null library	An attempt was made to build a library file containing no directory entries. Verify that the correct filenames 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.

D.4 PATCH ERROR MESSAGES

The following error messages can be output by the PATCH program. All messages are of the form:

```
?PATCH-n-message
```

where n represents the severity code of the error. Severity codes can be F (Fatal), I (Information) or W (Warning). Fatal errors cause the current command or statement to be ignored. You must enter another command. An informational message requires no further action. It is there for your benefit only. A warning message indicates an error condition that may affect execution at a later time. The condition causing the message may require some attention.

Message	Explanation
?PATCH-F-Insufficient memory	There was not enough free core to contain the device handler and the internal "overlay tables." This message should not occur under normal circumstances.
?PATCH-F-Read error	PATCH detected an input error in reading from the file. Check for read-locked or off-line devices.
?PATCH-F-Write error	PATCH detected an input error in writing to the file. Check for write-locked or off-line devices.
?PATCH-I-[+2K core]	The USR is swapping or PATCH needs more memory for overlay handling. PATCH continues executing normally. This message is for your information.
?PATCH-I-CHECKSUM=NNNNNN	PATCH prints out the checksum in response to the /D option after an "E" or "F" command has been issued. This message is for your information only.
?PATCH-W-Address not in segment	The specified address exceeds the limits of the particular overlay. Recheck the linker load map for the address and proper overlay segment.
?PATCH-W-Bottom address wrong	The contents of the address specified does not correspond to the first word in the standard RSTS/E overlay handler. Correct the line in error; specify the correct address using the x;B command.

Message	Explanation
?PATCH-W-CHECKSUM error	PATCH responds to an incorrectly entered checksum three times. A failure to enter the correct checksum on the third attempt will cause an automatic exit to the monitor. The file being patched has been changed. The incorrectly patched file should be deleted and the backup procedures repeated before attempting to patch the file a second time.
?PATCH-W-Illegal command	The response to the message "FILE NAME --" was incorrect. Check for a typing error in the command line. The file specification must be of the form: dev:filnam.ext/options
?PATCH-W-Illegal option	One of the options encountered in the entered file specification was not a recognized legal option.
?PATCH-W-Invalid overlay handler modification	An attempt was made to insert a zero value into the overlay handler tables for an overlaid program. A non-zero value must be given in conjunction with the ";O" command.
?PATCH-W-Invalid relocation register	An attempt was made to reference a relocation register outside the range 0-7. Relocation registers must be set within the range 0-7.
?PATCH-W-Invalid segment number	The specified segment number did not exist in the file being patched. Recheck the linker load map and command string to determine the overlay structure.
?PATCH-W-Must open word	The "@", "P," or "X" command was typed when no address was open.
?PATCH-W-Must specify segment number	The specified address exceeds the limits of the root segment.
?PATCH-W-No address open	The "LF", "^", "@", "X", "P", "C", or "A" command character was typed when no address was open. Check for a typing error in the command line.

Message	Explanation
?PATCH-W-Not in program bounds	An attempt was made to reference a location outside the limit defined by location 50 in block zero of the file. The value of the initial stack pointer for the program may also be beyond the last location of the program. Check for a typing error in the command line. Check the linker load map to determine where the program was loaded. Check the initial value of the stack pointer.
?PATCH-W-Odd address	An attempt was made to open an odd address as a word with the "/" command. Word addresses must be even numbers. Use "\" to open an odd address.
?PATCH-W-Odd bottom address	The bottom address specified or contained in location 42 of an overlay file was odd. The overlay handler must start on an even word boundary.
?PATCH-W-Program has no segments	An attempt was made to reference an overlay region in a program which was not identified as an overlaid program in the file specification, or an attempt was made to reference an overlay region in a program which has none.

D.5 PAT ERROR MESSAGES

The following messages can be output by PAT. All messages are of the form:

```
PAT-n-message
```

where n represents the severity code of the error causing the message. Possible severity codes are F (Fatal), I (Information) or W (Warning). Fatal errors cause the current command or statement to be ignored. You must enter another command. Information messages are for your benefit only. They require no further action. A warning message indicates an error condition that may affect execution at a later time. The condition causing the message may require some attention.

Message	Explanation
?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. Verify that the input filename is correct; check for a typing error in the command line. Reassemble or recompile the source to obtain a good object module and retry the operation.
?PAT-F-Correction file has bad RLD	A global symbol named in a relocatable record was not defined in the global symbol definition record. Reassemble the indicated file. If the condition persists, submit a Software Performance Report.
?PAT-F-Correction file has illegal record	The format of the correction file is not compatible with the object file format PAT requires. The format is not what the standard language processors should produce. Verify that the correction file has the proper format, and retype the command line.
?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 to PAT.
?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 read-locked or off-line devices.
?PAT-F-Illegal error	PAT has generated an illegal error message call. This is an internal software error condition. If the error persists, submit a Software Performance Report with the related console dialogue and any other pertinent information.

Message	Explanation
?PAT-F-Incompatible reference of 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. Choose RELOCATABLE or ABSOLUTE; and specify .PSECT or .CSECT. 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 filename is correct; check for a typing error in the command line. Reassemble or recompile the source and retry the operation.
?PAT-F-Input file has bad RLD	A global symbol named in a relocatable record was not defined in the global symbol definition record. Reassemble the indicated file. If the error persists, submit a Software Performance Report.
?PAT-F-Input file has illegal record	The format of the input file is not compatible with the object file format PAT requires. The format is not what the standard DIGITAL language processors should produce. Verify that the input file has the proper format, and retype the command line.
?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 to PAT.

Message	Explanation
?PAT-F-Input file read error	PAT detected an error while reading the input file. Input hardware can cause this error. Retry the command. If the error persists, submit a Software Performance Report with a copy of the console dialogue and any other pertinent information.
?PAT-F-Insufficient memory	There is not sufficient contiguous memory for PAT to use for the corrected output file.
?PAT-F-Only /C allowed	The input module or correction file specifications contain an illegal option. /C and /C:n are the only option forms PAT accepts.
?PAT-F-Output file full	There is not enough free space on the output volume for the corrected object file.
?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.
?PAT-F-Unable to locate module AAAAAA	The correction file has a module name that does not exist in the input file. PAT shows the name of the module in this message. 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 specifies more than two input files. PAT processes the first as the input module to be corrected and the second as the correction file. PAT ignores all other files.
?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. For the general command line format out1,out2,out3=input,correct PAT's output file must be in the "out1" position. PAT ignores all other output files.

Message	Explanation
?PAT-W-Correction file checksum error	PAT finds a checksum value that is different from the value for the /C correction file option. Mistyping the /C option value or specifying an invalid version of the correction file causes this error.
?PAT-W-Correction file check is NNNNNN	PAT responds to the /C option on the correction file with this message. NNNNNN is the octal value of the sum of all binary data composing the file. The message is for your information.
?PAT-W-Input file checksum error	PAT finds a checksum value that is different from the value for the /C input file option. Mistyping the /C option value or specifying an invalid version of the input file causes this warning.
?PAT-W-Input module checksum is NNNNNN	PAT responds to the /C option on the input module with this message. NNNNNN is the octal value of the sum of all binary data composing the file. The message is for your information.

RSTS/E FORTRAN IV
Utilities Manual
AA-2140B-TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

_____
_____
_____
_____
_____
_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____
_____
_____
_____
_____
_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____
_____
_____
_____
_____
_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Please cut along this line.

—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