February 1979

This document contains information that an assembly language programmer needs to use the capabilities of the VAX-11 MACRO assembly language efficiently.

VAX-11
MACRO User's Guide

Order No. AA-D033B-TE

SUPERSESSION/UPDATE INFORMATION:	This revised document supersedes the VAX-11 MACRO User's Guide (Order No. AA-D033A-TE)
OPERATING SYSTEM AND VERSION:	VAX/VMS V1.5
SOFTWARE VERSION:	VAX-11 MACRO V2.0

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, August 1978
Revised, February 1979

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
VAX	VMS	SBI
DECnet	IAS	PDT
DATATRIEVE	TRAX	

CONTENTS

FIGURES

CONTENTS

PREFACE

MANUAL OBJECTIVES

This manual describes how to use the VAX-11 MACRO assembly language. It is designed to enable users to assemble programs coded in VAX-11 MACRO. The features of the VAX-11 MACRO language are described in the VAX-11 MACRO Language Reference Manual.

INTENDED AUDIENCE

This manual is intended for all VAX-11 MACRO programmers. This manual assumes that the reader has had some assembly language programming experience and has read the VAX/VMS Primer. Chapter 3 of this guide is intended for experienced MACRO programmers who want to create shareable images.

STRUCTURE OF THIS DOCUMENT

This manual is organized into three chapters and one appendix, as follows:

- Chapter 1 provides an introduction to VAX-11 MACRO assembler for users who are not familiar with the operation of an assembler.
- Chapter 2 describes the MACRO command, which invokes the VAX-11 MACRO assembler.
- Chapter 3 describes how to write position-independent code for use in shareable images.
- Appendix A describes the VAX-11 MACRO error messages.

ASSOCIATED DOCUMENTS

The following documents are relevant to VAX-11 MACRO programming:

- VAX-11 MACRO Language Reference Manual
- VAX-11/780 Architecture Handbook
- VAX/VMS Primer
- VAX/VMS Command Language User's Guide
- VAX-11 Linker Reference Manual
- VAX-11 Symbolic Debugger Reference Manual

- VAX/VMS System Services Reference Manual
- VAX/VMS I/O User's Guide

For a complete list of all VAX-11 documents, including a brief description of each, see the VAX-11 Information Directory.

CONVENTIONS USED IN THIS DOCUMENT

The following conventions are observed in this guide, as in other VAX-11 documents:

- Brackets ([]) indicate that the enclosed argument is optional.
- Uppercase words and letters, used in formats, indicate that you should type the word or letter exactly as shown.
- Lowercase words and letters, used in formats, indicate that you are to substitute a word or value of your choice.
- Ellipses (...) indicate that the preceding item(s) can be repeated one or more times.

SUMMARY OF TECHNICAL CHANGES

This manual documents VAX-11 MACRO V2.0. This section summarizes the technical changes in the use of the assembler from Version 1.0. Technical changes in the VAX-11 MACRO language are documented in the VAX-11 MACRO Language Reference Manual.

The /CROSS and /NOCROSS qualifiers have been added to the MACRO command to control the cross-reference listing.

The SUPPRESSION function has been added to the /ENABLE and /DISABLE qualifiers to the MACRO command to allow suppression of the listing of unreferenced symbols in the symbol table.

You no longer need to specify macro library files in the MACRO command before the source files. In fact, macro library files should generally be specified after the source files. However, any MACRO command that follows the description in the Version 1.0 documentation will still work.

The formats of the listing file and of the diagnostic messages have been changed.

The assembler itself now runs in native mode rather than in compatibility mode. This does not change the object code that the assembler produces, which has always been native mode code, but is only an internal assembler change.

CHAPTER 1

INTRODUCTION

The VAX-11 MACRO language consists of the VAX-11/780 native mode instruction set and the assembler directives. The instruction set allows you to perform many types of data manipulation, such as add, compare, increment, move, and complement. The instructions are described in the VAX-11/780 Architecture Handbook. The assembler directives create and initialize data areas and provide tools for using the instruction set more effectively. The directives are described in the VAX-11 MACRO Language Reference Manual. This chapter provides an introduction to the assembler.

1.1 DEVELOPING A VAX-11 MACRO PROGRAM

You write a VAX-11 MACRO program as a sequence of assembly language statements in the following format:

```
label:    operator operand(s)    ; comments
```

The operator and operand are either 1) instructions selected from the VAX-11/780 instruction set and data needed by the instructions or 2) assembler directives (instructions to the assembler to guide the assembly process). The statement label, which is optional, identifies the statement line so that you can refer to the instructions or data on that line from other parts of the program. Comments, which are optional, explain what operations are being performed.

You use tabs and spaces to make the program more readable. When you have finished creating the program as a complete, edited file, you submit it as input to the VAX-11 MACRO language assembler. The VAX-11 MACRO language assembler processes (assembles) the language statements, converting them to an internal machine language code (object code). The object code is next processed by the linker, which combines your program units, making the program suitable for execution. Figure 1-1 illustrates the development of an executable VAX-11 MACRO program.

To develop a VAX-11 MACRO program you must follow the four steps illustrated in Figure 1-1. The commands associated with these steps are:

```
$ EDIT
$ MACRO
$ LINK
$ RUN
```

Chapter 2 of this manual describes the MACRO command. The VAX/VMS Command Language User's Guide describes the EDIT, LINK, and RUN commands.

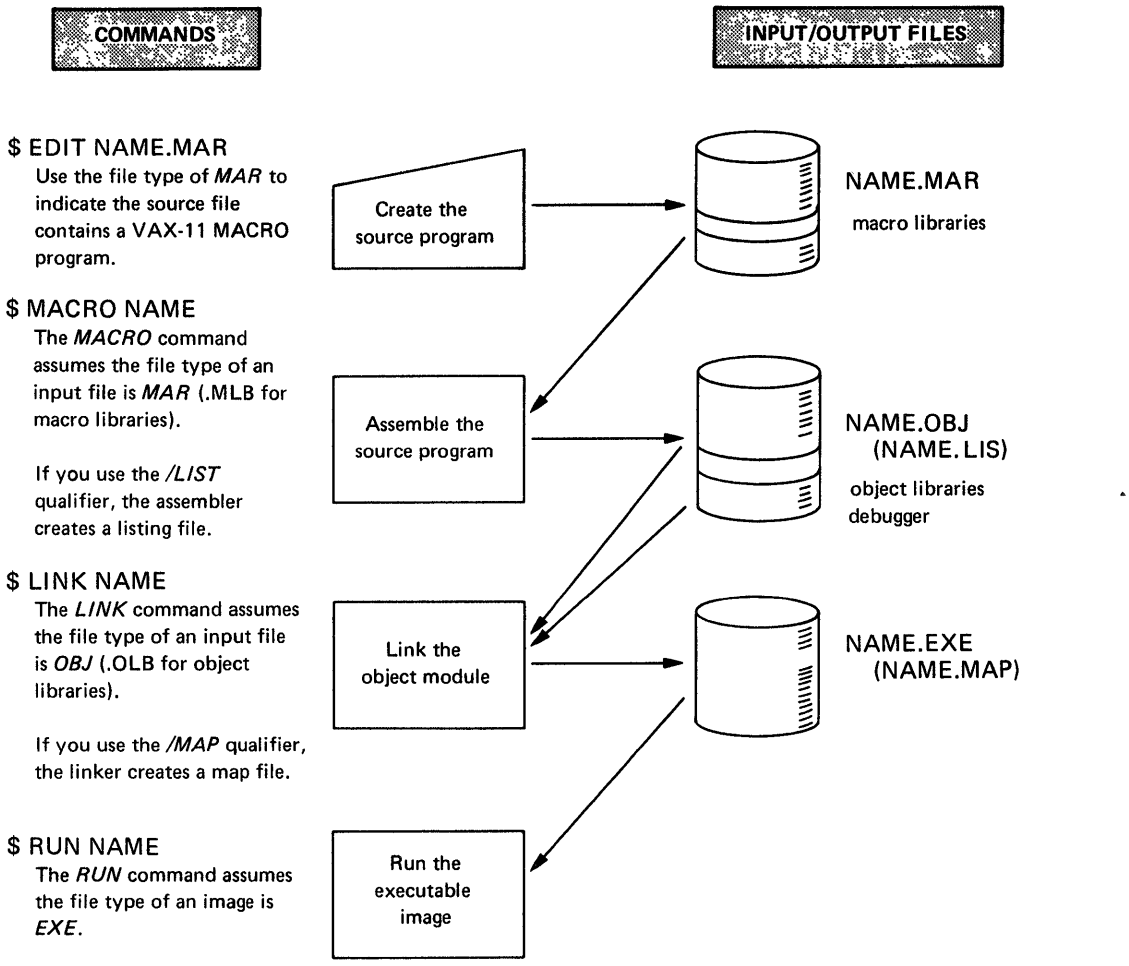

Figure 1-1 Developing a VAX-11 MACRO Program

VAX-11 MACRO allows you to use a modular approach to your program. You can create an entire program as a series of smaller independent subprograms or modules. Each module consists of a number of routines. A routine is a sequence of code that performs one procedure.

Modular programming facilitates program creation, debugging, maintenance, and enhancement as follows:

- You can write and test each routine independently of other routines. Then you can test the module consisting of these routines independently of other modules.
- Different programmers can develop and maintain different modules.
- Changing a program requires changing and testing only the module in which the change occurs.

VAX-11 MACRO assembles each module separately. Then the linker joins them all into a complete program.

1.2 VAX-11 MACRO ASSEMBLER

The VAX-11 MACRO assembler accepts information in one format (that is, your source program) and translates it into another format (that is, an object module). The assembler interprets and processes the assembly language statements, one at a time, and generates one or more computer instructions or data items. Because you originally use the editor to create a VAX-11 MACRO program in ASCII format, your program must be translated into a machine format that the computer can use. The VAX-11 MACRO assembler performs this translation, producing as output a new version of the program in object format, called an object module. You can request the VAX-11 MACRO assembler to produce a listing of the source program at the same time. Figure 1-2 illustrates the role of the assembler.

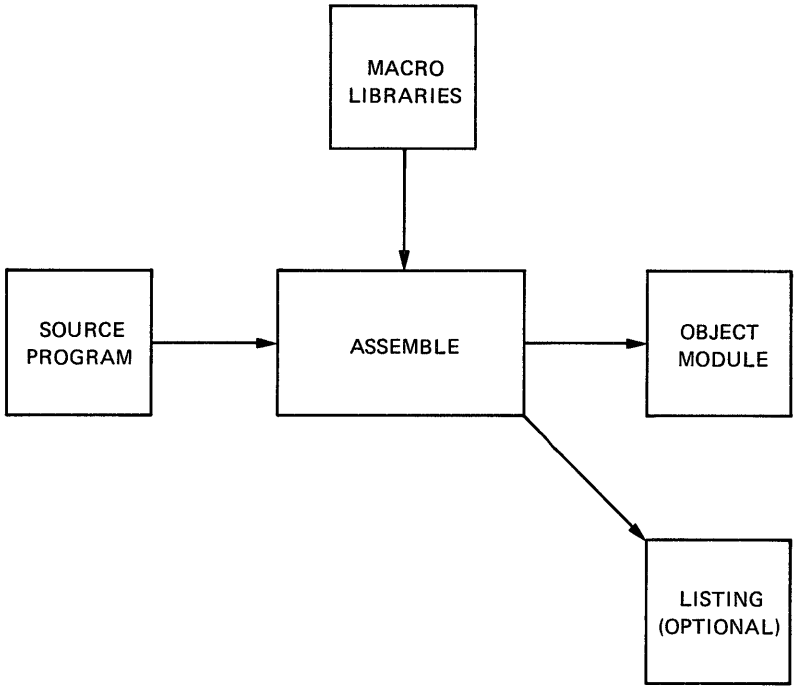

Figure 1-2 Function of a VAX-11 MACRO Assembler

During assembly processing, the VAX-11 MACRO assembler:

- Accounts for all instructions used within the source program and determines their relative positions within the program unit; it does this by means of a storage location counter
- Keeps track of all user-defined symbols and their respective values in a symbol table
- Converts assembly language mnemonics, user-defined symbols, and data values into their respective machine language (object code) equivalents

During processing, the assembler converts each program language statement into numerical data (the object code) and assigns the data a relative storage location. As the assembler translates and assigns each statement, it updates the value of the location counter

accordingly. The linker converts the relative storage locations assigned by the assembler to virtual storage locations in the computer's memory. Each location has an associated number called its address.

A VAX-11 MACRO assembly listing shows the addresses of memory locations and their contents as hexadecimal numbers. The hexadecimal numbers represent the machine language code that makes up the object module. See Section 2.2 for more information on the listing file.

1.3 USER-DEFINED SYMBOLS

User-defined symbols are symbolic names that you can use to:

- Identify the location of a routine
- Identify the location of data
- Represent a value

A symbol that identifies a location in memory is called a label. You can use labels to refer to locations without knowing exactly where they will be in memory.

You also can use a symbol to replace a constant used in several places in your program. This allows you to change the value referred to in several locations by simply redefining the symbol as a different value.

A symbol can be internal to one module; that is, the symbol is only referred to in the module in which it is defined. An internal symbol is also called a local symbol.

A symbol that is referred to in modules other than the one in which it is defined is called a global symbol. Global symbols are the key to modular programming. Global symbols provide communication between modules. You use a double colon (::) to define a global symbol used as a label and a double equal sign (==) to define a global symbol used to represent a value.

The assembler replaces each reference to a local symbol with the symbol's address or value. However, the assembler does not know the address of a global symbol defined in a different module. Therefore, it indicates to the linker that the symbol is global. The linker replaces each global symbol reference with the symbol's address or value.

As the assembler processes your module, it builds a symbol table. The symbol table contains all symbols used in the module with each symbol's address or value (when known). The symbol table is printed in the listing file (see Section 2.2.3). The assembler does not usually write the complete symbol table to the object module; instead it writes a symbol table that contains only global symbols. The linker uses the global symbol tables to resolve global symbol references. The VAX-11 Symbolic Debugger (see Section 1.7) also uses the object module symbol table. Consequently, you may want to include local symbols in the object module symbol table. To include a specific local symbol, you must specify it in the .DEBUG directive. To include all local symbols, you must specify the /ENABLE=DEBUG qualifier in the MACRO command or .ENABLE DEBUG in the source file.

There are two specialized kinds of global symbols: weak and universal symbols. Weak symbols do not have to be resolved by the linker (see the description of the .WEAK directive in the VAX-11 MACRO Language Reference Manual). Universal symbols are used in shareable images (see the description of universal symbols in the VAX-11 Linker Reference Manual).

Local labels are temporary labels (consisting of a number followed by a dollar sign) that you can use to refer to locations between symbolic labels (see the VAX-11 MACRO Language Reference Manual). Local labels are not symbols; local labels can be reused in the same object module. Consequently, local labels are not included in the symbol table and are not available to the linker or debugger.

1.4 MACROS

Macros are a very useful feature of the VAX-11 MACRO assembly language. A macro is any sequence of coding instructions that you want to recur in your program. You first define the macro and give it a name. Then, you can call the macro from any other part of your program by simply entering the macro name in the operator field of a statement line. You must define a macro before you can refer to it. The assembler directives that define macros are described in Chapter 6 of the VAX-11 MACRO Language Reference Manual.

Every time the assembler encounters the macro name, it inserts the code contained in the macro definition. This is called expanding a macro.

You can also define macros that contain conditional assembly directives. Each time the macro is expanded, the conditions are checked. Thus, you can generate several different code sequences from one macro.

In addition to using macros that you define, you can use system macros provided by the VAX/VMS operating system. These system macros perform useful functions such as calling system services. The VAX/VMS System Services Reference Manual describes how you can use system macros to call system services to perform, for example, file and record handling, process control, and memory management services.

Macros can be defined in a macro library. The system macros, for example, are defined in the system macro library. A macro library is a library consisting entirely of macro definitions. You can refer to macros in libraries in the same way that you refer to macros in your object modules. You must specify the name of the macro library in the MACRO command (with the /LIBRARY qualifier). You do not have to specify the name of the default library, the system macro library.

1.5 PROGRAM SECTIONS

You can segment your object module into a series of program sections. Using program sections allows you to write more modular programs, have increased error protection, and control the order in which your routines are stored in virtual memory. The assembler writes program section information into the object module, and the linker uses this information in creating an executable program image.

You specify the start of a program section and describe its attributes by using the .PSECT directive (see the VAX-11 MACRO Language Reference Manual). Within each module the assembler maintains several location counters -- one for each program section.

You can continue a previously defined program section by using a second .PSECT directive that specifies the same name as the .PSECT directive that defined the original program section.

Because the assembler does not know where each program section goes, all references between sections are relative to the base of the section. The linker resolves these references at link time.

You can use program sections to perform any of the following:

- Separate your object module into smaller sections of code. Each program section should contain a complete routine. This can increase the modularity of your program, making it easier to debug, maintain, and enhance.

- Allow different modules to gain access to the same data locations. (This is done in FORTRAN by using the COMMON statement.) If you specify the same program section name with the overlay (OVR) attribute in different modules, each program section shares the same virtual memory.

- Separate areas where you intend to write information from areas where you do not intend to write information. For example, if your program erroneously writes to an area with the no-write (NOWRT) attribute, a memory access violation occurs. Separating such areas in your program into program sections makes debugging your program easier because the program sections act as additional protection from miscoded instructions or logic errors.

- Identify sections of your object module to the debugger. The debugger uses the program section name to identify a location and to identify the section of the program being examined. Consequently, you should always specify names for all program sections. Do not use the default program sections that the assembler creates when you do not specify .PSECT or when you specify .PSECT with no program section name.

- Produce shareable program sections to use in shareable images. One copy of a shareable image on disk and in physical memory can be used by many processes at the same time. Several processes can gain access to the data in a shareable image. In addition, large programs that are used in many processes can be made into shareable images to improve system performance. See the VAX-11 Linker Reference Manual for more information on shareable images.

- Control the order in which program sections are stored in virtual memory; this can improve the performance of programs larger than your working set. Making frequently accessed program sections contiguous with each other in virtual memory increases the probability of having a frequently accessed program section in your working set.

The linker separates all program sections into groups with similar attributes. Within these groups the linker stores the program sections alphabetically by name.

Program sections with the same name and the overlay attribute are stored starting at the same address in virtual memory. Program sections with the same name and the concatenate attribute are concatenated in the order that they are specified to the linker.

The attributes you specify in the .PSECT directive describe but do not control the contents of the program section; you must ensure that the program section actually has those attributes. For example, you should not include instructions to be executed in a program section with the NOEXE (not executable) attribute.

1.6 LINKING MACRO PROGRAMS

The object module produced by the MACRO command may in itself be incomplete. It may need to be joined, or linked, with other object modules or library files to form a complete functioning program. The link operation:

- Joins together the object modules that use symbols with the object modules that define them
- Relocates individual object modules as necessary and assigns virtual memory addresses
- Produces an executable image and an optional map, as shown in Figure 1-3

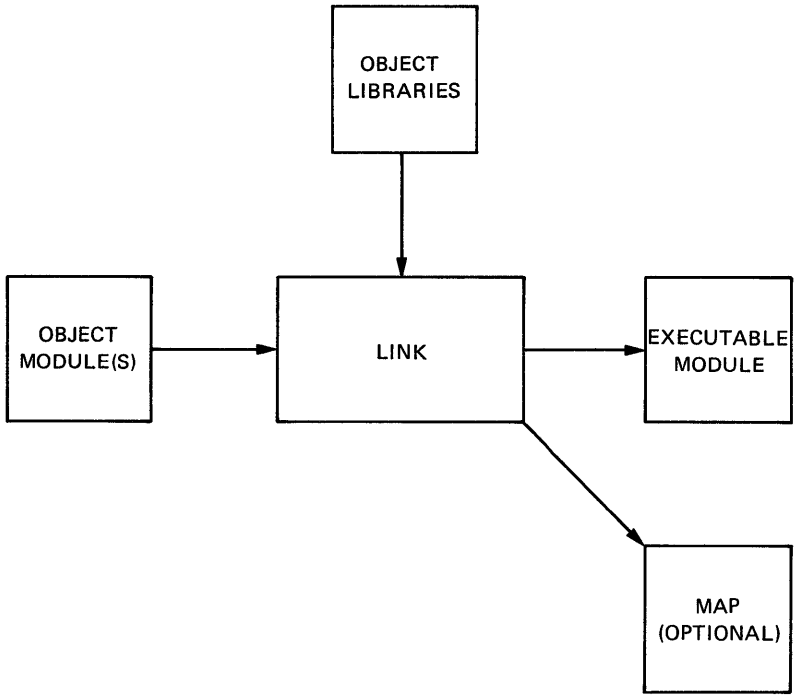

Figure 1-3 Link Functions

The link operation, in addition to joining object modules together, assigns virtual memory addresses to the relative addresses calculated by the VAX-11 MACRO assembler. Because the memory addresses of one object module must be relocated to accommodate the addresses used in another object module, the link operation serves to resolve all address changes. The result of the link operation is an image with all module links resolved and all virtual memory addresses and storage information assigned. The image, then, is a picture of what your program looks like just before execution.

An executable image is one that you can run on the system. Unless your program contains logic errors that prevent it from running properly (errors that the system cannot always detect), running the executable image of your program should produce the results you intended. However, if logic errors exist within your program, running the program will produce either erroneous results or none at all. If this is the case, you must study the source program, debug it, edit it, then perform the assembly and link operations again.

You can also link VAX-11 MACRO modules with subprograms written in other native mode languages, such as VAX-11 FORTRAN IV-PLUS. This capability gives you both the flexibility of assembly language programming and the ease of programming in a high-level language. For example, you can write a subprogram to perform data acquisition in VAX-11 MACRO and other subprograms to perform data analysis or file input/output in VAX-11 FORTRAN IV-PLUS.

In addition, the linker allows you to use object library files. These are files that contain already written, debugged, and linked subprograms and subroutines. Because you gain access to object library files at link time, their routines can be used by your program as needed.

1.6.1 Resolving Symbolic and Library References

The linker reads through all the object modules that you indicate as input to the LINK command. It gathers and evaluates information provided by the assembler that is necessary for program linking. For each input module, this information includes the object code, information needed for relocation, the relative address of the first instruction, the global symbols used, and the length of each program section.

One of the linker's functions is to resolve all global symbol references and library references in the joined routines.

During translation, the assembler notes which symbols in the object module are global. During linking, the linker keeps track of the global references and definitions found in all the object modules, and as linking proceeds, makes the appropriate correlations and modifies instructions or data as necessary. After linking, the linker outputs a list of all symbolic references that were not resolved (undefined globals) either because of a programming error or because some necessary object modules were not included in the LINK command.

References to library files also involve the use of global symbols. You gain access to the routines in a library by naming a routine as a global symbol in the source code of your program. You then link your program with the appropriate library file and the linker resolves the library references just as it does for any global symbol.

1.6.2 Program Relocation and Address Assignment

A second important function of the linker is to "fix" relative memory addresses so that they are virtual. The object module represents translated source instructions that have been assigned memory addresses relative to a base address of 0.

The linker assigns a base address to the image and fixes the base address of each program section.

1.7 DEBUGGING MACRO PROGRAMS

Debugging is the process of finding and correcting errors in executable programs; that is, in programs that have been assembled and linked without diagnostic messages, but that produce invalid results. (For information about diagnostic messages produced by VAX-11 MACRO, see Section 2.1.3 and Appendix A.)

The debugger provided with the VAX/VMS system is a symbolic debugger; it can refer to instructions and data by symbolic names. However, it can only gain access to the names that are included in the symbol table in the object module. By default, the debugger can gain access to global symbol and program section names. If you want to debug with local symbol names, you must specify the /ENABLE=DEBUG qualifier in the MACRO command or include .ENABLE DEBUG in the source code.

See the VAX-11 Symbolic Debugger Reference Manual for more information on debugging VAX-11 MACRO programs.

CHAPTER 2

USING VAX-11 MACRO

The MACRO command invokes the VAX-11 MACRO assembler. The assembler reads your source program; checks it for syntax errors; produces an object module; and, optionally produces a listing file. Section 2.1 describes the format of the MACRO command and Section 2.2 describes the listing file.

2.1 THE MACRO COMMAND

Format

```
$ MACRO[/qualifiers] file-spec-list[/qualifiers]
```

Parameters

/qualifiers

Command or file qualifiers that indicate special actions to be performed by the assembler (see Section 2.1.2).

file-spec-list

A file specification or list of file specifications that specify the source and macro library input files to be assembled into object modules (see Section 2.1.1). If the file specifications are separated by plus signs (+), the files are concatenated and assembled into one object module. If the file specifications are separated by commas (,), the files are assembled separately into individual object modules. The default file type is MAR for source files and MLB for macro library files.

The assembler reads your source files in the order in which you specify them. You can request the assembler to perform several assemblies with one command. The assembler, by default, produces an object module with the same file name as your first input file. You can use the /OBJECT qualifier to specify the file name of the object module. You can suppress the production of the object module by using the /NOOBJECT qualifier.

In interactive mode, the assembler does not, by default, produce a listing file; you must use the /LIST qualifier to specify a listing file. In batch mode, the assembler, by default, produces a listing file with the same file name as the first input file. You can use the /LIST qualifier to specify the file name of the listing file.

Examples

1. $ MACRO PART1+PART2+PART3

   The assembler concatenates the source files PART1, PART2, and PART3 and assembles them into one object module with a name of PART1. No listing file is created.

2. $ MACRO/LIST APROG,BPROG,CPROG

   The assembler independently assembles the three source files APROG, BPROG, and CPROG into object modules and listing files.

3. $ MACRO MYPROG/LIST+MLIB/LIBRARY

   The assembler uses the macro library MLIB to assemble the source file MYPROG and creates an output object module and listing files with the file name MYPROG.

The following sections describe the file specifications, command and file qualifiers, and how the assembler handles errors.

2.1.1 File Specifications

A file specification indicates the input file to be processed, or the output file to be produced.

Format

device:[directory]filename.filetype;version

Parameters

device

The physical device on which a file is stored or is to be written.

[directory]

The name of the directory under which the file is cataloged. The square brackets are required.

filename

The name of the file; filename can be up to 9 characters long.

filetype

The type of the file, describing the kind of data in the file; filetype can be up to 3 characters long.

version

The version number of the file. Versions are identified by a decimal number, which is incremented each time a new version of the file is created.

You need not explicitly state all elements of a file specification each time you assemble a program. The only part of the file specification that is always required is the file name. If you omit any other part of the file specification, a default value is used. Table 2-1 summarizes the default values.

Table 2-1
File Specification Defaults

Optional Element	Default
device	User's current default device
directory	User's current default directory
filetype	Depends on usage: Source input file MAR Macro library file MLB Object module OBJ Listing file LIS
version	Input: highest existing version Output: highest existing version plus 1

You can also specify a logical name rather than a complete file specification. See the VAX/VMS Command Language User's Guide for more information on logical names.

2.1.2 Qualifiers

Qualifiers specify that the assembler should perform the specified actions. Qualifiers can be used as either command qualifiers or file qualifiers. A command qualifier affects all the assemblies specified in the MACRO command. A file qualifier affects only the assembly that it qualifies.

All MACRO qualifiers except the /LIBRARY qualifier can be either command qualifiers or file qualifiers. The /LIBRARY qualifier can only be a file qualifier.

A qualifier can have one of the following formats:

```
/qualifier

/qualifier=function

/qualifier=(function1, function2, ..., functionn)
```

Table 2-2 lists the MACRO qualifiers, their possible functions, and their default functions. Note that some values have a long form and a short form. You can use either form; the effect is the same. Square brackets around the equal sign in the table indicate that the qualifier can appear with or without functions.

Table 2-2
VAX-11 MACRO Command Qualifiers

Qualifier	Functions		Negative Form	Default
	Long Form	Short Form		
/CROSS[=]	ALL	--	/NOCROSS	/NOCROSS
	DIRECTIVES	DIR		
	MACROS	MAC		
	OPCODES	OPC		
	REGISTERS	REG		
	SYMBOLS	SYM		
/DISABLE=	ABSOLUTE	AMA	/ENABLE=	/DISABLE= (AMA,DBG,LSB,SUP,FPT)
	DEBUG	DBG		
	GLOBAL	GBL		
	SUPPRESSION	SUP		
	TRACEBACK	TBK		
	TRUNCATION	FPT		
/ENABLE=	ABSOLUTE	AMA	/DISABLE=	/ENABLE=(GBL,TBK)
	DEBUG	DBG		
	GLOBAL	GBL		
	SUPPRESSION	SUP		
	TRACEBACK	TBK		
	TRUNCATION	FPT		
/LIBRARY	--	--	--	Not a library
/LIST[=]	file-spec	--	/NOLIST	/NOLIST (interactive mode) /LIST (batch mode)
/OBJECT[=]	file-spec	--	/NOOBJECT	/OBJECT
/SHOW[=]	BINARY	MEB	/NOSHOW[=]	/SHOW=(MC,CND,MD)
	CALLS	MC		
	CONDITIONALS	CND		
	DEFINITIONS	MD		
	EXPANSIONS	ME		

The following sections describe the VAX-11 MACRO command qualifiers in detail.

2.1.2.1 **The /CROSS and /NOCROSS Qualifiers** - The /CROSS and /NOCROSS qualifiers control whether a cross-reference listing is included in the listing file. If you specify the /CROSS qualifier, the listing file includes a cross-reference listing. Note that if you enter a MACRO command with the /CROSS qualifier interactively, you must also specify the /LIST qualifier. The /NOCROSS qualifier is the default; you need not specify it to have the cross-reference listing excluded.

Table 2-3 lists the functions that you can specify in a /CROSS qualifier. You can specify either the long form or the short form of the functions. If you specify the /CROSS qualifier with no functions, it is equivalent to /CROSS=(MAC,SYM). See Section 2.2.5 for a description of the format of the cross-reference listing. See the VAX-11 MACRO Language Reference Manual for a description of the .CROSS and .NOCROSS directives.

Table 2-3
/CROSS Qualifier Functions

Long Form	Short Form	Meaning
ALL	--	Includes directives, macros, opcodes, registers, and symbols in the cross-reference listing
DIRECTIVES	DIR	Includes directives in the cross-reference listing
MACROS	MAC	Includes macros in the cross-reference listing
OPCODES	OPC	Includes opcodes in the cross-reference listing
REGISTERS	REG	Includes register references in the cross-reference listing
SYMBOLS	SYM	Includes user-defined symbols in the cross-reference listing

2.1.2.2 **The /ENABLE and /DISABLE Qualifiers** - The /ENABLE and /DISABLE qualifiers have the same effect as the .ENABLE and .DISABLE assembler directives, respectively. They control the way that the assembler interprets your source program. The /ENABLE and /DISABLE qualifiers override any .ENABLE or .DISABLE directives in the source program. See the VAX-11 MACRO Language Reference Manual for more information on the .ENABLE and .DISABLE directives.

Table 2-4 lists the functions that you can specify in an /ENABLE or /DISABLE qualifier. You can specify either the long form or the short form of the functions. If you specify more than one function, you must enclose the function list in parentheses. If you use an /ENABLE or /DISABLE qualifier, you must specify at least one function in the qualifier.

Table 2-4
/ENABLE and /DISABLE Qualifier Functions

Long Form	Short Form	Default	Meaning
ABSOLUTE	AMA	/DISABLE	When ABSOLUTE is enabled, all PC relative addressing modes are assembled as absolute addressing modes
DEBUG	DBG	/DISABLE	When DEBUG is enabled, all local symbols are included in the symbol table in the object module for use by the debugger
GLOBAL	GBL	/ENABLE	When GLOBAL is enabled, all undefined symbols are considered to be external symbols; when GLOBAL is disabled, any undefined symbol that is not listed in a .EXTERNAL directive causes an assembly error
SUPPRESSION	SUP	/DISABLE	When SUPPRESSION is enabled, all symbols that are defined but not referred to are not listed in the symbol table; when SUPPRESSION is disabled, all symbols that are defined are listed in the symbol table
TRACEBACK	TBK	/ENABLE	When TRACEBACK is enabled, MACRO includes the program section names and lengths, module names, and routine names in the object module for use by the debugger; when TRACEBACK is disabled, MACRO excludes this information and, in addition, does not make any local symbol information available to the debugger
TRUNCATION	FPT	/DISABLE	When TRUNCATION is enabled, floating-point numbers are truncated; when TRUNCATION is disabled, floating-point numbers are rounded

2.1.2.3 **The /LIBRARY Qualifier** - The /LIBRARY qualifier indicates that the associated input file contains a macro library. The /LIBRARY qualifier affects only the input file that it qualifies.

2.1.2.4 **The /LIST and /NOLIST Qualifiers** - The /LIST and /NOLIST qualifiers control whether an output listing file is created. If you specify the /NOLIST qualifier, no listing file is created. If you specify the /LIST qualifier, a listing file is created. The /LIST qualifier determines the file specification of the output listing file.

If you enter the MACRO command interactively, the assembler does not, by default, create a listing file. If you execute the MACRO command in batch mode, however, the assembler does create a listing file by default.

If you specify the /LIST qualifier with a file specification, the assembler uses that file specification for the output listing file.

If you specify the /LIST qualifier without a file specification, the default file name depends on whether /LIST is used as a command qualifier or as a file qualifier. If /LIST is used as a command qualifier, the default file name is the name of the first input source file. If /LIST is used as a file qualifier, the default file name is the name of the file that /LIST qualifies.

2.1.2.5 **The /OBJECT and /NOOBJECT Qualifiers** - The /OBJECT and /NOOBJECT qualifiers control whether an object module is created. The /OBJECT qualifier is the default; you need not specify it to have an object module created. If you specify the /NOOBJECT qualifier, no object module is created.

If you do not specify either the /OBJECT or the /NOOBJECT qualifier, the assembler creates an object module with the same file name as the first input file.

If you specify the /OBJECT qualifier with a file specification, the assembler uses that file specification for the output object file.

If you specify the /OBJECT qualifier without a file specification, the default file name depends on whether /OBJECT is used as a command qualifier or as a file qualifier. If /OBJECT is used as a command qualifier, the default file name is the name of the first input file. If /OBJECT is used as a file qualifier, the default file name is the name of the file that /OBJECT qualifies.

2.1.2.6 **The /SHOW and /NOSHOW Qualifiers** - The /SHOW and /NOSHOW qualifiers have the same effect as the .SHOW and .NOSHOW assembler directives, respectively. They control what lines appear in the listing. Note that if you enter a MACRO command with a /SHOW or /NOSHOW qualifier interactively, you must also specify the /LIST qualifier. The /SHOW and /NOSHOW qualifiers have different effects depending on whether you specify them with or without functions.

If you specify /SHOW or /NOSHOW with functions, the qualifier controls the listing of source lines that are in conditional assembly blocks, macros, or repeat blocks. The /SHOW and /NOSHOW qualifiers override any .SHOW or .NOSHOW directives that are in the source program. Table 2-5 describes the /SHOW and /NOSHOW functions. You can specify either the long form or the short form of the functions. If you use more than one function, you must enclose the function list in parentheses.

Specifying either the /SHOW or /NOSHOW qualifier with no function is equivalent to starting your source file with an extra .SHOW or .NOSHOW directive, respectively. The listing count is incremented by a /SHOW qualifier and is decremented by a /NOSHOW qualifier. The listing count controls whether all source lines are listed. If the listing count is positive, all source lines are listed (including lines in conditional assembly blocks, macros, and repeat blocks). If the listing count is negative, no lines are listed. If the listing count is 0, all lines except lines in conditional blocks, macros, and repeat blocks are listed: these lines are listed depending on the values specified in .SHOW and .NOSHOW directives.

Table 2-5
/SHOW and /NOSHOW Qualifier Functions

Long Form	Short Form	Default	Function
BINARY	MEB	/NOSHOW	Lists macro expansions and repeat block expansions that generate binary code; BINARY is a subset of EXPANSIONS
CALLS	MC	/SHOW	Lists macro calls and repeat block specifiers
CONDITIONALS	CND	/SHOW	Lists unsatisfied conditional code associated with the conditional assembly directives
DEFINITIONS	MD	/SHOW	Lists macro and repeat range definitions that appear in an input source file
EXPANSIONS	ME	/NOSHOW	Lists macro and repeat range expansions

2.1.3 Diagnostic Messages

If the assembler encounters an error during assembly, it displays a diagnostic message. The assembler displays the message on the terminal (for interactive jobs) or in the batch log file (for batch jobs) and in the listing file.

Appendix A describes the VAX-11 MACRO diagnostic messages.

There are two levels of severity: error and warning. Object modules created with an error message cannot be linked into an image file. Object modules created with a warning message can be linked into an image file although the linker will display a diagnostic message.

The assembler displays diagnostic messages in the following format:

```
%MACRO-l-code, text
```

l
A severity code indicator. It has a value of E for an error or a value of W for a warning.

code
An abbreviation of the message text.

text
The explanation of the message.

For example:

```
%MACRO-E-ILLMASKBITS, Reserved bits set in ENTRY mask
```

The assembler displays on the terminal or batch log file the following information:

- The line from the listing that would precede the error message if there were a listing file. This line is often the source line that contains the error, but sometimes it is only the binary expansion of the source line.
- The error message itself.

If the assembler has detected any errors during the assembly process, it displays a diagnostic summary when the assembly is completed. It displays this summary on the terminal or batch log file and listing file. The summary contains the total number of errors and warnings with the line number and page number (enclosed in parentheses) of each. The assembler also displays at the end of the error summary a list of the file specifications in the MACRO command (see Section 2.2.6).

An example of a diagnostic summary follows.

```
$ MACRO/LIST PROG

There were 6 errors and 1 warnings, on lines:

100   (1)   1100   (1)   400   (2)   200   (3)   800   (3)   1200   (3)
400   (5)
/LIST PROG
$
```

2.2 LISTING FILE FORMAT

The listing file produced by VAX-11 MACRO has the following six parts.

- Table of contents (optional) and page headings
- Source statements and hexadecimal code
- Symbol table
- Program section synopsis
- Cross-reference listing (optional)
- Assembly summary

The following sections describe these six parts. Section 2.2.7 contains an example of a listing.

2.2.1 Table of Contents and Page Headings

If the source module contains any optional .SUBTITLE directives, VAX-11 MACRO prints a table of contents before the assembly listing. The table of contents lists all the subtitles specified in .SUBTITLE directives. The subtitle is listed with the source page number and the line number of the .SUBTITLE directive.

VAX-11 MACRO prints a new page in the listing file when it encounters a .PAGE directive in the source, when it encounters a new page in the source file, or when the existing page of the listing is filled. On the top of each page in the listing, VAX-11 MACRO prints two header lines. The first line of the header contains the following information:

- Title of the module specified in the .TITLE directive
- Comment after the title of the module in the .TITLE directive
- Date
- Time of day
- Assembler version identification
- Listing page number

The second line of the header contains the following information:

- The identifying information specified in the .IDENT directive (often used to specify a version number)
- Subtitle of the section of the module specified in the .SUBTITLE directive
- Source file creation date and time
- Source file specification
- Source page number

2.2.2 Source Statements and Hexadecimal Code

This section is the main part of the listing: it contains the source lines of the module and the hexadecimal code generated. Each line of code contains the following information:

- The source line, including comments
- The line number from the editor or, if the file has no line numbers, the sequence number of the line
- The location counter
- The hexadecimal code

The hexadecimal code is printed with the lowest address on the right. The hexadecimal code listed for an instruction contains, from right to left:

- The opcode
- The addressing mode for the first operand (if any)
- The addressing mode for the second operand (if any)
- The addressing mode for the third operand (if any)

The binary code for data storage is listed from right to left. The number of data items that are listed on one line depends on the size of the data type as follows:

Data Type	Number of Items per Line
Byte	12
Word	7
Longword	4
Quadword	1
ASCII	12 (characters)
Packed decimal string	24 (digits)

If an expression contains an externally defined symbol, the assembler lists the value of the expression followed by an apostrophe. The assembler evaluates the expression by assigning a value of 0 to the externally defined symbol. The apostrophe indicates that the linker will complete the evaluation of the expression.

VAX-11 MACRO also prints the diagnostic messages in this section of the listing. It prints each diagnostic message immediately after the line at which the error was detected. See Section 2.1.3 for a description of the diagnostic message format and Appendix A for a list of the VAX-11 MACRO diagnostic messages.

2.2.3 Symbol Table

The symbol table lists all symbols, except permanent symbols, that are defined or referred to in the module. The symbols are listed alphabetically, in three columns. The symbol's value (when known) is listed next to the symbol. If the symbol is assigned a value by a direct assignment statement or a directive (such as the .NARG directive), the symbol is separated from the value by an equal sign. If the symbol is defined externally (the value is unknown), the value is listed as a string of asterisks. The following letters are used in the symbol table to describe special attributes of symbols.

Letter	Meaning
D	The symbol is a local symbol that will be made available to the debugger.
G	The symbol is globally defined in a module.
R	The symbol is relocatable.
W	The symbol is a weak global symbol (specified in a .WEAK directive).
X	The symbol is defined externally.
U	The symbol is not defined (produced when .DISABLE GLOBAL has been specified and undefined symbol is not specified in .EXTERNAL).

If a symbol is defined externally or as a relocatable value, the number of the program section in which it appears first is printed. See Section 2.2.4 for information about program section numbers.

2.2.4 Program Section Synopsis

The program section synopsis lists the program sections, their size, their attributes, and their alignment. The program sections are listed in the order in which they are defined in the program. Each program section is assigned a number based on the order in which it is defined in the program: this number is printed after the size of the program section.

2.2.5 Cross-Reference Listing

The assembler lists the cross references separately for the following groups: symbols, macros, directives, opcodes, and registers. Within each group each item is listed alphabetically. For each item, the following information is listed:

- Symbol name
- Value
- Line number and page number of the symbol's definition
- Line number and page number of each reference to the symbol

You control which groups are cross referenced by specifying values in the /CROSS qualifier. You can exclude certain symbols from the cross-reference listing by using the .CROSS and .NOCROSS directives.

2.2.6 Assembly Summary

The assembly summary contains internal assembler performance indicators, a diagnostic summary, and the qualifiers and file specifications in the MACRO command.

The internal assembler performance indicators include the page faults, CPU time, and elapsed time for the different stages of the assembly. In addition, the indicators include the working set limit and the number of symbols, source lines, object records, and macros and the memory required to process these.

If the assembler detected any errors in the module, it prints the same diagnostic summary in the listing that it displays on the terminal. If no errors occurred, the assembler prints the following message in the assembly summary:

```
There were no errors or warnings.
```

The last line in the listing file shows the qualifiers and file specifications entered in the MACRO command.

2.2.7 Assembly Listing Example

The following pages show an example of a typical assembly listing.

```
CALC            - Routine to do simple arithmetic       23-JAN-1979 11:11:44    VAX-11 Macro V02.29              Page    0
Table of contents

   (1)     3200  Macro definitions
   (2)      100  Procedure entry point
```

Table of Contents

```
CALC            - Routine to do simple arithmetic       23-JAN-1979 11:11:44    VAX-11 Macro V02.29              Page   1
01                                                      23-JAN-1979 11:11:29    DB1:[JOSH]CALC.MAR;8             (1)

                        100         .TITLE  CALC    - Routine to do simple arithmetic
                        200         .IDENT  /01/
                        300
                        400 ;++
                        500 ; FUNCTIONAL DESCRIPTION:
                        600 ;
                        700 ;       This routine accepts two integers and an operator index as
                        800 ;       inputs, executes the requested arithemetic operation and
                        900 ;       returns the result.
                       1000 ;
                       1100 ; INPUT:
                       1200 ;
                       1300 ;       4(AP)           First integer
                       1400 ;       8(AP)           Second integer
                       1500 ;       12(AP)          Operator index - 0-addition, 1-subtraction,
                       1600 ;                       2-multiplication, 3-division
                       1700 ;       16(AP)          Address of result
                       1800 ;
                       1900 ; OUTPUT:
                       2000 ;
                       2100 ;       The operation is executed and the result stored at the address
                       2200 ;       contained in 16(AP)
                       2300 ;
                       2400 ;--
                       2500
                       2600         .ENABLE DEBUG                   ; Make symbols available to the
                       2700                                         ; debugger
                       2800
                       2900         .DEFAULT DISPLACEMENT WORD      ; Use word displacements on
                       3000                                         ; PC-relative references
                       3100
                       3200         .SUBTITLE       Macro definitions
                       3300 ;++
                       3400 ;       Define macro to use CASE instruction.
                       3500 ;
                       3600 ;       CASE    SRC,DISPLIST,TYPE,LIMIT,NMODE
                       3700 ;
                       3800 ;       Where:
                       3900 ;
                       4000 ;       SRC             Case selector
                       4100 ;       DISPLIST        List of displacements
                       4200 ;       LIMIT           Base value of the selector
                       4300 ;       TYPE            B-byte, W-word (default), L-long
                       4400 ;
                       4500 ;--
                       4600         .MACRO  CASE,SRC,DISPLIST,TYPE=W,LIMIT=#0,NMODE=S^#,?BASE,?MAX
                       4700         CASE'TYPE       SRC,LIMIT,NMODE'<<MAX-BASE>/2>-1
                       4800                                         ; Case instruction
                       4900 BASE:                                   ; Local label used to count args
                       5000         .IRP    EP,<DISPLIST>           ; To set up offset list
                       5100         .SIGNED_WORD    EP-BASE         ; Offset list
                       5200         .ENDR                           ;
                       5300 MAX:                                    ; Local label used to count args
                       5400         .ENDM   CASE
                       5500
                       5600
```

Source Statements and Hexadecimal Code (Part 1 of 2)

```
CALC            - Routine to do simple arithmetic              23-JAN-1979 11:11:44    VAX-11 Macro V02.29              Page   2
01              Procedure entry point                          23-JAN-1979 11:11:29    DB1:[JOSH]CALC.MAR;8               (2)

                                    0000   100          .SUBTITLE       Procedure entry point
                                00000000   200          .PSECT  RO_CODE,EXE,NOWRT
                                    0000   300 ;
                                    0000   400 ; Get the arguments from the argument list, perform the calculation
                                    0000   500 ; and return the result in the fourth argument and the status in R0
                                    0000   600 ;
                                001C' 0000   700          .ENTRY  CALC,^M<R2,R3,R4>       ; Routine ENTRY point
50          0000'8F     3C      0002   800          MOVZWL  #SS$_NORMAL,R0          ; For success, SS$_NORMAL
                                    0007   900                                          ; is defined in $SSDEF .
52            04 AC     D0      0007  1000          MOVL    4(AP),R2                ; Get argument from argument list
                                    000B  1100                                          ; R2 contains first argument
53            08 AC     D0      000B  1200          MOVL    8(AP),R3                ; R3 contains second argument
54            0C AC     D0      000F  1300          MOVL    12(AP),R4               ; R4 contains operator index
                                    0013  1400          CASE    R4,-                    ; Dispatch to evaluation routine
                                    0013  1500                  <ADD,SUB,MUL,DIV>       ;
                   24     11      001F  1600          BRB     ERR                     ; Calling routine specified an
                                    0021  1700                                          ; illegal operator index
53                 52     C1      0021  1800 ADD:     ADDL3   R2,R3,@16(AP)           ; (0) Calculate the sum
                10 BC             0024
                   1D     1D      0026  1900          BVS     ERR                     ; An overflow occurred
                   04     04      0028  2000          RET                             ; Return to calling program
52                 53     C3      0029  2100 SUB:     SUBL3   R3,R2,@16(AP)           ; (1) Form the difference
                10 BC             002C
                   15     1D      002E  2200          BVS     ERR                     ; An overflow occurred
                   04     04      0030  2300          RET                             ; Return to calling program
53                 52     C5      0031  2400 MUL:     MULL3   R2,R3,@16(AP)           ; (2) Calculate the product
                10 BC             0034
                   0D     1D      0036  2500          BVS     ERR                     ; An overflow occurred
                   04     04      0038  2600          RET                             ; Return to calling program
                   53     D5      0039  2700 DIV:     TSTL    R3                      ; (3) Check if divisor is 0
                   08     13      003B  2800          BEQL    ERR                     ; Avoid division by 0
52                 53     C7      003D  2900          DIVL3   R3,R2,@16(AP)           ; (3) Calculate the quotient
                10 BC             0040
                   01     1D      0042  3000          BVS     ERR                     ; An overflow occurred
                   04     04      0044  3100          RET                             ; Return to calling program
                10 BC     D4      0045  3200 ERR:     CLRL    @16(AP)                 ; Return 0 for overflow,
                                    0048  3300                                          ; division by 0, or illegal
                                    0048  3400                                          ; operator index
50                 00     D0      0048  3500          MOVL    #0,R0                   ; Indicate failure
                                    004B  3600          RET                             ; Return to calling program
                                    004C  3700          .END
```

Source Statements and Hexadecimal Code (Part 2 of 2)

```
CALC            - Routine to do simple arithmetic        23-JAN-1979 11:11:44    VAX-11 Macro V02.29          Page   3
Symbol table                                             23-JAN-1979 11:11:29    DB1:[JOSH]CALC.MAR;8                (2)

ADD             00000021 R  D  02
CALC            00000000 RG D  02
DIV             00000039 R  D  02
ERR             00000045 R  D  02
MUL             00000031 R  D  02
SSS_NORMAL      ********   X   02
SUB             00000029 R  D  02

                                        +----------------+
                                        ! Psect synopsis !
                                        +----------------+

PSECT name      Allocation          PSECT No.  Attributes
----------      ----------          ---------  ----------
. ABS  .        00000000  (    0.)  00 (  0.)  NOPIC   USR   CON   ABS   LCL NOSHR NOEXE NORD  NOWRT BYTE
. BLANK .       00000000  (    0.)  01 (  1.)  NOPIC   USR   CON   REL   LCL NOSHR   EXE   RD    WRT BYTE
RO_CODE         0000004C  (   76.)  02 (  2.)  NOPIC   USR   CON   REL   LCL NOSHR   EXE   RD  NOWRT BYTE
```

Symbol Table and Program Section Synopsis

```
CALC            - Routine to do simple arithmetic       23-JAN-1979 11:11:44  VAX-11 Macro V02.29          Page   4
Cross reference                                         23-JAN-1979 11:11:29  DB1:[JOSH]CALC.MAR;8          (2)


                                                        +-------------------------+
                                                        ! Symbol Cross Reference !
                                                        +-------------------------+

SYMBOL          VALUE        DEFINITION      REFERENCES...
------          -----        ----------      -------------
ADD             00000021-R   1800    (2)       1500    (2)
CALC            00000000-R   700     (2)
DIV             00000039-R   2700    (2)       1500    (2)
ERR             00000045-R   3200    (2)     #-1600    (2)    #-1900    (2)    #-2200    (2)    #-2500    (2)    #-2800    (2)
                                               #-3000    (2)
MUL             00000031-R   2400    (2)       1500    (2)
SS$_NORMAL      00000000-XR                    #-800     (2)
SUB             00000029-R   2100    (2)       1500    (2)
```

Cross-Reference Listing

```
                                +-------------------------+
                                ! Macros Cross Reference !
                                +-------------------------+

MACRO             SIZE          DEFINITION      REFERENCES...
-----             ----          ----------      -------------
CASE              1             4600   (1)      1400   (2)

                                +-------------------------+
                                ! Performance indicators !
                                +-------------------------+

Phase                     Page faults    CPU Time       Elapsed Time
-----                     -----------    --------       ------------
Initialization                      6    00:00:00.04    00:00:00.15
Command processing                 22    00:00:00.22    00:00:00.71
Pass 1                            161    00:00:00.62    00:00:01.21
Symbol table sort                   1    00:00:00.00    00:00:00.00
Pass 2                             87    00:00:00.30    00:00:00.56
Symbol table output                 3    00:00:00.01    00:00:00.00
Psect synopsis output               4    00:00:00.02    00:00:00.01
Cross-reference output             23    00:00:00.08    00:00:00.13
Assembler run totals              309    00:00:01.30    00:00:02.83

The working set limit was 150 pages.
1542 bytes (4 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 7 non-local and 2 local symbols.
94 source lines were read in Pass 1, producing 15 object records in Pass 2.
1 page of virtual memory was used to define 1 macro.

There were no errors or warnings.

/LIST/CROSS CALC
```


Cross-Reference Listing (Cont.) and Assembly Summary

CHAPTER 3

WRITING POSITION-INDEPENDENT CODE

An object module produced by VAX-11 MACRO is relocatable; that is, it can be linked anywhere in virtual memory. The linker modifies relocatable addresses so that they reflect the virtual memory locations in which the module will run. Once linked, the image can only be moved in virtual memory if the source code follows the restrictions described in this chapter. Source code that follows these restrictions, and thus can be moved in virtual memory, is called "position-independent code." Source code that does not follow these restrictions is called "position-dependent code." Images linked from position-dependent code will run correctly only at one virtual memory location.

Position independence is important if you are creating a shareable image. To use a shareable image, you must relink it with object modules. If the shareable image is position independent, the linker can place it anywhere in virtual memory. If the shareable image is position dependent, the linker must place it at a fixed virtual address. You cannot link object modules with two position-dependent, shareable images that share a virtual address.

The linker does not use the position-independent code (PIC) program section attribute to determine whether a shareable image is position independent. The linker assumes that when it is linking a shareable image, the shareable image is position independent unless the source code contains a .ADDRESS assembler directive or unless a base address was specified in the LINK command. Consequently, if you are linking a shareable image that is position dependent, specify a base address in the LINK command or use a .ADDRESS directive in the source code. Otherwise, the linker will assume that the image is position independent and the shareable image will not execute correctly. See the VAX-11 Linker Reference Manual for more information on linking shareable images.

Position independence depends on the addressing modes used in the source code and the way addresses are stored in the program. The remainder of this chapter assumes that you are familiar with the addressing modes described in Chapter 4 of the VAX-11 MACRO Language Reference Manual.

The following addressing modes involve only register references and are always position independent if the register's value is set by an instruction that is itself position independent.

Format	Mode
Rn	Register
(Rn)	Register deferred
(Rn)+	Autoincrement
@(Rn)+	Autoincrement deferred
-(Rn)	Autodecrement

The displacement addressing modes are position independent if the expression specifying the displacement is absolute and if the register's value is set by an instruction that is position independent itself. The displacement addressing modes are listed below.

Format	Mode
dis(Rn)	Displacement
@dis(Rn)	Displacement deferred

Relative and relative deferred addressing modes are position independent if the address expression is relocatable. Absolute addressing mode is position independent if the address expression is absolute (for example, an address in the system space). Because the linker converts general addressing mode to relative if the expression is relocatable and converts it to absolute if the expression is absolute, using general addressing mode ensures that the code is position independent. Table 3-1 summarizes the position independence or dependence of relative and absolute modes.

Table 3-1
Relative and Absolute Addressing Modes

Mode	Position Independence/Dependence	
	Relocatable Address Expression	Absolute Address Expression
Relative	Position independent	Position dependent
Relative Deferred	Position independent	Position dependent
Absolute	Position dependent	Position independent
General	Position independent	Position independent

The index addressing modes are position independent if the base mode is position independent and if the index register contains an absolute number (not an address).

In addition, to ensure position independence, you must make sure that no addresses are stored as data. For example, if you have a table of pointers, the code will be position dependent. But if you replace the table of pointers with a table of displacements from a relocatable address, then the code can be position independent.

The remainder of this chapter presents four examples showing the use of the different addressing modes to write position-independent code.

Example 1

```
        MOVL    #TABADDR,R0             ; POSITION-DEPENDENT CODE
        MOVAB   TABADDR,R0              ; POSITION-INDEPENDENT CODE
        MOVAB   IOC$GL_DEVLIST,R0       ; POSITION-DEPENDENT CODE
        MOVL    #IOC$GL_DEVLIST,R0      ; POSITION-INDEPENDENT CODE
```

This example demonstrates the use of relative and absolute modes in writing position-independent code. All the instructions in this example move an address to R0. The address TABADDR is a relocatable address; the address IOC$GL_DEVLIST is absolute. If the address is relocatable, relative mode is position-independent and absolute mode is not. But, if the address is absolute, absolute mode is position-independent and relative mode is not.

Example 2

```
CHARS:  .ASCII  \ABCDEFGHIJKLMNOPQRSTUVWXYZ\
          .
          .
          .
        MOVL    #4,R3                   ; PUT OFFSET OF LETTER E IN R3.
        MOVB    CHARS(R3),R0            ; POSITION-DEPENDENT CODE
        MOVAB   CHARS,R3                ; PUT ADDRESS OF CHARS IN R3.
        MOVB    4(R3),R0                ; POSITION-INDEPENDENT CODE
        MOVL    #4,R3                   ; PUT OFFSET OF LETTER E IN R3.
        MOVB    CHARS[R3],R0            ; POSITION-INDEPENDENT CODE
```

This example demonstrates the use of displacement and index modes in writing position-independent code. The address CHARS is a relocatable address. Compare the first addressing mode, which is position dependent, with the two following equivalent addressing modes, which are position independent.

Example 3

```
; SETTING UP A STRING DESCRIPTOR IN A POSITION-DEPENDENT WAY
        .ALIGN LONG
DESCRIP:
        .LONG   EOSTR-STR               ; LENGTH OF STRING.
        .ADDRESS        STR             ; CODE IS POSITION DEPENDENT
STR:    .ASCII  \AN ASCII STRING\       ; THE STRING
EOSTR:                                  ; THE END OF STRING
; TO ACCESS THIS DESCRIPTOR
        MOVAB   DESCRIP,R2              ; GET ADDRESS OF DESCRIPTOR
;
; SETTING UP A STRING DESCRIPTOR IN A POSITION-INDEPENDENT WAY
; BY CREATING THE STRING DESCRIPTOR ON THE STACK
        PUSHAB  STR                     ; POSITION-INDEPENDENT REFERENCE
                                        ; TO GET ADDRESS OF STRING ON THE
                                        ; STACK
        PUSHL   #EOSTR-STR              ; PUSH LENGTH OF STRING ON STACK
        MOVL    SP,R2                   ; GET ADDRESS OF DESCRIPTOR
;
; SETTING UP A LIST HEAD IN A POSITION-DEPENDENT WAY
QHEADA: .ADDRESS        QHEADA          ; THIS IS POSITION DEPENDENT
        .ADDRESS        QHEADA          ;
;
; SETTING UP A LIST HEAD IN A POSITION-INDEPENDENT WAY BY USING
; EXECUTABLE INSTRUCTIONS TO STORE ADDRESSES
QHEADB: .BLKA   2                       ; RESERVE 2 LONGWORDS FOR ADDRESS
                                        ; STORAGE
; SOURCE CODE TO STORE ADDRESSES
        MOVAB   QHEADB,R0               ; GET THE ADDRESS OF THE LIST HEAD.
        MOVL    R0,(R0)                 ; STORE THE FIRST ADDRESS (THE
                                        ; FORWARD LINK).
        MOVAL   (R0)+,(R0)              ; STORE THE SECOND ADDRESS (THE
                                        ; BACKWARD LINK).
```

This example demonstrates a way to avoid having absolute virtual addresses stored as data. Both string descriptors used in the VAX-11 procedure calling standard and the list head for the INSQUE and REMQUE instructions require absolute virtual addresses. To make code position independent, the addresses must be stored by executable instructions rather than as data in the source code.

Example 4

```
; CREATING A POSITION-DEPENDENT DISPATCH TABLE
DISPATBL:                                   ; LIST OF
        .ADDRESS        ROUTIN0             ; ABSOLUTE VIRTUAL
        .ADDRESS        ROUTIN1             ; ADDRESSES
        .ADDRESS        ROUTIN2             ; CAUSING CODE TO BE
        .ADDRESS        ROUTIN3             ; POSITION DEPENDENT

; ROUTIN2 IS ENTERED BY THE FOLLOWING INSTRUCTIONS
        MOVL            #<2*4>,R3           ; GET OFFSET OF ADDRESS
                                            ; OF ROUTIN2
        JSB             @DISPATBL[R3]       ; ENTER ROUTIN2
;
; CREATING AN EQUIVALENT OFFSET LIST USING THE CASE INSTRUCTION
; SOURCE CODE IS POSITION INDEPENDENT
DISPAT: CASEB           R3,#0,#3            ; CASE INSTRUCTION
10$:    .SIGNED_WORD    ROUTIN0-10$         ; LIST OF OFFSETS
        .SIGNED_WORD    ROUTIN1-10$         ; FROM PC.
        .SIGNED_WORD    ROUTIN2-10$         ; CODE IS
        .SIGNED_WORD    ROUTIN3-10$         ; POSITION INDEPENDENT.

; ROUTIN2 IS ENTERED BY THE FOLLOWING INSTRUCTIONS
        MOVL            #2,R3               ; GET OFFSET OF ROUTIN2 IN
                                            ; LIST OF OFFSETS
        BSBB            DISPAT              ; ENTER ROUTIN2 USING CASE
                                            ; INSTRUCTION.
```

This example demonstrates another way to avoid storing absolute virtual addresses as data. The dispatch table is a list of entry points to routines. This is a frequently used way to enter one of a series of routines, but the code is position dependent. The same functionality can usually be provided in a position-independent way by using the CASE instruction, which transfers control to a routine based on an offset to the PC.

APPENDIX A

DIAGNOSTIC MESSAGES

If the assembler encounters an error during an assembly, it displays a diagnostic message on the terminal or batch log file and in the listing file (if there is one). The general format of VAX-11 MACRO diagnostic messages is:

```
%MACRO-l-code, text
```

l
: A severity level indicator. It has a value of E for an error or a value of W for a warning.

code
: An abbreviation of the message text; the message descriptions in this appendix are alphabetized by this code.

text
: The explanation of the message.

For example:

```
%MACRO-E-ILLMASKBITS, Reserved bits set in ENTRY mask
```

Some input and output diagnostic messages are followed by an RMS error message.

Listed below are the diagnostic messages displayed by the VAX-11 MACRO assembler. Each message is accompanied by an explanation of the cause of the error and recommended user action to correct the error.

ADRLSTSYNX, Address list syntax error

Explanation: The address list in the .ADDRESS directive contained a syntax error.

User Action: Correct the syntax.

Severity: Error

ALIGNXCEED, Alignment exceeds PSECT alignment

Explanation: The .ALIGN directive specified an alignment larger than the program section alignment. For example, the .PSECT directive specified byte alignment (the default) and the .ALIGN directive specified a longword alignment. This message can also be caused by a .PSECT directive with an illegal alignment.

User Action: Correct conflicting alignments. The .PSECT directive should specify the largest alignment required in the program section.

Severity: Error

ARGTOOLONG, Argument too long

Explanation: An argument was more than 512 characters long.

User Action: Reduce the length of the argument.

Severity: Error

ASCTOOLONG, ACSII string too long

Explanation: The string in an .ASCIC directive was longer than 255 characters or the string in an .ASCID directive was more than 65535 characters.

User Action: Reduce the length of the string.

Severity: Error

ASGNMNTSYNX, Assignment syntax error

Explanation: A direct assignment statement contained a syntax error.

User Action: Correct the syntax.

Severity: Error

BADENTRY, Bad format for .ENTRY statement

Explanation: The .ENTRY directive did not specify an entry point name and an entry mask.

User Action: Correct the .ENTRY directive syntax.

Severity: Error

BADLEXARG, Illegal lexical function argument

Explanation: The argument to a macro string operator was invalid. String arguments can be macro arguments or strings delimited by angle brackets or the circumflex delimiters. Symbol arguments can be absolute symbols or decimal integers.

User Action: Correct the argument syntax.

Severity: Error

BADLEXFORM, Illegal format for lexical function

Explanation: The macro string operator contained a syntax error.

User Action: Correct the macro string operator syntax.

Severity: Error

BADLOGICPC, Internal logic error detected at PC xxxxx

Explanation: There was an internal error in the VAX-11 MACRO assembler; xxxxx indicates the value of the PC at the time the error was detected. The assembler does not produce an object module or listing file.

User Action: Retry the assembly. If the error is reproducible, notify your system manager to submit a Software Problem Report (SPR). The address displayed with the error message and the source program should be included in the SPR.

Severity: Error

BADVALUE, xxxxx is an invalid keyword value

Explanation: A command qualifier had an illegal value; xxxxx indicates the value specified in the command. The assembler does not produce an object module or a listing file.

User Action: Reenter the command with the correct syntax.

Severity: Error

BLKDIRSYNX, Block directive syntax error

Explanation: A conditional block or a repeat block directive contained a syntax error.

User Action: Correct the directive syntax.

Severity: Error

BRDESTRANGE, Branch destination out of range

Explanation: The address specified in the branch instruction was too far away from the current PC. Branch instructions with byte displacements have a range of from -128 bytes to +127 bytes from the current PC. Branch instruction with word displacements have a range of from -32768 bytes to +32767 bytes from the current PC.

User Action: Use a branch instruction with a word displacement instead of one with a byte displacement; use a jump (JMP) instruction instead of a branch instruction; or change the program logic so that the branch destination is closer to the branch instruction.

Severity: Error

CANTFINDMAC, Can't locate macro in macro libraries

Explanation: A macro name specified in a .MCALL directive was not defined in the macro libraries searched.

User Action: Specify, in the MACRO command, the macro library that defines the macro.

Severity: Error

CLOSEIN, Error closing file-spec as input

Explanation: The assembler encountered an I/O error when closing an input source or macro library file; file-spec is the file specification of the file being closed.

User Action: Retry the operation or make a new copy of the file and retry the operation with the copy.

Severity: Error

CLOSEOUT, Error closing file-spec as output

Explanation: The assembler encountered an I/O error when closing an output object or listing file; file-spec is the file specification of the file being closed.

User Action: Retry the operation. If the error is reproducible, notify your system manager.

Severity: Error

DATALSTSYNX, Data list syntax error

Explanation: The data list in the directive contained a syntax error. For example, the directive .LONG 3,,5 contains a data list syntax error because there is no data item between the two commas.

User Action: Correct the syntax of the data list.

Severity: Error

DATATRUNC, Data truncation error

Explanation: The specified value did not fit in the given data type. The assembler truncated the value so that it fit.

User Action: Reduce the value or the number of characters in an ASCII string or change the data type.

Severity: Warning

DIRSYNX, Directive syntax error

Explanation: The directive contained a syntax error.

User Action: Correct the syntax of the directive.

Severity: Error

DIVBYZERO, Division by zero error

Explanation: An expression contained a division by 0.

User Action: Change the values in the expression.

Severity: Warning

EMSKNOTABS, Entry mask not absolute

Explanation: The entry mask expression was not absolute or contained undefined symbols.

User Action: Change the values in the expression.

Severity: Error

ENDWRNGMAC, Statement ends wrong MACRO

Explanation: The .ENDM directive specified a different name than its corresponding .MACRO directive.

User Action: Correct the name in the .ENDM directive to ensure that the .ENDM directive and .MACRO directive correspond as required.

Severity: Error

EXPOVR32, Expression overflowed 32-bits

Explanation: The value of the expression could not be stored in a longword (32 bits). The assembler truncated the value to 32 bits.

User Action: Change the values in the expression.

Severity: Warning

FLTPNTSYNX, Floating point syntax error

Explanation: A floating-point constant contained a syntax error.

User Action: Correct the syntax of the constant.

Severity: Warning

GENERR, Generated ERROR: xxxxx message

Explanation: A .ERROR directive was assembled; xxxxx is the value of the expression specified in the directive; and message is the text specified in the directive.

User Action: Follow the instructions in the message.

Severity: Error

GENWRN, Generated WARNING: xxxxx message

Explanation: A .WARN directive was assembled; xxxxx is the value of the expression specified in the directive; and message is the text specified in the directive.

User Action: Follow the instructions in the message.

Severity: Warning

IFDIRSYNX, IF directive syntax error

Explanation: A conditional assembly directive contained a syntax error.

User Action: Correct the syntax of the directive.

Severity: Error

IFEXPRNTABS, IF expression not absolute

Explanation: The expression in a .IF directive was not an absolute expression or contained undefined symbols.

User Action: Change the values in the expression.

Severity: Error

IFLEVLXCED, IF nesting level exceeded

Explanation: The assembler encountered more than 31 levels of nested conditional assembly blocks.

User Action: Restructure the program to decrease nesting of conditional assembly blocks.

Severity: Error

ILLARGDESC, Illegal operand argument descriptor

Explanation: The operand descriptor in an .OPDEF directive was invalid.

User Action: Use one of the valid operand descriptors.

Severity: Error

ILLASCARG, Illegal ASCII argument

Explanation: The argument to an .ASCIx directive did not have enclosing delimiters or an expression was not enclosed in angle brackets.

User Action: Correct the syntax of the argument.

Severity: Error

ILLBRDEST, Illegal branch destination

Explanation: The destination of a branch instruction was not an address, for example, BRB 10(R9).

User Action: Change the destination of the branch instruction or use a jump (JMP) instruction.

Severity: Error

ILLCHR, Illegal character

Explanation: The source line contained a character that was illegal in its context.

User Action: Delete the illegal character.

Severity: Error

ILLDFLTARG, Illegal argument for .DEFAULT directive

Explanation: A .DEFAULT directive did not specify DISPLACEMENT or the displacement specified was not BYTE, WORD, or LONGWORD.

User Action: Correct the .DEFAULT directive.

Severity: Error

ILLEXPR, Illegal expression

Explanation: A radix unary operator was not followed by a number, or left and right angle brackets did not match in an expression.

User Action: Correct the syntax of the expression.

Severity: Error

ILLIFCOND, Illegal IF condition

Explanation: The condition specified in a conditional assembly was not a valid condition, or there were no symbols after a DIFFERENT or IDENTICAL condition.

User Action: Correct the syntax of the conditional assembly directive.

Severity: Error

ILLINDXREG, Invalid index register

Explanation: The base mode changed the value of the register and the index register was the same as the register in the base mode; the base mode was literal or immediate mode; or PC was used as the index register.

User Action: Correct the addressing mode.

Severity: Error

ILLMACARGNM, Illegal MACRO argument name

Explanation: The name in the .MACRO directive contained an illegal character.

User Action: Delete the illegal character.

Severity: Error

ILLMACNAM, Illegal MACRO name

Explanation: No macro name was specified in the .MACRO directive.

User Action: Specify the macro name in the .MACRO directive.

Severity: Error

ILLMASKBITS, Reserved bits set in ENTRY mask

Explanation: The register save mask in an .ENTRY or .MASK directive specified R0, R1, AP, or PC registers (corresponding to bits 0, 1, 12, and 13).

User Action: Remove these registers from the register save mask.

Severity: Error

ILLMODE, Illegal mode

Explanation: An invalid addressing mode for the instruction was specified.

User Action: Specify a legal addressing mode.

Severity: Error

ILLOPDEF, Illegal format for .OPDEF

Explanation: The .OPDEF directive had incorrect syntax.

User Action: Correct the .OPDEF directive syntax.

Severity: Error

ILLOPDEFVAL, Illegal value for opcode definition

Explanation: The value specified in the .OPDEF directive did not fit in two bytes.

User Action: Correct the value in the directive.

Severity: Error

ILLREGHERE, This register may not be used here

Explanation: This register cannot be used here, for example, PUSHL (PC).

User Action: Use another register.

Severity: Error

ILLREGNUM, Illegal register number

Explanation: A register name was not in the range R0 through R12 or was not the AP, FP, SP, OR PC register name.

User Action: Correct the illegal register name.

Severity: Error

ILLSYMLEN, Symbol exceeds 15 characters

Explanation: The symbol name was longer than 15 characters. The assembler truncated the name to 15 characters.

User Action: Truncate the name to 15 characters.

Severity: Warning

INSVIRMEM, Insufficient virtual memory

Explanation: The module being assembled has too many symbols and macro definitions for the virtual memory available or a macro definition called itself (a recursive definition). The assembler terminated the assembly.

User Action: Increase the virtual memory available by contacting the system manager; reduce the level of macro nesting; split the module into several smaller modules; or eliminate the recursive macro definition.

Severity: Error

INVALIGN, Invalid alignment

Explanation: No integer or keyword followed the .ALIGN directive.

User Action: Correct the syntax of the .ALIGN directive.

Severity: Error

LINETOOLONG, Line too long

Explanation: A source line in a macro definition was longer than 500 characters.

User Action: Restructure the source code so that the line is shorter.

Severity: Error

MACLBFMTERR, Macro library format error

Explanation: A format error occurred in the macro library.

User Action: Retry the assembly and, if the error still occurs, use the LIBRARY command (see the VAX/VMS Command Language User's Guide) to re-create the library from the source code.

Severity: Error

MAYNOTINDEX, This mode may not be indexed

Explanation: The base mode was register, immediate, or literal mode.

User Action: Change the addressing mode.

Severity: Error

MCHINSTSYNX, Machine instruction syntax error

Explanation: A syntax error occurred in an instruction, for example, MOVL, A.

User Action: Correct the instruction syntax.

Severity: Error

MISSINGEND, Missing .END statement

Explanation: There was no .END directive at the end of the module. The assembler inserted an .END directive after the last line.

User Action: Insert a .END directive.

Severity: Warning

MSGCMAIIF, Missing comma in .IIF statement

Explanation: The condition was not separated from the statement in an .IIF directive.

User Action: Insert a comma in the directive.

Severity: Error

MULDEFLBL, Multiple definition of label

Explanation: The same label was defined twice in the module.

User Action: Delete the second label definition or change one of the labels to a different symbol name.

Severity: Error

NOFORMALARG, No formal argument for .IRP/.IRPC

Explanation: There were no formal arguments in an .IRP or .IRPC directive.

User Action: Correct the syntax of .IRP or .IRPC directive.

Severity: Error

NOTDECSTRNG, Illegal character in decimal string

Explanation: A decimal string contained a character other than the digits 0 through 9 and a leading plus or minus sign.

User Action: Correct the syntax of the decimal string.

Severity: Error

NOTENABLOPT, Not a legal ENABLE option

Explanation: An argument to a .ENABLE or .DISABLE directive was not a legal option.

User Action: Delete the option or replace it with a legal option.

Severity: Error

NOTENUFOPR, Not enough operands supplied

Explanation: The instruction requires more operands than were specified in the statement.

User Action: Add the operands or change the instruction.

Severity: Error

NOTINANIF, Statement outside condition body

Explanation: A .IF_FALSE, .IF_TRUE, .IF_TRUE_FALSE, .IFF, .IFT, or .IFTF subconditional directive was not in a conditional assembly block.

User Action: Replace the subconditional directive with a conditional directive or delete the subconditional directive.

Severity: Error

NOTINMACRO, Statement not in MACRO body

Explanation: The .NARG directive was not in a macro definition or expansion.

User Action: Delete or move the line containing the .NARG directive.

Severity: Error

NOTLGLISTOP, Not a legal listing option

Explanation: The argument to a .SHOW, .NOSHOW, .LIST, or .NLIST directive was not a legal option.

User Action: Delete the illegal option or replace it with a legal option.

Severity: Error

NOTPSECTOPT, Not a valid PSECT option

Explanation: The attribute specified in the .PSECT directive was invalid.

User Action: Delete the invalid attribute or replace it with a valid one.

Severity: Error

OPENIN, Error opening file-spec as input

Explanation: The assembler encountered an I/O error when opening an input source or macro library file; file-spec is the file specification of the file being opened. This message is produced when the file cannot be found.

User Action: Retry the assembly or make a new copy of the input file and then retry the assembly.

Severity: Error

OPENOUT, Error opening file-spec as output

Explanation: The assembler encountered an I/O error when opening an output object module or listing file; file-spec is the file specification of the file being opened. This message is produced when the device is write locked or is not mounted.

User Action: Retry the assembly and, if the error is reproducible, notify your system manager.

Severity: Error

OPRNDSYNX, Operand syntax error

Explanation: An operand contained a syntax error.

User Action: Correct the operand syntax.

Severity: Error

PACKTOOLONG, Packed decimal string too long

Explanation: The numeric string in a .PACKED directive had more than 31 digits.

User Action: Reduce the length of the decimal string.

Severity: Error

PSECOPCNFLC, Conflicting PSECT options

Explanation: The values specified in a .PSECT directive conflicted with each other or were not the same as the values specified in a preceding .PSECT directive that specified the same program section name.

User Action: Correct the conflicting values in the .PSECT directive(s).

Severity: Error

PSECTBUFOVF, PSECT context buffer overflow

Explanation: The .SAVE_PSECT directive attempted to save a program section context when the program section context buffer was filled. A maximum of 31 program section contexts can be saved in the buffer.

User Action: Reduce the amount of program section nesting.

Severity: Error

PSECTBUFUND, PSECT context buffer underflow

Explanation: The .RESTORE_PSECT directive attempted to restore a program section context when the program section context buffer was empty.

User Action: Ensure that each .RESTORE_PSECT directive corresponds to a .SAVE_PSECT directive.

Severity: Error

READERR, error reading file-spec

Explanation: The assembler encountered an I/O error when reading an input source or macro library file; file-spec is the file specification of the file being read.

User Action: Retry the assembly, or create a new copy of the input file and then retry the assembly.

Severity: Error

REGOPSYNX, Register operand syntax error

Explanation: The addressing mode syntax contained an error.

User Action: Correct the addressing mode syntax.

Severity: Error

RMSERROR, RMS service error

Explanation: The assembler encountered an error during a VAX-11 RMS operation.

User Action: Retry the operation; consult the VAX-11 Record Management Services Reference Manual for more information.

Severity: Error

RPTCNTNTABS, Repeat count not absolute

Explanation: The repeat count in a .BYTE, .WORD, .LONG, .SIGNED_BYTE, or .SIGNED_WORD directive contained an undefined symbol or was a relative expression.

User Action: Replace the expression with an absolute expression that does not contain any undefined symbols.

Severity: Error

SYMDCLEXTRN, Symbol declared external

Explanation: A label definition or direct assignment statement specified a symbol that was previously declared external in a .EXTERNAL directive.

User Action: Delete the external declaration or change the name of the internal symbol.

Severity: Error

SYMDEFINMOD, Symbol is defined in module

Explanation: A .EXTERNAL directive specified a label that was previously defined in the module.

User Action: Delete the external declaration or rename the internal symbol.

Severity: Error

SYMNOTABS, Symbol is not absolute

Explanation: The argument in a macro string operator was a relative symbol or was undefined.

User Action: Ensure that symbol is defined as an absolute symbol.

Severity: Error

SYMOUTPHASE, Symbol out of phase

Explanation: A label definition specified a label that was defined later in the module; or a local label definition specified a local label that was defined later in the same local label block.

User Action: Ensure that the label is defined only once in the module or that the local label is defined only once in the local label block.

Severity: Error

TEXT, No input file given

Explanation: The macro command did not contain any source files; it contained only macro library files.

User Action: Specify a source file in the command line.

Severity: Error

TOOMNYARGS, Too many arguments in MACRO call

Explanation: The macro call contained more arguments than were specified in the .MACRO directive in the macro definition.

User Action: Ensure that the macro call corresponds to the macro definition.

Severity: Error

TOOMNYOPRND, Too many operands for instruction

Explanation: Too many operands were specified for the instruction.

User Action: Reduce the number of operands.

Severity: Error

TOOMNYPSECT, Too many PSECTs declared

Explanation: More than 255 user-defined program sections were declared.

User Action: Reduce the number of program sections.

Severity: Error

UNDEFSYMBOL, Undefined symbol

Explanation: A local label was referred to but not defined in a local label block; or, if GLOBAL was disabled, a symbol was referred to but not defined in the module or specified in a .EXTERNAL directive.

User Action: Define the local label or symbol, or specify the symbol in a .EXTERNAL directive.

Severity: Error

UNDEFXFRADR, Undefined transfer address

Explanation: The .END directive specified a transfer address that was not defined in the module or specified in a .EXTERNAL directive.

User Action: Define the transfer address or delete it from the .END directive.

Severity: Error

UNPROQUAL, Unprocessed qualifiers

Explanation: Either the /SHOW or the /CROSS qualifier was specified without the /LIST qualifier. The assembler does not process the source file or produce an object module.

User Action: Reenter the command with the /LIST qualifier.

Severity: Error

UNRECSTMT, Unrecognized statement

Explanation: The operator was not an opcode, directive, user-defined opcode, previously defined macro, or macro in a library.

User Action: Change the operator to a valid opcode, directive, or macro; or define the macro.

Severity: Error

UNTERMARG, Unterminated argument

Explanation: The string argument was missing a delimiter or the macro argument was missing an angle bracket.

User Action: Add the delimiter or angle bracket.

Severity: Error

UNTERMCOND, Unterminated conditional

Explanation: A conditional assembly block was not terminated by a .ENDC directive. The assembler inserted a .ENDC directive before the .END directive.

User Action: Add the .ENDC directive.

Severity: Error

WRITEERR, Error writing file-spec

Explanation: The assembler encountered an I/O error when writing to the output object module or listing file; file-spec is the file specification of the file being written.

User Action: Retry the assembly. If the error is reproducible, notify your system manager.

Severity: Error

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

W

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code_____
or
Country

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

RT/C SOFTWARE PUBLICATIONS TW/A14
DIGITAL EQUIPMENT CORPORATION
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS 01876

— Do Not Tear - Fold Here —