digital

VAX-11 MACRO
User's Guide

Order No. AA-D033C-TE

VAX11

March 1980

This document contains information required by an assembly language programmer to assemble VAX-11 MACRO programs and to use the VAX-11 MACRO assembly language efficiently.

VAX-11 MACRO
User's Guide

Order No. AA-D033C-TE

SUPERSESSION/UPDATE INFORMATION:	This revised document supersedes the VAX-11 MACRO User's Guide (Order No. AA-D033B-TE)
OPERATING SYSTEM AND VERSION:	VAX/VMS V02
SOFTWARE VERSION:	VAX/VMS V02

digital equipment corporation · maynard, massachusetts

First Printing, August 1978
Revised, February 1979
Revised, March 1980

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

CONTENTS

FIGURES

TABLES

PREFACE

MANUAL OBJECTIVES

The VAX-11 MACRO User's Guide describes how to use the VAX-11 MACRO assembler. The manual 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. The VAX-11 instruction set is described in the VAX-11 Architecture Handbook.

INTENDED AUDIENCE

This manual is intended for all VAX-11 MACRO programmers. It assumes that the reader has had some assembly language programming experience and has read the VAX/VMS Primer. Chapter 6 of this guide is intended for experienced VAX-11 MACRO programmers who want to create shareable images.

STRUCTURE OF THIS DOCUMENT

This manual is organized into six 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 the listing file produced by VAX-11 MACRO.
- Chapter 4 describes the essential elements of VAX-11 MACRO programs.
- Chapter 5 provides an overview of features of VAX-11 MACRO that allow programs to be modular and easy to understand.
- Chapter 6 describes how to write code for use in shareable images.
- Appendix A lists and explains the VAX-11 MACRO diagnostic messages.

ASSOCIATED DOCUMENTS

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

- VAX-11 MACRO Language Reference Manual
- VAX-11 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 and Index.

CONVENTIONS USED IN THIS DOCUMENT

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

Convention	Meaning
Uppercase words and letters	Uppercase words and letters, used in examples, indicate that you should type the word or letter exactly as shown.
Lowercase words and letters	Lowercase words and letters, used in format examples, indicate that you are to substitute a word or value of your choice.
quotation marks apostrophes	The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark.
[ ]	Square brackets indicate that the enclosed item is optional.
{ }	Braces are used to enclose lists from which one element is to be chosen.
...	A horizontal ellipsis indicates that the preceding item(s) can be repeated one or more times.
. . .	A vertical ellipsis indicates that not all of the statements in an example or figure are shown.

Convention	Meaning
RET or <RET>	A symbol with a 1- to 3-character abbreviation indicates that you press a key on the terminal, for example, RET .
CTRL/X or <CTRL/x>	The phrase <CTRL/x> indicates that you must press the key labeled CTRL while you simultaneously press another key, for example <CTRL/C>, <CTRL/Y>, <CTRL/O>. In examples, this control key sequence is shown as ^x, for example ^C, ^Y, ^O, because that is how the system echoes control key sequences.

Unless otherwise noted, all numeric values are represented in decimal notation.

Unless otherwise specified, you terminate commands by pressing the RETURN key.

SUMMARY OF TECHNICAL CHANGES

This manual documents VAX-11 MACRO Version 2.0, as released with Version 2.0 of VAX/VMS. This section summarizes the technical changes in the use of the assembler from earlier versions.

The /UPDATE qualifier has been added to the MACRO command to control the processing of update files and the audit trail listing.

The format of the listing file has been changed to accommodate the audit trail and update lines.

Technical changes in the VAX-11 MACRO language are documented in the _VAX-11 MACRO Language Reference Manual_.

CHAPTER 1

OVERVIEW OF ASSEMBLY

A VAX-11 MACRO source program is a sequence of assembly language statements. These statements may include instructions from the VAX-11 instruction set, which manipulate data (plus the data needed by these instructions), or assembler directives, which guide the assembly process. The VAX-11 instruction set is described in the VAX-11 Architecture Handbook; the VAX-11 MACRO assembler directives are described in the VAX-11 MACRO Language Reference Manual.

Before you can run a VAX-11 MACRO program on your system, its assembly language statements must be translated into a form that your machine understands: object code. Translation is the function of the VAX-11 MACRO assembler. The object code may need to be clarified or amplified by information or instructions from another program. Putting together different programs is the function of the VAX-11 Linker. Linked programs form an executable image, which can be executed using the DCL command RUN.

This chapter provides an overview of the functions of the assembler and linker. Details of the assembly process are presented in Chapter 2 of this manual. Details of the linking process are presented in the VAX-11 Linker Reference Manual. The LINK and RUN commands are described in the VAX/VMS Command Language User's Guide.

Figure 1-1 summarizes the process by which a source program is converted into an executable image.

1.1 ASSEMBLING VAX-11 MACRO PROGRAMS

Because you originally use an editor to create a VAX-11 MACRO source 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 code, called 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. You can request the VAX-11 MACRO assembler to produce a listing of the source program at the same time your program is assembled. Figure 1-2 illustrates the role of the assembler.

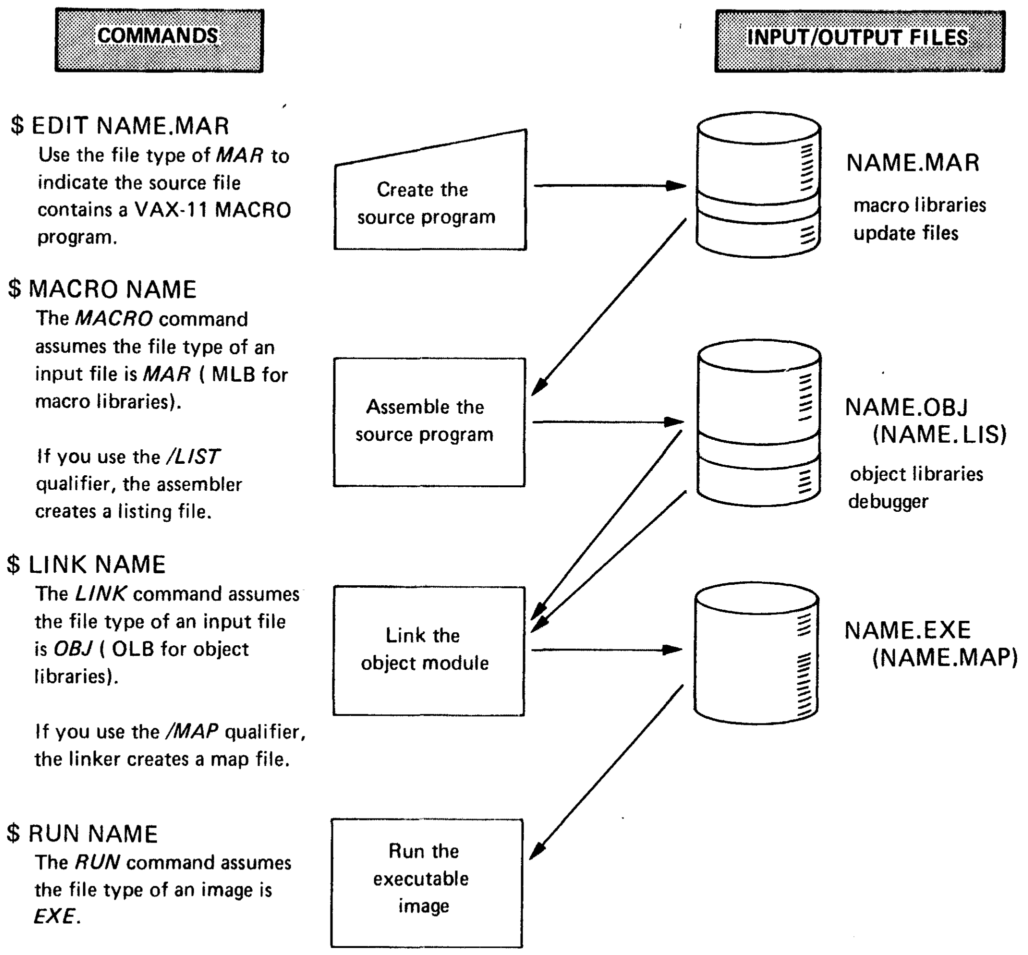
COMMANDS
INPUT/OUTPUT FILES
$ EDIT NAME.MAR
Use the file type of *MAR* to indicate the source file contains a VAX-11 MACRO program.
$ MACRO NAME
The *MACRO* command assumes the file type of an input file is *MAR* (MLB for macro libraries).
If you use the */LIST* qualifier, the assembler creates a listing file.
$ LINK NAME
The *LINK* command assumes the file type of an input file is *OBJ* (OLB for object libraries).
If you use the */MAP* qualifier, the linker creates a map file.
$ RUN NAME
The *RUN* command assumes the file type of an image is *EXE*.
Create the source program
Assemble the source program
Link the object module
Run the executable image
NAME.MAR
macro libraries
update files
NAME.OBJ
(NAME.LIS)
object libraries
debugger
NAME.EXE
(NAME.MAP)

Figure 1-1 Developing a VAX-11 MACRO Program

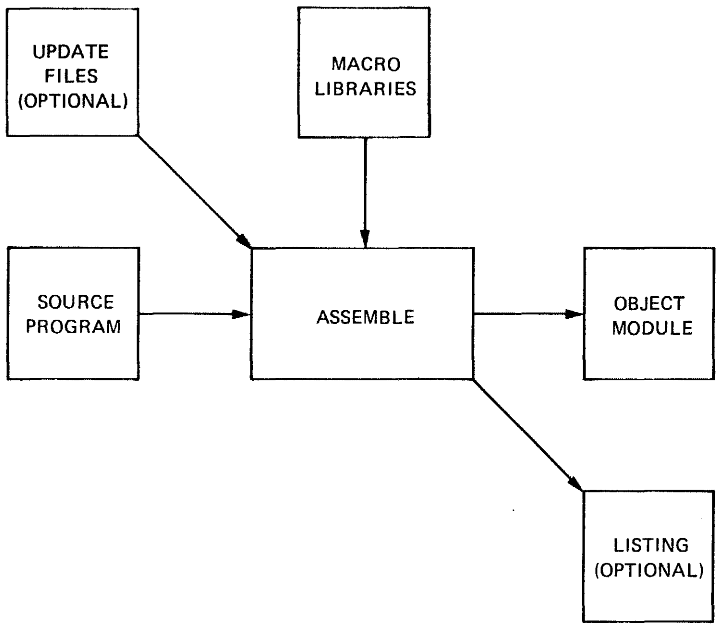

Figure 1-2 Function of the VAX-11 MACRO Assembler

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
- Optionally updates files by adding, replacing, or deleting lines
- Optionally keeps track of the status of updated lines by means of an audit trail
- Keeps track of all user-defined symbols and their respective values in a symbol table, described in Section 5.3.
- Converts assembly language mnemonics, user-defined symbols, and data values into their respective machine language (object code) equivalents

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 storage location counter accordingly. The linker will convert these relative storage locations 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 Chapter 3 for more information on the listing file.

1.2 LINKING VAX-11 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 the object modules that use symbols with the object modules that define them (See Section 1.2.1)
- Relocates individual object modules as necessary and assigns virtual memory addresses (See Section 1.2.2)
- Produces an executable image and an optional map, as shown in Figure 1-3

The link operation, in addition to joining object modules, 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.

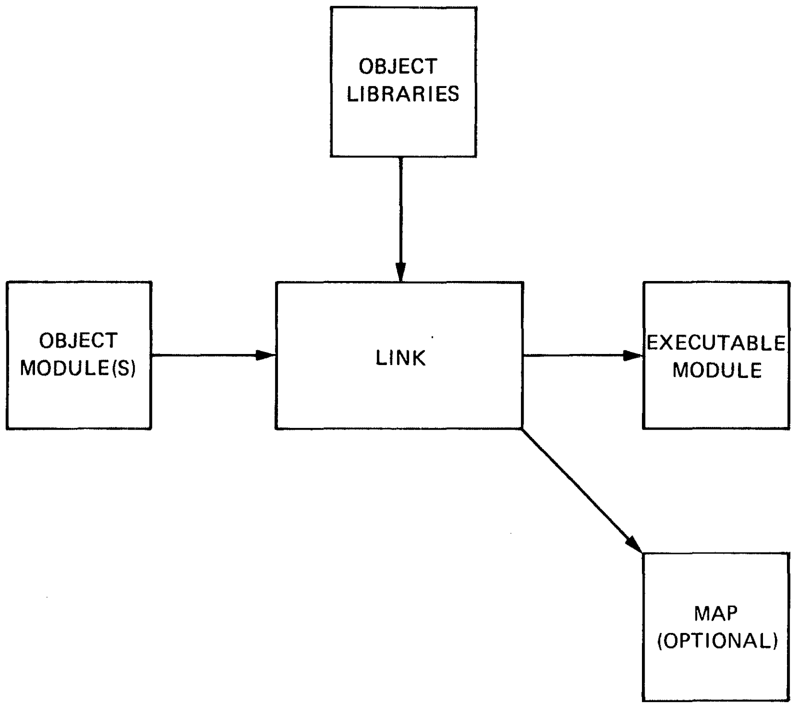

Figure 1-3 Link Functions

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 and VAX-11 BASIC. 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 one subprogram to perform data acquisition in VAX-11 MACRO and other subprograms to perform data analysis or file input/output in VAX-11 FORTRAN.

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.

1.2.1 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 global symbols) 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.

1.2.2 Program Relocation and Address Assignment

A second important function of the linker is to "fix" relative addresses in memory 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.

1.3 DEBUGGING VAX-11 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 have produced invalid results. (For information about diagnostic messages produced by VAX-11 MACRO, see Section 2.4 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 your program using 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

THE MACRO COMMAND

The MACRO command, typed in response to the DIGITAL Command Language (DCL) prompt, 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. This chapter describes the format of the MACRO command and Chapter 3 describes the listing file.

2.1 CONSTRUCTING THE COMMAND STRING

Format

```
     $ MACRO[/qualifier(s)] file-spec[/qualifier(s)]
```

/qualifiers

Command or file qualifiers that indicate special actions to be performed by the assembler (see Section 2.3).

file-spec

A file specification or list of file specifications that specify the source program and macro library input files to be assembled into object modules (see Section 2.2). 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 program 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.

When you invoke the assembler from your terminal 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.MAR, PART2.MAR, and PART3.MAR and assembles them into one object module with a name of PART1.OBJ. No listing file is created.

2. `$ MACRO/LIST APROG,BPROG,CPROG`

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

3. `$ MACRO MYPROG/LIST+MLIB/LIBRARY`

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

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

2.2 FILE SPECIFICATIONS

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

Format

```
device:[directory]filename.type;version
```

device

The physical device on which a file is stored or is to be written. The device name takes the form devcu, where:

```
dev = device code
c = controller designator
u = unit number
```

[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.

type

The type of the file, describing the kind of data in the file; type 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, controller, and unit	
directory	User's current default directory	
type	Depends on usage:	
	Source input file	MAR
	Macro library file	MLB
	Object module	OBJ
	Listing file	LIS
	Update file	UPD
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.

2.3 QUALIFIERS

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

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

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_REFERENCE[=]	ALL DIRECTIVES MACROS OPCODES REGISTERS SYMBOLS	-- DIR MAC OPC REG SYM	/NOCROSS_REFERENCE	/NOCROSS_REFERENCE
/DISABLE=	ABSOLUTE DEBUG GLOBAL SUPPRESSION TRACEBACK TRUNCATION	AMA DBG GBL SUP TBK FPT	/ENABLE=	/DISABLE= (AMA,DBG,LSB,SUP,FPT)
/ENABLE=	ABSOLUTE DEBUG GLOBAL SUPPRESSION TRACEBACK TRUNCATION	AMA DBG GBL SUP TBK FPT	/DISABLE=	/ENABLE=(GBL,TBK)
/LIBRARY	--	--	--	Not a library
/LIST[=]	file-spec	--	/NOLIST	/NOLIST (interactive mode) /LIST (batch mode)
/OBJECT[=]	file-spec	--	/NOOBJECT	/OBJECT
/SHOW[=]	BINARY CALLS CONDITIONALS DEFINITIONS EXPANSIONS	MEB MC CND MD ME	/NOSHOW[=]	/SHOW=(MC,CND,MD)
/UPDATE	--	--	--	No updates

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

2.3.1 The /CROSS_REFERENCE and /NOCROSS_REFERENCE Qualifiers

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

See Section 3.6 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.

Format

```
/CROSS_REFERENCE[=functions]
/NOCROSS_REFERENCE
```

Parameter

functions

Any of the functions listed in Table 2-3. You can specify either the long form or the short form of the function. If you specify multiple functions, you must separate them by commas and enclose them in parentheses. If you specify the /CROSS_REFERENCE qualifier without any functions, it is equivalent to specifying /CROSS_REFERENCE=(MAC,SYM).

Table 2-3
/CROSS_REFERENCE 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

2.3.2 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.

Format

```
/ENABLE=function(s)
/DISABLE=function(s)
```

function(s)

At least one of the functions listed in Table 2-4 must be specified when you use /ENABLE or /DISABLE. You can specify either the long form or the short form of the function. If you specify multiple functions, you must separate them by commas and enclose them in parentheses.

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

2.3.3 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.

2.3.4 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.

Format

```
/LIST[=file-spec]
/NOLIST
```

file-spec

The file specification to be used 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.

2.3.5 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.

Format

```
/OBJECT[=file-spec]
/NOOBJECT
```

file-spec

The file specification to be used for the object output 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. The default file type is OBJ.

2.3.6 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.

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.

Format

```
/SHOW[=function(s)]
/NOSHOW[=function(s)]
```

function(s)

Any of the qualifier functions listed in Table 2-5. Either the long form or the short form of the function can be used. If multiple functions are specified, they must be separated by commas and enclosed in parentheses.

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

2.3.7 The /UPDATE Qualifier

The /UPDATE qualifier enables the assembler to apply updates to the source file for which /UPDATE was specified. The updates are described in SUMSLP format command files, and the assembler performs the edit using a procedure similar to the batch-oriented text editor, SUMSLP. The /UPDATE qualifier is used only as a file qualifier.

Format

```
/UPDATE=(file-spec(s))
```

file-spec(s)

Specification of the file or files that are to be used to update the file qualified by the /UPDATE qualifier. The file type of these update files defaults to UPD. Parentheses need not be used if only one update file is specified.

When /UPDATE is specified and /LIST is also enabled, the output listing file shows the updated object module, plus an audit trail indicating the location of each deletion, addition, or change. The audit trail is described in Section 3.3. The effect of /UPDATE on line numbers is described in Section 3.2. The /UPDATE qualifier was specified in the command string that produced the sample output listings in Section 3.8.

For information on SUMSLP, see the VAX-11 Utilities Reference Manual.

When multiple update files are specified with the /UPDATE qualifier, the assembler merges their contents into a single list of updates before applying the updates to the source file. The rules for merging update files are described in the VAX-11 Utilities Reference Manual.

The /UPDATE qualifier cannot be used with the /LIBRARY qualifier, because a macro library is not a single source file.

2.4 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.

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. 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.

code

An abbreviation of the message text.

text
: The explanation of the message.

For example:

```
%MACRO-E-ILLMASKBIT, 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, warnings, and information messages with the line number and page number (enclosed in parentheses) of each. At the end of the error summary, the assembler displays a list of the file specifications in the MACRO command (see Section 2.1).

An example of a diagnostic summary follows.

```
$ MACRO/LIST PROG

There were 6 errors, 1 warnings, and 0 information messages on lines:

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

CHAPTER 3

THE VAX-11 MACRO LISTING FILE

The listing file produced by VAX-11 MACRO can have up to seven parts:

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

Sections 3.1 through 3.7 describe each of these parts. Section 3.8.2 contains an example of a listing.

3.1 TABLE OF CONTENTS AND PAGE HEADINGS

If the source module contains any optional .SUBTITLE directives, VAX-11 MACRO will print 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

3.2 SOURCE STATEMENTS AND HEXADECIMAL CODE

This section is the main part of the listing: it contains the source lines of the module and the binary 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. If /UPDATE has been specified, the listing numbers will be sequence numbers (even if the original source file had editor-generated line numbers). Inserted lines will be indicated by line numbers containing decimal points.
- The location counter
- The hexadecimal code

The hexadecimal code is printed with the lowest address on the right. The 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
Octaword	1/2
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 2.4 for a description of the diagnostic message format and Appendix A for a list of the VAX-11 MACRO diagnostic messages.

3.3 AUDIT TRAIL

The audit trail, optionally produced when the /UPDATE qualifier is specified for a file, occupies columns 1 through 16 of the main part of the listing, parallel to the listings of source line, line number, and hexadecimal code. It follows the forms used by the SUMSLP editor. Lines will be flagged as **NEW** if they have been added or changed, unless another audit trail is specified in the update file. When lines have been deleted, the next line after the deletion will contain an audit trail entry of the form -n, where n is the number of lines deleted.

The chapter on SUMSLP in the VAX-11 Utilities Reference Manual contains more information about audit trails and examples of their use.

3.4 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 3.5 for information about program section numbers.

3.5 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.

3.6 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_REFERENCE qualifier. You can exclude certain symbols from the cross-reference listing by using the .CROSS and .NOCROSS directives.

3.7 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 various stages of the assembly. In addition, the indicators include (1) the working set limit and (2) 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.

3.8 ASSEMBLY LISTING EXAMPLE

Section 3.8.1 gives a brief example of the effect of the /UPDATE qualifier on the listing file. Section 3.8.2 shows the complete listing file generated by assembling a source program; the /UPDATE qualifier was specified in the command line that produced this listing.

3.8.1 Effect of the /UPDATE Qualifier

Figures 3-1 and 3-2 show a source file and an update file containing corrections. These programs are assembled by the VAX-11 MACRO assembler using the file qualifier /UPDATE. The source program was originally edited with the SOS editor, which produces line numbers. Note that the line numbers are changed to sequence numbers by the assembler; the update function recognizes sequence numbers only.

Figure 3-3 shows an excerpt from the listing of the updated source file. This listing resulted from the following VAX-11 MACRO command string:

THE VAX-11 MACRO LISTING FILE

```
MACRO/LIST/SHOW=ME  SOURCEFIL/UPDATE=NEWDATA
```

```
 100   .TITLE  QUOTAS
 200   .PSECT  STRING_SYMBOLS,NOEXE     ; SYMBOL DEFINITIONS
 300                                    ; MAX AND MIN FOR V1 AND V2
 400                                    ; MAX AND MIN IN ALTERNATE LINES
 500   V1MAX_H==10
 600   V1MIN_H==6
 700   V2MAX_H==11
 800   V2MIN_H==5
 900
1000   MACROS: .PSECT  MACRODEF
1100           .MACRO  GETMORE  NUM,CAT,?LABEL ;
1200           .SAVE_PSECT
1300           .PSECT  STRING,NOEXE
1400   LABEL:  .ASCID  /MORE CAT NEEDED, NUMBER IN NUM/        ; MESSAGE CONSTRUCTED
1500           .RESTORE_PSECT
1600           PUSHAL  LABEL                    ; MESSAGE ADDRESS PUT ON STACK
1700           CALLS   #1,LIB$PUT_OUTPUT        ; CALL PROCEDURE FROM COMMON
1800                                            ; RUN TIME LIBRARY TO OUTPUT
1900                                            ; MESSAGE
2000           .ENDM   GETMORE
2100
2200
2300
2400
2500           .PSECT  COMPARE                  ; CALCULATIONS HERE
2600           .ENTRY  CALC,^M<R6,R7>
2700   V1_CAL: .IF NOT_DEFINED V1MAX_T          ; IF NO MAX ALREADY FOR V1_T
2800           .WARN                            ; NO V1'S IN LIST PROVIDED!
2900           JMP     V2_CAL                   ; NO V1 CALCULATION POSSIBLE
3000           .ENDC
3100           SUBL3   #V1MAX_H,#V1MAX_T,R2
3200           .IF GREATER_EQUAL <V1MAX_T-V1MAX_H>       ; IF HERE IS LESS THAN THERE,
3300           GETMORE          R2,V1  ; CALL MACRO GETMORE
3400                                            ; WITH REGISTER AND V1 AS ARGUMENTS
3500           .ENDC
3600   V2_CAL: .IF NOT_DEFINED V2MAX_T          ; IF NO MAX ALREADY FOR V2_T
3700           .WARN                            ; NO V2'S IN LIST PROVIDED!
3800           JMP     END_CAL                  ; NO MORE CALCULATIONS POSSIBLE
3900           .ENDC
4000           SUBL3   #V2MAX_H,#V2MAX_T,R3
4100           .IF GREATER_EQUAL <V2MAX_T-V2MAX_H>       ; IF HERE IS LESS THAN THERE,
4200           GETMORE          R3,V2           ; CALL MACRO GETMORE WITH
4300                                            ; REGISTER AND V2 AS ARGUMENTS
4400           .ENDC
4500    END_CAL:
4600           MOVL    #1,R0
4700           RET
4800           .END    CALC
```

Figure 3-1 Source Program

```
-4
-,,/16NOV1979/
V1MAX_T==12
V1MIN_T==5
V2MAX_T==10
V2MIN_T==6
/
$EXIT
```

Figure 3-2 Update Program

```
QUOTAS                                                              15-NOV-1979 18:03:52  VAX-11 Macro V02.38         Page   1
                                                                    15-NOV-1979 17:58:13  _DB1:[MARGERY]SOURCEFIL.MAR;28    (1)
                                      0000     1         .TITLE  QUOTAS
                                  00000000     2         .PSECT  STRING_SYMBOLS,NOEXE     ; SYMBOL DEFINITIONS
                                      0000     3                                          ; MAX AND MIN FOR V1 AND V2
                                      0000     4                                          ; MAX AND MIN IN ALTERNATE LINES
16NOV1979                   0000000C  0000    .1 V1MAX_T==12
16NOV1979                   00000005  0000    .2 V1MIN_T==5
16NOV1979                   0000000A  0000    .3 V2MAX_T==10
16NOV1979                   00000006  0000    .4 V2MIN_T==6
                            0000000A  0000     5 V1MAX_H==10
                            00000006  0000     6 V1MIN_H==6
                            0000000B  0000     7 V2MAX_H==11
                            00000005  0000     8 V2MIN_H==5
                                      0000     9
                                  00000000    10 MACROS: .PSECT  MACRODEF
                                      0000    11         .MACRO  GETMORE NUM,CAT,?LABEL  ;
                                      0000    12         .SAVE_PSECT
                                      0000    13         .PSECT  STRING,NOEXE
                                      0000    14 LABEL:  .ASCID  /MORE CAT NEEDED, NUMBER IN NUM/        ; MESSAGE CONSTRUCTED
                                      0000    15         .RESTORE_PSECT
                                      0000    16         PUSHAL  LABEL                    ; MESSAGE ADDRESS PUT ON STACK
                                      0000    17         CALLS   #1,LIB$PUT_OUTPUT        ; CALL PROCEDURE FROM COMMON
                                      0000    18                                          ; RUN TIME LIBRARY TO OUTPUT
                                      0000    19                                          ; MESSAGE
                                      0000    20         .ENDM   GETMORE
                                      0000    21
                                      0000    22
                                      0000    23
                                  00000000    24         .PSECT  COMPARE                  ; CALCULATIONS HERE
                                00C0' 0000    25         .ENTRY  CALC,^M<R6,R7>
                                      0002    26 V1_CAL: .IF NOT_DEFINED V1MAX_T          ; IF NO MAX ALREADY FOR V1_T
                                      0002    27         .WARN                            ; NO V1'S IN LIST PROVIDED!
                                      0002    28         JMP     V2_CAL                   ; NO V1 CALCULATION POSSIBLE
                                      0002    29         .ENDC
                 52   0C   0A   C3    0002    30         SUBL3   #V1MAX_H,#V1MAX_T,R2
                            00000002  0006    31         .IF GREATER_EQUAL <V1MAX_T-V1MAX_H>      ; IF HERE IS LESS THAN THERE,
                                      0006    32         GETMORE         R2,V1    ; CALL MACRO GETMORE
                                      0006               .SAVE_PSECT
                                  00000000               .PSECT  STRING,NOEXE
                 00000008'010E0000'  0000       30000$: .ASCID  /MORE V1 NEEDED, NUMBER IN R2/  ; MESSAGE CONSTRUCTED I
                 56 20 45 52 4F 4D   0008
                 44 45 45 4E 20 31   000E
```

Figure 3-3 Exerpt from Listing of Updated Source Program

3.8.2 Complete Assembly Listing

The following is a complete assembly listing generated by the command:

```
MACRO/LIST/CROSS   MATH/UPDATE=REPLINES
```

```
MATH                             - Routine to do simple arithmetic       15-NOV-1979 18:48:11  VAX-11 Macro V02.38         Page  0
Table of contents
   (1)      32      Macro definitions
   (1)      57      Procedure entry point
```

```
MATH                             - Routine to do simple arithmetic       15-NOV-1979 18:48:11  VAX-11 Macro V02.38         Page  1
01                                                                       15-NOV-1979 18:46:12  _DB1:[MARGERY.MACNEW]MATH.MAR;3   (1)

                                          0000     1         .TITLE  MATH   - Routine to do simple arithmetic
                                          0000     2         .IDENT  /01/
                                          0000     3
                                          0000     4 ;++
                                          0000     5 ; FUNCTIONAL DESCRIPTION:
                                          0000     6 ;
                                          0000     7 ;       This routine accepts two integers and an operator index as
;**NEW**                                  0000    .1 ;       inputs, executes the requested arithmetic operation, and
-1                                        0000     9 ;       returns the result.
                                          0000    10 ;
                                          0000    11 ; INPUT:
                                          0000    12 ;
                                          0000    13 ;       4(AP)           First integer
                                          0000    14 ;       8(AP)           Second integer
                                          0000    15 ;       12(AP)          Operator index - 0-addition, 1-subtraction,
                                          0000    16 ;                       2-multiplication, 3-division
                                          0000    17 ;       16(AP)          Address of result
                                          0000    18 ;
                                          0000    19 ; OUTPUT:
                                          0000    20 ;
                                          0000    21 ;       The operation is executed and the result stored at the address
;**NEW**                                  0000    .1 ;       contained in 16<AP>.
-1                                        0000    23 ;
                                          0000    24 ;--
                                          0000    25
                                          0000    26         .ENABLE DEBUG                   ; Make symbols available to the
                                          0000    27                                         ; debugger
                                          0000    28
                                          0000    29         .DEFAULT DISPLACEMENT WORD      ; Use word displacements on
                                          0000    30                                         ; PC-relative references
                                          0000    31
                                          0000    32         .SUBTITLE       Macro definitions
                                          0000    33 ;++
                                          0000    34 ;       Define macro to use CASE instruction.
                                          0000    35 ;
```

```
                               0000    36 ;       CASE    SRC,DISPLIST,TYPE,LIMIT,NMODE
                               0000    37 ;
                               0000    38 ;       Where:
                               0000    39 ;
                               0000    40 ;       SRC             Case selector
                               0000    41 ;       DISPLIST        List of displacements
                               0000    42 ;       LIMIT           Base value of the selector
                               0000    43 ;       TYPE            B-byte, W-word (default), L-long
                               0000    44 ;
                               0000    45 ;--
                               0000    46         .MACRO  CASE,SRC,DISPLIST,TYPE=W,LIMIT=#0,NMODE=S^#,?BASE,?MAX
                               0000    47         CASE'TYPE       SRC,LIMIT,NMODE'<<MAX-BASE>/2>-1
                               0000    48                                         ; Case instruction
                               0000    49 BASE:                                   ; Local label used to count args
                               0000    50         .IRP    EP,<DISPLIST>           ; To set up offset list
                               0000    51         .SIGNED_WORD    EP-BASE         ; Offset list
                               0000    52         .ENDR                           ;
                               0000    53 MAX:                                    ; Local label used to count args
                               0000    54         .ENDM   CASE
                               0000    55
                               0000    56
                               0000    57         .SUBTITLE       Procedure entry point
```

```
MATH                       - Routine to do simple arithmetic       15-NOV-1979 18:48:11  VAX-11 Macro V02.38        Page  2
01                          Procedure entry point                  15-NOV-1979 18:46:12  _DB1:[MARGERY.MACNEW]MATH.MAR;3   (1)
                           00000000    58         .PSECT  RO_CODE,EXE,NOWRT
                               0000    59 ;
                               0000    60 ; Get the arguments from the argument list, perform the calculation
                               0000    61 ; and return the result in the fourth argument and the status in R0
                               0000    62 ;
                          001C' 0000   63         .ENTRY  MATH,^M<R2,R3,R4>       ; Routine ENTRY point
          50   0000'8F    3C   0002    64         MOVZWL  #SS$_NORMAL,R0          ; For success, SS$_NORMAL
                               0007    65                                         ; is defined in $SSDEF
            52    04 AC   D0   0007    66         MOVL    4(AP),R2                ; Get argument from argument list
                               000B    67                                         ; R2 contains first argument
            53    08 AC   D0   000B    68         MOVL    8(AP),R3                ; R3 contains second argument
            54    0C AC   D0   000F    69         MOVL    12(AP),R4               ; R4 contains operator index
                               0013    70         CASE    R4,-                    ; Dispatch to evaluation routine
                               0013    71                 <ADD,SUB,MUL,DIV>       ;
                      44  11   001F    72         BRB     ERR                     ; Calling routine specified an
                               0021    73                                         ; illegal operator index
               53     52  C1   0021    74 ADD:    ADDL3   R2,R3,@16(AP)           ; (0) Calculate the sum
                  10 BC        0024
                      3D  1D   0026    75         BVS     ERR                     ; An overflow occurred
                               0028    76         $EXIT_S                         ; Return to calling program
               52     53  C3   0031    77 SUB:    SUBL3   R3,R2,@16(AP)           ; (1) Form the difference
                  10 BC        0034
                      2D  1D   0036    78         BVS     ERR                     ; An overflow occurred
                               0038    79         $EXIT_S                         ; Return to calling program
                          53   52   C5  0041    80 MUL:    MULL3   R2,R3,@16(AP)           ; (2) Calculate the product
                             10 BC      0044
                                1D  1D  0046    81         BVS     ERR                     ; An overflow occurred
                                        0048    82         $EXIT_S                         ; Return to calling program
                               53   D5  0051    83 DIV:    TSTL    R3                      ; (3) Check if divisor is 0
                               10   13  0053    84         BEQL    ERR                     ; Avoid division by 0
                          52   53   C7  0055    85         DIVL3   R3,R2,@16(AP)           ; (3) Calculate the quotient
                             10 BC      0058
                               09   1D  005A    86         BVS     ERR                     ; An overflow occurred
                                        005C    87         $EXIT_S                         ; Return to calling program
                             10 BC  D4  0065    88 ERR:    CLRL    @16(AP)                 ; Return 0 for overflow,
                                        0068    89                                         ; division by 0, or illegal
                                        0068    90                                         ; operator index
;**NEW**                        50  D4  0068    .1         CLRL    R0                      ; Indicate failure
-1                                  04  006A    92         RET                             ; Return to calling program
                                        006B    93                                         ; No reference to symbolic name
                                        006B    94         .END                            ; since MATH is not a main program.
```

```
MATH                            - Routine to do simple arithmetic       15-NOV-1979 18:48:11  VAX-11 Macro V02.38         Page   3
Symbol table                                                            15-NOV-1979 18:46:12  _DB1:[MARGERY.MACNEW]MATH.MAR;3    (1)

ADD               00000021 R  D  02
DIV               00000051 R  D  02
ERR               00000065 R  D  02
MATH              00000000 RG D  02
MUL               00000041 R  D  02
SS$_NORMAL        ********   X   02
SUB               00000031 R  D  02
SYS$EXIT          ********  GX   02

                                                  +----------------+
                                                  ! Psect synopsis !
                                                  +----------------+
PSECT name                      Allocation          PSECT No.  Attributes
----------                      ----------          ---------  ----------
.  ABS  .                       00000000  (    0.)  00 (  0.)  NOPIC   USR   CON   ABS   LCL NOSHR NOEXE NORD   NOWRT NOVEC BYTE
. BLANK .                       00000000  (    0.)  01 (  1.)  NOPIC   USR   CON   REL   LCL NOSHR   EXE   RD     WRT NOVEC BYTE
RO CODE                         0000006B  (  107.)  02 (  2.)  NOPIC   USR   CON   REL   LCL NOSHR   EXE   RD   NOWRT NOVEC BYTE
```

```
MATH                             - Routine to do simple arithmetic        15-NOV-1979 18:48:11  VAX-11 Macro V02.38          Page   4
Cross reference                                                           15-NOV-1979 18:46:12  _DB1:[MARGERY.MACNEW]MATH.MAR;3    (1)


                                                            +-------------------------+
                                                            ! Symbol Cross Reference !
                                                            +-------------------------+
SYMBOL          VALUE         DEFINITION                    REFERENCES...
------          -----         ----------                    -------------
ADD             00000021-R    74      (1)                    71      (1)
DIV             00000051-R    83      (1)                    71      (1)
ERR             00000065-R    88      (1)                  #-72      (1)       #-75      (1)       #-78      (1)       #-81      (1)       #-84      (1)
                                                           #-86      (1)
MATH            00000000-R    63      (1)
MUL             00000041-R    80      (1)                    71      (1)
SS$_NORMAL      00000000-XR                                #-64      (1)
SUB             00000031-R    77      (1)                    71      (1)
SYS$EXIT        00000000-XR                                  76      (1)         79      (1)         82      (1)         87      (1)
```

```
MATH                             - Routine to do simple arithmetic        15-NOV-1979 18:48:11  VAX-11 Macro V02.38          Page   5
Cross reference                                                           15-NOV-1979 18:46:12  _DB1:[MARGERY.MACNEW]MATH.MAR;3    (1)


                                                            +-------------------------+
                                                            ! Macros Cross Reference !
                                                            +-------------------------+
MACRO             SIZE           DEFINITION                   REFERENCES...
-----             ----           ----------                   -------------
$EXIT_S           1              76       (1)                 76       (1)         79       (1)         82       (1)         87       (1)
CASE              1              46       (1)                 70       (1)

                                                            +-------------------------+
                                                            ! Performance indicators !
                                                            +-------------------------+
Phase                           Page faults    CPU Time        Elapsed Time
-----                           -----------    --------        ------------
Initialization                           12    00:00:00.04     00:00:00.18
Command processing                       18    00:00:00.22     00:00:00.76
Pass 1                                  236    00:00:00.88     00:00:02.08
Symbol table sort                         2    00:00:00.01     00:00:00.00
Pass 2                                   76    00:00:00.41     00:00:01.16
Symbol table output                       1    00:00:00.02     00:00:00.02
Psect synopsis output                     3    00:00:00.02     00:00:00.02
Cross-reference output                   14    00:00:00.06     00:00:00.06
Assembler run totals                    365    00:00:01.67     00:00:04.29
```

```
The working set limit was 150 pages.
1878 bytes (4 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 8 non-local and 2 local symbols.
94 source lines were read in Pass 1, producing 15 object records in Pass 2.
2 pages of virtual memory were used to define 2 macros.

                                             +--------------------------+
                                             ! Macro library statistics !
                                             +--------------------------+
Macro library name                            Macros defined
------------------                            --------------
_DBA0:[SYSLIB]STARLET.MLB;1                  1

5 GETS were required to define 1 macros.

There were no errors, warnings or information messages.

/LIST/CROSS MATH/UPDATE=REPLINES
```

CHAPTER 4

ELEMENTS OF VAX-11 MACRO PROGRAMS

For a VAX-11 MACRO program to be assembled, linked, and executed successfully, it must be constructed according to certain rules, with various required elements in particular relationship to one another. This chapter outlines the essential elements of VAX-11 MACRO programs.

All of the VAX-11 MACRO directives used in this chapter are described in detail in the VAX-11 MACRO Language Reference Manual.

The comments included in the sample statements in this chapter are not required parts of the statements. See the VAX-11 MACRO Language Reference Manual for information on comments and other parts of VAX-11 MACRO statements.

4.1 ESSENTIAL PARTS OF A PROGRAM

A VAX-11 macro program must include the following three elements:

- An entry statement marking the transfer address of the procedure
- An end statement matching the entry statement
- An operation setting the contents of Register 0

These three elements are described in the following three sections.

4.1.1 Entry Statement

To run successfully, a VAX-11 MACRO program must contain an entry statement indicating the transfer address of the program. It can also include other entry statements. The program may assemble correctly without an entry statement, but it will probably fail when you try to execute (run) it.

The entry statement of a VAX-11 MACRO program has three elements -- a directive, a symbolic name, and a mask -- specified as in the following example:

```
    .ENTRY          PROGRAM,^M<R4,R5>
```

The .ENTRY directive notifies the assembler that this statement is an entry point of a procedure. PROGRAM is the symbolic name for this entry point. The symbolic name is a global relocatable symbol equal to the value of the location counter for the entry statement. This name must be repeated in the end statement (see Section 4.1.2) if your program is a main program. It can also be used with the .MASK and .TRANSFER directives and in other situations when you want to refer to the address of the entry point. It must not be the same as any other symbol used in your program.

The entry mask in this example is ^M<R4,R5>. ^M is an operator that sets a bit in the register mask for each register name or arithmetic trap enable specifier that follows. The specified registers are saved before the procedure is entered. If you specify multiple items after the ^M operator, you must enclose them in angle brackets and separate them by commas. For more information on register masks, see the description of procedure call instructions in the VAX-11 Architecture Handbook.

4.1.2 End Statement

A VAX-11 MACRO program must include an end statement indicating the transfer address of the program. The end statement has two elements: a directive and a symbolic name, specified as in the following example:

```
.END    PROGRAM
```

The .END directive notifies the assembler that this statement is the end of a procedure. PROGRAM is the same symbolic name of the program defined in the entry statement (see Section 4.1.1). If your program is a subprocedure, you should not include its symbolic name in the end statement.

4.1.3 Setting Register 0

For proper termination of the image after execution, you must include an operation before the end of your program that sets the contents of Register 0 (R0). In special cases, you may want to return a value in R0 that indicates an error condition. It most cases, however, you should return a value of SS$_NORMAL in R0 to indicate normal termination. You can do this in one of two ways:

- By including the following sequence of instructions before the end statement:

  ```
  MOVL    #SS$_NORMAL,R0
  RET
  ```

- By calling the $EXIT system service before the end statement, using the following call:

  ```
  $EXIT_S
  ```

The $EXIT system service is described in the VAX/VMS System Services Reference Manual.

You should set the value of R0 explicitly, even if you have not used R0 in your program. Do not assume that R0 contains a 1 when you enter a procedure.

4.2 ESSENTIAL ELEMENTS OF BLOCKS AND MACROS

VAX-11 MACRO programs typically include small structural units: conditional assembly blocks, macros, and repeat blocks. These units must contain certain elements to be assembled correctly. The essential elements of conditional assembly blocks, macros, and repeat blocks are described in the following three sections.

4.2.1 Essential Elements of Conditional Assembly Blocks

A conditional assembly block is a sequence of instructions that is only assembled if certain conditions exist when it is encountered. The basic conditional assembly block is delimited by two required statements. The block must be preceded by a statement containing an .IF directive. The block must be followed by a statement containing an .ENDC directive. The following example shows a basic conditional assembly block:

```
.IF DEFINED     MINIM
        .                               ; Assemble these
        .                               ; statements if
        .                               ; MINIM is defined
.ENDC
```

DEFINED is one of the condition tests that can be specified with the .IF directive. Condition tests are listed with the description of .IF in the VAX-11 MACRO Language Reference Manual. MINIM is the symbol whose state of being defined or not defined will determine whether (or, if there is a subconditional assembly block, how) the conditional block is assembled.

There are two variations on the basic conditional assembly block: the subconditional assembly block and the immediate conditional assembly block. Neither requires an .ENDC statement. See the descriptions of the .IF_x and .IIF directives in the VAX-11 MACRO Language Reference Manual for information on these blocks.

4.2.2 Essential Elements of Macros

A macro is a block of code that is to be assembled whenever the name of the macro is called in a program. For a macro to be assembled correctly, it must first be defined. A macro definition must include two statements -- a beginning statement containing the .MACRO directive and a final statement containing the .ENDM directive -- as shown in the following example:

```
.MACRO NAMEME
        .                              ; Assemble these statements
        .                              ; when macro NAMEME
        .                              ; is called
.ENDM  NAMEME
```

NAMEME is the name of the macro. It can be the same combination of characters as a user-defined symbol (see Section 5.3 and the VAX-11 MACRO Language Reference Manual) but such multiple use of names is not recommended. You will use NAMEME to call this macro from later points in your program, as follows:

```
NAMEME                                 ; call macro NAMEME
```

You do not need to include the macro name with the .ENDM directive, but including it makes your program easier to read.

4.2.3 Essential Elements of Repeat Blocks

A repeat block is a sequence of MACRO statements that generally occurs only within a macro. This sequence of statements includes a text that will be repeated according to the conditions that exist when the repeat block is encountered. The repeat block must be preceded by a statement containing the .REPEAT directive. The repeat block must be followed by a statement containing the .ENDR directive. These elements are arranged within a macro as follows:

```
.MACRO  NAMEME  NUMB,TEX
.REPEAT NUMB
.ASCII /TEX/
.ENDR
.ENDM  NAMEME
```

The number of repetitions is here represented by NUMB. When macro NAMEME is called, expressions are provided to replace the arguments NUMB and TEX. TEX is the text to be repeated.

4.2.4 Essential Elements of Indefinite Repeat Blocks

The .IRP and .IRPC directives are also used to cause repetitions; like .REPEAT, they are most useful inside macros. Each statement containing .IRP or .IRPC must be matched by a statement containing the .ENDR directive. For information on these directives, see the VAX-11 MACRO Language Reference Manual.

4.3 ELEMENTS IN RESTRICTED CONTEXTS

Some elements of VAX-11 MACRO programs can be used only in restricted contexts and will cause an error if encountered elsewhere. The directives described earlier in this chapter have this characteristic. Other elements that are only permitted under certain circumstances are listed in the following sections.

In addition, many VAX-11 MACRO directives require particular types of arguments and cannot be assembled correctly if an argument is specified incorrectly. See the directive descriptions in the VAX-11 MACRO Language Reference Manual for the parameter specifications of directives.

4.3.1 Elements Restricted to Macros

There are three string operators that can be used only within macros. They are the only VAX-11 MACRO operators that begin with the percent sign (%):

```
%EXTRACT
%LENGTH
%LOCATE
```

The .MEXIT directive, used to terminate a macro expansion under certain conditions, can only be used within macros and repeat blocks.

The .NARG directive, which determines the number of arguments in the current macro call, can only be used within a macro.

4.3.2 Restrictions Concerning Program Sections

The directives used to construct and manipulate program sections have certain requirements that must be met for successful assembly, linking, and execution.

- Each statement containing the .RESTORE_PSECT directive must correspond to a statement containing the .SAVE_PSECT directive. If you use, .RESTORE_PSECT when no program section context has been stored on the program section context stack using .SAVE_PSECT, an error results during assembly.
- If you have given a program section the attribute ABS (absolute), you cannot include in it any statements that generate binary code. Including such directives as .WORD and .ASCII in an absolute program section will cause errors when the program is linked. However, you can include statements that establish data structures (using .BLKx, for example) or define symbols.
- If you have given a program section the attribute NOEXE (Not Executable), you cannot include in it the transfer address of a procedure. The transfer address is defined using the .ENTRY and .END directives.

CHAPTER 5

FEATURES OF VAX-11 MACRO

VAX-11 MACRO programs should be written so that they are easy to read, easy to correct, and easy to combine with other programs. This chapter describes features of VAX-11 MACRO that help you to write better programs.

Details of the constructs introduced in this chapter are contained in the VAX-11 MACRO Language Reference Manual.

5.1 MODULES

VAX-11 MACRO allows you to use a modular approach to constructing programs. 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.

5.2 PROGRAM SECTIONS

You can segment your object module into a series of program sections. Using program sections allows you to 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 one location counter 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. If you specify the same program section name with the overlay (OVR) attribute in different modules, each program section will share the same virtual memory.
- Separate areas in which 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 will occur. 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. Avoid using 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, privileged or nonprivileged. 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. 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 that 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.

5.3 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 where they will be placed in virtual memory.

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

A symbol can be internal to one module; that is, the symbol is only understood 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, since they 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. 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 containing all symbols used in the module, with each symbol's address or value (when known). The symbol table is printed in the listing file, as described in Section 3.4. The assembler does not usually write the complete symbol table to the object module; it writes a 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 1.3) 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 using the .DEBUG directive. To include all local symbols, you must specify the /ENABLE=DEBUG qualifier in the MACRO command or the .ENABLE DEBUG directive in the source file.

There are two specialized kinds of global symbols: weak and universal. 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). Unlike symbols, local labels can be reused within the same object module. Consequently, local labels are not included in the symbol table and are not available to the linker or debugger.

5.4 MACROS

Macros are a very useful feature of the VAX-11 MACRO assembly language. The term "macro" is a short form of the word "macroinstruction." A macro is essentially one instruction that comprises a number of operations. You assign a name to each macro when you define it. You can put any sequence of coding instructions that you will need to use repeatedly into a macro; later, you can recall those instructions by entering the macro name in the operator field of a statement line.

A macro must be defined before you can refer to it. The assembler directives that define macros are described in Chapter 4 of this manual and in the VAX-11 MACRO Language Reference Manual.

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

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

CHAPTER 6

WRITING CODE FOR SHAREABLE IMAGES

If you are writing code for a shareable image, keep in mind the restrictions involved in writing position independent code and the consequences of storing address data in shareable images. These topics are discussed in the following sections.

6.1 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 certain rules concerning addressing modes and the storage of addresses. Source code that follows these rules, 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 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. Otherwise, the linker will assume that the image is position independent and the shareable image will not execute correctly. See the <u>VAX-11 Linker Reference Manual</u> 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. Addressing modes are described in Chapter 4 of the <u>VAX-11 MACRO Language Reference Manual</u> and in the <u>VAX-11 Architecture Handbook</u>.

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 6-1 summarizes the position independence or dependence of relative and absolute modes.

Table 6-1
Relative and Absolute Addressing Modes

	Position Independence/Dependence	
Mode	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).

The following two examples illustrate 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 of 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 not. If, however, the address is absolute, absolute mode will be position independent and relative mode will 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.

6.2 STORING ADDRESS DATA IN SHAREABLE IMAGES

If a shareable image contains an .ADDRESS directive specifying a relocatable address, the linker makes the image position dependent by means of deferred relocation. As described in the VAX-11 Linker Reference Manual, the linker creates a private copy of the image section containing the .ADDRESS directive.

To retain the benefits of a shareable image, you should use .ADDRESS with a relocatable address only in image sections that contain nonshareable data. These are image sections made from program sections that have the NOSHR and WRT attributes.

The following two examples show programming techniques used to avoid storing address data in an image.

Example 1

```
; SETTING UP A STRING DESCRIPTOR
        .ALIGN LONG
DESCRIP:
        .LONG   EOSTR-STR                   ; LENGTH OF STRING.
        .ADDRESS        STR                 ; CODE IS COPY ALWAYS
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. String descriptors used in the VAX-11 procedure calling standard and the list head for the INSQUE and REMQUE instructions require absolute virtual addresses. The addresses must be stored by executable instructions rather than as data in the source code.

Example 2

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

; 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. You can also use 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-ILLMASKBIT, Reserved bits set in ENTRY mask
```

Some input and output diagnostic messages are followed by a VAX-11 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 1000 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

ASGNMNTSYN, 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

BLKEXPNABS, Block expression not absolute

Explanation: The expression specifying the amount of storage to be allocated in a .BLKA, .BLKB, .BLKD, .BLKF, .BLKG, .BLKH, .BLKO, .BLKQ, or .BLKW 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

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

CANTLOCMAC, 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

CONFQUAL, Conflicting qualifiers

Explanation: The assembler encountered a combination of command qualifiers or of file qualifiers which cannot be put into effect.

User Action: Rewrite the command string in accordance with the limitations on the various qualifiers.

Severity: Error

DATALSTSYN, 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: An .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

IFEXPRNABS, IF expression not absolute

Explanation: The expression in an .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

ILLMACARGN, 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 a macro name in the .MACRO directive.

Severity: Error

ILLMASKBIT, 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

ILLOPDEFVL, 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 31 characters

Explanation: The symbol name was longer than 31 characters. The assembler truncated the name to 31 characters.

User Action: Truncate the name to 31 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: Contact the system manager to have the available virtual memory increased; reduce the level or 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

LINTOOLONG, Line too long

Explanation: A source line in a macro definition was longer than 1000 characters.

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

Severity: Error

MACLBFMTER, 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

MAYNOTINDX, This mode may not be indexed

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

User Action: Change the addressing mode.

Severity: Error

MCHINSTSYN, 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 an .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

NOFORMLARG, No formal argument for .IRP/.IRPC

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

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

Severity: Error

NOTDECSTRG, 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

NOTENABOPT, Not a legal ENABLE option

Explanation: An argument to an .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: An .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

NOTLGLISOP, 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

NOTPSECOPT, 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 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

PACTOOLONG, 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

PSECOPCNFL, 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

PSECBUFOVF, 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

PSECBUFUND, 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

RPTCNTNABS, 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

SYMDCLEXTR, 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

SYMDEFINMO, 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

SYMOUTPHAS, 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

TOOMNYPSEC, Too many PSECTs declared

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

User Action: Reduce the number of program sections.

Severity: Error

UNDEFSYM, 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 an .EXTERNAL directive.

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

Severity: Error

UNDEFXFRAD, Undefined transfer address

Explanation: The .END directive specified a transfer address that was not defined in the module or specified in an .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 an .ENDC directive. The assembler inserted an .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

INDEX

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______

— — — — Do Not Tear - Fold Here and Tape —

digital

No Posta
Necessar
if Mailed in
United Sta

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

BSSG PUBLICATIONS TW/A14
DIGITAL EQUIPMENT CORPORATION
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS 01876

— — — Do Not Tear - Fold Here —