EMULOGIC RELOCATABLE MACRO CROSS ASSEMBLER

MAN-1000-02

Edition 2 (December 1982)

The information contained herein is subject to change without notice. Emulogic, Inc., assumes no responsibility for any errors that may appear in this document.

Details and specifications concerning the use and operation of Emulogic equipment and software are included in various technical manuals available through local sales representatives.

Printed in U.S.A.

PREFACE

This manual contains reference material and procedures for developing programs to be run under the Emulogic Relocatable Macro Cross Assembler. The Emulogic cross assembler is a totally flexible and extremely powerful software package. It enables you to use the full capability of the PDP-11 MACRO-11 Assembler with the mnemonic language of the microprocessor for which you are writing the software. Assembler enhancements and differences that you should be aware of are documented herein.

The manual is intended to be used in conjunction with the standard documentation set for the PDP-11 MACRO-11 and the microprocessor chip assembler manual. Prior to reading this manual, you should become familiar with, or have access to these documents: the PDP-11 MACRO-11 Language Reference Manual, RT-11 Programmer's Reference Manual, RT-11 System User's Guide, RT-11 Software Support Manual, PDP-11 Processor Handbooks, and the microprocessor chip reference and user's manual.

This manual is organized as follows:

Chapter 1 introduces the Emulogic cross assembler and describes the two-pass assembly process.

Chapter 2 describes extensions and modifications to MACRO-11 character sets, directives, and formats.

Chapter 3 lists the components of the object module produced by the assembler. It also summarizes the linking process required to convert the object module into an executable image.

Chapter 4 summarizes the operating procedures necessary to run the assembler and the linker.

* * *

You should also refer to the chip supplement to this manual that corresponds to your microprocessor. The supplement contains the chip instruction set and specific assembler information you will need to write software for your microprocessor. Also included are sample output listings from the cross assembler.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The development sequence for creating programs to run on the ECL-3211 Microprocessor Emulation System involves five steps:

1. The character sets, symbols, terms, and expressions composing the assembler language elements are incorporated into standard assembler source program statements.

2. The source program statements are written into a file using one of the RT-11 editors to form an ASCII source file.

3. The Emulogic Relocatable Macro Cross Assembler assembles one or more ASCII source files into a single relocatable binary object file. It places object records in PDP-11 MACRO-11 object file format compatible with the RT-11 operating system. Assembler directives and macro directives control source statement processing during this assembly procedure.

4. Object modules are processed by the linker. The values of relocatable or external symbols are converted to absolute (LDA) format, and an executable load module is produced.

5. Finally, the executable image is loaded into memory and executed under the ECL-3211 emulation system.

Besides the binary object file, the assembler produces a file containing the table of contents, the assembly listing, the symbol table, and an optional cross reference table of symbols and macros. Additional features provided by the cross assembler include:

- Source and command string control of assembly and listing functions
- Device and filename specifications for input and output files
- Error listing on command output device
- Alphabetized, formatted symbol table listing
- Global symbols for linking object modules
- Conditional assembly directives
- Program sectioning directives
- User-defined macros and macro libraries
- Comprehensive system macro library

The cross assembler makes two passes during the assembly process.

FIRST ASSEMBLY PASS

During the first pass, the cross assembler locates and reads all required macros from the libraries, builds symbol tables and program section tables, and performs a partial assembly of each source statement.

The assembler first initializes all impure data areas (i.e., areas containing both code and data) that will be used internally for the assembly process. These areas include all dynamic storage and buffer areas used as file storage regions.

The assembler then calls a system subroutine that transfers a command line into memory. This command line, entered by the user, specifies all files to be used during assembly. After scanning the command line for proper syntax, the assembler initializes the specified output files, and opens them to determine whether valid output file specifications have been given in the command line.

Then, the assembler initializes a routine that retrieves source lines from the input file. If no input file is currently open, as is the case at the beginning of assembly, the assembler opens the next input file specified in the command line and determines the length of the instructions. It then starts assembling the source statements according to length.

At the end of the first assembly pass, the assembler reopens the output files. Such information as the object module name, program version number, and global symbol directory for each program section are written to the object file to be used later in linking the object modules. After producing the global symbol directory for a given program section, the assembler scans through the symbols tables to find all global symbols that are bound to that program section, and writes the directory records to the object file for these symbols. This process is repeated for each program section.

SECOND ASSEMBLY PASS

During the second pass, the cross assembler writes the object records to the output file while generating both the assembly listing and the symbol table listing for the program. An optional cross-reference listing may be produced at this time as well.

The assembler performs the same functions in this pass as in the first pass with the exception that all source statements containing assembler-detected errors are flagged with an error code as the assembly listing file is created.

The resultant object file contains all the object records, and also includes the relocation records that specify information necessary to later link the object file. The information in the object file, when passed to the task builder or linker, enables the global symbols in the object modules to be associated with absolute or virtual memory addresses, thereby forming an executable body of code.

CHAPTER 2

CROSS ASSEMBLER INSTRUCTION COMPONENTS

Source programs are composed of assembly language statements that incorporate the language elements and conform to the format and syntactical conventions of the PDP-11 MACRO-11 Assembler. (Refer to the PDP-11 MACRO-11 Language Reference Manual and the RT-11 Programmer's Reference Manual.)

This chapter describes the instruction components of the Emulogic cross assembler and notes the enhancements that have been added.

CROSS ASSEMBLER CHARACTER SET

The Emulogic cross assembler accepts the following characters in its source statements:

* The upper- and lower-case letters A through Z. Lower-case letters are converted to upper-case letters on input.
* The digits 0 through 9.
* A period (.) and a dollar sign ($) are reserved for use as PDP-11 system program symbols.
* The special characters listed in Table 2.1. While the normal function of these characters is as stated in the table, some of the functions may change depending on the microprocessor chip used. Refer to the chip supplement to this manual corresponding to your microprocessor for a description of any exceptions.

Table 2.1 Special Characters Used in Emulogic Source Programs

Character	Designation	Function
:	Colon	Label terminator
::	Double colon	Label terminator; defines the label as a global label
=	Equal sign	Direct assignment operator and macro keyword indicator
==	Double equal sign	Direct assignment operator; defines the symbol as a global symbol

Table 2.1 Special Characters Used in Emulogic Source Programs (Contd.)

Character	Designation	Function
	Tab	Item or field terminator
	Space	Item or field terminator
.	Period	Current location counter
,	Comma	Operand field separator
;	Semicolon	Comment field indicator
<	Left angle bracket	Initial argument or expression indicator
>	Right angle bracket	Terminal argument or expression indicator
+	Plus sign	Arithmetic addition operator
-	Minus sign	Arithmetic subtraction operator
*	Asterisk	Arithmetic multiplication operator
/	Slash	Arithmetic division operator
&	Ampersand	Logical AND operator
!	Exclamation point	Logical inclusive OR operator
''	Double quote	Double ASCII character indicator
'	Single quote	Single ASCII character or concatenation indicator
^	Up arrow or circumflex	Universal unary operator or argument indicator
\	Backslash	Macro call numeric argument indicator

PERMANENT SYMBOLS

Permanent symbols used by the cross assembler consist of the instruction mnemonics of the particular microprocessor for which the assembler was designed. For example, the permanent symbol table used by the Z80 cross assembler consists of the Zilog instruction mnemonics for the Z80

microprocessor. Because these symbols are a permanent part of the assembler image, they are not defined prior to being used in the operator field of a cross assembler statement.

When using a permanent symbol as a term of an expression, always specify its basic value. Do not substitute a zero for this value because zero changes the addressing mode.

The instruction sets for individual microprocessor chips are listed in corresponding chip supplements to this manual.

SYMBOLS FOR PDP-11 GENERAL REGISTERS

Symbols that designate the eight general registers of the PDP-11 processor are not used in cross assembler source statements. Likewise, expressions that refer to these general registers, specify register contents, or refer to the address mode should not be used in the source program. References to address modes in PDP-11 or RT-11 documentation are not applicable to the cross assembler.

ASSEMBLER DIRECTIVE ENHANCEMENTS

The ECL-3211 Cross Assembler uses all MACRO-11 macro directives and assembler directives. It also provides radix directive and radix control operator enhancements, as follows.

.RADIX Directive Enhancement

Although numbers used in a source program are initially considered to have octal values, the .RADIX directive allows you to specify alternate values throughout the entire program or specific portions of the program. The radix directive has the form:

```
.RADIX n
```

where: n is one of two radices: 8 or 16. If argument n is not specified, the default octal radix is assumed. Argument n is always read as a decimal value. Any value other than null or one of the two acceptable radices causes an error code (A) in the assembly listing.

A radix directive remains in effect until another directive is specified, as shown by the following example:

```
.RADIX 16       Begins a section of code having a
   .            hexadecimal radix
   .
   .
.RADIX          Reverts to octal radix
```

When radix 16 is in effect, any numeric value whose first character is A-F must be preceded by a zero. For example, 0D3 would be valid, but D3 would cause an error.

Temporary Radix Control Operator Enhancement

Once you have specified the default octal radix or the hexidecimal radix for a section of code, you may find that an alternate radix is more convenient or desirable.

There are four unary operators that allow you to temporarily declare an alternate radix for a single term, as shown by the following:

```
^Bn     where n is evaluated as a binary number
^On     where n is evaluated as an octal number
^Dn     where n is evaluated as a decimal number
^Hn     where n is evaluated as a hexadecimal
           number; if the first character of n
           is A-F, precede n with a zero
```

Temporary radix control operators can be used at any time regardless of the radix in effect or other radix declarations within the program. Because the unary operator affects only the term immediately following it, it can be used anywhere a numeric value is legal. The term or expression associated with the operator is evaluated during assembly as a 16-bit entity. The following are examples of temporary radix control operators:

```
^B00001101      Binary radix
^037            Octal radix
^D452           Decimal radix
^H0E2           Hexadecimal radix
```

CHAPTER 3

RELOCATING AND LINKING THE PROGRAM

The output of the cross assembler is an object module composed of relocatable machine language code, relocation information, and a corresponding global symbol table that defines the use of symbols within the program. To form an executable program, the object module must be processed by the Emulogic linker.

> NOTE: The standard Emulogic linker is ELINK2. However, some microprocessors require use of a modified version of this linker. Refer to the corresponding Chip Supplement to this manual for a description of any differences in linking your object modules. This manual describes the features and operation of the standard linker, ELINK2.

ELINK2 produces an executable load module with all locations resolved as absolute locations. This absolute load file (DEC LDA format) is the only loadable file format produced by ELINK2. With this exception, the linker operates the same as the RT-11 linker.

To allow the value of an expression to be fixed at link time, the cross assembler produces object file instructions and other required parameters. For relocatable expressions in the object module, the base of the associated relocatable program section is added to the value of the relocatable expression provided by the cross assembler. For external expression values, the value of the external term in the expression is determined and added to the absolute portion of the external expression as provided by the assembler.

All instructions requiring modification at link time are flagged in the assembly listing, as shown in the examples below. The apostrophe (') following the octal expansion of the instruction indicates that relocation is required. The letter "G" indicates that the value of an external symbol must be added to the absolute portion of an expression.

005065 000040'	CLR	RELOC	Assuming that the value of the symbol RELOC, 40, is relocatable, the relocation bias will be added to this value. (See Note 1)
005065 000000G	CLR	EXTERN	The value of the symbol EXTERN is assembled as zero and is resolved at link time.
005065 000000G	CLR	EXTERN+6	The value of the symbol EXTERN is resolved at link time and added to the absolute portion (+6) of the expression. (See Note 1)

NOTE 1: Use of complex forward references with chips that optimize may cause phasing errors. Refer to the corresponding chip supplement for specific information.

For a complete discussion of the linker, refer to the RT-11 System User's Guide (Section 11, Linker). For a complete discussion of the .LDA file, refer to the RT-11 Software Support Manual (Section 8, "File Formats").

CHAPTER 4

OPERATING PROCEDURES

This section of the manual provides supplementary operating procedures for running Emulogic's cross assembler and linker under the RT-11 operating system. A complete presentation of operating procedures, error messages, and corrective action is found in the RT-11 System User's Guide, and the PDP-11 MACRO-11 Language Reference Manual. Additional reference material can be found in the RT-11 Software Support Manual.

CALLING THE CROSS ASSEMBLER

To call the cross assembler from the system device, respond to the system prompt (a dot printed by the keyboard monitor) by typing:

```
.RUN  Xname<cr>
```

where: Xname designates the assembler program for a specific microprocessor chip

<cr> carriage return

Example: .RUN X68000<cr>

Invokes the 68000 cross assembler.

The cross assembler will respond with an asterisk (*) prompt. At this point, the assembler is ready to accept command string input and to perform an assembly. Everything typed to the left of the equal (=) sign is output; everything typed to the right of the = sign is input to the assembler. For example,

```
*TEST68.OBJ=TEST68.MSR<cr>
```

This produces an output object file TEST68.OBJ from source file TEST68.MSR.

```
*TEST68.OBJ,TEST68.LST=TEST68.MSR<cr>
```

This produces an output object file, TEST68.OBJ, and an output listing file, TEST68.LST, from source file TEST68.MSR. Refer to "Entering Command Strings."

TERMINATING THE CROSS ASSEMBLER

The cross assembler can be terminated at any time by typing ^C (Control C) from the keyboard. This is done by pressing

the 'CTRL' key and the 'C' key simultaneously. A ^C will be echoed on the console screen.

1. If you invoked the cross assembler and received the asterisk prompt but have not yet entered the command string, you can terminate the cross assembler by typing ^C. For example,

```
.RUN X68000<cr>
*^C
.
```

   System returns to the system monitor prompt.

2. If you have completed command string input and started an assembly, you can halt the assembly process at any time by typing ^C^C.

```
.RUN X68000<cr>
*TEST68.OBJ,TEST68.LST=TEST68.MSR<cr>
^C^C
.
```

   System returns to the system monitor prompt.

3. When assembly has been completed, the system displays the asterisk prompt. To return to RT-11, just enter ^C.

ENTERING COMMAND STRINGS

When the system displays the assembler prompt (*), it is waiting for you to enter a command string consisting of:

1. Output file specifications
2. An equals sign
3. Input file specifications

Type the command string using the following format and syntax:

```
(object_file),(list_file)/s:arg=source1 (,...,source6)/s:arg
```

where:

object_file	is the file specification of the binary object file to be produced by the cross assembler. The device for this file should not be TT or LP.
list_file	is the file specification for the assembly and symbol listing to be produced by the cross assembler
/s:arg	is a set of file specification options and arguments (See "File Specification Options")
source1	specifies the input source file. You can specify up to six source files (,...,source6).

Note: The parentheses in this format only indicate that the field is optional; the programmer should not actually enter the parentheses.

All output file specifications are optional. The system does not produce an output file unless the command string contains a specification for that file.

The system determines the file type of an output file specification by its position in the string, as determined by the number of commas in the string. For example, to omit the object file, you must begin the command string with a comma. A comma is not required after the final output file specification in the command string.

The following command string produces a listing but no binary object file. By including the /C option, cross reference tables will be included in the listing.

```
*,LP:/C=source_file_specification<cr>
```

If you enter a filename without an extension, the system assumes the default extension. For example, if the command string is entered as

```
*BETA,BETA=BETA<cr>
```

the cross assembler processes this as

```
*BETA.OBJ,BETA.LST=BETA.MSR<cr>
```

Table 4.1 lists the default values for each file specification.

Table 4.1 Cross Assembler Default File Specifications

File	Default Device	Default File Name	Default File Type
Object	DK:	Must specify	.OBJ
Listing	Same as object file	Must specify	.LST
Source1	DK:	Must specify	.MSR
Source2 thru source6	Same as preceding source file	Must specify	.MSR
System macro library	System device SY:	Cross Assembler name (e.g., X68000)	.SML
User macro library	DK: if first file; otherwise, same as for preceding source file	Must specify	.MSR

The following examples illustrate command strings typically given to the cross assembler.

```
*DK:SUM.OBJ,LP:=DK:SUM.MSR<cr>
```

Assembles source file SUM.MSR to generate object file SUM.OBJ. The assembly listing goes directly to the printer.

```
*,DK:SUM.LST=DK:SUM.MSR<cr>
```

Assembles source file SUM.MSR and generates a listing file SUM.LST. No object file is created. As illustrated here, the system does not produce an output file unless the command string contains a specification for that file. The system determines the file type of an output file by its position in the command string. The comma is used in place of the file that is to be omitted.

```
*,LP:/C=DK:SUM.MSR,SRC.MSR<cr>
```

Assembles source files SUM.MSR and SRC.MSR and produces a listing on the printer that includes a cross-reference table. No binary object file is generated.

The following command string example specifies an assembly that uses source file SRC.MSR and user macro library LIBR.MSR as input to produce an object file BINF.OBJ and a listing. The listing goes directly to the line printer.

```
*DK:BINF.OBJ,LP:=DK:SRC.MSR,LIBR.MSR<cr>
```

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

The default device for WRK.TMP is DK:. To cause the system to assign a different device, enter the command

```
.ASSIGN dev:  WF<cr>
```

where: dev: is the file-structured device that will hold WRK.TMP

FILE SPECIFICATION OPTIONS

When you assemble your source files, you may want to override certain MACRO directives in the source programs. You may also need to direct the cross assembler in handling specific files during the assembly. Table 4.2 lists the various options (i.e., /s:arg) you may use when entering file specifications in the command string.

Table 4.2 File Specification Options

Option	Usage
/L:arg /N:arg	Listing Control Switches. These options accept ASCII switch values (arg) which are equivalent in function and name to the arguments of the .LIST and .NLIST MACRO-11 directives specified in the source program. This switch overrides the arguments for these directives and remains in effect throughout assembly.
/E:arg /D:arg	Function Control Switches. These options accept ASCII switch values (arg) that are equivalent in function and name to arguments of the .ENABL and .DSABL directives specified in the source program. This switch overrides the arguments for these directives and remains in effect throughout assembly.
/M	Indicates input file is MACRO library file. If two or more macro libraries that contain definitions of the same macro name are included in the command string, the macro library that appears leftmost in the command string takes precedence.
/C:arg	Controls contents of cross-reference listing (See Table 4.3)
/P:1	Assembles the associated file during assembly pass 1 only.
/P:2	Assembles the associated file during assembly pass 2 only.

NOTE: /M and /P switches affect only the source file with which they are specified. The other options affect the entire command string.

CROSS-REFERENCE TABLE

A cross-reference table (CREF) lists all symbols, or a subset of the symbols, in a source program and identifies the statements that define and use symbols. The cross-reference listing is obtained at assembly time, if requested, and is output in the assembly .LST file.

A complete CREF listing contains the following five sections:

1. A cross-reference of program symbols: labels used in the program and symbols followed by an operator.

2. A cross-reference of MACRO symbols: those symbols defined by .MACRO and .MCALL directives.

3. A cross-reference of permanent symbols: all operation mnemonics and assembler directives.

4. A cross-reference of program sections: the names specified as operands of .CSECT or .PSECT directives.

5. A cross-reference of errors: all flagged errors from the assembly grouped by error type.

Any or all of these sections are included in the cross-reference listing by specifying the appropriate arguments with the /C:arg option.

Table 4.3 Cross-Reference Table /C Option Arguments

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

NOTE: Specifying /C with no arguments is equivalent to specifying /C:S:M:E.

To obtain a cross reference table at assembly time, include the /C:arg option in the command string. The table will be generated in the assembly listing file (.LST). The /C:arg option may be included at any point in the command string, after the first output file. Consider the following command strings:

```
*TEST.OBJ/C,TEST.LST=TEST.MSR

        or

*TEST.OBJ,TEST.LST/C=TEST.MSR

        or

*TEST.OBJ,TEST.LST=TEST.MSR/C
```

Any one of these command strings produces a cross reference table and includes it in the TEST.LST output file.

When you request a cross-reference listing, the system automatically generates a temporary file on device DK:. If a device other than DK: is required to contain the temporary CREF file, you can assign an alternate device for CREF.TMP by entering the command

```
.ASSIGN dev:CF<cr>
```

prior to invoking the cross assembler.

For additional information on cross-reference tables, refer to the PDP-11 MACRO-11 Language Reference Manual (Section 9), and to the RT-11 System User's Guide (Sections 10.4.3 through 10.4.4.2).

CALLING THE LINKER

As mentioned earlier in this manual, ELINK2 is the standard Emulogic linker. It processes cross assembler object modules and produces executable absolute load modules (with the .LDA extension). (Refer to the Chip Supplement corresponding to your target chip to ensure that your object code can be processed by ELINK2.)

ELINK2 operates the same as the RT-11 linker LINK, except for the following:

* The .SAV file is not created,
* An .LDA file is created,
* The linker /L option is selected by default to produce a formatted binary output file,
* The linker option /B:n defaults to n=0,
* The output load map is produced in hexadecimal notation, and
* The /Q option allows the user to specify the base address of up to 127 (decimal) named PSECTs.

To call the linker from the system device, respond to the system prompt (.) by typing:

```
.RUN ELINK2<cr>
```

When the linker responds with an asterisk (*),it is ready to accept the command string.

Enter the command string in this format:

(load_file),(load_map),(sym_table)=object_1/options(,...,object_n/options)

where:

load_file = output load module in .LDA format

load_map = output load map

sym_table = output symbol definition file

object_1 = input object module, library file, or symbol table created in a previous link. The user may enter more than one input file (,...,object_n/options)

Note: The parentheses in this format only indicate that the field is optional; the programmer should not actually enter the parentheses.

All output file specifications are optional. The system does not produce a specific output file unless the command string contains a specification for that file.

The system determines the file type of an output file specification by its position in the string, as determined by the number of commas in the string. For example, to omit the load module, begin the command string with a comma. A comma is not required after the final output file specification in the command string.

If you enter a file name without an extension, the linker assumes the default extension:

.LDA	output load module
.MAP	output load map
.STB	output symbol table file
.OBJ	input file

To create a load module, load map, and symbol table file from object file TEST.OBJ, you could enter the command string as

*TEST.LDA,TEST.MAP,TEST.STB=TEST.OBJ<cr>

or

*TEST,TEST,TEST=TEST<cr>

If you invoke the linker with the /Q option, you can specify the absolute base address for different named PSECTs. The syntax for this option is

/Q:n

where "n" is the number of PSECTs whose base addresses will be defined by the operator. "n" is a number between 1 and 177 octal (or 127. decimal). The default value is 8. decimal. By controlling this value, the operator can allocate an appropriate amount of space for the /Q table, thereby providing more space for the symbol table.

If the /Q option is excluded from the command line, all PSECTs are concatenated in the order of entry.

The linker responds to the /Q option with the prompt

```
*LOAD SECTION:ADDRESS?
```

At this point, you should enter the name of the PSECT and the base address (hexadecimal) at which you want to load the data, and then a carriage return.

> Note: Do not enter a space before the PSECT name, or the linker will respond with the error message
>
> ?LINK-W-load section NOT FOUND

For example, suppose that you enter the previous command line with the /Q option, as

```
*TEST,TEST,TEST=TEST/Q:5<cr>
```

The linker will ask you to name five PSECTs and to specify their base addresses:

```
*LOAD SECTION:ADDRESS?  BAS51:1000<cr>
*LOAD SECTION:ADDRESS?  RELEF:14C0<cr>
*LOAD SECTION:ADDRESS?  BRAL:2050<cr>
*LOAD SECTION:ADDRESS?  CREDL:3500<cr>
*LOAD SECTION:ADDRESS?  ORON:5A50<cr>
```

These entries specify five program sections (BAS51, RELEF, BRAL, CREDL, and ORON) and the corresponding base addresses (in hex).

If you enter a carriage return in response to the /Q prompt, the linker discontinues its prompting and begins the linking process.

TERMINATING THE LINKER

The linker can be terminated at any time using the same procedure as when terminating the assembler, using the ^C (Control C).

1. If you invoked the linker and received the asterisk prompt but have not yet entered the command string, you can terminate the linker by typing ^C. For example,

```
.RUN ELINK2<cr>
*^C
.
```

   System returns to the system monitor prompt.

2. If you have completed command string input and started the linking process, you can halt the linker at any time by typing ^C^C.

```
.RUN ELINK2<cr>
*TEST.LDA,TEST.MAP,TEST.STB=TEST.OBJ<cr>
^C^C
.
```

   System returns to the system monitor prompt.

3. When the linker has completed processing, the system displays the asterisk prompt. To return to RT-11, just enter ^C.

For a complete description of the linker, refer to the RT-11 System User's Guide. (Section 11, Linker).