CERN - Data Handling Division
DD/80/26

Horst von Eicken
September 1980

M 6 8 M I L

a

Cross Macro Assembler
for the
Motorola M 68000

DD-ag

CONTENTS

9-Sep-80

Cross Macro Assembler
for
Motorola 68000

INTRODUCTION

The Cross Macro Assembler described in this manual can be used to translate assembler source programs for the Motorola 68000 microprocessor into CUFOM, the CERN Universal Format for relocatable Object Modules. A linkage editor subsequently allows the combination and linking of several such modules into a new CUFOM module. A librarian permits the construction of CUFOM libraries, which can be placed in the input to the linkage editor, either upon request or automatically (if unresolved external references remain after all input to the linkage editor is processed, an automatic library call routine retrieves the modules required to resolve the references). A pusher finally translates a CUFOM module to the Motorola 'S' format needed for down-line loading from the host computer into the memory of the target machine.

The assembler is upward compatible with the M68000 Cross Macro Assembler provided by Motorola. Additional pseudo instructions are provided to allow the generation of relocatable object modules. The user is advised to use the following Motorola publications:

M68000 Cross Macro Assembler Reference Manual
M68KXASM(D3), Third Edition, September 1979

MC68000 16-Bit Microprocessor, User's Manual
Preliminary, September 1, 1979

as a base for the use of this assembler. This manual will restrict itself to the description of the extensions made to the definitions of the Motorola cross assembler. For the readers' convenience this will be done in complete chapters rather than by listing the explicit differences.

The main areas covered by this note are:

Expressions (generalized)
Pseudo-operations
Macro definitions
How to use the assembler, pusher,...

SYMBOLS and EXPRESSIONS

Symbols.

Symbols recognized by the assembler consist of one or more characters, the first eight of which are significant. The first character must be alphabetic (A through Z), each remaining character may be a letter, a digit (0 through 9) or an underscore (_).

Numbers recognized by the assembler include decimal, hexadecimal and octal values. Decimal numbers are specified by a string of decimal digits (0 through 9); hexadecimal numbers are specified by a dollar sign ($), followed by a string of hexadecimal digits (0 through 9, A through F); octal numbers are specified by a colon (:), followed by a string of octal digits (0 through 7).

One or more characters enclosed by apostrophes (') constitute a character string. Character strings are left-adjusted and zero-filled (if necessary), whether stored or used a immediate operands. Only strings of four or fewer characters may be used as immediate operands. (In order to specify an apostrophe within a character string, two successive apostrophes must appear where the single apostrophe is intended to appear.)

The assembler differenciates between absolute and relative symbols:

Absolute symbol:

1. The symbol is equated (EQU) or SET to an absolute value.

2. The symbol is defined in the absolute section of the program. The start of an absolute section is defined either by an ORG pseudo instruction or by a SECTION pseudo instruction, the type of which is absolute.

Relative Symbol:

1. The symbol is equated (EQU) or SET to a relative symbol.

2. The symbol is defined in a relative section of the program. The assembler recognizes the following relative sections:

   general section,
   relocatable section,
   common section and
   unique section.

   The different section types are explained in the chapter on pseudo instructions. The assembler will by default start each assembly in the general section. The RORG pseudo instruction initializes the general section.

3. The symbol is defined using an EXTERN pseudo instruction.

Expressions.

An expression is a combination of symbols, constants, algebraic operators, and parentheses. The expression is used to specify a value which is to be used as an operand. Expressions follow the conventional rules of algebra. There are no restrictions in the use of operators between these symbols. It may not be possible for the assembler to fully evaluate an expression containing relative symbols. Such an expression is forwarded in the CUFOM module and will finally be resolved by the loader (pusher).

Some pseudo instructions like EQU and SET however need expressions, which can be fully evaluated by the assembler. Such expressions are called "simple expressions" and must fulfil the following rules:

1. Relative symbols or expressions cannot be multiplied, divided, added, or operated on with the logical operators.
2. A relative symbol or expression may have an absolute value added to or subtracted from it. The result is relative.
3. A relative symbol or expression may be subtracted from another relative symbol or expression provided they are both defined in the same section. The result is absolute. External symbols do not belong to any specific section.

Absolute Addressing.

Since the M 68000 microprocessor allows two forms of absolute addressing,

- short absolute address (16 bit address) -
- long absolute address (24 bit address) -

the assembler has to take a decision as to which form to assume whenever it encounters a forward reference, i.e. a symbol which has not yet been defined. By default it will use the long absolute address to ensure correct handling of the symbol. For symbols that have already been defined, the problem does not occur if they belong to the absolute section. The assembler will always generate the appropriate short or long addressing form, based upon the size of the symbol's absolute address. For symbols defined in a relative section of the module the problem is however exactly the same, since the assembler does not know the final origin of the section and hence cannot make a safe assumption as to the final address of the symbol. It will therefore always use the long absolute address form. The pseudo instructions ORG, RORG, SECTION, FSHORT and FLONG will allow the programmer to override the assembler defaults.

A similar problem exists for conditional branches. If a forward reference is found in a conditional branch instruction, the assembler will use the two-word form of the instruction. Using the suffix .S for the branch instruction the programmer can force the assembler to generate the one-word form of the conditional branch instruction. The pseudo instructions BSHORT and BLONG allow additional control.

PSEUDO INSTRUCTIONS

Pseudo instructions discussed in this chapter are classed according to application as follows:

Module identification (IDENT and END)
Section control (ORG, RORG and SECTION)
Symbol definition (EQU and SET)
Module linkage (ENTRY and EXTERN)
Data generation and storage reservation (DC and DS)
Conditional assembly (ELSE, ENDC, ENDIF, IFEQ and IFNE)
Source stream control (INSERT)
Listing control (LIST, NOLIST, PAGE, SPC, STTL and TTL)
Object code control (BLONG, BSHORT, FLONG, FSHORT, NOOBJ)

A later chapter describes the definition and use of MACRO's.

NOTE: The format description for the pseudo instructions uses symbols which have the following syntactical meaning:

{ } Enclose optional fields of the pseudo instruction.

< > Enclose a symbol, called a syntactic variable.

Module Identification.

IDENT - Module Identification.

An IDENT pseudo instruction is the first statement of a module recognized by the assembler. It defines the name to be given to the module and has the following format:

Location	Operation	Variable Subfields
	IDENT	<symbol>

<symbol> The symbol defines the name of the CUFOM module.

END - End of Module.

An END pseudo instruction must be the last instruction of each module. It causes the assembler to terminate all counters, conditional assembly, or macro generation. It also causes the CUFOM module to be terminated.

Location	Operation	Variable Subfields
	END	{<trasym>}

<trasym> An optional symbol. If specified, it declares the module to

be a main program and the symbol to be the transfer symbol for the loader in the target machine. (Please note, that MACSBUG does not honour the transfer symbol.) The symbol, as well as being quoted in the END pseudo instruction, must be defined in the module, or an assembly error will be generated.

Section Control.

Section control pseudo instructions allow the programmer to divide a source module into separately controlled regions of a program, providing him with a means of changing origin and location counters. They establish a new section or resume use of an already established section. The section in use is the section into which code is subsequently assembled. A user may establish up to 16 sections. By default the assembler will always start with the general section. The assembler basically allows two types of section, namely:

Absolute Section.

The following pseudo instructions cause the assembler to establish or resume assembly in the absolute section:

Location	Operation	Variable Subfields
	SECTION	A,{<expression>}
	SECTION.L	A,{<expression>}
	SECTION.S	A,{<expression>}

or (for compatibility with the Motorola assembler)

Location	Operation	Variable Subfields
	ORG	{<expression>}
	ORG.L	{<expression>}
	ORG.S	{<expression>}

<expression> The optional expression may not contain any forward references and must evaluate to an absolute value. The program counter is changed to the value of the expression. If an expression is not defined, the program counter will be set to zero unless an absolute section has already been specified, in which case the program counter will resume with its last value.

If the suffix {.S} is appended to the pseudo instruction, the assembler will assume that as of now any forward reference can be achieved by using the short absolute addressing form of the instruction(this might be dangerous if the forward reference is to a symbol in a relative section). If no suffix or the suffix {.L} is appended it will use the long absolute addressing form for forward references.

Relative Section.

In order to allow structuring of the relative section the assembler and finally the linkage editor and pusher provide for different relocatable sections within the relative section. Whenever a new section is established by a SECTION pseudo instruction, the location counter will be set to zero. If the suffix .S is appended to the pseudo instruction establishing the section, the assembler will assume, that this section will finally be loaded into low address memory so that direct addressing of the symbols defined in that section may be accomplished through absolute short addresses. Whenever an already established section is resumed by a SECTION pseudo, the location counter will be reset to its last value.

Note: The suffix .S does not imply, that forward references in that section will also be resolved with absolute short addresses. The pseudo instruction FSHORT must be used for this purpose.

G (general) Section.

The section is relocatable and the linkage editor joins all general sections into a single one in the order in which they are processed. In a module only one general section can be defined; but in different modules G - sections may be named differently. The following pseudo instructions cause the assembler to establish or resume assembly in the G - section:

Location	Operation	Variable Subfields
{<symbol>}	SECTION{.S}	G

or (for compatibilty with the Motorola assembler)

Location	Operation	Variable Subfields
	RORG{.S}	

<symbol> An optional symbol to define a name for the G - section. By default the assembler will start generating code in the general section assuming a blank name. The first subsequent use of a SECTION pseudo instruction for the general section might change the default name.

Note: The general section is established by the assembler and it is assumed that it will not necessarily be loaded in low memory, hence long absolute addresses are generated. The user can override this by using a SECTION.S pseudo for the general section prior to any code generation for this section.

R (relocatable) Section.

The section is relocatable. If R - sections that have been processed by the linkage editor have the same name, they are joined into a single one (with that name) in the order in which they were processed. Unnamed R - sections are not joined. The following pseudo instruction causes the assembler to establish or resume assembly in an R - section:

Location	Operation	Variable Subfields
{<symbol>}	SECTION{.S}	R

<symbol> An optional symbol to define a name for the R - section.

C (common) Section.

The section is relocatable. If C - sections processed by the linkage editor have the same name, they are overlapped. Unnamed C - sections (blank common) are also overlapped. A warning message is printed by the linkage editor, if their sizes differ.

Location	Operation	Variable Subfields
{<symbol>}	SECTION{.S}	C

<symbol> An optional symbol to define a name for the C - section.

U (unique) Section.

The section is relocatable and must have a name. If other U - sections processed by the linkage editor have the same name, an error message is printed.

Location	Operation	Variable Subfields
<name>	SECTION{.S}	U

<symbol> A mandatory symbol to define the name of the U - section.

Symbol Definition.

EQU - Equate Symbol Value.

An EQU pseudo instruction permanently defines the symbol in the location field as having the value and attributes indicated by the simple expression in the variable field.

Location	Operation	Variable Subfields
<symbol>	EQU	<expression>

<symbol> A location symbol following the naming rules must be defined.

<expression> A simple expression with at most one relative symbol in the final expression. Forward references are not allowed.

SET - Set or reset symbol value.

A SET pseudo instruction defines the symbol in the location field as having the value and attributes indicated by the simple expression in the variable field. A subsequent SET using the same symbol redefines the symbol to the new value and attributes.

Location	Operation	Variable Subfields
<symbol>	SET	<expression>

<symbol> A location symbol following the naming rules must be defined.

<expression> A simple expression with at most one relative symbol in the final expression. Forward references are not allowed.

Module Linkage.

The pseudo instructions ENTRY and EXTERN do not define symbols but either declare symbols defined within a module as being available outside the module or declare symbols referred to in the module as being defined outside the module.

ENTRY - Declare Entry Symbols.

The ENTRY pseudo instruction specifies which of the symbolic addresses defined in the module can be referred to by modules assembled independently; ENTRY lists entry points to the current module.

Location	Operation	Variable Subfields
	ENTRY	<sym >,<sym >,...,<sym > 1 2 n

<sym >
i — Linkage symbol. Each symbol must be defined in the module as nonexternal (cannot be listed on an EXTERN pseudo instruction).

A list of all entry points declared in the module precedes the assembly listing.

Note: If the ENTRY pseudo declaring a symbol precedes the actual definition of the symbol, the assembler will handle the symbol as if it were going to be defined in the current section. With other words, if the current section is declared to be loaded in lower memory, abolute addresses for the symbols declared in the list of the ENTRY pseudo instruction will be short addresses, else they are long addresses. The user is therefore advised to use the ENTRY pseudo within the section for which the entry symbols are to be declared.

EXTERN - Declare External Symbols.

The EXTERN pseudo instruction lists symbols that are defined as entry points in independently assembled modules for which references can appear in the module being assembled.

Location	Operation	Variable Subfields
	EXTERN{.S}	<sym >,<sym >,....,<sym > 1 2 n

<sym >
i — Linkage symbol. These symbols must not be defined within the module.

The suffix .S, if appended to the EXTERN pseudo instruction, indicates to the assembler, that all symbols contained in the variable field will be finally located in lower memory and can be accessed using the absolute short address form. A list of all external symbols declared in the module precedes the assembly listing.

Data Generation and Storage Reservation.

DC - Define Constant.

Location	Operation	Variable Subfields
{<symbol>}	DC	<opr >,<opr >,....,<opr > 1 2 n
{<symbol>}	DC{.B}	<opr >,<opr >,....,<opr > 1 2 n
{<symbol>}	DC{.W}	<opr >,<opr >,....,<opr > 1 2 n
{<symbol>}	DC{.L}	<opr >,<opr >,....,<opr > 1 2 n

The function of the DC pseudo instruction is to define a constant in memory. The DC directive may have one operand, or multiple operands which are separated by commas. The operand field may contain the actual value (decimal, octal, hexadecimal, or character string). Alternatively, the operand may be a symbol or expression. The constant is aligned on a word boundary if word (.W) or long word (.L) is specified, or a byte boundary if byte (.B) is specified.

The following rules apply to size specifications on character strings:

DC.B If an odd number of bytes (characters) are entered, the odd byte on the right will be zero filled unless the next source statement is another DC.B or DS.B. In this case the next DC.B or DS.B will start in the odd byte on the right.

DC.W If an odd number of bytes (characters) are entered, the last word will be zero filled on the right to force an even byte count.

DC.L If less than a multiple of four bytes are entered, the last long word will be zero filled on the right to a multiple of four bytes.

DS - Define Storage.

Location	Operation	Variable Subfields
{<symbol>}	DS	<expression>
{<symbol>}	DS{.B}	<expression>
{<symbol>}	DS{.W}	<expression>
{<symbol>}	DS{.L}	<expression>

The DS pseudo instruction is used to reserve memory locations. The contents of the memory reserved are not initialized in any way. The expression must evaluate to an absolute value. Forward references are not allowed.

Conditional Assembly.

The pseudo instructions IFEQ and IFNE permit optional assembly or skipping of source code. Immediately following the test instruction are instructions that are assembled when the tested condition is true and skipped when the condition is false. Skipping is terminated either by a source statement count on the IF instruction, or by an ENDIF, an ELSE, or an END.

The statement count, when used, is decremented for instruction lines only; comment lines (identified by * in column one) are not counted. Determining the IF range with a statement count produces slightly faster assembly than using the ENDIF.

The result of an IF test is determined by the value of the expression in pass one of the assembler; the value of a relative symbol is relative to the origin of the section in which it was defined. The value of an external symbol is zero if the symbol was declared as external. If the symbol was defined relative to a declared external, the value is the relative value. IF's may be nested up to ten levels deep.

ENDIF - End of IF Range.

An ENDIF pseudo instruction (or ENDC for compatibility with the Motorola assembler) cause skipping to terminate and assembly to resume. When the sequence containing the ENDIF is being assembled, or is controlled by a statement count, the ENDIF has no effect other than to be included in the count.

Skipped instructions such as macro references are not expanded. Thus, any ENDIF that would have resulted from an expansion is not detected.

Location	Operation	Variable Subfields
{<if name>}	ENDIF	

<if name> <if name>, an optional symbol, defines the name of an IFEQ, IFNE, or ELSE sequence; or blank

Skipping of a sequence initiated by an IFEQ, IFNE, or ELSE that is assigned a name is terminated by an ENDIF specifying the same name. Skipping of a sequence initiated by an unnamed IFEQ, IFNE, or ELSE is terminated by an unnamed ENDIF.

ELSE - Reverse Effects of IF.

Through the ELSE instruction, the assembler provides the facility to reverse the effects of an IF test within the IF range. An ELSE detected during skipping causes assembly to resume at the instruction following the ELSE. An ELSE detected while a sequence is being assembled initiates skipping of source code following the ELSE. Skipping continues until either an END or an ENDIF for the sequence is detected.

Location	Operation	Variable Subfields
{<if name>}	ELSE	

<if name> <if name>, an optional symbol, defines the name of an IFE2, IFNE, or ENDIF sequence; or blank

An ELSE specifying the sequence by name terminates skipping of a sequence initiated by an IFE2 or IFNE with the same name. An unnamed ELSE terminates skipping of a sequence initiated by an unnamed IFE2 or IFNE. Skipped instructions such as macro references are not expanded; any ELSE that would have resulted from the expansion is not detected.

IFE2 - Test Expression is Equal Zero.
IFNE - Test Expression is Not Equal Zero.

The IFE2 and IFNE pseudo instructions test the value of the expression and assemble instruction in the IF range when the condition is satisfied.

Location	Operation	Variable Subfields
{<if name>}	IFE2	<expression>{,<line count>}
{<if name>}	IFNE	<expression>{,<line count>}

<if name> <if name>, an optional symbol, defines the name of the IFE2 or IFNE sequence; or blank

<expression> A simple expression without forward reference. If the expression is erroneous, an error message is printed and assembly continues with the next instruction.

<line count> Optional absolute value specifying an integer count of the number of statements to be skipped.

The <line count>, if specified, takes precedence over an <if name>, if specified at all.

Source Stream Control.

INSERT - Insert Secondary Source.

The INSERT pseudo instruction provides a means of obtaining source statements from a file other than that being used for input. The assembler transfers the text from this file and assembles it before taking the next statement from the interrupted source of statements.

Location	Operation	Variable Subfields
	INSERT	

There are no parameters for the INSERT pseudo instruction. The file to be used is specified when the assembler is called. The file will be rewound before using it.

Listing Control.

The pseudo instructions described in this section permit extensive control of the assembly listing format.

LIST - Select List Options.

The LIST pseudo instruction controls the content and format of the assembler listing. Use of the LIST pseudo instruction is optional. If not specified in a module, or if specified without parameters, the assembler will produce an output according to the default for each possible option.

Location	Operation	Variable Subfields
	LIST	{<op >,<op >,....,<op >} 1 2 n

<op >
i — Optional parameter. A list option or a list option prefixed by a minus sign. The unprefixed option selects the option; the prefixed option cancels it. Options are separated by commas and terminated by a blank.

DC — When DC is selected, the source line of the DC pseudo instruction and its expansion are listed, otherwise only the source line will be listed.
-DC is the default.

IF — When IF is selected, the source lines of the IFEQ, IFNE, ELSE, or ENDIF pseudo instructions and the skipped source statements in the IF range are listed, otherwise the pseudo instructions are listed, but not the skipped source statements.
-IF is the default.

MACRO — When MACRO is selected, the source line of the macro reference and the fully expanded macro body are listed, otherwise only the source line of the outermost macro reference of possibly nested macro calls is listed.
-MACRO is the default.

XOPC When XOPC is selected, the assembler will list the use of all opcodes in the cross reference list.
-XOPC is the default.

XPSE When XPSE is selected, the assembler will list the use of all pseudo instruction in the cross reference list.
-XPSE is the default.

XREG When XREG is selected, the assembler will list the use of all registers in the cross reference list.
-XREG is the default.

NOLIST - Cancel Listing.

The NOLIST pseudo instruction suppresses the printing of the assembly listing until a LIST pseudo instruction is encountered.

Location	Operation	Variable Subfields
	NOLIST	

PAGE - Top of Page.

The PAGE pseudo instruction advances printer paper to a new page before printing. Then page headings are printed and listing continues. The PAGE pseudo instruction does not appear on the program listing.

Location	Operation	Variable Subfields
	PAGE	

SPC - Space Between Source Lines.

The SPC pseudo instruction causes the assembler to output <count> blank lines to the assembly listing. The SPC pseudo instruction does not appear on the program listing.

Location	Operation	Variable Subfields
	SPC	<count>

<count> An absolute value.

TTL - Assembly Listing Title.
STTL - Assembly Listing Subtitle.

The TTL and STTL pseudo instructions allow the programmer to print a title and a subtitle on the top of each page of the listing. To this effect the assembler maintains internally two text strings which are set to blank at the beginning of pass one. In pass two, whenever a new page is started, these two text strings together with other information are printed in the page header. Specifying a title or subtitle merely means, that the contents of the corresponding internal text string is changed to the one specified with the TTL or STTL pseudo instruction. It does not imply an automatic start of a new page. The first specified title is in addition kept in a third internal text string and is copied into the title text string at the start of pass two. Neither the TTL nor the STTL pseudo instruction are listed in the assembly listing.

Location	Operation	Variable Subfields
	TTL	'<text>'
	STTL	'<text>'

<text> <text> specifies the text to be used as title or subtitle. Please note, that the text must be enclosed in apostrophes and cannot contain more than 60 characters.

Object Code Control

The pseudo instructions BLONG, BSHORT, FLONG, or FSHORT allow the programmer to influence the assembler's choice whenever forward references are encountered, be it for absolute addresses or relative branching instructions.

BLONG - Use Two-Word Conditional Branch.
BSHORT - Use One-Word Conditional Branch.

Location	Operation	Variable Subfields
	BLONG	
	BSHORT	

The two pseudo instructions BLONG and BSHORT allow the programmer to influence the assembler whenever it is assembling a conditional branch instruction the label of which contains a forward reference. By default the assembler will use the two-word instruction form allowing a larger relative address range. After a BSHORT pseudo instruction the assembler will generate the one-word relative branch instruction, unless the suffix .L has been appended to that branch instruction. The occurrence of a BLONG pseudo instruction forces the two-word relative branch instruction to be generated, unless the suffix .S has been appended to that branch instruction.

FLONG - Force Absolute Long Address.
FSHORT - Force Absolute Short Address.

Location	Operation	Variable Subfields
	FLONG FSHORT	

The two pseudo instructions FLONG and FSHORT allow the programmer to influence the assembler whenever it is assembling an absolute address the label of which contains a forward reference. By default the assembler will use the long absolute address form. After an FSHORT pseudo instruction the assembler will generate the absolute short address form. The occurence of a FLONG pseudo instruction forces the absolute long address form to be generated.

Note: The selected option, long or short absolute addresses, is only valid until the next occurence of a FLONG, FSHORT or SECTION pseudo instruction. The latter will reset it to the default, namely long absolute addresses.

NOOBJ - Suppress CUFOM Output.

The pseudo instruction NOOBJ suppresses the generation of a CUFOM module.

Location	Operation	Variable Subfields
	NOOBJ	

MACRO OPERATIONS

A macro definition is a sequence of source statements that are saved and then assembled whenever needed through a macro call. A macro call consists of the occurrence of the macro name in the operation field of a statement. It usually includes parameters to be substituted for formal parameters in the macro code sequence so that code generated can vary with each assembly of the definition.

Use of a macro requires two steps, definition of the macro, and calling of the definition.

A definition consists of three parts: heading, body, and terminator.

Heading — A macro definition is headed by a MACRO pseudo instruction stating the name of the macro. The heading optionally includes a LOCAL pseudo instruction indentifying symbols local to the definition.

Body — The body begins with the first statement in a definition that is not a LOCAL pseudo instruction or a comment line. The body consists of a series of symbolic instructions. All instructions other than END or another MACRO definition are legal within a definition. The assembler recognizes substitutable arguments in all fields of the source line. The macro argument \0 however can only be used in the operation field for referring to the data size subparameter in an opcode or pseudo instruction. The arguments \1 through \9 can appear anywhere in a source line. Ten is the maximum number of arguments that can be handled by any macro definition. Macro calls may be nested up to ten levels deep.

Terminator — An ENDM pseudo instruction terminates a macro definition.

ENDM - End Macro Definition.

Location	Operation	Variable Subfields
	ENDM	

An ENDM pseudo instruction terminates the macro definition.

LOCAL - Local Symbols.

The LOCAL pseudo instruction, which lists symbols local to the definition optionally follows the MACRO pseudo instruction.

Location	Operation	Variable Subfields
	LOCAL	<sym >,<sym >,....,<sym > (1, 2, 10)

<sym > (i) — List of local symbols. Symbols must be separated by commas. A blank terminates the list. The maximum number of local symbols is 10.

A symbol in the list is considered local to the macro; that is, it is known only within the macro definition. On each expansion of the macro, the assembler creates a new symbol for each local symbol and substitutes it for each occurrence of the local symbol in the definition. Thus invented symbols replace LOCAL-named symbols wherever they appear in a macro definition in a manner similar to the way substitutable parameters are replaced.

MACRO - Macro Heading.

A MACRO pseudo instruction tells the assembler to place the instructions forming the body of the macro in a table of macro definitions for assembly upon call, and to place the macro name in the symbol table.

Location	Operation	Variable Subfields
<m name>	MACRO	

<m name> — <m name>, a mandatory symbol, defines the name of the macro.

MACRO CALLS

A macro headed by a MACRO pseudo instruction can be called by an instruction in the following format:

Location	Operation	Variable Subfields
{<symbol>}	<m name>	<p >,<p >,....,<p > 1 2 i
{<symbol>}	<m name>{.S}	<p >,<p >,....,<p > 1 2 i
{<symbol>}	<m name>{.B}	<p >,<p >,....,<p > 1 2 i
{<symbol>}	<m name>{.W}	<p >,<p >,....,<p > 1 2 i
{<symbol>}	<m name>{.L}	<p >,<p >,....,<p > 1 2 i

<symbol> An optional location symbol.

<m name> Name of a previously defined macro.

<p >
i Parameter list composed of strings of characters. Parameters are separated by commas and terminated by a blank. Two consecutive commas constitute a null parameter. An explicit zero, if desired, must be entered.

If null parameters are interspersed with legal parameters, the correct positions must be established with commas. When the list terminates before the last possible parameter, all remaining parameters a considered null.

When the first character of a parameter is a left angular bracket (<), the assembler considers all the characters between it and the matching right angular bracket (>) as an embedded parameter. The assembler removes the outer pair of angular brackets before substituting the enclosed character string in a line. Embedded parenthetical items must be properly paired. A parenthetical item can contain blanks and commas.

Cross Macro Assembler
for
Motorola 68000

HOW TO USE THE ASSEMBLER

The assembler has been designed as a two pass assembler and is written in PASCAL. Currently it is installed on the IBM computers in the computer centre and can be called using the following command line under WYLBUR:

```
EXEC FROM #MICASM PUB
```

The EXEC program will ask the user for his options and will produce the necessary set of Job Control Language statements to assemble a source program and to produce a CUFOM module.

CUFOM modules generated by the assembler can now be linked together using the target independent CUFOM linkage editor by typing:

```
EXEC FROM #MICLINK PUB
```

This EXEC program will prompt the user for the names of the data sets which contain the CUFOM modules to be linked. It also allows the definition of a CUFOM library from which to satisfy undefined externals. Such a CUFOM library can be generated using the EXEC program:

```
EXEC FROM #MICLIB PUB
```

If the user now wants to produce information ready for down-line loading, he will have to type:

```
EXEC FROM #MICPUSH PUB
```

This EXEC program will request the name of the data set containing the CUFOM module to be pushed and will offer the user default origins for all the relative sections contained in the module. These defaults can be overridden by the user. If an absolute section exists in the module, the default origin for the first relative section is the address of the memory word immediately following the last load address of the absolute section. If there is no absolute section in the module, address zero is offered as origin. All other default origins are calculated using the origin and length of the preceding relative section. Thus a compact load can be achieved.

Should you experience any problems, encounter errors or want to suggest improvements for the software described above, please contact:

Horst von Eicken
DD-Division
C E R N
1211 GENEVA 23
Switzerland
Tel. (022) 83 23 63

Cross Macro Assembler
for
Motorola 68000

Availability of M68MIL

Program Availability and Charging

M68MIL is part of the CERN Program Library and may be used freely inside CERN. Universities, national laboratories and other non-profit organizations in Member States may receive M68MIL and its documentation freely and without charge, subject only to a limitation on the number of copies to be sent to any one institution free of charge.

Other organizations (commercial firms) are expected to pay a small fee to cover our immediate expenses in supplying the desired material. This fee is 150.-- SFrs. for M68MIL and includes the cost of a magnetic tape.

Universities and other non-profit organizations in non Member States normally pay a small fee as in the preceding paragraph, but the fee is waived in the case of laboratories collaborating on CERN experiments or engaged in a reciprocal exchange agreement with CERN.

CERN reserves the right to charge a higher fee, up to the full cost of developing the programs, or to refuse any request which it deems to be not in the interest of the Organization.

The source is available in card image form at 80 chars per card in either EBCDIC or ASCII on 9-track tape at densities of either 800, 1600 or 6250 bpi. Please specify which of the above you require and in addition the number of cards per physical block (default=10).

Conditions of Use

Programs and documentation are provided solely for the use of the organization to which they are distributed and may not be redistributed to any third party without the express agreement of CERN.

The material cannot be sold.

CERN should be given credit in all references, library documentation, and publications based on the programs.

CERN undertakes no obligation for maintenance of the programs, nor responsibility for their correctness, and accepts no liability whatsoever resulting from the use of its programs. Although we do not "support" the programs in a commercial sense, we will answer questions concerning the implementation or usage of the programs and we welcome suggestions for improvements.

Requests for M68MIL should be addressed to:

The Program Librarian
Data Handling Division
CERN

CH 1211 GENEVA 23

SWITZERLAND