COMMON ASSEMBLY LANGUAGE MACRO/32 PROCESSOR (CAL MACRO/32) LIBRARY UTILITY

Reference Manual

OS/32 Version 6.0 or higher

48-057 F00 R00

The information contained in this document is subject to change without notice. Concurrent Computer Corporation has taken efforts to remove errors from this document, however, Concurrent Computer Corporation's only liability regarding errors that may still exist is to correct said errors upon their being made known to Concurrent Computer Corporation.

The software described in this document is furnished under a license, and it can be used or copied only in a manner permitted by that license. Any copy of the described software must include all copyright notices, trademarks, or other legends or credits of Concurrent Computer Corporation and/or its suppliers. Title to and ownership of the described software and any copies thereof shall remain in Concurrent Computer Corporation and/or its suppliers.

The licensed program described herein may contain certain encryptions or other devices which may prevent or detect unauthorized use of the Licensed Software. Temporary use permitted by the terms of the License Agreement may require assistance from Concurrent Computer Corporation.

Concurrent Computer Corporation assumes no responsibility for the use or reliability of this software if used on equipment that is not supplied by Concurrent Computer Corporation.

Concurrent Computer Corporation, 2 Crescent Place

Oceanport, New Jersey 07757

Printed in the United States of America

TABLE OF CONTENTS

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

APPENDIXES

FIGURES

PREFACE

This manual describes the Perkin-Elmer Common Assembly Language (CAL) Macro/32 Processor, Program Number 03-339, and its use in defining macro instructions for frequently used sequences of assembler code, for creating a macro library, and for expanding macro instructions during the assembly process.

Chapter 1 introduces the CAL Macro/32 Processor and explains processor requirements, processor components, and summarizes macro processor features. Chapter 2 details macro instructions, macro definitions, special symbols, macro definition contents, and model statements. Chapter 3 explains how to write a macro instruction, macro instruction operands, and inner and outer macro instructions. Chapter 4 is an in-depth discussion of the conditional expansion of macro definitions. Additional CAL Macro/32 features are presented in Chapter 5, and the operation of the macro processor is discussed in Chapter 6. Chapter 7 details each macro library utility command.

Appendix A is a command summary of the CAL Macro/32 Processor utility commands. Appendix B contains the macro instructions and statements. Appendix C contains examples of macro expansion, and Appendix D presents the CAL Macro/32 Processor error messages. The CAL Macro/32 Processor now supports batch global set variable symbols, passes MLIBS and MCOPY from start options to all programs in the batch, and generates an END statement without terminating macro expansion if it is within the scope of an ASIS statement.

This manual replaces S29-408 and discusses revision R02.1 of the CAL Macro/32 Processor and the Macro Processor Utility. This revision applies to the OS/32 R06 software release and higher.

These manuals provide information related to the use of the various programs in the CAL Macro/32 Processor:

MANUAL TITLE	PUBLICATION NUMBER
OS/32 System Macro Library Reference Manual	48-006
OS/32 Library Loader Reference Manual	48-020
OS/32 Operator Reference Manual	48-030

Common Assembly Language/32 (CAL/32) Reference Manual	48-050
32-Bit Systems User Documentation Summary	50-003

For further information on the contents of all Perkin-Elmer 32-bit manuals, see the 32-bit Systems User Documentation Summary.

CHAPTER 1
CAL MACRO/32 PROCESSOR

1.1 INTRODUCTION

The Perkin-Elmer CAL Macro/32 Processor provides the user of CAL with a tool to standardize and efficiently generate programs. This ability is provided by the system macro library and any user designated special purpose libraries. Frequently used assembler code sequences are defined once and then appended to a macro library by a macro definition. By inserting a single source statement that the processor recognizes and expands, these code sequences are made available to all system users.

1.2 CAL MACRO/32 PROCESSOR REQUIREMENTS

The CAL Macro Processor requires:

- Any Perkin-Elmer 32-bit processor
- Main memory of 26kb above that memory required for the operating system and macro system table
- OS/32
- Source input, source output, and listing devices

1.2.1 Configuration Option

The CAL Macro Processor can use any Perkin-Elmer peripheral device that the operating system supports, if the program can run on the device and is capable of ASCII data transfer.

1.2.2 Relationship to Other Products

The CAL Macro Processor produces an expanded source stream that must be assembled with CAL (03-066R05 or higher).

1.3 CAL MACRO/32 PROCESSOR COMPONENTS

The CAL Macro Processor package consists of:

- CAL Macro Library Utility Object, 03-340M
- CAL Macro Processor Object, 03-339M
- OS/32 System Macro Library, 07-217
- CAL Macro/32 Processor and Macro Library Utility Reference Manual
- OS/32 System Macro Library Reference Manual

1.4 SUMMARY OF CAL MACRO/32 PROCESSOR FEATURES

The CAL Macro Processor offers these features:

- Positional, keyword, or mixed mode macro prototype statements
- Nested macro instructions
- Conditional macro expansion independent of assembler conditional statements
- Symbolic parameters that can vary the operation codes expanded to the assembler source stream
- A macro call (MCALL) instruction that allows the most used macro definitions to be called into memory at the start of macro processor execution, thus decreasing the number of library accesses necessary during the processor pass
- A macro trace (MTRAC) facility that allows the user to test a macro instruction expansion without going through the full assembly process
- User designation of macro libraries and the order in which they should be searched
- A system macro library of standard macro definitions
- A macro library utility program that builds and maintains the macro libraries. Special features allow adding and deleting macro definitions, generating a library table of contents, and copying macro definitions from existing macro libraries.

The power of the macro language is shown in its ability to:

- pass parameters to a macro and concatenate the parameter to characters in the macro to form new labels, operations, and parameters that allow the user to write macros for many "housekeeping" functions that normally would have to be performed in a program; and
- define symbols local to the macro, assign values to those symbols, and make complex tests on the values of those symbols or on parameters passed to the macro.

CHAPTER 2
PREPARATION OF A MACRO DEFINITION

2.1 INTRODUCTION

The following sections discuss what a macro instruction is, its interaction with the Common Assembly Language (CAL) Macro/32 Processor, and the preparation of a macro definition.

2.2 MACRO INSTRUCTIONS

A macro instruction is a single instruction that expands to a series of instructions. A macro instruction is written like an assembler instruction; but the output, when processed by the CAL Macro Processor Program, is in assembly language.

The rules for the syntax of a macro instruction are:

- Columns 1 through 8 contain a symbol or blanks.
- Columns 10 through 17 contain the macro name.
- At least one blank space must be on either side of the macro name.
- One blank space separates the label field from the macro call, and one blank space separates the macro call from the parameter.
- A comma must be specified to show omitted positional parameters.
- Keyword parameters can be written in any order.
- Subparameters are enclosed in parentheses and can only be positional.

The output of a macro instruction can be:

- machine instructions,
- another macro instruction,
- assembler instructions, or
- a combination of machine and assembler instructions.

This output process is the macro expansion. The assembler processes the output as if the user had written the expanded coding in detail. Macros are a valuable coding tool--they enable the user to avoid writing many system required details; thus, reducing the chance of errors. The user can use the standard Perkin-Elmer macros in the system macro library or write his own macros and store them in a user library.

2.3 MACRO DEFINITIONS

A macro definition is a series of user-written instructions in the macro language. The definition of a macro can include:

- machine instructions such as ADD, SUBTRACT, LOAD, or STORE;
- assembler instructions such as DS or DC;
- macro language instructions such as AIF, AGO, or SETA; or
- another macro instruction, which would be an inner macro.

The macro definition allows the user to assign a macro instruction to be used in the operation field of the macro instruction syntax. A macro instruction retrieves and processes the definition.

If the macro definition has no errors, processing the macro definition call results in generating zero or more assembler source statements that become part of the assembler source stream and appear in the assembler source stream immediately after the macro instruction. The assembler source statements the processor produces are referred to as generated (or expanded) statements. The macro expansion process involves the processor analyzing the macro instruction and definition, and generating assembler source statements.

2.3.1 Macro Definition Fields

A macro definition minimally consists of these fields:

- The name field begins in column 1. If the name field is omitted, column 1 must be blank.
- The operation field must start in column 10. If concatenation is not used, the maximum length of the operation field is eight characters. If concatenation is used, the operation field can exceed eight characters. Whether or not concatenation is used, the operation field of a generated statement must contain from one to five characters. If the generated statement is a macro instruction, the operation field can contain a maximum of eight characters.

- The operand field follows the operation field, separated by at least one blank. The operand field can extend as far as column 71 on a single statement line and can be continued in another line by inserting a nonblank character in column 72 (the continuation field). A generated CAL statement, however, cannot contain an operand field extending beyond a single statement line. Continuation can be invoked only from the operand field.

- The comment field follows the operand field, separated by at least one blank column. This field contains user comments.

- The continuation field follows the comments field. In a keyword or mixed mode prototype statement, each operand can appear on a single line if it is followed by a comma, and a nonblank character appears in the continuation field.

- The identification/sequence field occupies columns 73 through 80. The user has the option of identifying and maintaining the sequence of the source field.

Figure 2-1 illustrates the fields used for writing macro instructions.

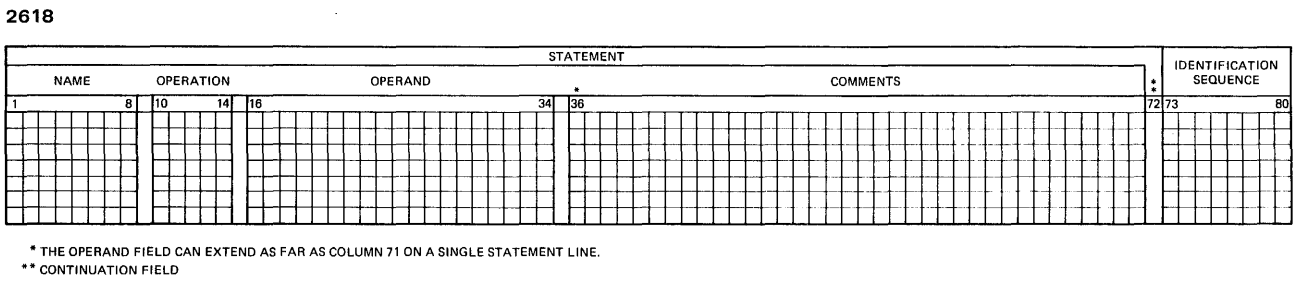


Figure 2-1 Fields Used for Writing Macro Instructions

2.4 SPECIAL SYMBOLS

The following symbols have special meanings to the CAL Macro Processor:

SYMBOL	MEANING
%	A percent sign identifies variable symbols used within the macro definition.
:	A colon is a concatenation symbol; or, if it is found in column 1 followed by an asterisk in column 2, it identifies comments internal to the macro definition.
&	An ampersand identifies sequence symbols used in a macro definition.

For each of these symbols except the ampersand (&), if two symbols are input, only one symbol is output.

2.4.1 Variable Symbols

Macro language allows the user to define variable symbols, assign values to variable symbols, and test the values of variable symbols. The macro processor uses variable symbols as symbolic parameters, system symbols, and set symbols. It uses variable symbols like the assembler uses symbolic names. Variable symbols can be used in arithmetic expressions, binary or Boolean expressions, or character expressions. These guidelines apply to variable symbols:

- The first character in a variable symbol must be a percent sign (%).
- The second character in a variable symbol must be a letter.
- The remaining zero to six characters can be letters or digits.
- Characters 2, 3, and 4 must not form the word SYS because these letters define system variables.

Valid examples of variable symbols are:

```
%REG      %AREA      %LOC      %LOC1      %A123
%INDEX    %LABLE     %NAME     %LIST
```

2.4.1.1 Local, Global, and Batch Global Variable Symbols

Variable symbols can be local, global, or batch global. Parameters are always local. If a variable symbol is batch global, the value assigned to it in one program can be used in another program in the same batch. The macro processor initializes batch global variables when they are encountered first in the batch. The macro processor does not initialize subsequent definitions of batch global variables in that batch. If a variable symbol is global, the value assigned in one macro can then be used in another macro. The macro processor initializes global variables when they are first encountered in a program. The processor does not reinitialize subsequent definitions of global variables in that program. Local variables are initialized each time they are defined. They must be defined before they can be used, and their values do not carry from one macro definition to another.

New values can be reassigned to local, global, and batch global variable symbols by these macro language instructions:

INSTRUCTION	MEANING
SETA	Assign arithmetic value.
SETB	Assign binary or Boolean value of 0 (false) or 1 (true).
SETC	Assign character value.

Section 4.3 details these SET variable symbol statements.

2.4.1.2 Defining Variable Symbols

A variable symbol is defined explicitly in the body of the macro definition. It is assigned a value, which can be changed, in the macro body. These macro language statements define and initialize variable symbols:

STATEMENT	MEANING
LCLA	Local arithmetic; initial value 0
LCLB	Local binary or Boolean; initial value 0
LCLC	Local character; initial value null ''
GBLA	Global arithmetic; initial value 0
GBLB	Global binary or Boolean; initial value 0
GBLC	Global character; initial value null ''

STATEMENT	MEANING
BGBLA	Batch global arithmetic; initial value 0
BGBLB	Value batch global binary or Boolean; initi 1 value 0
BGBLC	Batch global character; initial value null '

Sections 4.2.1 and 4.2.2 detail these
local, global, and batch global
SET variable symbol declaration statements.

2.4.2 Concatenation Symbols

Variable symbols can be concatenated on the left or right with any other characters to form a new string. This concatenation can occur in the label, operation, or operand fields. The ability to concatenate is advantageous because the macro expansion can be different in different calls.

When concatenating to the right of a variable symbol, a colon indicates the concatenation. The colon is optional except when the next character is:

- alphanumeric,
- a colon, or
- a left parenthesis [( ].

Valid examples of concatenation symbols are:

```
ABC%SYM        %SYM:(R4)        %SYM:ABC
```

2.4.3 Sequence Symbols

Sequence symbols are used to branch within a macro. They can appear in a statement name field to vary the statement processing sequence. These guidelines apply to sequence symbols:

- The first character must be an ampersand (&).
- The second character must be a letter.
- The remaining zero to six characters can be letters or digits.
- Sequence symbols can appear in the name field of any statement not containing a symbol except a prototype statement, an ACTR, BGBLA, BGBLB, BGBLC, GBLA, GBLB, GBLC, LCLA, LCLB, LCLC, or MACRO instruction.

Valid examples of sequence symbols are:

```
&DONE       &END        &MORE       &A123
&NEXT       &AGAIN      &LOOKUP
```

2.5 MACRO DEFINITION CONTENTS

The contents of a macro definition are written in this sequence:

1. A macro header, indicated by the word MACRO, is written in column 10.

2. A prototype statement, the method by which the macro is called, is written next. If a macro call is to have a label or parameters, they must be defined in the prototype statement.

3. The macro definition body can contain optional model statements, which are assembler instructions. The macro body can also contain any macro instructions. In general, macro instructions declare variables and assign values to variables, contain unconditional branches, and conditional branches. Examples are:

   - Local variable (LCLx), global variable (GBLx), and batch global variable (BGBLx) declaration statements

   - MEXIT, MNOTE, SETx, AIF, and AGO instructions

4. A macro definition is terminated with the macro trailer, MEND.

This example illustrates a macro definition:

```
NAME        OPERATION       OPERAND          TYPE

            MACRO                            1
%NAME       MOVE        %FROM,%TO            2
%NAME       ST          1,HOLD               3
            L           1,%FROM              3
            ST          1,%TO                3
            L           1,HOLD               3
            MEND                             4
```

LEGEND FOR TYPE COLUMN

1 = macro header
2 = macro prototype
3 = model statement
4 = macro trailer

Macro definitions can be placed in the source stream or in special libraries. If they are defined in the source program, they must be written before any calls to the macro. If a macro is defined in a library, the appropriate logical unit must be assigned to define the macro library and a given MLIBS statement. See Section 5.1.5.

2.5.1 Macro Header and Trailer Statements

The macro header, which indicates the beginning of a macro definition, must be the first statement. The MACRO statement is written in this format:

NAME	OPERATION	OPERAND	COLUMN 72
optional	MACRO	version number and date, or any other optional description	blank

The macro trailer, which indicates the end of a macro definition, must be the last statement in the definition. The MEND statement is written in this format:

NAME	OPERATION	CPERAND	COLUMN 72
a sequence symbol or blank	MEND	blank	blank

2.5.2 Macro Instruction Prototype Statements

The macro is called by the macro prototype statement. If the macro call is to have any labels or parameters, they must be defined in the prototype statement. The parameters can be positional, keyword, or mixed mode.

The prototype statement specifies the mnemonic operation code and general format to be used when writing any macro instructions referring to this definition. This statement must immediately follow the macro header, MACRO.

2.5.2.1 Positional Macro Instruction Prototype Statements

A positional macro instruction prototype statement contains positional parameters that must be specified in the defined order. The positional prctctype statement is written in this format:

NAME	OPERATION	OPERAND
a symbolic parameter or blank	a 1- to 8-character mnemonic operation code	zero or more symbolic parameters separated by commas

The symbolic parameters used in the macro definition represent the name field and operands of the corresponding macro instruction. A symbolic parameter is a variable symbol consisting of a percent sign followed by from one to seven letters or numbers, the first of which must be a letter.

A macro instruction must use the mnemonic operation code to refer to this macro definition. If two macro definitions use the same mnemonic operation code, the first definition the macro processor encounters is expanded; the second definition is flagged with an error message. If a macro definition uses the same mnemonic operation code as a valid assembler or machine instruction, the macro processor treats it as a macro instruction. To override the macro instruction and obtain the machine instruction, use the ASIS statement. See Section 5.1.1.

A positional prototype statement can be continued if the operand field extends beyond column 71. Section 3.3.1 explains continuation rules. This example illustrates a positional macro instruction prototype statement:

NAME	OPERATION	OPERAND
%NAME	MOVE	%FROM,%TO

2.5.2.2 Keyword Macro Instruction Prototype Statements

In a keyword prototype statement, any symbolic parameter in the operand field of the prototype statement can have a standard value assigned to it. If a symbolic parameter is omitted when the macro instruction is written, the macro processor substitutes the standard value. This substitution allows the user to omit symbolic parameters whose values are not to be changed when writing a keyword prototype statement. The operands in a macro instruction that reference a macro definition with a keyword prototype statement can be written in any sequence. A keyword prototype statement can be continued to the next statement. Section 3.3.1 explains the rules for continuation. The keyword prototype statement is written in this format:

NAME	OPERATION	OPERAND
any symbolic parameter or blank	a 1- to 8-character mnemonic operation code	one or more operands separated by commas and consisting of a symbolic parameter, immediately followed by an equal sign followed (optionally) by a standard value.

With the exception of variable symbols, whatever can be used as an operand in a macro instruction can be used as a standard value. Following is an example of a macro using a keyword prototype macro statement:

```
     NAME      OPERATION      OPERAND                              TYPE

                MACRO                                               1
     %SYM       MOVE         %FROM=,%TC=,%HOLD=HOLD1,%R=1          2
     %SYM       ST           %R,%HOLD                              3
                L            %R,%FROM                              3
                ST           %R,%TO                                3
                L            %R,%HOLD                              3
                MEND                                               4
     A          MOVE         FROM=SOURCE,R=5,TO=DEST               5
     A          ST           5,HOLD1                               6
                L            5,SOURCE                              6
                ST           5,DEST                                6
                L            5,HOLD1                               6
```

LEGEND FOR TYPE COLUMN

1 = macro header
2 = macro prototype
3 = model statement
4 = macro trailer
5 = macro instruction
6 = generated statement

2.5.2.3 Mixed Mode Macro Instruction Prototype Statements

Macro instructions can be defined with positional and keyword parameters. When a macro is called, the positional parameter must be specified first in the defined order. The positional parameters are followed by keyword parameters in any order. Omitted positional parameters must be indicated by a comma, except when all trailing positional parameters are omitted. The mixed mode prototype statement is written in this format:

NAME	OPERATION	OPERAND
any symbolic parameter or blank	a 1- to 8-character mnemonic code	one or more positional parameters followed by one or more keyword parameters (see Section 2.5.2.2)

The following is an example of a mixed mode prototype statement:

```
NAME        OPERATION        OPERAND
%SYM          EX4          %P1,%P2,%P3,%K1=,%K2=
                .
                .
                .
```

The following is a macro instruction with the second positional parameter omitted. It is coded as:

```
EX4       ABC,,3,K1=ALPHA
```

The following is a macro instruction with the second and third positional parameters omitted. It is coded as:

```
EX4       ABC,K2=BETA
```

A mixed mode instruction can be continued to the next statement. Section 3.3.1 explains continuation rules.

2.6 MODEL STATEMENTS

The CAL Macro Processor expands the desired source statements from model statements. Any number of model statements can appear within a given macro definition. Model statements consist of four fields:

- Name
- Operation
- Operand
- Comments

These fields correspond to the same fields in the expanded source statement. The CAL Macro Processor does not check the expanded source statements for the correct assembler syntax.

2.6.1 Name Field

The name field, which begins in column 1, can be blank or it can contain a:

- name,
- variable symbol, or
- sequence symbol.

It can also contain a name concatenated with a variable symbol or a variable symbol concatenated with one or more other variable symbols.

2.6.2 Operation Field

At least one blank character separates the operation field from the name field. It can contain:

- a machine instruction,
- an assembler instruction,
- a macro instruction, or
- a variable symbol.

It can also contain a name concatenated with a variable symbol or a variable symbol concatenated with one or more other variable symbols.

Variable symbols cannot be used to generate more than one field at a time, macro prototypes, or these instructions:

ACTR	END	MCALL	MTRAC
AGC	GBLA	MCOPY	NDEFS
AIF	GBLB	MDEFS	NOLIB
ANOP	GBLC	MEND	NTRAC
ASIS	LCLA	MEXIT	SETA
BGBLA	LCLB	MLIBS	SETB
BGBLB	LCLC	MNOTE	SETC
BGBLC	MACRO	MPAUS	

2.6.3 Operand Field

At least one blank character separates the operand field from the operation field. It can contain:

- names,
- variable symbols, or
- constants.

It can also contain names concatenated with other symbols or variable symbols concatenated with one or more other variable symbols.

2.6.4 Comments Field

At least one blank character separates the comments field from the operand field. This field cannot extend beyond column 71 of a model statement. The comments field can contain any combination of characters and it is passed to the expanded statement exactly as it appears in the model statement. Variable symbols in a comment field have the value substituted. Example:

NAME	OPERATION	OPERAND	COMMENTS
field	field	field	field

2.6.5 Using Symbolic Parameters in Model Statements

A symbolic parameter is defined in the prototype statement and is assigned a value when the argument calls the macro. A symbolic parameter cannot have its value changed in the macro definition body. It is always local to the macro definition; the same parameter name can be in several macros.

By varying values given to symbolic parameters, the user can vary the statements generated for each macro instruction. If a symbolic parameter appears in a model statement, it must have been defined in the name or operand field of the prototype statement or an expansion error occurs. The following example demonstrates how the macro instruction operands that invoked the definition replace the symbolic parameters of the model statement:

```
NAME      OPERATION     OPERAND            TYPE

          MACRO                            1
%NAME     MOVE          %FROM,%TO          2
%NAME     ST            1,HOLD             3
          L             1,%FROM            3
          ST            1,%TO              3
          L             1,HOLD
          MEND                             4

SYMBCL    MOVE          HERE,THERE         5

SYMBOL    ST            1,HOLD             6
          L             1,HERE             6
          ST            1,THERE            6
          L             1,HOLD             6
```

LEGEND FOR TYPE COLUMN

1 = macro header
2 = macro prototype
3 = model statement
4 = macro trailer
5 = macro instruction
6 = generated statement

2.6.5.1 Concatenation Rules

A symbolic parameter in a model statement is concatenated with other symbolic parameters or variable symbols immediately preceding or succeeding the symbolic parameter. An example is:

```
    ST%SUFF  4,%DEST%SUFF
```

ST%SUFF means concatenate ST with %SUFF to form STH (when %SUFF is 'H'). %DEST%SUFF means concatenate %DEST with %SUFF.

NOTE

The concatenation is implicit here.

If a symbolic parameter is to be concatenated with a letter, number, or left parenthesis, the symbolic parameter must be immediately followed by a colon. In this case, the characters corresponding to the symbolic parameter replace the symbolic parameter and colon. Carefully distinguish between array references or sublist notation and concatenation with a left parenthesis in a model statement:

```
    ST    1,%TO:(1)
```

means concatenate %TO with (to from THERE(1).

```
    ST    1,%TO(1)
```

is the first element of the array %TO.

For a single colon to appear in a statement, use two consecutive colons. The following example illustrates the rules for concatenating symbolic parameters:

```
NAME        OPERATION       OPERAND                 TYPE

            MACRO                                   1
%NAME       MOVE        %TY,%FROM,%TO,%X1           2
%NAME       ST%TY       1,HOLD                      3
            L%TY        1,%FROM:A                   3
            ST%TY       1,%TO:(%X1)                 3
            L%TY        1,HOLD                      3
            MEND                                    4

SYMBOL      MOVE        H,HERE,THERE,13             5

SYMBOL      STH         1,HOLD                      6
            LH          1,HEREA                     6
            STH         1,THERE(13)                 6
            LH          1,HOLD                      6
```

LEGEND FOR TYPE COLUMN

1 = macro header
2 = macro prototype
3 = model statement
4 = macro trailer
5 = macro instruction
6 = generated statement

CHAPTER 3
WRITING MACRO INSTRUCTIONS

3.1 INTRODUCTION

A macro instruction is an instruction used to declare variables, and to assign values tc variables, unconditional branches, and conditional branches. The following sections detail rules for writing macro instructicns, types of macro instructions, inner and outer macro instructions, and levels of macro instructions.

3.2 RULES FOR WRITING MACRO INSTRUCTIONS

The following sections explain the rules for writing macro instructions.

3.2.1 Macro Instruction Name Field

A name, if used in the macro instruction name field, does not appear in a generated statement unless a specific symbolic parameter appears in both the name fields of the prototype statement and expanded model statement.

3.2.2 Macro Instruction Operation Field

The operation field contains a mnemonic operation code. This code must be the same prototype statement code contained in a macro definition. This macro definition appears in a macro library, or it previously appeared in the source stream. The macro processor uses the applicable macro definition to determine the statements to be expanded to the assembler source stream. If a macro definition in the processor source stream and a macro definition in a macro library have the same mnemonic operation code, the definition in the source stream is processed.

3.2.3 Macro Instruction Operand Field

Any combination of characters can be used as a macro instruction operand if these rules are followed:

- A quoted string is a character sequence enclosed in paired apostrophes. The paired apostrophes are the first and last apostrophes in a quoted string. The first even numbered apostrophe, not immediately preceding another apostrophe, must

terminate a quoted string. If an apostrophe is used as a character in a quoted string, it must be expressed as two consecutive apostrophes.

For example:

- In the quoted string, 'ABCDEF', the apostrophes preceding the A and following the F are paired apostrophes.

- In the quoted string, 'DON''T', the apostophes preceding the D and following the T are paired apostrophes. The apostrophes between the N and T generate a single apostrophe as a part cf the string.

- Paired parentheses consist of a left and following right parentheses without any other intervening parentheses. If paired parentheses are nested, identify each pair; then, find the left parenthesis and the following right parenthesis with no other intervening parenthesis. The maximum pairs of parentheses that can be nested is 15. When considering paired parentheses, ignore a single parenthesis enclosed in paired apostrophes.

  For example:

  - In the expression, (A-B), the parentheses preceding the A and following the B are paired.

  - In the expression, ((A-B)-C), the leftmost and rightmost parentheses are paired; the innermost parentheses are also paired.

- A percent sign (%) identifies a symbolic parameter unless it appears between paired apostrophes. Use two consecutive percent signs to ensure that a single percent sign appears in a statement.

- A comma (,) indicates the end of an operand, unless it is placed between paired apostrophes.

- A blank indicates the end of the operand field, unless it is placed between paired apostrophes.

- A colon (:) indicates concatenation, unless it appears between paired apostrophes. Use two consecutive colons to ensure that a single colon appears in a statement.

3.3 MACRO INSTRUCTION OPERANDS

A macro instruction invokes a given macro definition. Macro instructions take three forms: positional, keyword, or mixed mode. These three forms correspond to the three forms of prototype statements. See Section 2.5.2. The macro instruction statement format is:

NAME	OPERATION	OPERAND
symbol or blank	a mnemonic operation	positional: zero or more operands separated by commas keyword: one or more operands separated by commas and consisting of a keyword, immediately followed by an equal sign, followed (optionally) by a value mixed mode: positional operands followed by keyword operands

- Positional Macro Instructions

  - In a macro instruction containing positional operands, the placement of the symbolic parameters in the operand field of the prototype statement determines the placement of operands.

- Keyword Macro Instructions

  - A keyword operand is the portion of a symbolic parameter that does not include the percent sign. Anything that can be used as an operand value in a positional macro instruction can be used as a value in a keyword macro instruction. Each keyword operand in a macro instruction must consist of a keyword immediately followed by an equal sign (=), followed (optionally) by a value.

  - Operands in a keyword macro instruction can be written in any order or can be omitted. If a keyword operand is omitted, its delimiting comma can also be omitted.

- Mixed Mode Macro Instructions

  - In a mixed mode macro instruction, all positional operands must be placed before any keyword operands.

Example:

```
| NAME          OPERATION        OPERAND          COLUMN 72
|
| symbol or     MOVE             A,B,LEN=80       blank
| blank
```

In this example, A and B are positional operands, and LEN=80 is a keyword operand.

Apply these general rules to the operand field of a positional, keyword, or mixed mode macro instruction:

- A comma must follow each operand, but a comma does not follow the last operand.
- A single comma, followed immediately by another comma, indicates that an operand does not exist.
- If a continuation character (Section 3.3.1) is not present, use at least one blank space to indicate the end of the operand field.
- Use a comma to indicate omitted positional operands.

3.3.1 Continuation of Macro Instructions

Macro instruction statements can be continued if the operand field extends beyond cclumn 71. To continue macro instruction statements:

- Column 72 must contain a nonblank character.
- Each operand can appear on a separate line.
- A comma must follow each operand except the last.

The following example illustrates continuation of a macro instruction containing keyword operands:

```
|  NAME          OPERATION        OPERAND          COLUMN 72
|
|  symbol or     MOVE             FROM=HERE,          X
|  blank                          TO=THERE,           X
                                  BYTES=80
```

The following example illustrates continuation of a macro instruction containing mixed mode operands:

```
NAME        OPERATION        OPERAND          COLUMN 72
symbol        MOVE           A,                  X
                             B,LEN=80
```

The guidelines for continuation lines, omitted positional and keyword operands, and operand sublists apply to mixed mode operations as they apply to positional and keyword operands, respectively.

3.3.2 Omitted Operands

These guidelines apply to omitted positional operands:

- If an operand is omitted from a positional macro instruction, the comma must be present. This comma represents the comma that would have separated it from the next value.
- If the symbolic parameter corresponding to an omitted positional operand is referenced in an expanded model statement, a null character value replaces the symbolic parameter in the expanded statement.
- If the last operand is omitted, the comma separating the last operand from the previcus operand can be omitted.

These guidelines apply to omitted keyword operands:

- If the prototype statement assigned a standard value to a symbolic parameter and the macro instruction does not contain the corresponding keyword, the standard value replaces the symbolic parameter.
- If the prototype statement did not assign a standard value to a symbolic parameter and the macro instruction does not contain the corresponding keyword, a null character value replaces the symbolic parameter.
- If a symbolic parameter appears in the operand field of the prototype statement and the macro instruction contains the corresponding keyword, the value assigned to the keyword replaces the symbolic parameter.

3.3.3 Sublists

A sublist consists of one or more operands separated by commas and enclosed in paired parentheses. The entire sublist, including the parentheses, is one macro instruction operand.

- Positional Macro Operand Sublists:

  - If %SP is a symbolic parameter in a prototype statement and the corresponding operand in a macro instruction is a sublist, then:

    %SP(n)

    is used to reference the nth operand of the sublist; where n, which can be a decimal integer or any arithmetic expression that resolves a decimal integer, is greater than or equal to 1.

  - If the nth operand of the sublist is omitted, then %SP(n) refers to a null character value.

  - If the sublist notation is used, but the operand in the macro instruction is not a sublist, then %SP(1) refers to the operand and any other sublist notation references a null character.

  - When using sublist notation, the left parenthesis must immediately follow the last character of the symbolic parameter. The following example illustrates the use of operand sublists:

Example:

```
 NAME       OPERATION      OPERAND                    TYPE

            MACRO                                      1
%SYMBOL     CLEAR          %R1,%FIELD,%X2              2
%SYMBOL     LIS            %R1,0                       3
            ST             %R1,%FIELD(1):(%X2)         3
            ST             %R1,%FIELD(2):(%X2)         3
            ST             %R1,%FIELD(3):(%X2)         3
            MEND                                       4

CLRAREA1    CLEAR          6,(FLD,FLDA,FLDB),13        5

CLRAREA1    LIS            6,0                         6
            ST             6,FLD(13)                   6
            ST             6,FLDA(13)                  6
            ST             6,FLDB(13)                  6
```

LEGEND FOR TYPE COLUMN

1 = macro header
2 = prototype statement
3 = model statement
4 = macro trailer
5 = macro instruction
6 = generated statement

3.4 INNER/OUTER MACRO INSTRUCTIONS

When mnemonic operation code for a given macro definition appears as the operation field of a model statement in another macro definition, the model statement (see Section 2.6) is an inner macro instruction and the macro instruction referring to the containing definition is an outer macro definition. The macro definition that corresponds to an inner macro instruction generates the statements that replace the inner macro instructions. Recursion is permitted, and recursive macros expand properly.

The corresponding values of the outer macro instructions replace the symbolic parameters used in an inner macro instruction.

An inner macro instruction cannot reference a single member of an outer macro instruction sublist unless the inner macro instruction references the operand containing the entire sublist and the macro definition corresponding to the inner macro instruction contains a reference to the sublist member.

Keyword, positional, or mixed mode instructions can be used as model statements in keyword, positional, or mixed mode macro definitions. The following illustrates the use of inner macro instructions:

STORE is an outer macro.

ADD is an inner macro.

```
    NAME      OPERATION      OPERAND                    TYPE

              MACRO                                      1
    $SYM      ADD        %REG,%FLD                       2
              L          %REG,%FLD(1):(13)               3
              A          %REG,%FLD(2):(13)               3
              A          %REG,%FLD(3):(13)               3
              MEND                                       4

              MACRO                                      1
              STORE      %R1,%FIELD,%R1A,%FIELDA         2
              ADD        %R1A,%FIELDA                    3
              LR         %R1,%R1A                        3
              ST         %R1,%FIELD(1)                   3
              ST         %R1,%FIELD(2)                   3
              MEND

              STORE      6,(F1,F2),7,(FA1,FA2,FA3)       5

              L          7,FA1(13)                       6
              A          7,FA2(13)                       6
              A          7,FA3(13)                       6
              LR         6,7                             6
              ST         6,F1                            6
              ST         6,F2                            6

    LEGEND FOR TYPE COLUMN

    1 = macro header
    2 = prototype statement
    3 = model statement
    4 = macro trailer
    5 = macro instruction
    6 = generated statement
```

3.4.1 Levels of Macro Instructions

A macro definition corresponding to an outer macro instruction can contain any number of inner macro instructions. The outer macro instruction is a first level macro instruction. Each of the inner macro instructions is a second level macro instruction. A macro contained within the macro definition corresponding to a second level macro instruction is a third level macro instruction, etc.

The number of macro instruction levels that can be used depends on the definition's complexity and the amount of available memory.

3.4.2 Macro Instructions in Conditional Assembly

CAL conditional assembly, such as IFZ, IFNZ, cannot be evaluated at macro processing time since the values of EQUs are not known to the macro processor. Hence, any macros within conditional code will always be expanded, regardless of whether CAL will actually generate the expanded code. Normally, this would be no problem since CAL would not assemble the expanded code if the conditional failed. However, certain macros; e.g., PURE, IMPUR, also set CAL macro global flags that are used by other macros. These flags are set regardless of whether CAL assembles the statements in the assembly.

It is advisable not to use IFZ or IFNZ to generate such macros as PURE or IMPUR.

Example:

```
FLAG    EQU     0
        IFNZ    FLAG
        PURE
        ENDC
```

In the previous example, CAL will not generate the PURE statement; however CAL macro will set a global flag within the PURE macro, thus affecting other macros. An example of an alternate approach to code such a macro is:

Example:

```
        MACRO
        SETFLAG
        GBLB        %FLAG
%FLAG   SETB        0
        MEND
        MACRO
        ISPURE
        GBLB        %FLAG
        SETFLAG
        AIF         (%FLAG) &PURE
        MEXIT
&PURE   PURE
        MEND
```

In the previous example, a call to ISPURE with %FLAG set to 0 will not generate the PURE statement. A call to ISPURE with %FLAG set to 1 will generate the PURE statement.

CHAPTER 4
CONDITIONAL EXPANSION OF MACRO DEFINITIONS

4.1 INTRODUCTION

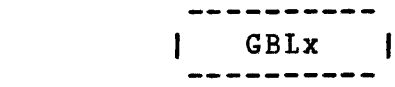
The following sections detail the conditional expansion of a macro definition. Global, batch global, and local SET variable symbol declaration statements, the SET variable symbol statements, attributes, sequences symbols, and system variable symbols are discussed.

4.2 GLOBAL, BATCH GLOBAL, AND LOCAL SET VARIABLE SYMBOL DECLARATION STATEMENTS

Variable symbols, known as SET variable symbols, must be declared before they can be used. Variable symbols can be declared as:

- local to a given macro definition,
- global to all macro definitions in a program, or
- (batch) global to all macro definitions in all programs in the batch

as they are invoked in a given macro processor pass. These symbols are declared by way of the global SET variable symbol (GBLx), batch global SET variable symbol (BGBLx), or local SET variable symbol (LCLx) declaration statements. Only a SETx instruction can change the SET variable symbol value. Section 2.4.1.1 gives additional information on local, global, and batch global variable symbols.



4.2.1 Global SET Variable Symbol Declaration (GBLx) Statement

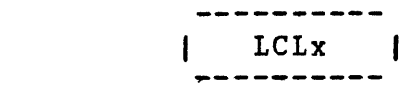
The GBLx symbols communicate values between macro definitions or different uses of the same macro definition in a program. A GBLx symbol must be declared as global each time it is used in a macro definition. GBLA, GBLB, and GBLC statements reference arithmetic, binary, and character variables, respectively, and assign values to them. The initial values of the GBLA, GBLB, and GBLC variable symbols are 0, 0, and null character values, respectively. The initial value is only assigned when a macro definition, which contains a particular global SET variable symbol, is first invoked. Subsequent GBLx instructions have no effect on the value assigned to a GBLx symbol. The format of the GBLx statement is:

NAME	OPERATION	OPERAND
blank	GBLA, GBLB, or GBLC	one or more variable symbols used as global SET variable symbols, separated by commas

4.2.2 Batch Global SET Variable Symbol Declaration (BGBLx) Statement

The BGBLx symbols communicate values between programs in the batch. Within a program, BGBLx symbols perform the same function as GBLx symbols. A BGBIx symbol is initialized at the first declaration of that symbol in the batch. The initial values of the BGBLA, BGBLB, and BGBIC variable symbols are 0, 0, and null character values, respectively. The value assigned to the BGBLx symbol is available to all successive programs in which it is declared as a BGBLx symbol. The format of the BGBLx statement is:

NAME	OPERATION	OPERAND
blank	BGBLA, BGBIB, or BGBLC	one or more variable symbols used as batch global SET variable symbols, separated by commas.



4.2.3 Local SET Variable Symbol Declaration (LCLx) Statement

The LCLx symbols communicate values within the same usage of a particular macro definition. A local SET variable symbol is only declared in the macro definition that it is used in. It is reset to its initial value each time that macro definition is invoked. The initial values of LCLx symbols are the same as those for global SET variable symbols (0, 0, and null character values). The format of the LCLx instruction is:

NAME	OPERATION	OPERAND
blank	LCLA, LCLB, or LCLC	one or more variable symbols used as local SET variable symbols, separated by commas

4.2.4 Declaring SET Variable Symbols

The following rules apply to declaring SET variable symbols:

- If the same SET variable symbol is declared local in more than one macro definition, it becomes a different symbol for each definition in which it is used.
- If the same SET variable symbol is declared global or batch global in one or more macro definitions and local in others, it is one symbol wherever it is declared global or batch global and a different symbol wherever it is declared local.
- If the same SET variable symbol is declared batch global in one or more macro definitions and global in others, it is one symbol wherever it is declared batch global and a different symbol wherever it is declared global.

All batch global, global, or local declarations must immediately follow the macro prototype statement or other BGBLx, GBLx, or LCLx statements.

4.2.5 Declaring SET Variables as Arrays

A batch global, global, or local SET variable can be declared as an array. The format is:

NAME	OPERATION	OPERAND
blank	BGBLA, BGBLB, BGBLC, GBLA, GBLB, GBLC, LCLA, LCLB, or LCLC	variable symbol followed by an integer enclosed in parentheses

The integer is the highest subscript; the lowest subscript is 0. The number of elements is one greater than the integer dimension.

Example:

NAME	CPERATICN	OPERAND
blank	GBLA	%AR1(9)

This statement declares a global arithmetic set variable %AR1 as an array of 10(=9+1) elements.

Ten elements of %AR1 are %AR1(0),%AR1(1),...,%AR1(9).

4.3 SET VARIABLE SYMBOL (SETx) STATEMENTS

The SETx statements alter the values of the variable symbols that the BGBLx, GBLx, or LCLx declaration statements declared as SET variable symbols. These SET statements assign arithmetic, character, and binary or Boclean values to SETA, SETC, and SETB variables, respectively.

Arithmetic expressions can be a single term or an arithmetic combination of terms. The arithmetic operators used in combining terms are:

- addition (+),
- subtraction (-),
- multiplication (*), and
- division (/).

An arithmetic expression cannot contain two operators or two terms in succession, nor can it begin with the multiplication or division operators. This procedure evaluates arithmetic expressions:

- Each term is given its numerical value.
- The arithmetic operations are performed from left to right. Unary plus or minus is evaluated first.
- Multiplication and division are performed before addition and subtraction.
- Parentheses can be used to redefine the order of evaluation. Parenthesized sequences can be nested to a limit of 15 levels of parentheses. Parentheses required for sublist and substring notation count toward this limit of 15. The parenthesized portions or an arithmetic expression are evaluated first. If there is more than one level of parentheses, the inner-most level is evaluated first.
- In division only, the integer portion of the quotient is retained; for example: 91/25 yields 3. If an expression exceeds the maximum range of values, it is flagged as an expansion error.

Section 4.3.1 details the relationship of arithmetic expressions to SETA statements.

These guidelines apply to character expressions:

- The maximum number of characters that can be assigned to a SETC symbol is eight.
- A character expression consists of a type attribute or any combination of up to 255 characters, enclosed in apostrophes.
- More than one character expression can be concatenated into a single character expression by placing a colon between the terminating apostrophe of the first expression and the leading apostrophe of the next one. Use two apostrophes to represent an apostrophe that is a part of a character string.
- By specifying a specific character string, smaller substrings can be extracted from larger strings.

Section 4.3.2 details the relationship of character expressions to a SETC statement.

Binary or Boolean variables have the value of 0 (false) or 1 (true). A binary variable can be assigned a value by evaluating a relational expression enclosed in parentheses. The value of the relational expression is either true or false.

A binary variable can also be assigned a value as a result of logical comparisons. Both operands must be the same type; for example, both must be character expressions. Logical expressions can be formed by using the operations AND, OR, NOT. The expression is evaluated from left to right; AND is evaluated before OR.

Section 4.3.3 details binary or Boolean values assigned to a SETB statement.

```
----------
|  SETA  |
----------
```

4.3.1 SET Arithmetic Variable (SETA) Statement

The SETA instruction assigns an arithmetic value to a SETA symbol or array element. Its format is:

NAME	OPERATION	OPERAND
a SETA symbol or array element	SETA	an arithmetic expression

The arithmetic expression is evaluated as a signed 32-bit arithmetic value in the range of -2,147,483,648 to +2,147,483,647. The value is assigned to the SETA variable symbol in the name field. The expression can consist of one term or an arithmetic combination of terms. The terms that can be used are:

- signed integer constants,
- variable symbols,
- count attributes, and
- number attributes.

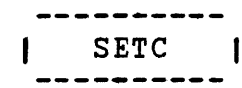
4.3.1.1 Using SETA Variable Symbol

When using a SETA variable symbol, its assigned arithmetic value is substituted for the SETA symbol when the symbol is used in an arithmetic expression. If the SETA symbol is not used in an arithmetic expression, the arithmetic value is converted to a signed integer with leading zeros removed. If the value is zero, it is converted to a single zero. A SETA variable symbol (with a positive value) can be used with a symbolic parameter to refer to an operand in a sublist or array vehicle. This example illustrates the use of SETA variable symbols:

```
 NAME      OPERATION          OPERAND                    TYPE

           MACRO                                         1
           MVBYT         %TO=,%FROM=,%REG=13             2
           LCLA          %ALOC(2)                        3
%ALOC(1)   SETA          1                               4
%ALOC(2)   SETA          %ALCC(1)+%ALOC(1)               4
           LB            %REG,%FROM%ALOC(1)              5
           STB           %REG,%TO%ALOC(2)                5
           MEND                                          6

           MVBYT         FROM=HERE,TO=THERE              7

           LB            13,HERE1                        8
           STB           13,THERE2                       8
```

LEGEND FOR TYPE COLUMN

1 = macro header
2 = macro prototype
3 = local declaration
4 = SETx instruction
5 = model statement
6 = macro trailer
7 = macro instruction
8 = generated statement



4.3.2 SET Character Variable (SETC) Statement

The SETC statement assigns a character value to a SETC variable symbol. The format of the SETC statement is:

NAME	OPERATION	OPERAND
a SETC symbol or array element	SETC	one operand consisting of a type attribute, character expression, substring notation, or a concatenation of substring notations and character expressions

If a SETA symbol appears in the operand of a SETC statement, the resulting unsigned character represents the decimal value, with leading zeros removed. If a SETB symbol appears, the result is the character 0 or 1.

Variable symbols can be concatenated with other characters in a SETC operand. Use two percent signs to represent a single percent sign that is not part of a variable symbol.

The maximum number of characters that can be assigned to a SETC symbol is 8.

4.3.2.1 Substring Notation

Substring notation allows assigning a part of a character value to a SETC variable symbol indicating in the operand field of a SETx statement:

- the character value or an expression representing the character value, and
- the part of the character value to be assigned to the SETC variable symbol.

The combination of the previous items, referred to as substring notation, consists of a character expression, immediately followed by two arithmetic expressions separated by a comma and enclosed in parentheses. The two arithmetic expressions indicate the beginning and ending characters in the substring. Positive

values refer to the beginning of the string; negative values refer to the end of the string. This example illustrates the use of substring notation:

```
NAME        OPERATION        OPERAND

%CLOC1        SETC          'BASEADDR'
%CLOC2        SETC          '%CLOC1'(1,4)
%CLOC3        SETC          '%CLOC1'(5,-1)
```

In the previous example, the character value, 'BASEADDR' is assigned to the SETC variable, %CLOC1. The character values 'BASE' and 'ADDR' are assigned to the SETC variables %CLOC2 and %CLOC3 respectively.

If a substring requests more characters than are contained in the character string, only the characters in the string are assigned. The maximum size of a substring is 255 characters. The maximum size of the expression the character value is developed from is 255 characters.

Character variables and substring notation are valuable in scanning arguments for occurrences of quotes, parentheses, or special characters.

4.3.2.2 Using SETC Variable Symbols

The character value assigned to a SETC symbol is substituted for the SETC symbol when it is used in the name, operation, or operand field of a statement.

Character expressions can be concatenated with substring notations in the operand field of a SETC instruction. If a substring notation follows a character expression, the two can be concatenated by placing a colon between the terminating apostrophe of the character expression and the opening apostrophe of the substring notation. If a substring notation precedes a character expression or another substring notation, the colon is not necessary for concatenation.

If a SETC variable symbol is used in the operand field of a SETA instruction, the character value of the SETC symbol must be one or more decimal digits; or, it is flagged as an expansion error. This example illustrates the use of the SETC variable statements:

```
NAME     OPERATION          OPERAND                          TYPE

         MACRO              SAVE REGISTERS                    1
         SAVE               %REG                              2
         LCLA               %ALOC                             3
         LCLC               %REGS,%REGS1,%REGS2               3
%ALOC    SETA               %REG                              4
%REGS    SETC               'R':'%ALOC'                       5
%ALOC    SETA               %ALOC+1                           4
%REGS    SETC               '%REGS':'R':'%ALOC'               5
%ALOC    SETA               %ALOC+1                           4
%REGS    SETC               '%REGS':'R':'%ALOC'               5
%ALOC    SETA               %ALOC+1                           4
%REGS    SETC               '%REGS':'R':'%ALOC'               5
%REGS1   SETC               '%REGS'(1,2)                      4
%REGS2   SETC               '%REGS'(2,2)                      5
         ST                 %REGS1,HOLD%REGS2                 6
%REGS1   SETC               '%REGS'(3,4)                      5
%REGS2   SETC               '%REGS1'(2,2)                     5
         ST                 %REGS1,HOLD%REGS2                 6
%REGS1   SETC               '%REGS'(5,6)                      5
%REGS2   SETC               '%REGS1'(2,2)                     5
         ST                 %REGS1,HOLD%REGS2                 6
%REGS1   SETC               '%REGS'(7,8)                      5
%REGS2   SETC               '%REGS1'(2,2)                     5
         ST                 %REGS1,HOLD%REGS2                 6
         MEND                                                 7

         SAVE               2                                 8

         ST                 R2,HOLD2                          9
         ST                 R3,HOLD3                          9
         ST                 R4,HOLD4                          9
         ST                 R5,HOLD5                          9
```

LEGEND FOR TYPE COLUMN

1 = macro header
2 = macro prototype
3 = local declaration
4 = SETA statement
5 = SETC statement
6 = model statement
7 = macro trailer
8 = macro instruction
9 = generated statement

The previous example defines the macro 'SAVE'. When the macro SAVE is invoked, it generates assembly code to save four registers beginning with the register designated in the operand field of the macro instruction.

The macro instruction 'SAVE2' generates assembly code to save registers 2 through 5 in the memory locations designated HOLD2 through HOLD5. This operation is accomplished by first building a character local variable containing 'R2R3R4R5' and using concatenation and substring notation to expand the proper values to the generated assembly code.

To build the assembly instruction, 'ST R2,HOLD2', this procedure is used:

1. The character string 'R2R3R4R5' is built in %REGS.

2. The statement "%REGS1 SETC '%REGS'(1,2)" extracts the first two characters (or 'R2') from %REGS and places them in %REGS1.

3. The statement "%REGS2 SETC '%REGS1'(2,2)" extracts the second character (or '2') from %REGS1 and places it in %REGS2.

4. In the model statement "ST %REGS1,HOLD%REGS2", R2 is substituted for %REG1 and 2 is substituted for %REGS2 and concatenated with HOLD to form HOLD2. The resulting statement is "ST R2,HCLD2".

| SETB |

4.3.3 SET Binary Variable (SETB) Statement

The SETB statement assigns the value true (binary 1) or false (binary 0) to a SETB variable symbol. The format of the SETB instruction is:

NAME	OPERATION	OPERAND
a SETB symbol or array element	SETB	a 0 or 1 or logical expression enclosed in parentheses

A logical expression consists of one term or a logical combination of terms. The terms that can be used are:

- arithmetic relations,
- character relations, and
- SETB variable symbols.

The logical operators used in combining the terms are OR, AND, and NOT.

A logical operator must always separate a logical expression that contains two consecutive terms. The expression can contain two successive operators only if the first operator is OR or AND, and the second operator is NOT. A logical expression can begin with the operator NOT; but, it cannot begin with OR or AND. A logical expression is evaluated to determine if it is true or false; the SETB variable symbol is then assigned the value of 1 or 0.

An arithmetic relation consists of two arithmetic expressions enclosed in parentheses, connected by a relational operator. A character relation consists of two character values enclosed in apostrophes connected by a relational operator. The six relational operators are:

- EQ is equal.
- GE is greater than or equal.
- GT is greater than.
- LE is less than or equal.
- LT is less than.
- NE is not equal.

At least one blank must precede and succeed the relational and logical operators.

Any arithmetic expression permitted in the operand field of a SETA instruction can be used in the operand field of a SETB instruction as a part of an arithmetic relation.

Any expression permitted in the operand field of a SETC instruction can be used as a character value in the operand field of a SETB instruction. In resolving a SETB expression using character relations, two character values are considered equal only when they are of equal length and contain the same characters; for example: '20' is less than '020'. When two character values are of unequal length, the shorter value is always less then the longer one; that is: 'Z' is less than 'AA'.

Logical expressions are evaluated as follows:

- Each term is evaluated and given its logical value.
- Each parenthesized expression is evaluated and given its logical value.
- The computed result of the entire operand (1 for true, 0 for false) is the value assigned to the SETB variable symbol.

The logical expression in the operand field of a SETB instruction can be parenthesized. Parenthesized sequences of terms can be nested to a limit of 15 levels of parentheses. The parenthesized portions of a logical expression are evaluated first. If there is more than one level of parentheses, the innermost level is evaluated first.

4.3.3.1 Using SETB Variable Symbols

The logical value assigned to a SETB variable symbol replaces the SETB symbol in the operand field of a conditional instruction or another SETB instruction. If a SETB symbol appears in the operand field of a SETA instruction or in arithmetic relations in the operand fields of conditional or SETB instructions, the logical values 0 and 1 are converted to the arithmetic values +0 and +1. If a SETB symbol appears in the operand field of a SETC instruction or in a character relation in the operand field of a conditional or SETB instruction, the logical values 0 and 1 are converted to the character values 0 and 1. This example illustrates the use of the SETB instructions:

```
NAME          OPERATICN          OPERAND                    TYPE

              MACRO                                         1
              MOVE               %FROM,%TO                  2
              LCLA               %ALO1                      3
              LCLB               %BLO1                      3
              LCLC               %CLO1                      3
%BLO1         SETB               ('%FROM' NE '%TO')         4
%ALO1         SETA               %BLO1+1                    5
%CLO1         SETC               'A '(%ALO1,%ALO1)          6
              L                  13,%FROM%CLO1              7
              ST                 13,%TO                     7
              MEND                                          8

              MOVE               HERE,HERE                  9

              L                  13,HERE                   10
              ST                 13,THERE                  10

              MOVE               HERE,THERE                 9

              L                  13,HEREA                  10
              ST                 13,HERE                   10
```

LEGEND FOR TYPE COLUMN

```
 1 = macro header
 2 = macro prototype
 3 = local declaration
 4 = SETB statement
 5 = SETA statement
 6 = SETC statement
 7 = model statement
 8 = macro trailer
 9 = macro instruction
10 = generated statement
```

4.4 ATTRIBUTES

Attributes can be assigned to macro instruction operands. These attributes can be referenced only in conditional instructions or expressions. Each of the three attributes has an associated notation:

ATTRIBUTE	NCTATION
type	T'
count	K'
number	N'

If an inner macro instruction operand is a symbolic parameter, the operand attributes are the same as those of the corresponding outer macro instruction operand. A symbol appearing as an inner macro instruction operand is not assigned the same attributes as the same symbol appearing as an outer macro instruction operand. Section 3.4 details inner and outer macro instructions.

If a macro instruction operand is a sublist, the attributes of the sublist or each element in the sublist can be referenced. Section 3.3.3 details sublists.

4.4.1 Type Attribute (T')

The type attributes of the macro instruction operand are:

- The letter A represents an alphanumeric operand.
- The letter N represents a numeric operand. (A signed integer is recognized as numeric.)
- The letter U represents a null operand.

The type attribute can be used wherever a character expression could be used; but, the type attribute must occur alone (that is, not concatenated with anything), and it must not be enclosed in quotes.

4.4.2 Count Attribute (K')

The count attribute value is the number of characters in a macro instruction field. It includes all characters in the operand except the delimiting commas. The count attribute of an omitted operand is zero. The count attribute can be referred to:

- in the operand field of a SETA instruction, or
- in arithmetic relations in the operand fields of SETB, or conditional instructions in a macro definition.

4.4.3 Number Attribute (N')

The number attribute is a value equal to the number of operands in an operand sublist. The number is equal to one plus the number of delimiting commas within the sublist. If the macro instruction operand is not written in sublist notation, the

number attribute is one. If the operand is omitted, the number attribute is zero. The number attribute can be referred to:

- in the operand field of a SETA instruction, or
- in arithmetic relations in the operand fields of SETB or conditional instructions in a macro definition.

This example illustrates attributes:

Given this macro prototype and instruction:

```
MOVE   %REG,%FROM,%TO,%HCLD

MOVE   13,HERE,(THERE1,THERE2)

T'%REG   = N     K'%REG   = 2     N'%REG   = 1
T'%FROM  = A     K'%FROM  = 4     N'%FROM  = 1
T'%TO    = A     K'%TO    = 15    N'%TO    = 2
T'%HOLD  = U     K'%HOLD  = 0     N'%HOLD  = 0
```

4.5 CONDITIONAL AND UNCONDITIONAL BRANCH INSTRUCTIONS

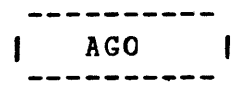
The conditional and unconditional branch instructions conditionally and unconditionally alter the macro definition processing sequence.

4.5.1 Conditional Branch (AIF) Instruction

The AIF instruction conditicnally alters the macro definition statement processing sequence. Its format is:

NAME	OPERATION	OPERAND
a sequence symbol or blank	AIF	logical expression enclosed in parentheses, immediately followed by a sequence symbol

Any expression used in a SETB instruction operand field can be used in an AIF instruction operand field. The expression is evaluated to determine if it is true or false. If the expression is true, the statement that the sequence symbol in the operand field named is the next statement processed. If the expression is false, the next sequential statement is processed. Section 4.7 gives examples of the AIF instruction. Section 2.4.3 details sequence symbols.



4.5.2 Unconditional Branch (AGO) Instruction

The AGO instruction unconditionally alters the macro definition statement processing sequence. Its format is:

NAME	OPERATION	OPERAND
a sequence symbol or blank	AGO	a sequence symbol

The statement the sequence symbol names in the operand field is the next statement processed.

The sequence symbol in an AGO or AIF instruction operand field must appear in a statement name field in the same macro definition as the AGO or AIF instruction; or, it causes an expansion error. Section 4.7 gives examples of using the AGO instruction.

4.5.3 Computed AGO and AIF Statements

An AGO or AIF statement operand is extended to include a:

- character variable,
- array element, or
- parameter.

The value of the character variable, array element, or parameter must be a valid sequence number.

Example:

```
            LCLC    %C
%C          SET     '&SEQSYM'
&AGAIN      ANOP
            AGO     %C
             .
             .
             .
&SEQSYM     ANOP
%C          SETC    '&SEQ2'
            AGO     &AGAIN
&SEQ2       ANOP
```

The character value %C is redefined to &SEQ2. The second time the AGO is executed, a branch to &SEQ2 is executed:

```
             MACRO
             EX2     %P
             AGO     %P
              .
              .
              .
&ABC         ANOP
              .
              .
              .
&DEF         ANOP
             MEND
              .
              .
              .
             EX2      &ABC
```

takes a branch to &ABC, while:

```
             EX2      &DEF
```

takes a branch to &DEF.

The ampersand must be included in the macro call parameter. To avoid the ampersand in the macro call, use a string variable and concatenate the ampersand in the macro definition. Many of the system macros use this technique for codes:

```
            MACRO
            EX3     %AP=
            LCLC    %C
%C          SETC    '&':'%AP'
            AGO     %C
             .
             .
             .
&SRO        ANOP
             .
             .
             .
&SRW        ANOP
             .
             .
             .
            EX3     AP=SRO
            EX3     AP=SRW
```

4.6 CONDITIONAL INSTRUCTION LOOP COUNTER (ACTR) INSTRUCTION

The maximum count of AIF and AGO branches that can be executed in a macro definition is 32767. The ACTR instruction assigns a count other than 32767 as the maximum number of AIF and AGO branches executed within a macro definition. The format of the ACTR instruction is:

NAME	OPERATION	OPERAND
blank	ACTR	any expression that can appear in the operand field of a SETA instruction

The ACTR instruction can only appear immediately after global and local declaration statements. This instruction causes a counter to be set to the value in the operand field. The counter is checked for zero or negative value; if the counter is not zero or negative, it is decremented by one each time an AIF or AGO branch is executed. If the count is zero before decrementing, the entire nest of macro definitions is terminated and the next source statement is processed. An ACTR statement in a macro definition affects only the definition in which it appears.

When a macro definition calls an inner macro definition, the current value of the branch count is saved and a new count of 32767 is set up for the inner macro definition (unless the inner macro contains an ACTR instruction). When processing in the inner definition is completed and a return is made to the higher definition, the saved count is restored.

4.7 NO OPERATION (ANOP) INSTRUCTION

When the sequence symbol in an AIF or AGO instruction must reference a statement already containing a symbol (other than a sequence symbol) in the name field, the ANOP instruction is used. The format of the ANOP instruction is:

NAME	OPERATION	OPERAND
a sequence symbol	ANOP	blank

The ANOP instruction is placed before the statement that the branch is to be made to and the sequence symbol is placed in the ANOP instruction name field. This placement has the same effect as branching to the statement immediately following the ANOP instruction. The following example shows the use of conditional instructions:

```
NAME      OPERATION       OPERAND                                  TYPE

          MACRO                                                     1
          SAVE            %REG,%HOWMANY                             2
          LCLA            %ALOC,%ALCC1                              3
          LCLC            %REGS                                     3
          AIF             ((T'%REG NE 'N') AND                      4
                          (T'%REG NE 'U'))&END
          AIF             ((T'%HOWMANY NE 'N') AND
                          (T'%HOWMANY NE'U'))&END                   4
          AIF             ( NOT ((%REG GE 0) AND
                          (%REG LE 15)))&END                        4
          AIF             (T'%HOWMANY EQ 'U')&NULL                  4
          AIF             ( NOT ((%HOWMANY GE 1) AND
                          (%HOWMANY LE 16)))&END                    4
&NULL     ANOP                                                      5
          AIF             (T'%REG EQ 'U')&ZERO                      4
%ALOC     SETA            %REG                                      6
&ZERO     ANOP                                                      5
          AIF             (T'%HOWMANY EQ 'N')&NOTALL                7
%ALOC1    SETA            16-%ALOC                                  6
          AGO             &CKSIZE                                   8
&NOTALL   ANOP                                                      5
%ALOC1    SETA            %HOWMANY                                  6
&CKSIZE   AIF             (%ALOC+%ALOC1 GT 16)&END                  4
&SETNAME  ANOP                                                      5
%REGS     SETC            'R':'%ALOC'                               6
          ST              %REGS,HCLD%ALOC                           9
%ALOC1    SETA            %ALOC1-1                                  6
          AIF             (%ALOC1 EQ 0) &END                        4
%ALOC     SETA            %ALOC+1                                   6
          AGO             &SETNAME                                  8
&END      ANOP                                                      5
          MEND                                                     10
```

LEGEND FOR TYPE COLUMN

```
 1 = macro header
 2 = macro prototype
 3 = local declaration
 4 = conditional branch
 5 = no operation
 6 = SETA instruction
 7 = conditional
 8 = unconditional branch
 9 = model statement
10 = macro trailer
```

The previous example defines the macro SAVE. This macro generates assembler code to save a number of registers (designated by %HOWMANY in the prototype statement) beginning with the register designated by %REG. If the argument used in place of %REG is omitted in the macro instruction, R0 is assumed to be the beginning register. If the argument used in place of %HOWMANY is omitted in the macro instruction, it is assumed that all registers beginning with %REG are to be saved. These examples show some possible expansions of the SAVE macro:

```
NAME     OPERATION      OPERAND          TYPE

         SAVE           10,4             1

         ST             R10,HOLD10       2
         ST             R11,HOLD11       2
         ST             R12,HOLD12       2
         ST             R13,HOLD13       2

         SAVE           13               1

         ST             R13,HOLD13       2
         ST             R14,HOLD14       2
         ST             R15,HOLD15       2

         SAVE           ,2               1
         ST             R0,HOLD0         2
         ST             R1,HCLD1         2
```

LEGEND FOR TYPE COLUMN

1 = macro instruction
2 = generated statement

4.8 MACRO DEFINITION EXIT (MEXIT) INSTRUCTION

The MEXIT instruction terminates the current macro definition expansion. Its format is:

NAME	OPERATION	OPERAND
a sequence symbol or blank	MEXIT	not used

If a MEXIT instruction is processed in a macro definition for an outer macro instruction, the next statement in the source is processed next. If a MEXIT instruction is processed in a macro definition for an inner macro instruction, the next statement after the inner macro instruction in the macro definition is processed next. Section 3.4 details inner and outer macro instructions. This example illustrates the use of the MEXIT instruction:

```
NAME      OPERATION      OPERAND                    TYPE

          MACRO                                     1
%SYM      MOVE           %FROM,%TO,%REG,%HOLD       2
          AIF            (T'%REG EQ 'N')&OK         3
          MEXIT                                     4
&OK       ANOP                                      5
%SYM      ST             %REG,%HOLD                 6
          L              %REG,%FROM                 6
          ST             %REG,%TO                   6
          L              %REG,%HOLD                 6
          MEND                                      7
```

LEGEND FOR TYPE COLUMN

1 = macro header
2 = macro prototype
3 = conditional branch
4 = macro exit
5 = no operation
6 = model statement
7 = macro trailer

In the previous example, a numeric argument must represent the symbolic parameter %REG when the macro instruction is written. To assure this representation, the %REG type attribute is compared to 'N'. If it is not 'N', the MEXIT instruction is executed, resulting in the termination of any further macro expansion.

4.9 REQUEST FOR MESSAGE (MNOTE) INSTRUCTION

The MNOTE instruction generates a macro message. Its format is:

NAME	OPERATION	OPERAND
a sequence symbol or blank	MNOTE	an optional integer expression followed by a comma, followed by any combination of characters enclosed in apostrophes

The characters between the apostrophes are printed on the source listing when the MNOTE instruction is processed. If a symbolic parameter appears between the apostrophes, its value replaces it when the message is printed.

Use two apostrophes to represent a single apostrophe to be printed as part of the message in the source listing. Use two percent signs to represent a single percent sign to be printed as part of the message in the source listing. Use two colon signs (::) to represent a single colon sign to be printed as part of the message in the source listing. MNOTE statements can have a maximum of two continuation statements.

The optional integer expression is the end of task code returned when the macro processor terminates. The default value for this integer expression is 0. The highest value that any executed MNOTE statement specifies is the value returned. This example illustrates using the MNOTE instruction:

```
NAME     OPERATION          OPERAND                        TYPE

         MACRO                                             1
         MOVE        %FROM,%TO,%REG,%HOLD                  2
         AIF         (T'%REG NE 'N')&E2                    3
         AIF         (T'%HOLD EQ 'U')&E1                   3
         ST          %REG,%HOLD                            4
         L           %REG,%FROM                            4
         ST          %REG,%TO                              4
         L           %REG,%HOLD                            4
         MEXIT                                             5
&E1      MNOTE       8,'SAVE AREA NOT DEFINED'             6
         MEXIT                                             5
&E2      MNOTE       4,'REGISTER NOT NUMERIC'              6
         MEND                                              7
         MOVE        FROM,TO,REG,HOLD                      8
*        MNOTE       'REGISTER NOT NUMERIC'                9
         MOVE        HERE,THERE,4                          8
*        MNOTE       'SAVE AREA NOT DEFINED'               9
```

The end of task code is 8.

LEGEND FOR TYPE COLUMN

1 = macro header
2 = prototype statement
3 = conditional branch
4 = model statement
5 = macro exit
6 = macro note
7 = macro trailer
8 = macro instruction
9 = generated statement

4.10 SYSTEM VARIABLE SYMBOLS

The macro processor automatically assigns values to system variable symbols. These system variable symbols are local variable symbols. The five variable symbols:

- %SYSLIST
- %SYSINDX
- %SYSMAC
- %SYSTIME
- %SYSDATE

can be used in the name, operation, or operand fields of statements in macro definitions. System variable symbols cannot be defined as symbolic parameters or SET symbols. Although these system symbols are the only system symbols defined at this time, do not use any variable symbol that starts with %SYS because of future expansion.

%SYSLIST

4.10.1 %SYSLIST Symbol

%SYSLIST provides an alternative to symbolic parameters for referring to positional macro instruction operands. Both %SYSLIST and symbolic parameters can be used in the same positional macro definition.

%SYSLIST (n) refers to the nth positional macro operand. If the nth operand is a sublist, then %SYSLIST (n,m) refers to the mth operand in the sublist. Any arithmetic expression allowed in a SETA instruction operand field can represent n and m.

If the value of n is zero, then %SYSLIST (n) is assigned the value specified in the macro instruction name field, unless that value is a sequence symbol.

The type and count attributes of %SYSLIST(n) and %SYSLIST(n,m) and the number of attributes of %SYSLIST(n) and %SYSLIST can be used in conditional instructions. N'%SYSLIST refers to the total positional operands in a macro instruction statement. N'%SYSLIST(n) refers to the number of operands in a sublist. If the nth operand is null, n is zero. If the nth operand is not a sublist, N' is one. The following example illustrates the use of %SYSLIST:

```
NAME      OPERATION      OPERAND                          TYPE

          MACRO                                            1
%SYM      MOVE           %FROM,%TO,%REG,%HOLD              2
          LCLA           %AL1                              3
%AL1      SETA           1                                 |
tC1610    AIF            ('%SYSLIST(%AL1)' EQ '')&ERR      4   |
%AL1      SETA           %AL1+1                            5
          AIF            (%AL1 LE 4) &CKNUL                4   |
          AGO            &OK                               6
&ERR      MNOTE          'PARAMETER%AL1 MISSING'           7   |
          MEXIT                                            8
&OK       ANOP                                             9
          STA            %REG,%HOLD                       10
          LDA            %REG,%FROM                       11
          STA            %REG,%TO                         10
          LDA            %REG,%HOLD                       10
          MEND                                            11

MOV1      MOVE           HERE,THERE,13                    12

*         MNOTE          'PARAMETER 4 MISSING'            12


LEGEND FOR TYPE COLUMN

 1 = macro header
 2 = macro prototype
 3 = local declaration
 4 = conditional branch
 5 = SETA
 6 = unconditional branch
 7 = macro note
 8 = macro exit
 9 = no operation
10 = model statement
11 = macro trailer
12 = generated statement
```

The use of %SYSLIST in the previous example avoids having to write a separate conditional branch instruction to check each argument for a null condition.

| %SYSINDX |

4.10.2 %SYSINDX Symbol

The value of the %SYSINDX variable symbol can be concatenated with other characters to create unique names for statements generated from the same model statement. %SYSINDX is assigned the numerical value for the first macro instruction that the macro processor processes. It is incremented by one for each subsequent macro instruction processed, whether inner or outer.

If %SYSINDX is used in a model statement, SETC, or MNOTE instruction or in a character relation in a SETB or AIF instruction, the value substituted for %SYSINDX is the number of the macro instruction being processed. If %SYSINDX appears in an arithmetic expression, the value used for %SYSINDX is an arithmetic value. Throughout one use of a macro definition, the %SYSINDX value is constant, independent of any inner macro instruction in that definition. The following example illustrates the use of %SYSINDX:

```
NAME          OPERATICN    OPERAND                    TYPE

              MACRO                                    1
%SYM          MOVE         %FROM,%TO                   2
%SYM          STA          13,HOLD%SYSINDX             3
              LDA          13,%FROM
              STA          13,%TO                      3
              LDA          13,%HOLD%SYSINDX            3
              B            HOLD%SYSINDX+ADC            3
HOLD%SYSINDX  DAC          0                           3
              MEND                                     4

MOV1          MOVE         ASLOT,BSLOT                 5

MOV1          STA          13,HOLD0001                 6
              LDA          13,ASLOT                    6
              STA          13,BSLCT                    6
              LDA          13,HOLD0001                 6
              B            HOLD0001+ADC
HOLD0001      DAC          0                           6

MOV2          MOVE         XSLOT,YSLOT                 5
MOV2          STA          13,HOLD0002                 6
              LDA          13,XSLOT                    6
              STA          13,YSLCT                    6
              LDA          13,HOLD0002                 6
              B            HOLD002+ADC
HOLD0002      DAC          0                           6


LEGEND FOR TYPE COLUMN

1 = macro header
2 = macro prototype
3 = model statement
4 = macro trailer
5 = macro instruction
6 = generated statement
```

In the previous example, the variable symbol %SYSINDX is concatenated with the letters HOLD to form a unique label each time the macro is invoked.

| %SYSMAC |

4.10.3 %SYSMAC Symbol

The system variable %SYSMAC differs from the system variable %SYSINDX because the inner macro calls change their value. Upon returning from an inner macro call, %SYSINDX returns to its original value; while, %SYSMAC remains at the incremented value because of the inner macro. An example of this useful feature is: INNER generated a label using %SYSMAC which was referred to in CUTER by using %SYSMAC. This example shows the relationship between %SYSINDX and %SYSMAC:

```
    NAME          OPERATION     OPERAND                 TYPE

                  MACRO                                   1
                  INNER                                   2
                  IMPUR                                   3
HOLD%SYSINDX      DAS           1                         3
                  PURE                                    3
                  MEND                                    4

                  MACRO                                   1
%SYM              MOVE          %FROM,%TO                 2
                  INNER                                   5
%SYM              STA           13,HOLD%SYSMAC            3
                  LDA           13,%FROM                  3
                  STA           13,%TO                    3
                  LDA           13,HOLD%SYSMAC            3
                  MEND                                    4
```

LEGEND FOR TYPE COLUMN

1 = macro header
2 = macro prototype
3 = model statement
4 = macro trailer
5 = inner macro call

```
%SYSINDX   %SYSMAC     NAME      OPERATION     OPERAND

1          1                     MOVE          A,B
2          2                     INNER
2          2                     IMPUR
2          2          HOLD0002   PAS           1
2          2                     PURE
1          2                     STA           13,HOLD0002
1          2                     LDA           13,A
1          2                     STA           13,B
1          2                     LDA           13,HOLD0002
3          3                     MOVE          C,D
4          4                     INNER
4          4                     IMPUR
4          4          HOLD0004   DAS           1
4          4                     PURE
3          4                     STA           13,HOLD0004
3          4                     LDA           13,C
3          4                     STA           13,D
3          4                     LDA           13,HOLD0004
```

NOTE

After the inner call, %SYSMAC retains its value.

4.10.4 %SYSTIME Symbol

%SYSTIME is an eight character string system variable whose value is the time of day that the macro processor was invoked. System generation determines the form.

4.10.5 %SYSDATE Symbol

%SYSDATE is an eight character string system variable whose value represents the date that the macro processor was invoked. System generation determines the form.

4.11 AREAD STATEMENT

The syntax for the AREAD statement is:

```
%SYMBOL AREAD
```

where:

%SYMBOL is a character variable or array variable large enough to contain the character string (a minimum of 80 characters).

When a macro containing an AREAD is called, the next source line is read into the character variable. The string handling capability of the macro processor can be used to process the line. The following macro demonstrates this capability:

```
          MACRO
          REED
          LCLC  %C(9)                        80 characters
  %C      AREAD                              READ NEXT-LINE AFTER CALL
          AIF   ('%C'(1,2) EQ 'CR')&BYTE     check for character
          DB    X'00'                        NOT CR
          MEXIT
  &BYTE   ANOP
          DB X'0D'
          MEND
           :
          REED
  CR
  +       DB X'0D'
          REED
  ABC
  +       DB        X'00'
```

4.12 SUBSTRING NOTATION IN MODEL STATEMENTS

Substring notation is allowed in model statements. The following macro demonstrates this capability:

```
        MACRO
        EXAMPLE         %P
        LCLC            %TYPE
%TYPE   SETC            'TF'
        AIF             ('%P' EQ 'T')&TRUE
        B%TYPE<2,2>C    ALPHA
        MEXIT
&TRUE   B%TYPE<1,1>C    BETA
        MEND
```

where:

```
        EXAMPLE         T
+       BTC             ALPHA
        EXAMPLE         F
+       BFC             BETA
```

NOTE

Because of possible ambiguities in interpretation, substring notation in a model statement always applies only to the immediately preceding symbolic variable or parameter not enclosed in quotes.

CHAPTER 5
ADDITIONAL CAL MACRO/32 FEATURES

5.1 INTRODUCTION

The operation codes of the features discussed in this section are macro processor pseudo operation codes. Because they are macro processor pseudo operation codes, the processor recognizes and acts on them so they are not passed to the assembler source stream.

5.1.1 As Is (ASIS) Instruction

The ASIS instruction tells the macro processor not to treat the next line or lines as macros even though these lines might be macros. This feature is useful when a macro is used to redefine a CAL operation code. The format of the ASIS instruction is:

NAME	OPERATION	OPERAND
sequence symbol or blank	ASIS	blank, decimal integer, BEGIN, or END

Blank indicates the next statement only. Decimal integer means the next n statements. BEGIN means until an ASIS END is encountered. END indicates termination of ASIS BEGIN.

The END statement does not cause termination of a macro expansion if it appears within an ASIS statement.

The following instructions cannot occur in the scope of ASIS:

ACTR	BGBLC	LCLC	MEXIT	NOLIB
AGO	GBLA	MACRO	MLIBS	NTRAC
AIF	GBLB	MCALL	MNOTE	SETA
ANOP	GBLC	MCOPY	MPAUSE	SETB
BGBLA	LCLA	MDEFS	MTRAC	SETC
BGBLB	LCLB	MEND	NDEFS	

5.1.2 Macro Call (MCALL) Instruction

The MCALL instruction permits the cited macros to be called as they appear in the library rather than as they appear in the processor source stream. This method avoids time-consuming rewinds in a magnetic tape search or multiple disc access in a disc search. The format of the MCALL instruction is:

NAME	OPERATION	OPERAND
blank	MCALL	MACRO,MACRO,...,MACRO

A particular order does nct have to be specified in the MCALL instruction; the specified macro definitions are fetched from the library in the order that they are found. If a macro definition cited in the MCALL instruction is not found in any library, the request for that definition is ignored. All libraries named in the MLIBS instruction are searched for the cited macro definitions. The MCALL instruction statement cannot occur within a macro definition.

5.1.3 Macro Copy (MCOPY) Statement

The MCOPY statement enables source text to be copied from a specified logical unit (lu) or file descriptor at any point in a macro definition or program. The copied source text can be any arbitrary text, such as a:

- set of global variables used in several macros,
- complete macro definition or a set of macro definitions,
- part of a macro definition,
- set of equates, or
- subroutine.

Copying starts at a specified point or at the beginning of the file. Copying terminates when:

- a /* in columns 1 and 2 is encountered,
- an END statement is encountered, or
- an end of file (EOF) is reached.

An END statement, if used as a terminator for MCOPY, will also terminate macro expansion.

The syntax of the MCOPY statement is:

```
ignored     MCOPY     [label]     [,lu number]
                                  [,file descriptor]
```

where:

If a label is not specified, the file is not rewound and everything is copied until a /* in columns 1 and 2 or an EOF is reached.

If a label is specified, search the file from its present position until a label cf the following form is found:

**LABEL

If an EOF is encountered, the file is first rewound and searched once more for the label. If found, copying starts from the next statement until the next terminator is encountered.

If the file is omitted, copying is performed from the current file, where the current file is initially lu 7. It can be changed with a START option, or the file or lu from the most recent preceding MCOPY statement at the same level.

MCOPY statements can be nested up to 16 levels, provided sufficient logical units are available. For example: an MCOPY can copy a macro definition containing an MCOPY statement to copy global variables. The first MCOPY statement copies the initial part of the macro definition until the nested MCOPY statement is encountered. Then, the first MCOPY statement resumes. While the inner MCOPY statement is copying, two logical units are open.

Operands of MCOPY passed to the macro processor through the MCOPY start options are passed to each program in the batch.

5.1.4 Macro Definitions (MDEFS) Instruction

The MDEFS instruction controls which macro statements are sent to the CAL file. Its format is:

NAME	OPERATION	OPERANDS
blank	MDEFS	one or more codes separated by commas

The optional letter N preceding the code means to suppress sending the statement; whereas, the code enables sending. The codes and their meanings are:

CODE	MEANING	MINIMUM ABBREVIATION
(N)DEFS	(NO) macro definitions	D ND
(N)INNER	(NO) inner macro calls	I NI
(N)OUTER	(NO) outer macro calls	O NO
ALL	send all of the above	ALL
NONE	send none of the above	NONE

Example:

```
MDEFS NI,ND
```

suppresses macro definitions and inner macro calls to be sent to CAL. MDEFS can be specified as a START option. If specified as a START option, the MDEFS statement has no effect. MDEFS cannot occur in a macro definition. Section 6.2 details START options.

5.1.5 Macro Libraries (MLIBS) Instruction

The MLIBS instruction designates the file descriptor or decimal lu numbers where the macro libraries, necessary for a given macro processor source stream, reside. The format of the MLIBS instruction is:

NAME	OPERATION	OPERAND
blank	MLIBS	one or more file descriptors or lu numbers separated by commas

If any macro libraries (including the system macro library) are used, this statement must appear as a START option or in the macro processor source stream prior to invoking any macros residing on a library. The operand field must contain at least one argument. The libraries are searched in the order that their file descriptor or lu numbers appear in the START option or MLIBS instruction. The absence of a START option or MLIBS instruction in a processor source stream indicates to the processor that all required macro definitions for that source stream exist as part of the source stream. The maximum number of file descriptors or logical units that can be actively designated as libraries at any one time is 15. The MLIBS instruction statement cannot occur within a macro definition. See Section 6.2 for START options.

The files and logical units passed to the macro processor through the MLIBS start options are passed to each program in the batch.

5.1.6 Macro Listing (MLIST) Instruction

The MLIST instruction controls which macro statements are sent to the list device. Its format is:

NAME	OPERATION	OPERAND
blank	MLIST	one or more codes separated by commas

The optional letter N preceding the code means to suppress sending the statements; whereas, the code enables sending the statement. The codes and their meanings are:

CODE	MEANING	MINIMUM ABBREVIATIONS
(N) DEFS	(NO) macro definitions	D ND
(N) INNER	(NO) inner macro calls	I NI
(N) OUTER	(NO) outer macro calls	O NO
(N) CAL	(NO) CAL code	C NC
(N) GENCAL	(NO) generated code	G NG
(N) MNOTE	(NO) MNOTE	M NM
ALL	send all	ALL
NONE	send NONE	NONE

Example:

```
MLIST ND,NI
```

suppresses macro definitions and inner macro calls to be sent to the list device. MIIST can be specified as a START option. If it is specified as a START option, the MLIST statement has no effect. MLIST cannot occur in a macro definition.

5.1.7 Pause (MPAUS) Instruction

The MPAUS instruction permits the user to pause the macro processor. The pause can occur anywhere in the input stream; but, it cannot occur between a macro statement and a following macro prototype. An example of the MPAUS instruction is:

NAME	OPERATION	OPERAND
blank	MPAUS	blank

```
 -------
| MTRAC |
 -------
```

5.1.8 Macro Trace (MTRAC) Instruction

The diagnostic instruction, MTRAC, determines the effective conditional branches and the SET variable symbols values within the macro logic. The format of the MTRAC instruction is:

NAME	OPERATION	OPERAND
blank	MTRAC	blank

The MTRAC instruction can occur anywhere in the source stream. It causes the macro trace feature to be enabled for all subsequent macro instructions processed. Each time a macro is invoked, a map of its expansion, detailing the path of the conditional branches and the values of the SET variable symbols, is written to the designated trace output device (lu 3). If the MTRAC statement occurs within a macro definition, the trace feature is enabled every time the corresponding macro is invoked and remains enabled until a no trace (NTRAC) statement is encountered in a macro definition expansion or in the source stream.

5.1.9 No Libraries (NOLIB) Instruction

The NOLIB instruction suppresses searching all or some macro libraries previously designated by the MLIBS statement. The lu numbers of the libraries that searching is to be suppressed for are contained in the NOLIB statement operand field. If the operand field is blank, all library searching is suppressed. The format of the NOLIB instruction is:

NAME	OPERATION	OPERAND
blank	NOLIB	one or more file descriptors or lu numbers separated by commas

The NOLIB instruction is useful in these situations:

- If memory size is such that all macro definitions invoked in a given processor source stream can be made memory resident at the same time:

  - the MCALL instruction brings all the macro definitions into memory prior to their use; then,

  - the NOLIB instruction is invoked with a blank operand field, causing all library searches to be suppressed.

  If erroneous operation codes are present in source statements, the NOLIB instruction prevents unnecessary library searches.

- If memory size is not sufficient to allow all macro definitions necessary for a given processor pass to be made memory resident:

  - the MCALL instruction brings as many macro definitions as possible into memory; (ideally, these include the most frequently used macro definitions); then,

  - the NOLIB instruction is invoked to suppress searching the libraries from which the memory-resident macro definitions were extracted.

  This technique minimizes the necessary processing time for library searches in a limited memory environment. The NOLIB instruction statement cannot occur within a macro definition.

5.1.10 No Trace (NTRAC) Instruction

The NTRAC instruction causes the macro trace feature to be disabled. If NTRAC occurs in a macro definition, the macro trace feature is disabled each time that macro is invoked.

NAME	OPERATION	OPERAND
blank	NTRAC	blank

CHAPTER 6
OPERATION OF THE CAL MACRO/32 PROCESSOR

6.1 INTRODUCTION

This chapter demonstrates the operation of the CAL Macro/32 Processor (MACRO) and OS/32.

The CAL Macro Processor is available as a 32-bit object program.

6.1.1 Device Assignments

The CAL Macro Processor uses these device assignments:

AS LU 1	Source input
AS LU 2	Expanded source output
AS LU 3	Listing, trace output, and error messages (see Appendix D)

Any other available logical unit (lu) can be used for libraries.

6.1.2 Memory Requirements

The macro processor requires approximately 26kb of memory over and above what the operating system uses and the macro processor itself requires for any table space. The macro processor requires table space for stcring:

- programmer macros,
- library macros defined by an MCALL statement,
- global symbols, and
- any parameters associated with the macro instruction currently being processed.

Thus, the amount of table space required depends on the program being processed.

6.2 OPERATION OF THE MACRO PROCESSOR UNDER OS/32

Prior to using the macro processor, it must be established as a user task with OS/32 Link. Refer to the OS/32 Operator Reference Manual and the OS/32 Link Reference Manual.

Once the macro processor is established, it is loaded by OS/32 with the command:

```
LOAD taskid,fd
```

where:

taskid is the name assigned to the macro processor task partition.

fd is the file descriptor or device mnemonic containing the established macro processor.

The macro processor task is made the currently selected task for making device assignments, starting with the command:

```
TASK taskid
```

Assign all logical units that the macro processor is to use:

LU	COMMAND	DEVICE
source input	AS 1,SOURCE.MAC	disc
list output	AS 3,PR:	printer
source output	AS 2,SOURCE.CAL	disc
macro library*	AS 7,MAG1:	magnetic tape

*The macro library is not a required assignment; when used, it can be assigned to any available LU number.

Issue the operating system START command to execute the macro processor:

```
START
```

The macro processor then executes. When the entire source program has been processed, the macro processor issues an SVC 3, end of task code 0 to the operating system if no errors are

detected. The following end of task codes may be returned by the macro processor:

```
0   :  No errors
2   :  Errors in macro expansion
4   :  Invalid start option
254:  Insufficient memory
```

Any other return code is due to an MNOTE statement in the listing which set the return code.

6.3 I/O ERRORS

When the macro processor detects an I/O error during its operation, it prints this message on the system console:

```
I/O ERROR xxdd
```

where:

xx is a 2-digit hexadecimal value representing the device status.

dd is the device number causing the error.

The possible values of xx are:

DEVICE STATUS CODE	MEANING
C0	illegal function
A0	device unavailable
90	end of medium
88	end of file
84	unrecoverable error
82	parity/recoverable error
81	unassigned lu

Examples:

```
9085    end of medium on magnetic tape
A004    card reader offline
C062    READ attempted from the line printer
```

Refer to the appropriate operating system manual for details on the status of each device.

6.4 START OPTIONS

START options are provided for these functions:

FUNCTION	START OPTION
MLIST	(option, option,...option)
MDEFS	(option, option,...option)
MLIBS	(fd or lu number) the order specifies the search order
MCOPY	fd or lu number
NTRAC	globally turns off all trace facilities
BATCH	batch mode

When invalid start options are specified, a message indicating which option is in error is logged to the user console. The task is terminated with an end of task code 4.

Examples:

```
START     ,MLIBS=(8,9,10)
START     ,MCOPY=7,B,NTRAC
START     ,NTRAC,MLIST=(NONE,GEN)
```

6.5 CAL MACRO/32 PROCESSCR TERMINATION

The CAL Macro Processor goes to end of task under the following conditions:

- An END statement is encountered outside the scope of ASIS, and BATCH is not specified in the start option or in line.
- An end of file is encountered following an END statement which is in the scope of ASIS, and BATCH is not specified in the start option or in line.
- A BEND statement is encountered.
- An end of file is encountered following an END statement (which is generated or in the scope of ASIS), and BATCH is specified in the start option or in line.

CHAPTER 7
MACRO LIBRARY UTILITY PROGRAM

7.1 INTRODUCTION

The Perkin-Elmer Macro Library Utility Program (03-340) provides the capabilities for establishing and maintaining the system macro library and/or any user designated macro libraries. It can:

- create a new library (ESTABLISH command),
- maintain an existing library (GET command),
- include new macro definitions into a library (INCLUDE command),
- delete macro definitions from a library (DELETE command),
- list macro definitions from a library to a device file (LIST command),
- print the directory (names of the macros) of a library to a device or file (DIRECTORY command),
- save an updated library to a permanent file (SAVE command),
- facilitate magnetic tape positioning (FF, BF, RW, WFM commands), and
- accept comment statements (*).

7.2 MACRO LIBRARY

A macro library is a 256-byte record file with this format:

- Header record
- Index records
- Macro definitions

7.2.1 Header Record

The header record, the first record in any macro library, contains information on the last date the library was modified, the size of the library, the type of medium, and user comments.

The header record has this format:

BYTES	DESCRIPTION
1-8	date library was last modified
9	medium code "D" (disc) "M" (magnetic tape or tape cassette)
10	blank
11-12	last record of the current library
13-80	comments or blanks
81-256	blanks

The ESTABLISH command creates the header record or the GET command retrieves it from an existing library.

7.2.2 Index Records

Index records follow the header record. Index records locate macro definitions within a library. The number of index records depends on the number of macro definitions in the library. The macro library utility adjusts the number of records as definitions are INCLUDEd or DELETEd. Each index record contains from 0 to 21 entries. The last entry is marked with an internal end-of-block mark. The format of each entry is:

BYTES	DESCRIPTION
1-8	macro name - the one to eight character mnemonic name used when the macro is invoked.
9-10	zero
11-12	relative record number of the macro definitions

7.2.3 Macro Definitions

Each macro definition starts on a 256-byte record boundary. The definition records are packed into a variable number of variable length logical records (81 bytes maximum). A carriage return (CR) designates the end of a logical record. To facilitate packing, single blanks replace the logical records having leading and trailing blanks. No logical record is split between two physical records. The records are ASCII.

7.3 COMMAND FORMAT

Each macro library utility command has this format:

```
command arg1,arg2,...argn
```

The command verb begins in the first nonblank space of a line and specifies the operation to be performed. The arguments modify the command. The command verb can be specified as the entire verb or any sequence of leading characters making the verb unique.

Example:

```
DE

DEL

DELE

DELET

DELETE
```

All of these verb forms can be accepted for DELETE. The verb form DEM is not accepted for DELETE.

The arguments are separated from the command by at least one blank and from each other by commas or blanks. To obtain the default value if a required argument is omitted, the comma must be included. An end-of-record indicator or 80 bytes, whichever comes first, terminates the command line.

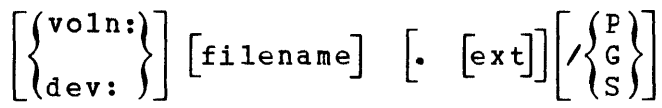
If multiple lines are needed for a command, the last nonblank character of the line to be continued must be a comma. The arguments continue on the next nonblank character of subsequent lines.

Example:

```
command arg1,arg2,
argi,...,argj,
argk,...,argn
```

7.4 MACRO LIBRARY UTILITY COMMANDS

The standard format for the user-specified file descriptor (fd) is:



voln: or dev:	is a disc volume or device name from one to four characters long.
filename	is a filename from 1 to 8 characters long.
.ext	is the extension name and is from one to three characters long preceded by a period.
P G S	are single alphabetic characters representing the file class. They are: P private file G group file S system file

The following sections detail each macro library utility command.

7.4.1 BF Command

The BF command backspaces a magnetic tape device. The device is assigned. The number of specified filemarks are repositioned and closed.

Format:

```
BF file descriptor, [decimal number]
```

Parameters:

file descriptor	is the file descriptor of the device to be backspaced.
decimal number	is the number of filemarks to be backspaced. If the number of filemarks to be backspaced is not specified, the default is 1.

Programming Considerations:

Ensure that filemarks are on the tape. Do not attempt to backspace a magnetic tape that does not have a beginning filemark.

Error Messages:

NO FILE DESCRIPTOR SPECIFIED	Indicates that the file descriptor of the device to be backspaced was not specified.

| DELETE |

7.4.2 DELETE Command

The DELETE command deletes macro definitions from a macro library.

Format:

```
DELETE macro,...
```

Parameters:

macro	is the one to eight character name of the macro.

Programming Considerations:

The requested macros are deleted from the macro library. An updated library taskid.001 is assigned; the message:

```
UPDATED TEMPORARY LIBRARY VOL:taskid.001 NOW AVAILABLE
```

appears. This library can be further modified. When all modifications have been completed, use the SAVE command.

Error Messages:

NO LIBRARY PRESENT	Indicates that a macro library is not present. Use the ESTABLISH or GET commands.
MACRONAME NOT FOUND	Indicates that the macroname was not found.

7.4.3 DIRECTORY Command

The DIRECTORY command writes the names of all macros in the library to an output device.

Format:

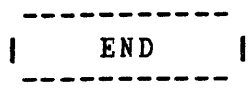
```
DIRECTORY [file descriptor]
```

Parameters:

file descriptor	is the file or device where the names of all macros are written. If the file or device is not specified, the default is the system console.

Programming Considerations:

The date the library was last modified and the comments from the header are also written. The index blocks are searched for macro names. If the file descriptor does not exist, it is allocated and assigned. If the file descriptor does exist, the directory is written to the end-of-file. The file is closed at the end of the DIRECTORY command.



7.4.4 END Command

The END command normally terminates the macro library utility.

Format:

```
END
```

Programming Considerations:

If the temporary library task-id.001 was not saved, this information message:

```
REMINDER!  SAVE YOUR CURRENT LIBRARY
```

is issued. A second END command deletes that file and goes to end of task. If changes were not made to a library, a normal end of task is taken.

7.4.5 ESTABLISH Command

The ESTABLISH command creates a new macro library.

Format:

```
ESTABLISH file descriptor ,comments
```

Parameters:

file descriptor	is the filename or device of the new macro library.
comments	represents up to 63 characters of user comments.

Programming Considerations:

If the temporary library task-id.001 was not saved, this information message:

```
REMINDER! SAVE YOUR CURRENT LIBRARY
```

is issued once. A second ESTABLISH, GET, or END command is performed. If changes to a previous library were not made, a new library is created and the previous unchanged library is closed.

Error Messages:

INSUFFICIENT MEMORY FCR THIS LIBRARY	Not enough memory space exists for this library.

```
-----------
|    FF     |
-----------
```

7.4.6 FF Command

The FF command spaces a magnetic tape forward to a filemark.

Format:

```
FF file descriptor [,decimal number]
```

Parameters:

file descriptor	is the device to be repositioned.
decimal number	is the number of filemarks. If the number of filemarks is not specified, the default is one.

Programming Considerations:

The device is assigned, repositioned (the number of filemarks), and closed.

Error Messages:

NO FILE DESCRIPTOR SPECIFIED	A file descriptor was not specified.

7.4.7 GET Command

The GET command obtains an existing macro library for updating.

Format:

```
GET file descriptor
```

Parameters:

file descriptor	is the existing macro library.

Programming Considerations:

If the temporary library task-id.001 is present, the information message:

```
REMINDER!  SAVE YOUR CURRENT LIBRARY
```

is issued once. A second GET, ESTABLISH, or END command is performed. If changes were not made to a previous library, the new library is obtained and the previous unchanged library is closed.

Error Messages:

NO FILE DESCRIPTOR SPECIFIED	A file descriptor was not specified.

LU 1 file descriptor DOES NOT EXIST--CANNOT ASSIGN

RECCRD LENGTH NOT 256. NOT A MACRO LIBRARY.

INSUFFICIENT MEMORY FOR THIS LIBRARY	Not enough memory space for this library.

```
-----------
| INCLUDE |
-----------
```

7.4.8 INCLUDE Command

The INCLUDE command includes new macro definitions into a library.

Format:

```
INCLUDE file descriptor [,macro][,macro]...
```

Parameters:

file descriptor	is the source filename.
macro	is the one to eight character macro name. If the character macro name is not specified, the default is all macros on the source file.

Programming Considerations:

The macro(s) specified (all if none specified) are included in the current macro library. The old library is closed and the updated temporary library task-id.001 is assigned as the current library; and, the messages:

```
XXX MACROS INCLUDED FROM file descriptor
UPDATED TEMPORARY LIBRARY VOL:task-id.001 NOW AVAILABLE
```

are issued. This library must be saved to become permanent. Macro definitions to be included in a library must be reasonably debugged and syntactically correct. A minimum of error checking is performed on the macro statements. An invalid CAL statement could cause problems and should be avoided. Any file that CAL Macro can process with NO ERRORS can be used safely as input to the macro library utility. Macros should be tested with CAL Macro before they are placed in a library.

A single blank replaces leading and trailing blanks to conserve library space and provide for a faster running of CAL Macro. Macro definitions with sequence numbers, while accepted, should be used since blank compression cannot be performed.

Error Messages:

NO FILE DESCRIPTOR SPECIFIED

DUPLICATE MACRO macroname	The macro already exists in the library.
ILLEGAL MACRO NAME name	The macro prototype statement is invalid.

The following statements inside a macro definition cause an error:

MACRO STATEMENT ENCOUNTERED AT LINE XXX - Macro

MEND STATEMENT ENCOUNTERED AT LINE XXX - Mend

BEND STATEMENT ENCOUNTERED AT LINE XXX - Bend

END STATEMENT ENCOUNTERED AT LINE XXX - End

EOF ENCOUNTERED IN A MACRO DEFINITION

The assumption is that there is a missing MACRO or MEND statement. If any of these messages occur, check the library carefully to find which macros have actually been included. Also, column 72 of the statement preceding the MEND must be blank. A statement that extends past column 72 causes the MEND statement to be treated as a continuation. However, an END statement in the range of an ASIS statement does not cause the error, nor does it terminate the INCLUDE command.

```
-----------
|   LIST    |
-----------
```

7.4.9 LIST Command

The LIST command writes macro definitions to an output device.

Format:

LIST [file descriptor] [,macro],...

Parameters:

file descriptor	the file or device where the macro definitions are written. If a file or a device is not specified, the default is the system console.
macro	names of the macros to be written. If the names are not specified, the default is all macros in the library.

Programming Considerations:

Macro definitions are written to the output device in the order that the LIST command specifies. If the file does not exit, it is allocated and assigned. If the file does exit, the macro definitions are written to the end of the file. The file is closed at the end of the LIST command and one of these messages is written:

n MACROS LISTED TO file descriptor

or

n MACROS LISTED TO NEW FILE file descriptor

Error Messages:

NO LIBRARY PRESENT	no macro library is present (use the GET or ESTABLISH commands).
name MACRO NOT FOUND	a macro was not in the library
THIS LIBRARY CONTAINS INVALID MACROS	the library has to be rebuilt

7.4.10 PAUSE Command

The pause command pauses the macro library utility.

Format:

```
PAUSE
```

7.4.11 RW Command

This command rewinds a magnetic tape. The device is assigned and closed.

Format:

RW file descriptor

Parameters:

file descriptor	is the device name of the magnetic tape to be rewound.

Error Messages:

NO FILE DESCRIPTOR SPECIFIED

7.4.12 SAVE Command

The SAVE command saves the temporary updated macro library to a permanent file or device.

Format:

SAVE {file descriptor / *}

Parameters:

file descriptor	is the name of a new file.
*	is the same file descriptor used in the last GET or SAVE command.

Programming Considerations:

The SAVE file descriptor causes a new file descriptor to be allocated and assigned. If the file already exists, this message appears:

file descriptor EXISTS. DELETE AND REALLOCATE?

Expected responses are:

YES or NO

Any other response issues the message:

PLEASE ANSWER YES OR NO

A NO response does nothing more and invites the next command. A YES response first deletes the old library. If the old library was a private disc file, the temporary library task-id.001 is then renamed to the old library. If a rename is not possible

(different disc, magnetic tape), then the temporary library task-id.001 is copied to the new file or device. In either case, the new library is assigned as the current library.

If SAVE * is issued, this action occurs without the prompt message. SAVE * should not be issued for a file that cannot be renamed or allocated (system or group file or magnetic tape).

If the SAVE is successful, the message:

UPDATED LIBRARY file descriptor NOW AVAILABLE is written.

Error Messages:

NO FILE DESCRIPTOR SPECIFIED

7.4.13 WFM Command

The WFM command writes a filemark to a magnetic tape.

FORMAT:

WFM file descriptor

Parameters:

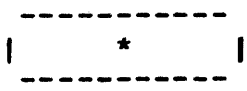
file descriptor	is the device to write a filemark.

Programming Considerations:

The device is assigned; a filemark is written; and the device is closed. No further repositioning occurs. The magnetic tape is then positioned after the filemark.

Error Messages:

NO FILE DESCRIPTOR SPECIFIED



7.4.14 Comments

Any character string starting with an asterisk (*) in column 1 is treated as a comment.

Format:

```
* any string of characters
```

7.5 OPERATION WITH A MACRO LIBRARY ON MAGNETIC TAPE

The positioning and automatic repositioning of a magnetic tape by the macro library utility are important if the library resides on magnetic tape. This list shows positioning before and after each command:

COMMAND	BEFORE	AFTER
GET	Beginning of header record	Beginning of first macro or library (first record after index records)
COPY	Beginning of first macro or library	Same
INCLUDE	Beginning of first macro or library	Past filemark at end of library
DELETE	Beginning of first macro or library	Past filemark at end of library
SAVE	Assumption (see other commands)	Beginning of first macro new library (same as GET) beginning and ending file marks automatically written
DIRECTORY	No change	No change

7.6 OPERATION OF A MACRO LIBRARY UTILITY UNDER OS/32

Before the macro library utility can be used, it must be established as a user task with OS/32 Link. See the OS/32 Operator Reference Manual and the OS/32 Link Reference Manual.

An EXPAND factor should be included in Link. To process the command, 256 bytes are needed and an additional 256 bytes are needed for every 21 macros in a library.

Example:

```
EXPAND 6 is sufficient to process 100 macros.
```

Once the macro library utility is established, it is loaded by the operating system with the command:

```
LOAD taskid,fd
```

where:

taskid	is the name assigned to the macro library utility partition.
fd	is the file descriptor or device mnemonic containing the established macro library utility.

The macro library utility task is made the currently selected task with the command:

```
TASK taskid
```

Device assignments do not have to be made. However, if a device other than CON: is to be used for command input and message output, the input and output can be assigned to LU 5 and LU 6, respectively.

The macro library utility is executed with the command:

```
START
```

If the files taskid.001, taskid.002, or taskid.003 exist, they must be renamed or deleted. These files are used as scratch files for the macro library utility.

APPENDIX A
COMMAND SUMMARY

BF file descriptor,[decimal number]

backspaces a magnetic tape to a filemark.

DELETE macro,...

deletes macro definitions from a macro library.

DIRECTORY [file descriptor]

writes the names of all macros in the library to an output device.

END

normally terminates the macro library.

ESTABLISH file descriptor ,comments

creates a new macro library

FF file descriptor [,decimal number]

forward spaces a magnetic tape to a filemark.

GET file descriptor

obtains an existing macro library for updating.

INCLUDE file descriptor [,macro] [,macro]...

includes new definitions into a library

LIST [file descriptor] [,macro],...

writes macro definitions to an output device

PAUSE

pauses the Macro Library Utility

RW file descriptor

rewinds a magnetic tape

SAVE {file descriptor / *}

saves the temporary updated macro library to a permanent file or device

WFM file descriptor

writes a filemark to a magnetic tape

APPENDIX B
INSTRUCTION STATEMENT SUMMARY

NAME	OPERATICN	OPERAND
symbol	AIF	Logical expression enclosed in parentheses immediately followed by a sequence symbol
	ACTR	Any expression that can appear in the operand field of a SETA instruction
symbol	AGO	A sequence symbol
sequence symbol	ANOP	Blank
symbol	ASIS	Blank, decimal integer, BEGIN, or END
	BGBLA, BGBLB, BGBLC	One or more variable symbols used as batch global SET variable symbols, separated by commas
	GBLA, GBLB, GBLC	One or more variable symbols used as global set variable symbols, separated by commas
	LCLA, LCLB, LCLC	One or more variable symbols used as local SET variable symbols, separated by commas
	MACRO	Version number and date or any other optional description. Column 72 is blank.
sequence symbol	MEND	
	MCALL	MACRO,MACRO,...,MACRO
	MDEFS	One or more codes separated by commas
sequence symbol	MEXIT	

NAME	OPERATICN	OPERAND
	MLIBS	One or more file descriptors or lu numbers separated by commas
	MLIST	One or more codes separated by commas
sequence symbol	MNOTE	An optional integer expression followed by a quoted message string separated by a comma
	MPAUS	
	MTRAC	
	NCLIB	One or more file descriptors or lu numbers separated by commas
	NTRAC	
SETA symbol or array element	SETA	An arithmetic expression
SETB symbol or array element	SETB	0, 1, or logical expression enclosed in parentheses
SETC symbol or array element	SETC	One operand consisting of a type attribute, character expression, substring notation or concatenation, or substring notations and character expressions

APPENDIX C
EXAMPLES OF MACRO EXPANSION

Example 1: Expansion of the Macro Instruction COMPR

The following example shows the expansion of the macro instruction, COMPR. COMPR compares two byte-oriented fields (%FLD1 and %FLD2) of equal length (%SIZE) for a normal condition (%PEQ.NE) of equal or nonequal. If the normal condition is not met, a branch is taken to an error routine (%ERRTN).

The expansion of the COMPR macro instruction is subject to this restriction:

> The operand replacing the symbolic parameter %EQ.NE must not be 'EQ' or 'NE'.

If this restriction is not met, further macro expansion is suppressed and the appropriate error message is passed to the source stream.

```
          MACRO                                                          1
%NAME     COMPR  %FLD1,%FLD2,%SIZE,%EQ.NE,%ERRTN                         2
          GBLB   %BG01,%BG02                                             3
          AIF    (('%EQ.NE' NE 'EQ') AND ('%EQ.NE' NE 'NE'))&ERR1        4
          STM    11,@@@RHOLD           SAVE REGS 11 THRU 15             5
          AIF    ('%EQ.NE' EQ 'EQ')&EQ01                                  6
          BAL    12,@@@CMPNE           GO CHECK FOR NOT EQUAL.          7
          AGO    &SETARGS                                                 8
&EQ01     ANOP                                                           9
          BAL    12,@@@CMPEQ           GO CHECK FOR EQUAL.              10
&SETARGS  ANOP                                                           11
          DAC    A(%FLD1)              BASE ADDRESS OF FIELD 1.         12
          DAC    A(%FLD2)              BASE ADDRESS OF FIELD 2.         13
          DAC    %SIZE                 FIELD SIZE FOR COMPARE.          14
          DAC    A(%ERRTN)             ADDRESS OF ERROR ROUTINE.        15
          AIF    ('%EQ.NE' EQ 'EQ')&EQ02                                  16
          AIF    (%BG01 EQ 1)&GETOUT                                      17
          AIF    (%BG02 EQ 1)&SHORT                                       18
          B      6*ADC+@@@RHOLD        BYPASS COMPARE SUBROUTINE        19
          AGO    &CMPNE                                                   20
&SHORT    ANOP                                                           21
          B      @@@NE001+4            BYPASS COMPARE SUBROUTINE        22
&CMPNE    ANOP                                                           23
@@@CMPNE  AIS    12,ADC-2              ASSURE CORRECT ALIGNMENT OF      24
          AGO    &SUBR                                                    25
```

```
&EQ02     ANOP                                                          26
          AIF     (%BG02 EQ 1)&GETOUT                                   27
          AIF     (%BG01 EQ 1)&SHORT1                                   28
          B       6*ADC+@@@RHOLD        BYPASS COMPARE SUBROUTINE.      29
          AGO     &CMPEQ                                                30
&SHORT1   ANOP                                                          31
          B       @@@EQ001+4            BYPASS COMPARE SUBROUTINE.      32
&CMPEQ    ANOP                                                          33
@@@CMPEQ  AIS     12,ADC-2              ASSURE CORRECT ALIGNMENT OF     34
&SUBR     ANOP                                                          35
          NAI     12,-ADC               ADDRESS OF ARGUMENT LIST.       36
          LDA     15,0(12)              R15=ADDR OF 1ST FIELD.          37
          LDA     14,ADC(12)            R14=ADDR OF 2ND FIELD.          38
          LDA     13,2*ADC(12)          R13=FIELD SIZE.                 39
          LDA     11,3*ADC(12)          R11=ADDR OF ERROR ROUTINE.      40
          AHI     12,4*ADC              POINT R12 TO RETURN ADDR.       41
          STA     12,5*ADC+@@@RHOLD     SAVE RETURN ADDR.               42
%NAME     LB      12,0(15)              R12=BYTE FROM FIELD 1.          43
          CLB     12,0(14)              COMPARE WITH SAME BYTE
                                        IN FIELD 2.                     44
          BNE     %NAME:B               GET OUT IF THEY'RE NOT EQUAL.   45
          AIS     15,1                  BUMP ADDR OF FIELD 1.           46
          AIS     14,1                  BUMP ADDR OF FIELD 2.           47
          SIS     13,1                  DECREMENT FIELD SIZE.           48
          BNZ     %NAME                 CHECK NEXT BYTE IF
                                        FIELD SIZE > 0                  49
          AIF     ('%EQ.NE' EQ 'EQ')&EQ03                               50
%NAME:A   LM      12,ADC+@@@RHOLD       FIELDS ARE =,
                                        RESTORE REGS & ERROR            51
          B       0(11)                 EXIT.                           52
%NAME:B   LDA     11,5*ADC+@@@RHOLD     FIELDS ARE NOT =,
                                        R11 - NORMAL RETURN             53
@@@NE001  B       %NAME:A               GO RESTORE REGS AND GET OUT     54
%BG01     SETB    1                                                     55
          AIF     (%BG02 EQ 1)&GETOUT                                   56
@@@RHOLD  DAS     6                     REGISTER & RETURN SAVE AREA     57
          MEXIT                                                         58
&EQ03     ANOP                                                          59
          LDA     11,5*ADC+@@@RHOLD     FIELDS ARE =,
                                        R11 = NORMAL RETURN             60
%NAME:B   LM      12,ADC+@@@RHOLD       RESTORE REGS (ALSO HERE
                                        IF FLDS NOT =)                  61
@@@EQ001  B       0(11)                 EXIT.                           62
%BG02     SETB    1                                                     63
          AIF     (%BG01 EQ 1)&GETOUT                                   64
@@@RHOLD  DAS     6                     REGISTER & RETURN SAVE AREA     65
          MEXIT                                                         66
&ERR1     ANOP                                                          67
          MNOTE   'INVALID COMPARE TYPE'                                68
          MEXIT                                                         69
&GETOUT   ANOP                                                          70
          MEND                                                          71

CMPREQ1   COMPR   FLDA,FLDB,26,EQ,ERROR1                                72
          STM     11,@@@RHOLD           SAVE REGS 11 THRU 15.
          BAL     12,@@@CMPEQ           GO CHECK FOR EQUAL.
```

```
            DAC     A(FLDA)               BASE ADDRESS OF FIELD 1.
            DAC     A(FLDB)               BASE ADDRESS OF FIELD 2.
            DAC     26                    FIELD SIZE FOR COMPARE.
            DAC     A(ERROR1)             ADDRESS OF ERROR ROUTINE.
            B       6*ADC+@@@RHOLD        BYPASS COMPARE SUBROUTINE.
@@@CMPEQ    AIS     12,ADC-2              ASSURE CORRECT ALIGNMENT OF
            NAI     12,-ADC               ADDRESS OF ARGUMENT LIST.
            LDA     15,0(12)              R15=ADDR OF 1ST FIELD.
            LDA     14,ADC(12)            R14=ADDR OF 2ND FIELD.
            LDA     13,2*ADC(12)          R13=FIELD SIZE.
            LDA     11,3*ADC(12)          R11=ADDR OF ERROR ROUTINE
            AHI     12,4*ADC              POINT R12 TO RETURN ADDR.
            STA     12,5*ADC+@@@RHCLD     SAVE RETURN ADDR.
CMPREQ1     LB      12,0(15)              R12=BYTE FROM FIELD 1.
            CLB     12,0(14)              COMPARE WITH SAME BYTE IN FIELD 2.
            BNE     CMPREQ1B              GET OUT IF THEY'RE NOT EQUAL.
            AIS     15,1                  BUMP ADDR OF FIELD 1.
            AIS     14,1                  BUMP ADDR OF FIELD 2.
            SIS     13,1                  DECREMENT FIELD SIZE.
            BNZ     CMPREQ1               CHECK NEXT BYTE IF FIELD SIZE > 0.
            LDA     11,5*ADC+@@@RHOLD     FIELDS ARE =, R11 = NORMAL RETURN.
CMPREQ1B    LM      12,ADC+@@@RHOLD       RESTORE REGS (ALSO HERE IF FLDS NOT =).
@@@EQ001    B       0(11)                 EXIT.
@@@RHOLD    DAS     6                     REGISTER & RETURN SAVE AREA.
CMPRENE1    COMPR   AFLD,BFLD,91,NE,ERR01
            STM     11,@@@RHOLD           SAVE REGS 11 THRU 15.                 73
            BAL     12,@@@CMPNE           GO CHECK FOR NOT EQUAL.
            DAC     A(AFLD)               BASE ADDRESS OF FIELD 1.
            DAC     A(BFLD)               BASE ADDRESS OF FIELD 2.
            DAC     91                    FIELD SIZE FOR COMPARE.
            DAC     A(ERR01)              ADDRESS OF ERROR ROUTINE.
            B       @@@NE001+4            BYPASS COMPARE SUBROUTINE.
@@@CMPNE    AIS     12,ADC-2              ASSURE CORRECT ALIGNMENT OF
            NAI     12,-ADC               ADDRESS OF ARGUMENT LIST.
            LDA     15,0(12)              R15=ADDR OF 1ST FIELD.
            LDA     14,ADC(12)            R14=ADDR OF 2ND FIELD.
            LDA     13,2*ADC(12)          R13=FIELD SIZE.
            LDA     11,3*ADC(12)          R11=ADDR OF ERROR ROUTINE.
            AHI     12,4*ADC              POINT R12 TO RETURN ADDR.
            STA     12,5*ADC+@@@RHOLD     SAVE RETURN ADDR.
CMPRNE1     LB      12,0(15)              R12=BYTE FROM FIELD 1.
            CLB     12,0(14)              COMPARE WITH SAME BYTE IN FIELD 2.
            BNE     CMPRNE1B              GET OUT IF THEY'RE NOT EQUAL.
            AIS     15,1                  BUMP ADDR OF FIELD 1.
            AIS     14,1                  BUMP ADDR OF FIELD 2.
            SIS     13,1                  DECREMENT FIELD SIZE.
            BNZ     CMPRNE1               CHECK NEXT BYTE IF FIELD SIZE 0.
CMPRNE1A    LM      12,ADC+@@@RHCLD       FIELDS ARE =, RESTORE REGS AND ERROR.
            B       0(11)                 EXIT.
CMPRNE1B    LDA     11,5*ADC+@@@RHOLD     FIELDS ARE NOT =, R11 = NORMAL RETURN.
@@@NE001    B       CMPRNE1A              GO RESTORE REGS & GET OUT.
CMPREQ2     COMPR   F1,F2,13,EQ,ERROR2                                          74
            STM     11,@@@RHOLD           SAVE REGS 11 THRU 15.
            BAL     12,@@@CMPEQ           GO CHECK FOR EQUAL.
            DAC     A(F1)                 BASE ADDRESS OF FIELD 1.
            DAC     A(F2)                 BASE ADDRESS OF FIELD 2.
```

```
          DAC     13                  FIELD SIZE FOR COMPARE.
          DAC     A(ERROR2)           ADDRESS OF ERROR ROUTINE.
CMPRNE2   COMPR   FLD1,FLD2,52,NE,ERR02                              75
          STM     11,@@@RHOLD         SAVE REGS 11 THRU 15.
          BAL     12,@@@CMPNE         GO CHECK FOR NOT EQUAL.
          DAC     A(FLD1)             BASE ADDRESS OF FIELD 1.
          DAC     A(FLD2)             BASE ADDRESS OF FIELD 2.
          DAC     52                  FIELD SIZE FOR COMPARE.
          DAC     A(ERR02)            ADDRESS OF ERROR ROUTINE.

COMPARE   COMPR   FIELDA,FIELDB,39,LT,ERR03                          76
*         MNOTE   'INVALID COMPARE TYPE'
```

SUMMARY:

When the first equal compare (EQ) is made, the subroutine '@@@CMPEQ' is expanded. When the first request for a nonequal compare (NE) is made, the subroutine '@@@CMPNE' is expanded. All subsequent uses of the COMPR macro instruction in the same source stream result in the expansion of a BAL to the applicable subroutine, followed by an argument list. When the macro instruction COMPR is first invoked (for an equal or a nonequal compare), a register storage area identified by the label '@@@RHOLD' is defined. Registers 11 through 15 are used in executing the subroutines; but, registers 12 through 15 are restored before exiting. The original contents of register 11 are available to the user in location @@@RHOLD.

ANALYSIS:

- Statements 1 through 71 constitute the macro definition.
- Statements 72 through 76 show 5 possible calls of the macro.
- Statement 72 represents the initial call of the COMPR macro instruction in the current source stream and the initial use of the EQ operand in a COMPR instruction in this source stream. This representation of the statement results in the full expansion of the @@@CMPEQ subroutine and the register storage area '@@@RHOLD' appended to the '@@@RHOLD' subroutine.

  When the @@@CMPEQ subroutine is completed, the binary global '%BG02' is set to true, indicating the presence of the subroutine.
- Statement 73 represents the initial use of the NE operand in a COMPR instruction in this source stream, which results in the full expansion of the @@CMPNE subroutine. The text of %BG02 in statement 56 causes the definition of the register storage area to be suppressed and further macro expansion to be terminated. When the expansion of the @@@CMPNE subroutine is completed, the '%BG01' binary global is set to true, indicating the subroutine.

- Statement 74 represents a subsequent use of the EQ operand and results in the expansion of a BAL to @@@CMPEQ followed by the applicable argument list.
- Statement 75 represents a subsequent use of the NE operand and results in the expansion of a BAL to @@@CMPNE followed by the applicable argument list.
- Statement 76 shows the results of an invalid compare type.

Example 2: Expansion of the Macro Instruction PRIME:

The following example illustrates the expansion of the macro instruction PRIME. PRIME generates a table of prime numbers %LENGTH long, beginning with the prime numbers %PRIME1.

```
                MACRO                                          1
                PRIME     %PRIME1,%LENGTH                      2
                LCLA      %AL1,%AL2,%AL3                       3
                AIF       ('%LENGTH' LT '1')&ER2               4
                AIF       ('%PRIME1' LT '2')&ER1               5
%AL1            SETA      %PRIME1                              6
%AL2            SETA      %AL1/2                               7
&LOOP1          AIF       (%AL2 EQ 1)&GETREST                  8
                AIF       (%AL1/%AL2*AL2 EQ %AL1)&ER1          9
%AL2            SETA      %AL2-1                              10
                AGO       &LOOP1                              11
&GETREST        ANOP                                          12
                DAC       %AL1                                13
%AL3            SETA      %AL3+1                              14
                AIF       (%AL3 LT %LENGTH)&LOOP2             15
                MEXIT                                         16
&LOCP2          ANOP                                          17
%AL1            SETA      %AL1+1                              18
%AL2            SETA      %AL1/2                              19
&LOOP3          AIF       (%AL2 EQ 1)&GETREST                 20
                AIF       (%AL1/%AL2*%AL2 EQ %AL1)&LOOP2      21
%AL2            SETA      %AL2-1                              22
                AGO       &LOOP3                              23
&ER1            ANOP                                          24
                MNOTE     '%PRIME1 NOT A PRIME NUMBER'        25
                MEXIT                                         26
&ER2            ANOP                                          27
                MNOTE     ' LENGTH LT 1'                      28
                MEND                                          29
                PRIME     13,5                                30
DAC 13
DAC 17
DAC 19
DAC 23
DAC 29
                PRIME 55,20                                   31
*MNOTE '55 NOT A PRIME NUMBER'
```

SUMMARY:

If the operand replacing the symbolic parameter %PRIME1 is not a prime number or if %LENGTH is less than 1, macro expansion is suppressed and an error message is passed to the source stream.

ANALYSIS:

- Statements 1 through 29 constitute the macro definition.
- Statements 30 and 31 show two possible expansions of the macro instruction.
- Statement 30 requests a table of prime numbers, 5 address-length constants long, beginning with the number 13. Because 13 is a prime number, the expansion of the macro instruction takes place.
- Statement 31 requests a table of prime numbers, 20 address-length constants long, beginning with the number 55. Because 55 is not a prime number, the macro expansion is suppressed and the error message results.

APPENDIX D
CAL MACRO/32 PROCESSOR ERROR MESSAGES

ERRCR CODE	MEANING	EXPLANATION
1	STRING TOO LONG	The length of a string literal in a macro definition exceeds 255 characters.
2	ILLEGAL CHARACTER	An illegal character was encountered in the input stream and was changed to a '#'.
3	UNEXPANDABLE MACRO	A macro invocation was encountered which cannot be expanded because of errors in the definition.
4	OPCODE NOT ALLOWED TO BE GENERATED	An operation code allowed only in a source statement was obtained through substitution of a value for a variable symbol.
5	INVALID OPCODE	The length of an operation code exceeds eight characters or the operation code is missing or not followed by a blank.
6	ILLEGAL OPCODE	An operation code allowed only in a macro definition was encountered in open code.
7	UNDEFINED VARIABLE SYMBOL	A variable symbol, not declared in a BGBLx, GBLx, or LCLx statement or in a macro prototype, was encountered in a SETx, AIF, or model statement.

ERROR CODE	MEANING	EXPLANATION
8	UNDEFINED SEQUENCE SYMBOL	A sequence symbol was declared in the operand field of an AIF or AGO statement, but does not occur in the name field of any statement in the macro definition.
9	UNDEFINED KEYWORD PARAMETER	A keyword was encountered in a macro instruction operand that does not correspond to any keyword in the macro prototype.
10	MULTIPLY DEFINED MACRO NAME	A macro definition of the same name was encountered previously, or a macro was invoked before it was defined.
11	MULTIPLY DEFINED SETx SYMBOL	A SETx variable symbol was defined more than once in an LCLx, BGBLx, or GLBx statement, or it has the same name as a parameter or system variable.
12	MULTIPLY DEFINED SEQUENCE SYMBOL	A sequence symbol of the same name occurs in the same name field of a previous statement.
13	MULTIPLY DEFINED PARAMETER	A symbol parameter in a macro prototype occurs more than once or has the same name as a system variable.
14	ILLEGAL PARAMETER SEQUENCE	A keyword parameter precedes a positional parameter in a macro prototype or instruction.
15	ILLEGAL OPCODE	A BGBLx, GBLx, or LCLx SEQUENCE statement does not precede all executable statements in a macro definition, or an ACTR statement does not immediately follow the declaration statements.

ERROR CODE	MEANING	EXPLANATION
16	ILLEGAL VARIABLE SYMBOL	A variable symbol is longer than seven characters or has a first character that is not alphabetic.
17	ILLEGAL SEQUENCE SYMBOL	A sequence symbol is longer than seven characters or has a first character that is not alphabetic.
18	ILLEGAL DECLARATION	The syntax of a BGBLx, GBLx, or LCLx statement is incorrect. Either an operand is not a variable symbol or a comma is missing.
19	ILLEGAL MACRO NAME	The length of a macro name exceeds eight characters or corresponds to a reserved opcode, or the syntax of the operand field of an MCALL statement is incorrect.
20	MISMATCHED SETx TYPE	The types of the SETx variable statements and the variable symbol in the name field of that operation do not match.
21	MISSING SETx SYMBOL	The name field of a SETx statement is blank or does not contain a variable symbol.
22	MISSING SEQUENCE SYMBOL	A sequence symbol is missing in the name field of an ANOP statement or in the operand field of a AGO or AIF statement.
23	ILLEGAL NAME FIELD	The statement name field in a macro definition contains a token forbidden in that position.
24	NON-BLANK NAME FIELD	Self-explanatory.
25	MISSING OPERAND	Self-explanatory.

ERROR CODE	MEANING	EXPLANATION
26	ILLEGAL ARITHMETIC EXPRESSION	The syntax of OP, an arithmetic expression, cannot be parsed.
27	ILLEGAL BOOLEAN EXPRESSION	The syntax of a Boolean expression cannot be parsed.
28	ILLEGAL CHARACTER EXPRESSION	The syntax of a character expression cannot be parsed.
29	EXPRESSION STACK OVERFLOW	An arithmetic, Boolean, binary, or character expression was encountered that exceeds 15 levels of parentheses with two operations pending at each level.
30	INVALID ATTRIBUTE FUNCTION	The operand of an attribute function is not a symbolic parameter.
31	ILLEGAL ATTRIBUTE FUNCTION	A type attribute function occurs in an arithmetic expression.
32	TYPE FUNCTION NOT ALONE	A type attribute function is not the only element in a character expression.
33	ILLEGAL SUBLIST NOTATION	The syntax of a sublist notation is incorrect.
34	MISMATCHED PARENTHESES	The parentheses in a macro instruction, or prototype, or character expression do not balance.
35	MISMATCHED QUOTES	The quotes in a macro instruction or prototype, or in a character expression or model statement do not balance.
36	MISSING COMMA	A comma is missing.
37	ILLEGAL QUCTE	An illegal quote was encountered in the name field or operation field of a model statement.

ERRCR CODE	MEANING	EXPLANATION
38	ILLEGAL SYMBOL	A model statement contains an illegal character not enclosed in quotes.
39	UNRECOGNIZABLE SYMBOL	An expression contains an illegal character.
40	BATCH IN ILLEGAL POSITION	A batch pseudo-op was encountered that is not the first statement of the program.
41	END BEFORE MEND	An end pseudo-op was encountered in a macro definition.
42	BEND BEFORE MEND	A bend pseudo-op was encountered in a macro definition.
43	ILLEGAL BEND	A bend pseudo-op was encountered but no preceding batch pseudo-op occurred.
44	ILLEGAL SUBSTRING EVALUATION	In substring notation, the value of the second expression is less than the value of the first expression.
45	ILLEGAL SUBLIST EVALUATION	An expression in sublist notation evaluates to a negative number.
46	ARITHMETIC OVERFLOW	An arithmetic expression evaluates to a number outside the range +2, 147, 483, 647.
47	ACTR OVERFLOW	The arithmetic expression in an ACTR statement evaluates to a number greater than 32,767.
48	ATTEMPT TO DIVIDE BY ZERO	Self-explanatory.

ERROR CODE	MEANING	EXPLANATION
49	ILLEGAL ARITHMETIC OPERATION	An operand of an arithmetic operation is a SETC variable statement or symbolic parameter whose value contains other than an optional sign followed by numerics.
50	ILLEGAL LOGICAL UNIT	The operand of an MLIBS or NOLIB statement is not numeric or an attempt was made to assign an LU that was previously assigned or not present.
51	ILLEGAL LABEL	The label in the operand field of an MCOPY statement is blank or contains an illegal character.
52	ACTR RUNOUT	Loop counter runout.
53	DICTIONARY SPACE FULL	Insufficient memory space was allocated for the macro processor's dynamic tables.
54	UNRECOGNIZABLE LINE	A line was encountered in the input that is neither blank nor a comment nor a continuation card, but has a missing operation code.
55	TOO MANY SYMBOLS	A symbol table overflow.
56	BUFFER OVERFLOW	A macro statement is over 256 bytes.
57	ILLEGAL SUBSCRIPT	A subscript is not arithmetic.
58	MISSING SUBSCRIPT	Self-explanatory.
59	ILLEGAL ARRAY EXPRESSION	Self-explanatory.
60	DIMENSION OF ARRAYS TOO LARGE, INCREASE MEMORY	Increase memory.

ERROR CODE	MEANING	EXPLANATION
61	ILLEGAL TYPE FOR AIF OR AGO	The variable symbol is not a type character.
62	DIFFERENT TYPE DECLARATIONS FOR GLOBAL VARIABLE	A global variable name was declared twice with different types.
63	SUBSCRIPT OUT OF RANGE	Out of range subscript was detected.
64	NONEXISTENT SEQUENCE SYMBOL FOR COMPUTED AIF OR AGO; INVALID CODE FOR SYSTEM MACRO	Self-explanatory.
65	MORE THAN 16 MACRO LIBRARIES	Self explanatory.
66	MORE THAN 16 NESTED MCOPY STATEMENTS	Self explanatory.
67	ILLEGAL OPTION FOR MLIST OR MDEFS	Self explanatory.
68	MACRO COMMENT	A statement with :* in columns 1 and 2 is outside the macro definition.
69	ASIS ERROR	The statement is not allowed in the range of ASIS.

GLOSSARY

batch global set variable (BGBLx) symbol

The BGBLx symbol communicates values between macro definitions or between different usages for the same macro definitions in different programs. It must be declared as batch global each time it is used in a macro definition.

conditional branch (AIF) instruction

The AIF instruction alters the macro definition statement processing sequence.

conditional instruction loop counter (ACTR) instruction

The ACTR instruction assigns a count other than 32767 as the maximum number of AIF and AGO branches executed within a macro definition.

conditional instructions

Conditional instructions are instructions that can vary a macro instruction at each invocation.

count attribute

The count attribute is the number of all characters in a macro instruction field. It includes all characters in the operand plus apostrophes; but, it does not include delimiting commas.

generated statements (expanded statements)

Generated statements are the assembler statements the processor processes.

global SET variable (GBLx) symbol

The GBLx symbol communicates values between macro definitions or between different usages for the same macro definition in a program. It must be declared as global each time it is used in a macro definition.

header record

A header record is the first record in any macro library. It contains: the last date the library was modified, the library size, the medium type, and user comments.

index records

Index records are records used to locate macro definitions within a library. Each index record contains from 0 to 21 entries.

inner macro instruction

When the mnemonic operation code for a given macro definition appears as the operation field of a model statement in another macro definition, the model statement is an inner macro instruction.

keyword

A keyword is the portion of a symbolic parameter that does not include the percent sign.

keyword macro instruction

A keyword macro instruction is a specific type of macro instruction in which each operand must consist of a keyword immediately followed by an equal sign (=), followed (optionally) by a value.

local SET variable (LCLx) symbols

The LCLx symbols communicate values within the same usage of a particular macro definition. It is only declared in the macro definition that it is used in and it is reset to its initial value each time that macro definition is invoked.

macro call (MCALL) instruction

The MCALL instruction permits the cited macros to be called as they appear in the library rather than as they appear in the processor source stream.

A macro definition is a series of user written statements in the macro language. The language enables the user to assign a mnemonic operation code to the definition. This mnemonic operation code causes the definition to be invoked. A macro definition minimally consists of: a name, operation,

operand, comments, continuation, and identification/sequence field used for writing macro instructions.

macro definition header (MACRO)

A macro definition header indicates the beginning of a macro definition. It must be the first statement.

macro definition trailer (MEND)

A macro definition trailer indicates the end of a macro definition. It must be the last statement in the definition.

macro definitions (MDEFS) instruction

The MDEFS instruction controls which macro statements are sent to the CAL file.

macro instruction

A macro instruction is a single instruction that expands to a series of instructions. It invokes and processes a given macro definition. The instruction can be positional, keyword, or mixed operand, corresponding to the three forms of macro prototype statements.

macro instruction prototype statement

A macro instruction prototype statement specifies the mnemonic operation code and general format to be used when writing any macro instructions referring to this definition.

macro library

A macro library is a 256-byte record file containing: a header record, index records, and macro definitions.

macro libraries (MLIBS) instruction

The MLIBS instruction designates the file descriptor or decimal LU numbers where the macro libraries, necessary in a given macro processor source stream, reside.

macro trace (MTRAC) instruction

The MTRAC instruction, a diagnostic instruction, determines the effective conditional branches and the SET variable symbols values within the macro logic.

MCOPY statement

The MCOPY statement enables source text to be copied from a specified LU or file descriptor at any point in a macro definition or program.

MEXIT instruction

The MEXIT instruction terminates the current macro definition expansion.

mixed mode macro instructions

Mixed mode macro instructions are a specific type of macro instruction in which all positional operands must be placed before any keyword operands.

mnemonic operation code

A mnemonic operation code is a user-assigned code that enables the macro definition to be invoked.

model statements

Model statements are statements from which the CAL Macro Processor expands the desired source statements. The four fields of a model statement are: name, operation, operand, and comments.

no libraries (NOLIB) instruction

The NOLIB instruction suppresses searching all or some macro libraries previously designated by the MLIBS statement.

no operation (ANOP) instruction

The ANOP instruction is used when the sequence symbol in an AIF or AGO instruction must reference a statement already containing a symbol (other than a sequence symbol) in the name field.

no trace (NTRAC) instruction

The NTRAC instruction causes the macro trace feature to be disabled.

number attribute

A number attribute is a value equal to the number of operands in an operand sublist. The number is equal to one plus the number of delimiting commas appearing within the sublist.

outer macro instruction

When the mnemonic operation code for a given macro definition appears as the operation field of a model statement in another macro definition, the macro instruction referring to the containing definition is an outer macro definition.

pause (MPAUS) statement

The MPAUS statement permits the user to pause the macro processor.

positional macro instructions

Positional macro instructions are a specific type of macro instruction in which placement of the symbolic parameters in the operand field of the macro prototype statement determines placement of operands.

The MNOTE instruction generates a macro message.

sequence symbols

Sequence symbols are symbols that can appear in a statement name field to vary the statement processing sequence.

SET arithmetic variable (SETA) instruction

The SETA instruction assigns an arithmetic value to a SETA symbol or array element.

SET binary variable (SETB) instruction

The SETB instruction assigns the value true (binary 1) or false (binary 0) to a SETB variable symbol.

SET character variable (SETC) statement

The SETC statement assigns a character value to a SETC variable symbol.

SET variable statement (SETx) instruction

The SETx instruction alters the variable symbols value that the BGBLx, GBLx, or LCLx declaration statements declared as SET variable symbols.

sublist

A sublist is one or more operands separated by commas and enclosed in paired parentheses. The entire sublist, including the parentheses, is one macro instruction operand.

substring notation

Substring notation allows a part of a character value to be assigned to a SETC variable symbol indicating in the operand field of a SETx instruction the character value or an expression representing the character value to be assigned to the SETC variable symbol.

symbolic parameters

A symbolic parameter is a variable consisting of a percent sign (%) followed by from one to seven letters or numbers, the first of which must be a character. These parameters, used in the macro definition, represent the name field and operands of the corresponding macro instruction.

type attributes

Type attributes are attributes of a macro instruction operand that can be used whenever a character expression could be used; but the type attribute must occur alone (that is, not concatenated with anything) and it must not be enclosed in quotes.

unconditional branch (AGO) instruction

The AGO instruction alters the sequence in which macro definition statements are processed.

%SYSDATE macro variable symbol

The %SYSDATE symbol is an eight character string system variable whose value represents the date that the macro processor was invoked.

%SYSINDX macro system variable symbol

The %SYSINDX is a system variable whose values can be concatenated with other characters to create unique names for statements generated from the same model statement.

%SYSLIST macro system variable symbol

The %SYSLIST is a system variable that provides an alternative to symbolic parameters for referencing to positional macro instruction operands. Both %SYSLIST and symbolic parameters can be used in the same positional macro definition.

%SYSMAC macro system variable symbol

%SYSMAC is a system variable that differs from the system variable %SYSINDX because the value changes due to inner macro calls.

%SYSTIME macro system variable symbol

%SYSTIME is an eight character string system variable whose value is that time of day that the macro processor was invoked.

INDEX

A B

C D E F G H

I J K L

M N

O P Q

R

S T U V

W X Y Z

Concurrent Computer Corporation

PUBLICATION COMMENT FORM

We try to make our publications easy to understand and free of errors. Our users are an integral source of information for improving future revisions. Please use this postage paid form to send us comments, corrections, suggestions, etc.

1. Publication number _____

2. Title of publication _____

3. Describe, providing page numbers, any technical errors you found. Attach additional sheet if necessary. _____

4. Was the publication easy to understand? If no, why not? _____

5. Were illustrations adequate? _____

6. What additions or deletions would you suggest? _____

7. Other comments: _____

From _____ Date _____

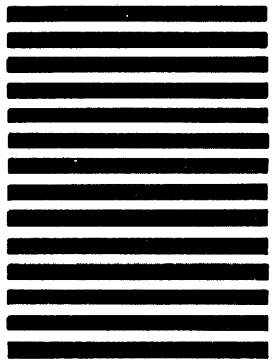
Position/Title _____

Company _____

Address _____

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 22 OCEANPORT, N.J.

POSTAGE WILL BE PAID BY ADDRESSEE

Concurrent Computer Corporation
2 Crescent Place
Oceanport, NJ 07757

ATTN:
TECHNICAL SYSTEMS PUBLICATIONS DEPT.

FOLD FOLD

STAPLE STAPLE