IBM 1130/1800 Macro Assembler Programming

This manual describes how to use the Macro Assembler of the 1130 Disk Monitor System, Version 2, and the 1800 Multiprogramming Executive Operating System, Versions 2 and 3. It should be used by programmers who have a basic knowledge of the assembler language and use this language to write programs for these systems.

This publication is supplemental to the 1130 and 1800 Assembler Language manuals and should be used in conjunction with them. This manual describes the 1130/1800 Macro Assembler: the definition and usage of the macro instruction, features of macro and assembler-language programming, and creation of a language for a specific need; the Macro Update Program; and the error messages, error flags, and warning flags of the Macro Assembler and Macro Update Program.
First Edition (June, 1970)

This edition applies to the IBM 1130 Disk Monitor System, Version 2, and the IBM 1800 Multiprogramming Executive Operating System, Versions 2 and 3, and to all subsequent versions and modifications until otherwise indicated in new editions or Technical Newsletters. Significant changes and/or additions to the specifications contained in this publication are being made from time to time; therefore, before using this publication in connection with IBM systems, consult the latest SRL Newsletter, Order Number GN20-1130 or Order Number GN20-1800, for editions that are applicable and current.

Requests for copies of IBM publications should be made to the IBM Branch Office serving your locality.

Forms are provided at the back of this publication for the reader's comments. If the forms have been removed, comments may be addressed to IBM Corporation, Programming Publications, Department D78, Monterey and Cottle Roads, San Jose, California 95114.

© Copyright International Business Machines Corporation 1970
Preface

This publication is a guide for assembler-language programmers of the 1130 Disk Monitor System, Version 2, or the 1800 Multiprogramming Executive Operating System, Version 3. It is supplemental to, and should be used in conjunction with, the 1130 and 1800 Assembler-Language manuals.

The first chapter, "Introduction," discusses the fundamentals of the Macro Assembler: what it is, how it operates, how much main storage it requires, how fast it performs, what types of macros there are, how to incorporate macros into your system; and, briefly, the Macro Update Program and error detection.

The second chapter, "The Macro Instruction," discusses how to define and use a macro instruction.

The third chapter, "Macro Assembler Features," discusses conditional assembly pseudo-operations, the ANOP, SET, and PURG pseudo-operations, automatic name generation, concatenation, optional remarks, indirect parameter substitution, the division operator, and the symbolic tag field. At the end of this chapter is a section on programming techniques. This section includes a sample program and programming tips. The sample program is for 1800 MPX but is also valid for the 1130 (an 1130 DM2 sample program is in Appendix A).

The fourth chapter, "Macro Assembler Language," describes how the Macro Assembler can be used to create a language for a specific purpose. The example is for the 1800 MPX system but is a general illustration that is also valid for 1130 users.

The fifth chapter, "The Macro Update Program," describes how you can set up and maintain your macro libraries through various statements. These statements may refer to whole libraries, macros within the libraries, or statements within the macros.

The sixth chapter, "Errors and Warnings," discusses the various error messages, error flags, and warning flags you may receive when using the Macro Assembler and the error messages you may receive when using the Macro Update Program.

Marginal notes have been included in this publication to allow easy reference to matter within the text.

The coding forms used in this manual are: for Macro Assembler statements, the 1130/1800 Assembler Coding Form, Order Number GX33-8000; for other statements, the General Purpose Card Punching Form, Order Number GX20-8030.
Required Reading

1130 publications:
Assembler Language manual, Order Number GC26-5927

1800 publications:
Assembler Language manual, Order Number GC26-5882
System Introduction, Order Number GC26-3718

Suggested Reading

1130 publications:

1800 publications:

Error Messages and Recovery Procedures manual, Order Number GC26-3727, for information on the warning flags, error codes, and error messages.
INTRODUCTION .................................................. 1
The 1130/1800 Macro Assembler ............................... 1
  The Macro Instruction ........................................ 1
  Pseudo-Operations .......................................... 3
  Nested Macro Definitions and Calls ......................... 3
  Main-Storage Requirements .................................. 3
  Macro Assembler Performance ................................ 4
Macro Update Program .......................................... 4
Error Messages, Error Flags, and Warning Flag .............. 4

THE MACRO INSTRUCTION ...................................... 5
Defining a Macro Instruction .................................. 5
  The Definition Prototype Statement ......................... 5
  An Example of Macro Definition .............................. 6
Using a Macro Instruction ..................................... 7
Substituting a Character String for a Parameter .......... 10
Continuing Calls to Additional Records ...................... 12

MACRO ASSEMBLER FEATURES ................................. 15
Conditional Assembly Pseudo-Operations ..................... 15
  AIF, AIFB Pseudo-Ops ...................................... 15
  AGO, AGOB Pseudo-Ops ..................................... 17
Unspecified Parameter Checking ............................. 17
  Special Considerations Using AIFB and AGOB ............... 18
ANOP Pseudo-Operation ....................................... 19
SET Pseudo-Operation ........................................ 20
PURG Pseudo-Operation ........................................ 22
Automatic Name Generation ................................. 22
Concatenation .................................................. 23
Optional Remarks ............................................. 25
Indirect Parameter Substitution ............................. 26
Division Operator ............................................. 27
Symbolic Tag Field .......................................... 27
Programming Techniques ...................................... 28
  Checking for Blank Parameters .............................. 28
Restrictions on AIF, AIFB, and SET Pseudo-Operations .... 29
Label and Blank Parameter Checking Using AGO .............. 30
Macro Parameter Substitution ............................... 32
Sample Program .............................................. 33

MACRO ASSEMBLER LANGUAGE ................................. 37

THE MACRO UPDATE PROGRAM ................................. 47
Initializing Disk Space ..................................... 47
Specifying the Macro Library ............................... 48
Joining Macro Libraries Physically ......................... 48
Joining Macro Libraries Logically ......................... 49
Updating a Macro in a Library .............................. 50
Renaming a Macro in a Library .............................. 51
Defining a Macro During a Macro Update Run ............... 51
  ADD Statement ............................................. 52
Deleting a Macro From a Library ............................ 53
Punching Source Statements ................................. 54
Inserting a Statement in a Macro ......................... 54
Deleting a Statement from a Macro ........................ 55
Obtaining a Listing of Macro Libraries by Statements or Macros 56
  Special Requirements on the Use of Automatic Name Generation in Nested Definitions 57
Designating Comments ....................................... 58
Terminating a Macro Update Run ............................. 58
Sequencing MUP Control Statements ......................... 59
Tables

Table 1. Error Flags .............................................. 64
Table 2. Macro Assembler Error Codes and Messages. .......................... 65
Table 3. Macro Update Program Error Messages ............................. 66
The 1130/1800 Macro Assembler

The 1130/1800 Macro Assembler allows you to condense a sequence of assembler-language coding that you use over and over again into one instruction, a macro instruction.

If you use the Macro Assembler, you can generalize a sequence of coding and then modify it slightly each time it is used. You code the sequence only once, defining macro parameters that cause the appropriate code to be generated for a particular use. The exact code generated when the macro instruction is used is based on the conditional assembly, automatic name generation, and/or parameter substitution facilities of the Macro Assembler.

The Macro Assembler also allows you to define a language that is unique to your application; such a language may be simple and/or meaningful enough to be used by a person other than a professional programmer.

THE MACRO INSTRUCTION

A macro instruction, or macro, is a source program statement. When the Macro Assembler encounters a macro, it expands the macro by processing a sequence of assembler-language statements. This sequence must have been defined in a macro definition before it can be used.

When you define a macro, you specify its operation code (macro name), its parameters, and the sequence of assembler-language statements to be processed when the Macro Assembler encounters the macro name in a source program.

Using the Macro Assembler, you can define two kinds of macros, temporary macros and stored macros. You should define every macro as a temporary macro until you are sure that it will execute properly. If it does execute properly, and you want to store it, you can include it in a macro library.

TEMPORARY MACROS

A temporary macro can be used only during the assembly of the program in which it is defined. This kind of macro isn't saved by the system; if you want to use it in another program, you have to define it again during assembly of that program. You do not have to define stored macros in order to use temporary macros.

If temporary macros are to be defined, you will need an *OVERFLOW SECTORS control statement. Two new parameters, N2 and N3, have been added to this statement. The N1 parameter remains the same. N2 is the number of sectors you allocated for the overflow of macro parameters from main storage to
disk. This parameter can be zero, and space will not be required if the overflow from a macro that is defined or called with another macro definition never exceeds 100 words. The required size of N2 may be estimated by using the following formula:

\[ \text{Number of words} = 3 + N + \sum_{i=1}^{N} \frac{1}{2} (m_i + 1) \]

N is the number of parameters and \( m_i \) is the number of characters per parameter. For example, the call \text{EXPND} \text{ALPHA,BETA,C} would be computed as:

\[ 3 + 3/2 + (5+1)/2 + (4+1)/2 + (1+1)/2 = 12 \text{ words} \]

The remainders of individual terms are ignored. N3 is the number of sectors you allocated for temporary macro definitions. You can estimate the number of sectors needed by dividing the total number of statements in all macro definitions within the assembly by 40. If you want to retain remarks, you may have to increase N3 to accommodate them. For further information on the *OVERFLOW SECTORS statement, see the 1130 Programming and Operator’s Guide, Order Number GC26-3717, or the 1800 Programmer’s Guide, Order Number GC26-3720.

STORED MACROS

A stored macro, on the other hand, resides on disk in a macro library and can be used by any program. When you assemble a program that uses or defines stored macros, you must specify which macro library you wish to use during the assembly with the *MACLIB Macro Assembler control statement.

The macro library specified on the *MACLIB statement must be defined and initialized before it can be used by the Macro Assembler. For details on how to initialize a macro library, refer to the discussion on initializing disk space in the chapter "The Macro Update Program." The *MACLIB statement enables you to access one library selectively without having to access all of the stored macros. Multiple macro libraries may be accessed in one assembly if you logically concatenate the libraries before the assembly. Additional information on macro libraries may be found in the chapter on the Macro Update Program.

You cannot define a stored macro within an assembly unless you have used an *MACLIB statement. If your program attempts to call a stored macro which cannot be found in the specified library, the macro call will be flagged as an illegal operation. The format of the *MACLIB is as follows:

```
*MACLIB LNAME
```

<table>
<thead>
<tr>
<th>1-10</th>
<th>11-20</th>
<th>21-30</th>
<th>31-40</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234567890</td>
<td>1234567890</td>
<td>1234567890</td>
<td>1234567890</td>
</tr>
</tbody>
</table>

2 1130/1800 Macro Assembler Programming
LNAME is the name of the macro library to be used in the current assembly. For further information on the *MACLIB statement, see the 1130 Programming and Operator's Guide, Order Number GC26-3717, or the 1800 Programmer's Guide, Order Number GC26-3720.

**PSEUDO-OPERATIONS**

Pseudo-operations, or pseudo-ops, extend the capabilities of the assembly process. Pseudo-ops are written like assembler-language statements, but they are used to provide information to the Macro Assembler rather than to generate executable code for the program. They can appear anywhere within an assembler-language program, providing you follow the other rules for their use.

Several new pseudo-ops have been included for use in programs that use the Macro Assembler. These are described in detail under "Features of the Macro Assembler" and perform the following functions:

- Defining the beginning and the end of a macro definition.
- Determining during an assembly whether certain statements are to be processed, based on a specific condition.
- Permitting a program label to be set to two or more different values at different points in a program without a multiple-definition error condition.
- Logically removing a macro from a library.

**NESTED MACRO DEFINITIONS AND CALLS**

A macro can be defined or called within another macro definition. This process is called nesting. The nesting of macro definitions is limited only by the physical size of your system. You may want to nest definitions to allow the dynamic definition (with decisions) of an inner macro when you are expanding the outer macro. This can also be used to advantage to conserve library space if the same code is being used in both macros (the inner one can be called only by the outer one).

The nesting of macro calls is limited to 20 calls per nest. You may wish to nest calls to macros so that values from one macro can be passed to the other macro.

**MAIN-STORAGE REQUIREMENTS**

**THE 1130**

In the 1130 Disk Monitor System, Version 2, the Macro Assembler functions as and is fully compatible with the 1130 Disk Monitor System, Version 2, Assembler. However, the Macro Assembler requires 8K words of main storage for the macro capability. The symbol table in main storage has been reduced in size to accommodate approximately 750 words for the resident macro processor. Assemblies that almost
exhaust the symbol table area of the Assembler will probably require symbol table overflow sectors for successful assembly using the Macro Assembler (see the 1130 DM2 Programming and Operator's Guide, Order Number GC26-3717). The pseudo-operations SET, ANOP, AIF, and AGO (discussed later) are also available to the 4K 1130 user.

THE 1800

The 1800 MPX Macro Assembler requires a VCORE of 5140 words. It occupies no more main storage than, and is fully compatible with, the 1800 MPX Version 1 Assembler.

MACRO ASSEMBLER PERFORMANCE

If you have a program that has been assembled successfully under the 1130 Disk Monitor Version 2 Assembler or the 1800 MPX Version 1 Assembler, the performance of the Macro Assembler will be approximately equal to that of the earlier Assemblers. Assembly time will be greater for new programs if the Macro Assembler encounters either a macro definition, a macro call, or an invalid op code. In the case of an invalid op code, the Macro Processor of the Macro Assembler will search the temporary macros and/or the specified macro library to determine whether the unrecognized op code is the name of a macro.

Macro Update Program

The Macro Update Program assists you in initializing and maintaining macro libraries. Using this program, you can add or delete macros from your library, alter those that are already in your library, physically or logically join two libraries, and perform other functions necessary to maintaining macro libraries.

Error Messages, Error Flags, and Warning Flag

During the assembly process, the Macro Assembler checks for source program errors. If an error is detected, an error flag or an error code and message are printed. If a questionable instruction is encountered, it is flagged with the warning flag (Q). At the end of each assembly, the Macro Assembler prints a message to indicate the number of errors and warnings it encountered during that assembly. Any errors in the Macro Update Program are detected by the Disk Utility Program (DUP) for the 1130 or the Disk Management Program (DMP) for the 1800.
Defining a Macro Instruction

A macro definition can appear any time after the Macro Assembler has completed processing the control statements. The definition must appear before the first call to the macro.

The first statement in a macro definition in an assembly must be the MAC (for a temporary macro) or the SMAC (for a stored macro) statement. The last statement of a macro definition must be the MEND statement. The pseudo-op names MAC, SMAC, and MEND must appear in the op code fields of the statements.

No label and no operand are required on these statements. Column 35 of the MAC and SMAC statements can be used to specify that remarks are retained so that they can be printed on listings (see "Optional Remarks"). Note that if you exercise this option, you will need additional disk space to accommodate these remarks. Comment statements (* in column 21) within the definition are always retained and listed within the expansion.

THE DEFINITION PROTOTYPE STATEMENT

The statement immediately following the MAC or SMAC statement is called the definition prototype statement. This statement contains the macro name in its op code field. The macro name may be from one to five characters long. You cannot use a period, a comma, and a left or right parenthesis in macro names. A macro name can contain embedded blanks or can consist of all blanks. An invalid macro name is flagged as an illegal op code.

If you define a stored macro with a name that is already in the library named in the *MACLIB statement, it will be flagged as an invalid macro name. A temporary macro of the same name will not be flagged; it will be expanded when the macro is called since the temporary macros are searched before the stored macros. An apostrophe should not be used in the name of a stored macro because once this macro has been stored, it cannot be modified or removed from your library.

The label and operand fields of the definition prototype statement contain the names of parameters which are supplied when the macro is used. A parameter name can be any valid assembler-language symbol; an invalid parameter name is flagged as an op code error. You may use an op code as a parameter, but it must be done carefully because substitution will occur for all uses of the op code. Parameter names in the operand field are separated by commas. Do not include a blank within a parameter name or between parameters, as a blank terminates the parameter list. If the label field is blank, it is ignored when the macro is used. The definition prototype statement must
definition

consist of one source statement; no continuation is allowed. A maximum of 20 parameter names is allowed.

The text of the macro follows the definition prototype statement and is a sequence of assembler-language instructions, calls to other macros, and/or pseudo-ops. The statements in the text may contain the parameter names specified in the definition prototype statement. During assembly, the parameters specified in the call to the macro are substituted positionally for the corresponding parameter names in the text statements (see "An Example of Macro Definition"). Whenever the Macro Assembler prints macro definition prototype statements, five-digit decimal sequence numbers are printed to the left of each statement.

obtaining a listing

The list control pseudo-operations (see the 1130 Assembler Language manual, Order Number GC26-5927, or the 1800 Assembler Language manual, Order Number GC26-5882) can be used within a macro definition to control the listing of a macro call. If you use an 1800, you may inhibit printing by turning sense switch 2 to the ON position.

parameter substitution

You can substitute a parameter into any field or subfield of a text statement in the macro definition. A parameter substituted into the operand field of a text statement may be any valid assembler-language expression. The number of characters in the parameter name has no relationship to the number of characters actually substituted, except in the case of format and tag fields. Both the parameter name and the parameter substituted must be exactly one character long in order for the parameter to be substituted correctly into the format or tag field. Note that the number of characters in a parameter on a call may also be significant. For example, an increase in the number of characters caused by the substitution of longer parameters during the expansion may cause the operand field (columns 35-71) to be exceeded. Any information beyond column 71 will be ignored.

special characters

The slash, comma, period, plus sign, minus sign, and asterisk retain their usual meaning. When a blank occurs in an operand field (except where permitted by the assembler language), the rest of the operand field is ignored. Special characters used in Macro Assembler statements must conform to the character code summaries as listed in the 1130 Assembler Language manual, Order Number GC26-5927, or the 1800 Assembler Language manual, Order Number GC26-5882.

AN EXAMPLE OF MACRO DEFINITION

The temporary macro SUM is completely defined by the following sequence. A definition of the macro SUM is shown below and the call and statements generated by the call are shown in the next section.
Using a Macro Instruction

After a macro has been defined, you call it by using its name as an op code and specifying in the label and/or operand field the parameters to be substituted for the parameter names in the definition prototype statement. The parameter names must follow the order outlined under "The Definition Prototype Statement." If a parameter name is omitted (two commas in a row or a trailing comma), it is ignored when the macro is used.

When the Macro Assembler encounters a macro instruction, it processes the statements in the macro definition text with the parameters you have specified for substitution.

The assembled instructions are listed along with the macro call in the assembly listing. Any statement within a program that is a result of a macro expansion is flagged with a plus sign to the left of the label field of the Macro Assembler listing.

A sample definition of the macro SUM was illustrated in the previous section. Below is an example of a call to that macro, and the code generated by that call.
Parameters specified in the call to SUM are substituted for the dummy parameters of the SUM definition prototype statement. LOOP is substituted for NAME, 2 for X, 10 for COUNT, FROM for LIST, and TEMP for STOR. There's nothing special about the parameter in the label field of the call; it's substituted just like any other parameter. In fact, it need not even be used as a label in a macro definition; it can be used as an op code or an operand. For example, the macro DIVBY could be defined as follows (note that a while A appears in the label field of the definition prototype statement, it is used as an operand in the text of the macro):

When you call a macro, you may want to leave out one or more of the parameters. If the parameter to be left out would ordinarily be in the label field of the call, just leave it out. If the parameter would ordinarily appear followed by a comma in the operand field, leave out the character string to be substituted, but include the comma that would ordinarily follow the parameter. If the parameter ordinarily comes at the end of the operand field, you may leave it out. If you leave out one or more parameters that ordinarily come at the end, you may leave out all the commas that immediately precede them. If you anticipate omitting parameters that will be parts of AIF or AIFB expressions, refer to the section "Unspecified Parameter Checking" for restrictions on this operation.

The following examples illustrate the omission of parameters.
The macro DCS generates DC statements. For example:

```
M A C
D C S
D C
D C
D C
M E N D
```

would generate

```
D C S
D C
D C
D C
```

Omitting parameter B in a call

```
D C S
D C
D C
```

would cause

```
D C
D C
D C
```

to be generated. Note that the operand of the second DC statement is blank because its associated parameter was omitted.

You should be extremely careful when omitting parameters because nothing is substituted for the parameter that is missing. Consider the following example; in the first call, the parameter is not missing and the code is generated properly. In the second call, the second parameter is missing and the generated code is in error.
Substituting a Character String for a Parameter

You can substitute a character string containing embedded blanks and special characters (such as commas, periods, or slashes) for a macro parameter name by enclosing the string in parentheses. This makes it easy to pass a set of parameters to a nested macro call. However, you must be careful in passing character strings to ensure that parameters do not exceed the record length, as any information beyond column 71 will be ignored.

Assume in the example given below that SEE is a previously defined macro with a maximum of three parameters specified in the operand field. If another macro SCAN calls SEE, parameters may be passed to SEE as shown:
Without this facility the same SCAN macro would have to be defined as follows in order to generate the above statement sequence. Notice that this second method may be restrictive in passing parameters, because a definition prototype statement may have no more than 20 parameters.

You can also use character-string substitution to generate messages. An example of this is given below.

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>F</th>
<th>T</th>
<th>Operands &amp; Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>'SCAN'</td>
<td>MACRO, DEFINITION</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MAC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCAN</td>
<td>D, H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEE</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MEND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>'SCAN'</td>
<td>TABLE, (VAL1, VAL2, VAL3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>TABLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEE</td>
<td>VAL1, VAL2, VAL3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Macro Instruction 11
Continuing Calls to Additional Records

A call to a macro may be too long to fit on one record, so the Macro Assembler allows calls to be continued onto one or more additional records. When you call a macro, you indicate that there is another record in the call by placing any character other than a blank in the format field (column 32). You can then continue the macro call beginning in column 35 of the next record. The first 31 columns of the next record are ignored; column 32, if used, specifies continuation of the call onto still another record.

Note that on continued calls, you must be careful in substituting parameters that have more characters than those in the original definition, because you may cause the operand field to be exceeded on either the CALL record or any subsequent continuation records. Therefore, some of your values may be lost (see "Macro Parameter Substitution" in the section on programming techniques).

When a macro call is continued onto an additional record, the Macro Assembler will stop its parameter scan of the first record when either there is a blank (not within parentheses) in the operand field or the operand extends to and includes column 71. The next character in your parameter list is assumed to be in column 35 of the next record.

Using parentheses and continuation records, you can replace any number of consecutive fields in a macro definition statement. Look at this example:
Here's a call to the ONION macro:

```
Label Operation F T 35 40 45 50 55
21 25 27 30 32 33
1...1 MAC
1...1 ONI ON
A...1 LD
B...1
1...1 S.TO
1...1 M. E. N. D
```

A punch other than a blank (the S in the above example) in the format field means that the macro call is continued on a second record.

The following code is generated from the above macro call.

```
Label Operation F T 35 40 45 50 55
21 25 27 30 32 33
NAME LD ADDR LD XYZ
S.R.T.
S.TO.
```

In the macro call, the instruction NAME LD ADDR replaces the label. The remaining fields, LD XYZ, are displaced to the right. The Macro Assembler treats these fields as remarks because in this example there are two blanks after ADDR.
Conditional Assembly Pseudo-Operations

You may want different calls to the same macro to produce different lines of assembled code, depending on some condition to be examined during the assembly. Conditional assembly pseudo-ops allow you to do this. These pseudo-operations do not generate any executable code and do not modify the address counter.

Applications which require slight code modifications to a general technique need be coded only once using conditional assembly pseudo-ops within macro calls. This saves time for the programmer.

AIF, AIFB PSEUDO-OPS

Two conditional-assembly pseudo-ops, the "assemble if" and "assemble if back" pseudo-ops AIF and AIFB, have the following format:

- An optional label.
- The op code (AIF, AIFB).
- In the operand field, a left parenthesis, an expression, one or more blanks, a condition, one or more blanks, another expression, a right parenthesis, a comma, and a name.

The two expressions can be any valid assembly expressions. The name should be a valid assembler-language symbol or may be left blank. It may also be any combination of from one to five characters if this combination is used in the label field of one of these pseudo-ops: AIF, AIFB, AGO, AGOB, ANOP, PURG, LIST, EJCT, HDNG, MEND, END, or SPAC. All symbols used within AIF or AIFB statement expressions must have been predefined or the statement will be flagged with a U (undefined symbol). If the name is left blank, the statement will be flagged with a warning flag (Q).

The condition must be one of the following:

- EQ—Equal to
- GT—Greater than
- LT—Less than
- NE—Not equal to
- GE—Greater than or equal to
- LE—Less than or equal to

During assembly, the condition statement between the parentheses is evaluated. If it is true, the AIF statement causes all the following statements to be skipped (and not processed) until the Macro Assembler finds a statement with a label corresponding to the symbol specified in the AIF statement. If the statement between the parentheses is false, the assembly continues with the statement immediately.
following the AIF statement. The AIF statement may be used anywhere in a assembler-language program.

The AIFB (AIF back) statement functions as the AIF statement, except that the Macro Assembler returns to the beginning of the current (innermost within a nest) macro definition being expanded (called) before searching for a label. Unlike the AIF statement, the AIFB statement may occur within a macro definition only; it is flagged as an illegal op code if it appears outside of a macro definition. If the search is unsuccessful, the MEND statement will terminate the search and the expansion of that macro.

If the name subfield of the AIF or AIFB operand is left blank and the label search is to be performed, statements are skipped until the first statement with no label is encountered, at which time assembly continues. In any case, when a label search is performed, the search can continue until an END statement is encountered; the END statement will be processed and flagged with a Q. If the AIF statement is in a definition, a MEND statement will terminate the search.

Let's look at an example of the use of AIF.

The assembly of this code depends on the values of X+1 and Y when the Macro Assembler evaluates them. If they are not equal, all the instructions shown above are processed. If they are equal, only the following three instructions are assembled.

The assembly of this code depends on the values of X+1 and Y when the Macro Assembler evaluates them. If they are not equal, all the instructions shown above are processed. If they are equal, only the following three instructions are assembled.

16 1130/1800 Macro Assembler Programming
Notice that the second AIF has a blank name field. As a result, the first instruction following this statement with a blank label field is assembled.

**AGO, AGOB PSEUDO-OPS**

Two other instructions are used along with AIF and AIFB to effect conditional assembly. These pseudo-ops, AGO and AGOB, cause unconditional branching and have the following format:

- An optional label.
- The op code (AGO, AGOB).
- A valid assembler-language symbol or five blanks in the operand field. If the name field is left blank, the statement will be flagged with a warning flag (Q).

The name should be a valid assembler-language symbol, or may be left blank. It may also be any combination of from one to five characters if this combination is used in the label field of one of these pseudo-ops: AIF, AIFB, AGO, AGOB, ANOP, PURG, LIST, EJCT, HDNG, MEND, END, or SPAC.

The AGO statement causes the Macro Assembler to skip (and not process) statements following the AGO statement until it encounters a statement with a label corresponding to the symbol specified in the AGO statement. See the ANOP section of this manual for an example of the use of the AGO instruction. The AGO statement may be used anywhere in an assembler-language program.

The AGOB (AGO back) statement functions as the AGO statement, except that the Macro Assembler returns to the beginning of the current (innermost within a nest) macro definition being expanded before performing the label search. If the search is unsuccessful, the MEND statement will terminate the search. Unlike the AGO statement, the AGOB statement may occur within a macro definition only; it is flagged as an illegal op code if it appears outside of a macro definition.

If the operand of the AGO or AGOB statement is left blank and the label search is to be performed, statements are skipped until the first statement with no label is encountered. In any case, when a label search is performed, the search can continue until an END statement is encountered. Like the AGOB statement, if the AGO statement is in a definition, a MEND statement will terminate the search.

**UNSPECIFIED PARAMETER CHECKING**

The name searching technique used by the AIF and AGO pseudo-ops may be utilized in checking for unspecified parameters.

Assume that the COUNT parameter on the following prototype statement is a count of how many data words are to be moved from one area to another.
If COUNT is not specified in a call to MOVE, the name search prompted by the AGO COUNT statement will be terminated on the ANOP statement that follows immediately, because a blank was substituted for COUNT and the ANOP has a blank label field.

If COUNT is specified, the COUNT that is a label on an ANOP statement will be replaced with the COUNT specified in the call. Thus, the name search prompted by the AGO COUNT statement will terminate on the ANOP statement that has COUNT as a label.

SPECIAL CONSIDERATIONS USING AIFB AND AGOB

Note that if the AIFB or the AGOB causes a second assembly of the same code, multiple label definition errors may occur. It is your responsibility to ensure that the label to be skipped to is either unique or not entered in the symbol table, that is, a label on an AIF, AIFB, AGO, AGOB, SPAC, EJCT, HDNG, LIST, MEND, END, PURG, or ANOP statement. Also note that with the capability of the AIFB and AGOB, you can put the Macro Assembler into a loop. This will occur if the conditions never get changed, thus causing the Macro Assembler to loop between the AIFB and AGOB statements. The call below will cause the AIFB expression to be evaluated always as true (8 LE 20) because the AIFB to A will cause a branch to the first statement labeled A within the macro. Thus, the Macro Assembler will loop interminably between "A SET X" and the AIFB statement.

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>F</th>
<th>T</th>
<th>FF</th>
<th>TT</th>
<th>FF</th>
<th>TT</th>
<th>FF</th>
<th>TT</th>
<th>FF</th>
<th>TT</th>
<th>FF</th>
<th>TT</th>
<th>FF</th>
<th>TT</th>
<th>FF</th>
<th>TT</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>MAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>MOVE FROM,T0,COUNT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>AG0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>COUNT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>SET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>AG0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>B,EGIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COUNT</td>
<td>ANOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>SET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B,EGIN</td>
<td>ANOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MEND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

18 1130/1800 Macro Assembler Programming
ANOP Pseudo-Operation

The purpose of the ANOP pseudo-op is to provide a label which an AIF, AGO, AIFB, or AGOB can reference to resume assembling. Assembling an ANOP label has the same effect as assembling the instruction immediately following it. The label on an ANOP is not placed in the symbol table, so statements other than AIF, AGO, AIFB, and AGOB can't use it as a reference. This is also true of other labels as discussed previously under "Special Considerations Using AIFB and AGOB."

The format of the ANOP statement is:

- A label.
- The op code ANOP.

The ANOP pseudo-operation allows you to associate temporary and permanent labels with the same instruction. Thus, the temporary label can be used to clarify a conditional assembly sequence while the permanent label can be used to clarify the instruction sequence.

The following is an example of a way in which ANOP might be used. In this example, A is assumed to have been defined prior to the AIF statement.
When A is less than 0, the generated code is:

\[
\begin{align*}
\text{SET} & 01 \\
L & D \\
S & T \ C h
\end{align*}
\]

When A is greater than or equal to 0, the generated code is:

\[
\begin{align*}
\text{SET} & 01 \\
L & D \\
S & T \ C h
\end{align*}
\]

ANOP is useful when you're using the SET pseudo-op. An example of this usage is given in the "SET Pseudo-Operation" section.

SET Pseudo-Operation

SET allows you to assign a value to a symbol and, later in the assembly, assign another value to the same symbol without a multiple label definition error resulting. The symbol retains the value of the last SET statement associated with it from the first pass of the Macro Assembler until the Macro Assembler encounters an associated SET in the second pass. You can't use the EQU statement this way because the EQU statement is not processed on the second pass of the Macro Assembler and, consequently, cannot be used to change the value of a symbol during the assembly.

The format of the SET statement is:

- A label.
- The op code SET.
- A valid assembler-language expression in the operand field.

The label is set equal to the value of the expression in the operand field. Any symbols used within the expression on a SET statement must have been predefined, or the statement will be flagged with a U (undefined symbol).
Here's an example of the use of SET. Suppose A is the starting address of some data to be sent to disk, and B is the address of the end of the data. Assume we know the data will take up no more than two sectors and we want to set SECT equal to the number of sectors. The Macro Assembler automatically calculates the value of SECT in the following statement sequence.

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>F T</th>
<th>Operands &amp; Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>25</td>
<td>27</td>
<td>B-A</td>
</tr>
<tr>
<td></td>
<td>EQU</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>25</td>
<td>27</td>
<td>N-320</td>
</tr>
<tr>
<td></td>
<td>SET</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>25</td>
<td>27</td>
<td>(K,L.E.,0).ONE</td>
</tr>
<tr>
<td></td>
<td>AIF</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>25</td>
<td>27</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>SET</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>25</td>
<td>27</td>
<td>0.K</td>
</tr>
<tr>
<td></td>
<td>AGO</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>25</td>
<td>27</td>
<td>ONE</td>
</tr>
<tr>
<td></td>
<td>ANOP</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>25</td>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SET</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>25</td>
<td>27</td>
<td>(next instruction)</td>
</tr>
</tbody>
</table>

In the above example, N, the difference between B and A, is compared to 320 by the AIF instruction. If the difference is greater than 320, the first SET following the AIF statement sets SECT to 2. The AGO then causes the assembly to continue around the next SET and the assembly proceeds.

If the difference (N) is less than or equal to 320, AIF causes the assembly to continue at ONE. This is equivalent to a continuation at the second SET following the AIF statement, since ONE is an ANOP instruction. Notice that it is impossible to branch directly to the correct SET instruction, since two SET instructions in the sequence contain the label SECT. If the AIF statement specified a branch to SECT, the Macro Assembler would continue processing with the next statement having SECT in its label field—in this case, the wrong instruction.

Here's another example of a macro that uses SET.
Given variables A and B, the TAB macro defines a constant equal to A times B. It next defines a constant equal to this product times B. It continues this way until the result reaches a specified value, C. Note that if A and B are equal, TAB builds a table of powers of B.

PURG Pseudo-Operation

The PURG pseudo-operation removes the specified macro name from the macro library associated with the assembly by the *MACLIB control statement. PURG causes operations to occur only on the library associated with the *MACLIB control statement; it does not affect any other library even if it has been concatenated to the associated library (see MUP section on "Joining Macro Libraries Logically").

You can then define another macro with the same name, but the space occupied by the purged macro isn't available for reuse until the next DMP/DUP macro update job is performed on the library (see "Macro Update Program"). The space is reclaimed by any macro update function run on that library.

The format of the PURG statement is:

- Optional label (can be used as a target for pseudo-ops).
- Op code PURG.
- Macro name in the operand field.

The macro name must be enclosed in apostrophes (the first apostrophe must be in column 35). If the macro name is not properly formatted, is missing, or cannot be found, the PURG statement will be flagged with a warning flag (Q) and the PURG operation will not be performed.

Automatic Name Generation

If your macro definition contains a label that isn't a parameter of the macro, and if you call the macro more than once in a given assembly, you'll get a multiple-definition error for that label. You can get around this problem by making all labels used in macros parameters of the macros, but then you have to supply all the labels every time you call a macro. An easier method to use is automatic name generation.

Instead of writing out a complete label in the macro definition, you write from one to four alphameric characters, the first of which must be alphabetic, followed by an apostrophe. Each time a macro is called in a given assembly, the Macro Assembler replaces the apostrophe with a number—a different number each time.

The number of digits added to your label always causes a five character label to be generated. For example, repeated uses of the label P' result in the labels P0001, P0002, P0003, and so on. Repeated uses of the label PAM' result in PAM01, PAM02, PAM03, and so on.

The digits inserted into your label by the above method are determined by a counter maintained by the Macro Assembler. When the Macro Assembler encounters a macro that utilizes the automatic name generation feature, the counter is
incremented by one and remains at that value until the expansion of that macro is complete. This causes a problem when other macro calls are included in the first macro (that is, the nesting of macros) and one of them is called twice. MAC3 in the diagram at the left is called twice within the nest. If MAC3 contains labels in the form of letters followed by an apostrophe, the same set of labels will be generated twice because the counter is not incremented until MAC1 is completed.

The way around this problem is to place an apostrophe before the characters in the label. The Macro Assembler replaces this apostrophe with a different alphabetic character (beginning with A) each time a macro is called within a given nest. Thus, if the label 'RAB' is used in a macro and the macro is called four times in the same nest, the labels generated are ARAB, BRAB, CRAB, and DRAB.

You can use apostrophes at both the beginning and the end of a label; in fact, this is the simplest way of ensuring that you don't get multiply-defined labels. The label 'C' can result in labels from AC001 through TC999 (T rather than Z because only 20 levels of macro nesting are allowed). The label 'SPY' can result in labels from ASPY1 through TSPY9.

Note that if you alternately call macros with automatic name generation, the numbers of the generated labels will also alternate. For example, if you call two macros alternately and the first has a label C' and the second has a label D', the resulting labels would be C0001, D0002, C0003, D0004, and so on.

The Macro Assembler also automatically generates labels in the remarks field (not on comment records) of assembler-language statements. For this reason, you must be careful when using apostrophes within a remark.

Concatenation

By concatenating two parts of a field, you can join a parameter to a character string, two parameters, or two character strings. You can use either a period or an ampersand as a concatenator. The ampersand functions as a concatenator only if it appears as the first character in a parameter of a definition prototype statement. Otherwise, it functions as a plus operator. If you use the ampersand as a concatenator, you may use six characters, including the ampersand, for the parameter name.

The following example shows several uses of period concatenation.
Using the above definition with the macro calls below, the code following the calls would be generated.

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>F</th>
<th>T</th>
<th>Operands &amp; Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>'MOVE'</td>
<td>0</td>
<td>2</td>
<td>MACRO, CALL</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>'FIRS'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>'FIRS'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>'ABLE'</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>'ABLE'</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notice that in the above examples, parameter substitution does not occur for A and B in the names A.B.C and B.B.C. You need to be careful when you are using any op code as a parameter, because substitution will occur for all uses of that op code.

The following examples show several ways in which the ampersand can be used as a concatenator. Notice in the second example that the B in the macro definition is not preceded by an ampersand and, therefore, the B is a parameter, but the &B is not.
Because a period is used frequently as a decimal point when writing DEC and XFLC statements, the Macro Assembler inhibits period concatenation when writing these statements. The ampersand concatenation feature may still be used.

### Optional Remarks

When you define a macro, you specify whether or not you want remarks on the macro definition statements to be reproduced each time the macro is expanded. If you want to keep the remarks, place any nonblank character in column 35 of the MAC or SMAC statement. The disk space required for the macro is increased according to the amount of space required for your remarks. Comment statements within the definition are always retained and listed within the expansion.

If parameters occur in remarks, parameter substitution is performed for the remarks also. This substitution includes automatically-generated names. The Macro Assembler also substitutes parameters into comments statements (asterisk in column 21), but it does not substitute automatically-generated names into comments statements. On such statements, the apostrophe is treated like any other character.

All records are truncated following column 71 of the record, and no error indication is given.

---

**Macro Assembler Features 25**
Indirect Parameter Substitution

The indirect parameter substitution feature allows you to substitute one parameter for another when a macro is expanded. You do this by specifying a semicolon followed by any valid assembler-language expression, instead of the parameter you wish to replace, in the macro call.

The value of this expression is evaluated by the Macro Assembler and is considered to be the position of the replacement parameter in this macro call. For example, if the expression is evaluated as 3, the parameter in the third position is the replacement parameter. Remember, in determining the position of the replacement parameter, the label field of the call is the first parameter.

The position number and semicolon are counted as one parameter. If the replacement parameter position referenced is not specified in the macro call, an empty parameter is substituted. An indirectly-specified parameter may select as many as 19 other indirectly-specified parameters. If this limit is exceeded, an empty parameter is substituted, and the macro call is flagged with a syntax error indicator. Processing continues with the next statement.

If a symbol within a parameter substitution specification is not defined before its use, the substituted expression is evaluated as zero, the referenced parameter is evaluated as a blank parameter, and processing continues.

The following examples demonstrate the use of indirect parameter substitution.

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>F</th>
<th>T</th>
<th>Operands &amp; Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MAC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LO, C</td>
<td>BRT, B</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LO, C</td>
<td>BSC, L</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MEND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BRT, B</td>
<td>; A, LOC1, LOC2, LOC3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BSC, L</td>
<td>LOC2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LOC3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In a nested macro call, if the symbol following the semicolon is a parameter of the outer macro, the parameter must be concatenated to the semicolon for recognition by the Macro Assembler. (Concatenation was described previously under "Concatenation").
The Macro Assembler interprets any slash in the operand field as a division operator, unless it can be interpreted as a hexadecimal number indicator. A hexadecimal constant is indicated by a slash in column 35. A slash preceded by an operator is interpreted as the hexadecimal indicator. A slash preceded by a term is interpreted as a division operator. A division operator may be immediately followed by a + or - to indicate whether the divisor is positive or negative. If no + or - follows the division operator, the divisor is assumed to be positive. A division operator followed immediately by a multiplication operator is flagged as a syntax error (S). Division by the internal address register (IAR) is allowed in an absolute assembly; in a relocatable assembly, it is flagged as a relocation error (R).

Each division operation within each term is performed from left to right. The 16-bit quotient is the result of a division operation; the remainder is lost.

The Macro Assembler performs all operations in an expression algebraically. For example:

(1) 3+2*4/2 = 7  (2) 5*2/3+9/-3 = 0

In example 1 the entire term 2*4/2 is evaluated from left to right before it is combined with 3. In example 2 the term 5*2/3 is evaluated (left to right) first. Then the term 9/-3 is evaluated and combined with the first term. Note that since the result of a division is always an integer with the remainder ignored, the first term in example 2 if written as 2/3*5 would be evaluated as 0 but 3 would result when the term is written as 5*2/3.

Note that division by zero results in a zero quotient and a warning flag. A relocation error flag (R) will be issued if either the dividend or divisor is relocatable. If two consecutive division operators are found in a single term, the term will be replaced by zero. For example, 27/9/3 will not be correctly evaluated and will be replaced by zero; the statement in which it occurred will be flagged as a syntax error (S).

Symbolic Tag Field

If you wish to change an index register designation once you have coded that portion of your program, you can do this by using the symbolic tag field feature. You specify the tag field with a one-character symbol which is defined in the assembly by means of an EQU or SET statement (see the 1130 Assembler Language manual, Order Number GC26-5927 or the 1800 Assembler Language manual, Order Number GC26-5882).

You may change the value of the tag symbol dynamically when using the SET pseudo-operation to define the tag field. The tag symbol retains the value of the last SET statement associated with it from the first pass of the Macro Assembler until the Macro Assembler encounters an associated SET statement in the second pass. The EQU statement is not read on the second pass of the Macro Assembler and,
consequently, cannot be used to change the value of the tag symbol during assembly.

The following example illustrates use of the symbolic tag field in instructions and the code generated by those instructions.

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>F</th>
<th>T</th>
<th>Operands &amp; Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>SET</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOP1</td>
<td>LD</td>
<td>LA</td>
<td>TABL1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ST.</td>
<td>LA</td>
<td>TABL2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MD.</td>
<td>A</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SET</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Programming Techniques

The following items should help you in using the Macro Assembler.

**CHECKING FOR BLANK PARAMETERS**

It is generally desirable to simplify macro calls by defining macros so that parameters (preferably the last parameter in the call) may be optionally omitted. Passing a blank parameter to the macro can cause a special sequence of code or no code to be generated. This blank parameter in the call causes no substitution to occur for the parameter when the macro is expanded. Consider the following examples:
The SET statement is used in Example 2 to avoid Macro Assembler errors that can result from passing blank parameters. Note the use of $ in the label field. Use of $, a, and # in macro labels may help prevent conflict with other labels used in the program. These characters may also be used in conjunction with the automatic name generation feature.

**RESTRICTIONS ON AIF, AIFB, AND SET PSEUDO-OPERATIONS**

We have said that symbols used either in the SET operand field or in the AIF or AIFB expression must have been defined before the referencing SET, AIF, or AIFB statement is processed. Since symbols used as above in AIF, AIFB, and SET are evaluated at the time they are first encountered, they are flagged with the U error even if the symbols are defined later in the program. This error flagging has been implemented to help ensure that code generated by SET, AIF,
or AIFB statements is the code intended by the user. See Example 3.

// JOB 01 JAN 70 00.517 HRS
// * EXAMPLE 3 ILLUSTRATES THE RESTRICTION ON THE AIF OPERATION
// * THAT A SYMBOL MUST BE DEFINED PRIOR TO ITS USE IN
// * THE SET, AIF, OR AIFB OPERATIONS
// ASM SAMPL 01 JAN 70 00.517 HRS
*LIST
*OVERFLOW SECTORS **2
MAC  C  BEGIN MACRO DEFINITION
TABLE  MAC  C  AIFP  (A  LE  0),EXIT  EXIT  MACRO  IF  A  LE  0
00001  CONT  ANOP
00002  DC  /00F0+A  CONSTANT  DEFINED
00003  A  SET  A+1  INCR  A
00004  AIFB  (A  LE  /F),CONT  END  TABLE  IF  A  GT  15
00005  EXIT  MEND  END  MACRO  DEFINITION
*  IF  A  IS  NOT  DEFINED  PRIOR  TO  THE  CALL
*  TO  TABLE,  NO  TABLE  WILL  BE  GENERATED
*  SINCE  A  IS  EVALUATED  AS  ZERO

0000 0 1000  BEGIN  NOP  TABLE  MACRO  CALL
  MAC  C  U+  AIF  (A  LE  0),EXIT  EXIT  MACRO  IF  A  LE  0
0001 30 059C98C0  EXIT
0004 0000  END  BEGIN

001 ERROR(S) AND 000 WARNING(S) IN ABOVE ASSEMBLY.

LABEL AND BLANK PARAMETER CHECKING USING AGO

The Macro Assembler does not have an explicit method for character string comparison. However, the AGO and ANOP pseudo-operations provide a means of label checking both within and outside of a macro. Consider Example 4.

// JOB 01 JAN 70 00.525 HRS
// * EXAMPLE 4 ILLUSTRATES THE USE OF THE AGO AND ANOP OPERATIONS
// * TO HANDLE BLANK MACRO PARAMETERS WITHOUT CAUSING ASSEMBLER
// * ERRORS TO OCCUR
// DUP 01 JAN 70 00.525 HRS
*DELET  D  MACRO  *****
*DFILE  MACRO  0007
WILL  RESERVE  AT  SCTR  ADDR  0380
DMP  FUNCTION  COMPLETED
*MACRO  UPDATE
BUILD 'MACRO'
  0380 0005  **  LIBRARY  END  **
0000
ENDUP
UPDATE  COMPLETED
// ASM SAMPL 01 JAN 70 00.531 HRS
*MACLIB  MACRO
*LIST
SMAC  C  BEGIN MACRO DEFINITION
LABEL DISK  FUN1,ARE1,ERR1
00001  LABEL LIBF  DISKN  CALL TO DISKN SUBR
00002  AIF  (FUN1 EQ 0),READ  TEST  FOR  READ  FUNC
00003  AIF  (FUN1 EQ 3),WRITE  TEST  FOR  WRITE  FUNC
00004  AIF  (FUN1 EQ 0),TEST  TEST  FOR  TEST  FUNC
00005  LIST  ON
00006  =  ILLEGAL  REQUEST
00007  LIST  AG0  EXIT  EXIT  MACRO
0008  READ  ANOP

30 1130/1800 Macro Assembler Programming
Macro Assembler Features 31
Note that if a macro parameter, which may be a character string greater than one character, is used in an AIF statement to check for character values, a syntax flag will be generated by such a statement and the value of the substituted expression will be zero. See Example 5.

MACRO PARAMETER SUBSTITUTION

When special features such as call continuation and indirect parameter substitution are used in nested macro calls you must account for such special instances as you code your macro. If, for example, you anticipate that a nested macro call may be a continued call, you must pass the continuation indicator and ensure that any parameter string from the first call is completed on each nested continuation record. See Example 6. If you wish to pass a symbolic value to a nested macro call and an indirect substitution expression and that symbol appears next to the semicolon, you must concatenate the symbol to the semicolon. See Example 7.
SAMPLE PROGRAM

The following sample program illustrates three macros: their definitions, calls to them, and the code generated.
**a move macro**

```assembly
macro definition
MOVE FROm,TO,CNT
LABEL MOVE FROM,TO,CNT
LIST OFF
00002 AIF (CNT LE 0),NDMVE
00003 LIST OFF
00004 LABEL LDX L1 -CNT INITIALIZE LOOP COUNTER
00005 LOUP LD L1 FROM+CNT GET WD TO BE MOVED
00006 STD L1 TO+CNT MOVE IT
00007 MDX 1 1 RUMP MOVE (LOOP) COUNTER
00008 MDX LOOP LOOP UNTIL MOVE COMPLETED
00009 NDMVE LIST OFF
MEND
```

**a message macro**

```assembly
macro definition
MSG TEXT
MAC MSG TEXT
00001 ONE S TEXT 'E
MEND
```

**a parameter checking macro**

```assembly
macro definition
VKUS PARAM
00001 AGO PARAM
00002 ANOP
00003 #VKUS SET 0
00004 AGO QUIT
00005 PARAM ANOP
00006 #VKUS SET 1
QUIT MEND
```

**DC-generating macros**

```assembly
macro call
00001 LIST OFF
00002 VKUS LABEL (VKUS EQ 0),BLANK
00003 AIF
00004 LIST OFF
00005 LABEL EQU *
00006 BLANK LIST OFF
00007 Z SET CNT
00008 X SET 3
00009 RETRN AIF (Z EQ 0),QUIT
00011 X SET XI
00012 Z SET Z-1
00013 AGOB RETRN
00014 QUIT LIST OFF
MEND
```
The 1 indicates the overflow sector necessary to house temporary macro definitions in this assembly.

If CNT is less than or equal to 0, the definition is not expanded.

Sequence numbers of definition statements are printed for easier visual perception.

The move loop is generated with your program's parameters.

This AlF checks the call to the DCS macro to determine if you want a label to be associated with the first DC.

You can significantly alter a printed listing when a macro is called through use of the LIST pseudo-op.

All statements resulting from a macro expansion are flagged with a plus in column 20.

This is a call to DCS without the label specified.

Note how use of the LIST pseudo-op caused this expansion to be printed.

This is a call to DCS with the label specified.

NOTES

By using parentheses, the parameter TEXT (see the MSG definition) is replaced by a string of characters into a DMES statement.
The features of the Macro Assembler permit you to create a language that can help programmers who are new to your installation or who don't understand the more detailed aspects of certain operations. This specialized language may also be used by programmers who know the operation and want to simplify their job or who need to interact with persons other than programmers. Such a specialized language may also be used to help others learn to write simple programs for your system.

Consider the following 1800 MPX example.

Mr. Jones is a programmer who works for a sports information service center and is in charge of all baseball statistics. He has been assigned the following project in response to a customer requisition.

<table>
<thead>
<tr>
<th>SPORTS INFORMATION SERVICE CENTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>programmer: A.B. Jones</td>
</tr>
<tr>
<td>date needed: September 10, 1972</td>
</tr>
<tr>
<td>customer: 126-3381-07</td>
</tr>
</tbody>
</table>

Problem:
1. Read a batch of data cards having the following format:
   columns 1–20 Player's name
   25–28 Number of at-bats
   30–33 Number of singles
   35–38 Number of doubles
   40–43 Number of triples
   45–48 Number of home-runs
2. Compute the player's batting average, slugging percentage, and home-run ratio.
3. Print all the information mentioned in 1 and 2.
4. Terminate execution of the program when a data card containing an asterisk (*) in column 1 is read.

Mr. Jones knows that in order to answer this request, he will have to use some macros from the general purpose library, use some macros from his own library, and use the following computations in his solution.

Batting average=number of hits number of times at bat.
For example, if hits=13 and at-bats=64, 13 ÷ 64=.203
The batting average would be .203.

Slugging percentage=(number of singles + 2 x number
of doubles + 3 x number of triples + 4 x number
of home runs) ÷ number of at-bats.
For example, if at-bats=64, singles=8, doubles=3,
triples=1, and home runs=1, the slugging percentage
would be [8+2(3)+3(1)+4(1)] ÷ 64 or 21 ÷ 64 or .328
Home run ratio = number of at-bats ÷ home runs.
For example, if at-bats = 64 and home runs = 4, the home run ratio would be 64 ÷ 4 or 16.

Mr. Jones then looks in the guide to general purpose macros and determines which ones he wants to use. The guide lists the macros in a library that have been built by all the programmers who work with Mr. Jones. These macros are ones that are used frequently by the programmers, such as, read a card and print a line.

The following listing demonstrates how this library of macros was created. In this case the library has been named SYSTM.

```plaintext
// JOB 00 JAN 00 00:235 IRS
// DMP 00 JAN 00 00:235 IRS
*MACRO UPDATE
DMPP FUNCTION COMPLETED
WILL RESERVE AT SCTA ADDR 04CO
DMP FUNCTION COMPLETED
BUILD 'SYSTM',
NAME LABEL, INPUT, OUTPUT
ADD 'DCBIN'
04CO 0005 DCBIN
X 00001 LABEL CALL DCBIN
X 00002 DC INPUT
X 00003 DC OUTPUT
00004 ** MACRO END **
ADD 'BINDC'
04CO 0019 BINDC
X 00001 LABEL CALL BINDC
X 00002 DC INPUT
X 00003 DC OUTPUT
00004 ** MACRO END **
NAME LABEL, CNTL, AREA, WDCNT
ADD 'LLIST'
04CO 0020 LLIST
X 00001 LABEL DC 0
X 00002 DC 0
X 00003 BSS 4
X 00004 DC CNTL
X 00005 DC AREA
X 00006 DC WDCNT
X 00008 BSS WDCNT
00009 ** MACRO END **
NAME LABEL, LIST, CNTL, AREA, WDCNT
ADD 'RCRD'
04CO 0034 RCRD
X 00001 LABEL CALL CARON
X 00002 DC LIST
X 00003 LD L LIST
X 00004 BSS L **-4,2
00005 ** MACRO END **
ADD 'PRINT'
04CO 006F PRINT
X 00001 LABEL CALL PRNTN
X 00002 DC LIST
X 00003 LD L LIST
X 00004 BSS L **-4,2
00005 ** MACRO END **
NAME LABEL, INPUT, OUTPUT, CHRCT
ADD 'HOLPR'
04CO 0094 HOLPR
X 00001 LABEL CALL HOLPR
X 00002 DC A+1
X 00003 DC INPUT
X 00004 DC OUTPUT
X 00005 DC A+1
X 00006 DC AREA
X 00007 A+2 DC CNTL
X 00008 DC CHRCT
00009 ** MACRO END **
NAME LABEL, CNTL, AREA, LABL1, LBL2
ADD 'SLIST'
04CO 009A SLIST
X 00001 LABEL DC 0
X 00002 DC 0
X 00003 BSS 4
X 00004 DC 0
X 00005 DC AREA
X 00006 DC L LIST
X 00007 ** MACRO END **
04CO 009D ** LIBRARY END **
00020
ENDUP
UPDATE COMPLETED
```

38 1130/1800 Macro Assembler Programming
He then determines which macros from his library, which he has named JONES, are required. This library contains macros that he has written because he finds them useful in his job of providing baseball statistics.

The following listing demonstrates how his library was created.

```assembly
*FILE O JONES 10
WILL RESERVE AT SCRN ADDR O4CA
*MACRO UPDATE
BUILD *JONES*
NAME LABEL FROM: TO COUNT
ADD *MOVE* O4CA 0005 MOVE
  X 0001 0LX L1 FROM
  X 0002 0LX L2 TO
  X 0003 0LX 3 COUNT
  X 0004 *A LO 1 0
  X 0005 STD 2 0
  X 0006 MX 1 1
  X 0007 MX J 1
  X 0008 MX 3 -1
  X 0009 MX *A
NAME LABEL X COUNT LIST STORE
ADD *SUM* O4CA 0034 SUM
  X 0001 LABEL 0LX X - COUNT
  X 0002 S LA 1 6
  X 0003 B A LKS T - COUNT
  X 0004 STD L STORE
  X 0005 MX X 1
  X 0006 MX B 1
  X 0007 ** MACRO END **
NAME LABEL AB HITS BAVG
ADD *BA* O4CA 0056 BA
  X 0001 0LX L HITS
  X 0002 0LX H
  X 0003 0LX A
  X 0004 0STD L BAVG
  X 0005 MX ++1
  X 0006 0LX D
  X 0007 ** MACRO END **
ADD *COEU* O4CA 0076 COEU
  X 0001 CCE EQU LABEL+1
  X 0002 CCE EQU LABEL+5
  X 0003 CCE EQU LABEL+10
  X 0004 CCE EQU LABEL+15
  X 0005 CCE EQU LABEL+20
  X 0006 CCE EQU LABEL+25
  X 0007 CCE EQU LABEL+30
  X 0008 CCE EQU LABEL+35
  X 0009 CCE EQU LABEL+40
  X 0010 CCE EQU LABEL+45
  X 0011 CCE EQU LABEL+50
  X 0012 CCE EQU LABEL+55
  X 0013 CCE EQU LABEL+60
  X 0014 CCE EQU LABEL+65
  X 0015 CCE EQU LABEL+70
  X 0016 CCE EQU LABEL+75
  X 0017 ** MACRO END **
NAME LABEL NUMBR BY PROD
ADD *MPY* O4CA 0099 MPY
  X 0001 0LABEL D L NUMBR
  X 0002 0H C
  X 0003 0SLT 1 6
  X 0004 0STD L PROD
  X 0005 MX ++1
  X 0006 C 1 DC
  X 0007 ** MACRO END **
NAME LABEL AB HR RESULT
ADD *HRATE* O4CA 011A HRATE
  X 0001 0LABEL D L AB
  X 0002 0KTE 1 6
  X 0003 SLA 1K
  X 0004 0L HR
  X 0005 0STD L RESULT
  X 0006 ** MACRO END **
NAME LABEL TEXT X
ADD *MSG1* O4CA 0135 MSG1
  X 0001 LDL DC 'AB'-+1
  X 0002 'Y' DMES X TEXT
  X 0003 'Z' BES 0
  X 0004 ** MACRO END **
NAME LABEL TEXT TEXT2 X
ADD *MSG2* O4CB 00DC MSG2
  X 0001 LDL DC 'AB'-+1
  X 0002 'Y' DMES X TEXT1
  X 0003 'Z' BES 0
  X 0004 'Z' BES 0
  X 0005 ** MACRO END **
```

21 27 35 Format: LABEL MOVE FROM: TO COUNT FROM = address of the data to be transferred TO = address at which the data will be transferred COUNT = number of words to be transferred The MOVE macro transfers a block of data from one area to another.

21 27 35 Format: LABEL SUM X: COUNT LIST STORE X = index register to be used COUNT = number of words to add LIST = address of the first word to be added STORE = address where the sum shall be placed The SUM macro calculates the sum of a block of contiguous words and places the result into a word designated by the STORE parameter.

21 27 35 Format: LABEL BA AB HITS BAVG AB = address of the word which contains the number of at-bats HITS = address of the word which contains the number of hits BAVG = address where the batting average shall be placed The BA macro calculates a batting average and places it into a word designated by the BAVG parameter.

21 27 35 Format: LABEL CDEQU The CDEQU macro generates a series of EQU statements tailored for the processing of data cards. LABEL = address of the word count word which precedes the buffer into which a data card will be read Since EQU statements do not increase the size (in words) of a program, it is not wasteful if all of the labels generated by CDEQU are not used.

21 27 35 Format: LABEL MPY NUMBR BY PROD NUMBR = address of one number to be multiplied BY = second number (not an address) to be multiplied PROD = address of the word where the product will be stored The MPY macro multiplies two given numbers and stores the result into a word designated by the PROD parameter.

21 27 35 Format: LABEL HRATE AB HR RESULT AB = address of the word which contains the number of at-bats HR = address of the word which contains the number of home-runs RESULT = address where the home-run ratio will be stored The HRATE macro calculates a home-run ratio and places the result into a word designated by the RESULT parameter.

21 27 35 Format: LABEL MSG1 TEXT X TEXT = text, using DMES syntax, of the message X = 0 if 1053 message, 1 if 1443 message The MSG1 macro facilitates the printing of a message. A word count and DMES statement are generated.

21 27 35 Format: LABEL MSG2 TEXT TEXT2 X The MSG2 macro is the same as the MSG1 macro except that two DMES statements (for longer messages) are generated.
Mr. Jones has now selected all the macros he is going to use, and the next step is for him to write the coding. The following is a list of the source coding Mr. Jones decided to use. Note that he could have further reduced the coding by using more complex macros. (In order to aid understanding, the macros in this sample are not complex; therefore, the coding required by Mr. Jones is far more extensive than would be needed in actual applications.)
Mr. Jones then submitted the coding punched on cards to his system operator and requested that the job be performed and a listing of the operation supplied. The following listing shows the assembly and execution of his program. (The explanation of the coding is given to the side of the listing; the macro instructions are enclosed in rectangles.)
Move and convert the four columns for triples (40 through 43) and store the result into location TRPLS.

Move and convert the four columns for homers (45 through 48) and store the result into location HOMRS.

Compute SNGLE+DBLES+TRPLS+HOMRS and store the result into location HITS.

Compute the batting average and store the result into location BTAVG.

Compute the home-run ratio and store the result into location RATIO.

In preparing for the slugging percentage calculation, multiply DBLES by 2 and store the result in location DBLES.

Multiply HOMRS by 4 and store the result in HOMRS.

Compute SNGLES+DBLES+TRPLS+HOMRS and store the result into location HITS.

Compute the slugging percentage and store the result into location SPTC.

Move 4 words beginning at WKBUF+2 to words 50-53 of the card buffer. This puts the batting average into the card buffer which will then be converted to 1443 code and printed.

Convert and move the slugging percentage to columns 55-58 of the card buffer.
Macro Assembler Language 43
Convert and move the home-run ratio to columns 60-63 of the card buffer.

**Supress leading zeros to facilitate an easy-to-read printout by replacing all leading zeros (0/2000 in card code) with blanks (0 in card code). Do this for the at-bat field.**

- Suppress leading zeros in the singles field.
- Suppress leading zeros in the doubles field.
- Suppress leading zeros in the triples field.
- Suppress leading zeros in the home-runs field.

Supppress leading zeros in the home-runs field

Suppress the first character in the batting average field since batting averages are always written with 3 digits. (The first character will always be zero.)
From this example you should realize how the use of macros with meaningful names helped Mr. Jones and his fellow workers make efficient use of their system. Similarly, you can design your macros and libraries to aid your programmers and others who must work with the programmers.
The Macro Update Program

The Macro Update Program (MUP) assists you in maintaining macro libraries. It performs the following functions:

- Initializes disk space for macro libraries.
- Adds macros to libraries.
- Deletes macros from libraries and reclaims the space they occupied.
- Joins macro libraries, physically or logically.
- Renames macro libraries.
- Obtains a listing of the contents of macro libraries by macro name or by macro name and statement.
- Inserts or deletes statement(s) within a macro.
- Provides macro definition source statements on cards.

To call the Macro Update Program, you must first load the Disk Utility Program (for DM2) or the Disk Management Program (for MPX) into main storage. After it has been loaded, you use an *MACRO UPDATE control statement to call MUP. Following this statement, you should use MUP control statements to indicate the functions you want to perform. The function field must begin in column one, and at least one blank is required to separate it from the operand field. If you leave the first column blank, the statement will be ignored. If you want to specify more than one operand, you must separate the operands by commas and leave no blanks within or between them. Any unused columns in MUP control statements are reserved for system use. With the exception of the NAME function (described later in this chapter), control statement continuation is not allowed in MUP.

Note that once a Macro Update function has been started, it should not be aborted, because an incompletely modified library may cause the library to be unusable.

All special characters used in the Macro Update Program must conform to the character code summaries as listed in the 1130 Assembler Language manual, Order Number GC26-5927, or the 1800 Assembler Language manual, Order Number GC26-5882.

Initializing Disk Space

Before initializing disk space, you must reserve a data file to serve as your macro library. This can be accomplished by means of an *DFILE statement or an *STOREDATA statement. For a detailed description of *DFILE and *STOREDATA, see the Programming and Operator’s Guide (for DM2), Order Number GC26-3717, or the Programmer’s Guide (for MPX), Order Number GC26-3720.
To initialize disk space for a macro library, you must use a BUILD statement. The library name in the BUILD statement must be the same as the one defined in the *DFILE or *STOREDATA statement. After you have named a library, you must use that library's name in all LIB, BUILD, JOIN, or CONCAT statements (described later) that refer to that library. You must initialize disk space before requesting the Macro Assembler or the Macro Update Program to operate on a specific library. The format of a BUILD statement is as follows:

```
  LNAME is the name of the macro library that was reserved by the *DFILE or *STOREDATA statement (discussed above). LNAME is a 1-5 character name for the macro library. The apostrophes are delimiters and, as such, are required by MUP for the names of all macros and libraries. You can use alphabetic characters A-Z and the characters 0-9, #, @, and $ within your library name. The digits 0-9 may not be used as the first character. A library name is considered a symbol, and therefore, it must conform to the rules for symbols as stated in the Assembler Language manual, Order Number GC26-5927 (for the 1130) or Order Number GC26-5882 (for the 1800). If LNAME applies to a previously initialized library containing macros, the function purges the library and reinitializes it.
```

**Specifying the Macro Library**

To specify the macro library to be operated on, you use the LIB statement. The format of a LIB statement is as follows:

```
  LNAME (discussed above) is the name for the macro library.
```

**Joining Macro Libraries Physically**

If you want to physically join a macro library to the end of the library specified in the last BUILD or LIB statement,
you use a JOIN statement. For example, if you wanted to physically join LIB06 to LIB05, you would use the following statements.

These statements cause the contents of LIB06 to be added (physically copied) to the end of the contents of LIB05. This does not change the contents of library LIB06.

If the first library cannot accommodate the library being joined to it, the JOIN operation is suppressed. Neither library is changed and processing continues with the next LIB or BUILD statement. If you specify a new library name in a BUILD and then join an existing library to it, you physically copy the existing library.

Joining Macro Libraries Logically

If you want to logically join a macro library to the library specified in the last BUILD or LIB statement, you use a CONCAT statement. This statement allows you to maintain individual libraries, and then unite them for assembly purposes without using additional disk space.

For example, if you wanted to logically join LIB22 to LIB15, you would use the following statements.

These statements cause both libraries to be available for assembly purposes when LIB15 is referenced by the Macro Assembler. However, in any Macro Update Program operation, only the library named on the last LIB or BUILD statement is operated upon.

It is possible to concatenate a multiplicity of libraries, so that several libraries may be available to the Macro Assembler even though the assembly references only one. You
would perform this multiple concatenation by concatenating library B to library A, library C to B, library D to C, library E to D, and so on until all the libraries that you wanted were linked together. You can concatenate only one library to any other library, but you can concatenate up to a total of 16 libraries, making a total of up to 17 available for assembly purposes. If the Macro Assembler does not find a macro and has searched through 17 libraries, the statement containing the name of the macro will be flagged as an op code violation and the assembly will continue.

If you want to disconnect a library that has been concatenated to another library, you can use the following statements.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The CONCAT statement would cause the library (LIB22) that has been concatenated to library LIB15 to be disconnected. The physical contents of the libraries remain unchanged.

Updating a Macro in a Library

If you want to alter a macro that has been stored in the library specified in the last BUILD or LIB statement, you specify that macro with the UPDATE function. An example of specifying the macro TAXES might be as follows.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The UPDATE statement is normally followed by an INSERT or DELETE statement. Descriptions of these two statements are discussed under "Inserting Statements in a Macro" and "Deleting Statements From a Macro."
Renaming a Macro in a Library

If you want to rename a macro in a library, you use the RENAME statement. Once you have renamed a macro, the original name is lost, and the macro can be referenced only by its new name. The format for the RENAME statement is as follows:

```
NAME P 1  P 2  P 3  .  .  .  P n

RENAME 'BOLTS', 'RIVET'
```

This statement would cause the macro BOLTS to be renamed RIVET. BOLTS represents the name of the current macro; it must be enclosed in apostrophes and separated from the new name by a comma. RIVET represents the new name for the macro. The macro must now be referenced by this new name. BOLTS can now be used as a name for another macro. The RENAME statement may be followed by INSERT or DELETE statements.

Defining a Macro During a Macro Update Run

You can define a macro during a macro update run by using essentially the same method you use during an assembly. The differences are that you don't use MAC or SMAC and MEND statements, and you don't use a definition prototype statement. Instead, you use a NAME statement to name the parameters to be used in a subsequent definition and an ADD statement to name the macro (discussed below). The format of the NAME statement is as follows:

```
NAME P 1  P 2  P 3  .  .  .  P n
```

P1 is the symbol for parameter 1, P2 is the symbol for parameter 2, and so on.

Note that parameter 1 is the parameter used in the label field of the macro call. Parameter 2 would be the first parameter in the operand field. If you need more than one record for all your parameters (limited to 20 possible), use as many NAME records as needed immediately following the first, and continue the parameters on these records.
such a case, a comma must not follow the last parameter on each record.

If your NAME statement does not have sufficient parameter names for the parameters in the macro being processed, one of the following will occur:

- On input, the additional parameters will be assumed to be standard Macro Assembler variables instead of macro parameters.

- On output, the operation will be aborted and a D117 error message printed. This also occurs if a parameter that is used in the format or tag fields is given a name that is greater than one character.

- On listing, extra parameters will be replaced by // N where N is the number (1-20) of the parameter. A D117 error message will be printed. Note that the parameter number may be truncated if it exceeds the field length.

The names specified in the NAME statement are used in all subsequent operations until the next LIB or BUILD statement is encountered or another NAME statement is read.

**ADD STATEMENT**

After the NAME statement, you use the ADD statement. The ADD statement adds the macro to the library specified in the last LIB or BUILD. The ADD statement in conjunction with the NAME statement performs the same function as the definition prototype statement of the Macro Assembler definition. The format of the ADD statement is as follows:

```
ADD 'PARTS'
```

This statement causes the macro PARTS to be added to the end of the macro library specified in the last BUILD or LIB statement.

Following the ADD statement, you place the assembler-language source statements that you want to include in the definition. The macro being thus defined by the ADD function is terminated by the occurrence of a MUP control statement. An example of how to define a macro during a macro update run is as follows:
If there is not sufficient space in the library to accommodate the macro you want to store, the macro is not added. Processing continues with the next LIB or BUILD statement; the library is not changed by this occurrence.

If you define another macro within a macro definition, then the MAC or SMAC and MEND statements of the nested macro are included in the definition of the outer macro. Thus, if a macro is defined with an ADD statement and its source statements include a MAC or SMAC and MEND statement, then every time it is called in an assembly, a macro definition is generated.

The definition prototype statement cannot be used in a macro defined by the ADD function except in conjunction with a MAC or SMAC statement. If one is present, no error is diagnosed. Note also that macro source statements stored during a MUP run are not diagnosed for errors.

Deleting a Macro From a Library

If you want to delete a macro from the library named in the last LIB or BUILD statement, you use a PURGE statement. This function deletes the macro that is specified, and automatically reclaims the space it occupied. The format of a PURGE statement is as follows:

```
1-10  11-20  21-30  31-40
1234567890 1234567890 1234567890 1234567890
PURGE 'WAGES'
```
This statement causes the macro WAGES to be deleted from the library named in the last LIB or BUILD statement and reclaims the space WAGES occupied.

The Macro Update Program PURGE statement should not be confused with the Macro Assembler PURG statement. The Macro Assembler PURG statement does not automatically reclaim the space occupied by the macro named in that statement. Instead, the space is reclaimed by running the Macro Update Program. Any macro update run affecting a particular library will reclaim the space occupied by a macro deleted from that library with the Macro Assembler PURG statement. The PURG and PURGE statements cause operations to occur in regard to the specific library named in the last LIB, BUILD, or *MACLIB statement. They do not affect any other library.

Punching Source Statements

If you want to punch the source statements of a specified macro definition, you can use the OUTPUT statement. This statement must have been preceded by a NAME statement. Blank cards must be available for punching as soon as the OUTPUT statement is read.

Care should be taken in preparing the NAME statement since the OUTPUT function will be aborted if a parameter is defined incorrectly or is left undefined (see NAME statement described previously). The definition to be punched must be part of the library specified in the last LIB or BUILD statement. The format of the OUTPUT statement is as follows:

```
  1-10 | 11-20 | 21-30 | 31-40 |
     |      |      |      |
  1234567890 | 1234567890 | 1234567890 | 1234567890 |

use OUTPUT 'SALES'
```

SALES is the name of the macro definition to be punched. This statement also causes an ADD 'SALES' statement to be punched prior to the source statement to facilitate loading of the definition at a later time. The source statements will contain an identification (first three characters of the macro name) and a sequence number in columns 73-80.

Inserting a Statement in a Macro

If you want to insert additional macro definition source statements into a macro, you can use the INSERT statement. This statement must be preceded by a RENAME or UPDATE statement and a NAME statement (discussed previously) that specifies the macro to be modified and its parameters, and it must be followed by the source statements to be inserted. The format of an INSERT statement is as follows:
### Inserting Statements Into a Macro

**Format**

`INSERT NNNNN`

**Notes**

- **NNNNN** is a decimal integer (maximum of 32767) up to five digits long. It references a macro definition source statement sequence number. The source statements are inserted after NNNNN; so, if you want the statements inserted before any other statements in that macro, you must specify NNNNN as zero.

- Whenever **MUP** or the Macro Assembler prints macro definition source statements, five-digit decimal sequence numbers are printed to the left of each statement. These sequence numbers are referenced by **INSERT** and **DELETE** statements. Any statements inserted into a library by the ADD, **INSERT**, or **DELETE** (described in the next section) function are flagged with an X when the definition is printed. Macro definition source statements are automatically sequenced when the **ADD** function is used, and resequenced when the **INSERT** or **DELETE** functions are used.

- You can insert only as many statements as the library has room for. In other words, if you have 25 statements to add to a macro, and there is enough space to accommodate only 15 of those statements, those fifteen statements will be added. A DI03 LIBRARY OVERFLOW message will be printed; processing will continue with the next **LIB** or **BUILD** statement. If you want to include the ten statements left out of your macro after regaining enough space to accommodate them, you will have to alter your **INSERT** control statement because of the resequencing of the statements in the macro library that has occurred.

### Deleting a Statement From a Macro

**Deletion**

If you want to delete one or more statements from a macro, you can use the **DELETE** statement. The **DELETE** statement must be preceded by a **RENAME** or **UPDATE** statement that specifies the macro to be altered and a **NAME** statement (discussed previously) that specifies parameter names used for alteration. The format of a **DELETE** statement is as follows:

**Format**

```
DELETE MNNNN,NNNNN
```
Obtaining a Listing of Macro Libraries by Statements or Macros

The SELECT control statement is used to control the Macro Update Program printed output and remark inclusion. The output is always printed on the list (principal) printer, and MUP control statements are always printed. You can also specify in a SELECT statement that remarks are to be included with any macro text statement being placed on disk. An example of the print format may be seen in the sample program that is at the end of this chapter.

The format of the SELECT statement is as follows:

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1-10</td>
<td>11-20</td>
<td>21-30</td>
<td>31-40</td>
</tr>
<tr>
<td>12345678901234567890123456789012345678901234567890</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SELECT M, P, I, C, N</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

no parameters—suppresses printing of macro headers and macro text.

M—causes headers to be printed.

P—causes text and headers to be printed. If you select this option, it should have been preceded by a NAME statement (discussed previously). On an OUTPUT operation with this option, the 1800 will not list the macro text. Note that if a LIB, BUILD, or ENDUP statement is encountered and no name statement is available, SELECT P will be reset and the text will not be printed. On the 1800, if sense switch 2 is ON, printing will be suppressed.

I—causes headers to be printed except during an ADD, INSERT, or DELETE operation, in which case, the macro headers and text are printed. If you do not specify a SELECT statement, I is assumed.
On the 1800, if sense switch 2 is ON, printing will be suppressed.

C--indicates that remarks are to be included with any macro text statement being placed on disk.

SELECT N

N--indicates that both of the following conditions are true (see also the section following):

- You want to update a nested definition.
- The statement(s) inserted uses the automatic name generation feature.

You may use any combination of parameters in a SELECT statement; however, I will override M and P will override either I or M, or both. Each two consecutive parameters must be separated by a comma, with no embedded blanks. If you use more than one SELECT statement, the latest is assumed, and the prior SELECT statements are overridden.

SPECIAL REQUIREMENTS ON THE USE OF AUTOMATIC NAME GENERATION IN NESTED DEFINITIONS

When automatic name generation is used with nested definitions, the indicators for automatic name generation (leading and/or trailing apostrophes) must be suppressed until the call (expansion) of the outer macro occurs. Otherwise, the automatic name generation feature will not function properly. To suppress this feature, the statements in a nested definition must be stored as data. In a nested definition, the MAC statement(s) of the inner nested definitions indicates that the Macro Update Program (on an ADD statement) or the Macro Assembler (on an SMAC statement) should store the statements between the MAC and its associated MEND.

When inserting statements that use automatic name generation in a nested definition, you must precede the INSERT or DELETE statements with a SELECT statement that includes N as one of its parameters. This should be done to indicate the Macro Update Program should store these inserted statements as data. The SELECT N option should be followed by another SELECT option (without N as a parameter) when all inserts to the nested definition are completed. If suppressing the automatic name generation feature by SELECT N is not done properly, the consequences will not be observed until an assembly with the call (expansion) of the nested macro is attempted.
Consider the following example:

LIB 'LIB01'
SELECT P
NAME LABEL,A,B,C

UPDATE COMPLETED

The macro MOVE is a nested definition and uses the automatic name generation feature. Hence, a SELECT N statement must precede any updates (statements for insertion into the macro MOVE) that use the automatic name generation feature. Failure to do so will result in the symbol actually being expanded at this time, which is not desired.

Note that any updates using automatic name generation outside the macro MOVE should not be preceded by a SELECT N statement, since the automatic name generation will be suppressed. In the above example, an insert that uses automatic name generation made between statement number 00003 and statement number 00010 should be preceded by a SELECT N statement. An insert made elsewhere in the macro MOVE that does not use automatic name generation should not be preceded by a SELECT N.

Designating Comments

A period in column one of any record designates that record as a MUP comment and it is printed on the list (principal) printer. If, however, an error has occurred, and MUP is ignoring all statements until the next LIB or BUILD statement, comments will also be ignored and not printed. Note also that MUP comment records are never included in your macro library.

Terminating a Macro Update Run

To terminate a Macro Update Program run properly, you must use an ENDUP statement. This statement must be the last MUP control statement used, or else none of the statements that follow it will be processed. You don't specify any parameters in an ENDUP statement; the format is as follows:
The UPDATE COMPLETED message (see the following sample program, note 9) indicates that the Macro Update run has been properly terminated. This does not imply that all the operations requested were successful.

Sequencing MUP Control Statements

The following table illustrates the required order of MUP control statements. Each x in the matrix indicates that the function at the top of the column must precede the statement to the left side of the x. MUP statements not included in this table have no precedence requirements.

<table>
<thead>
<tr>
<th></th>
<th>BUILD or LIB</th>
<th>RENAME or UPDATE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBRARY FUNCTIONS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JOIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONCAT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAME STATEMENT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACRO FUNCTIONS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPDATE</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PURGE</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RENAME</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUTPUT</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATEMENT FUNCTIONS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSERT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRINT FUNCTIONS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANY SELECT P</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Note that a new LIB or BUILD function removes the previous NAME statement and a new one must be used. On any function where you use a SELECT P option, you must have a NAME statement available to SELECT P or else the print or punch operation will not be as you wanted it.

Making Efficient Use of the Macro Update Program

The Macro Update Program searches sequentially through the library it is operating on for the macro specified in a RENAME, OUTPUT, UPDATE, or PURGE statement. It joins
libraries and adds macros to the end of each library. Consequently, the printing may be voluminous while the Macro Update Program is positioning for the requested function. Therefore, to make the most efficient use of the program, you should request the macro functions in the order in which the macros appear in the library, and the JOIN and ADD functions after all other functions for that library have been performed.

For example, suppose LIB01 contains four macros: MAC1, MAC2, MAC3, and MAC4, respectively and you want to perform an OUTPUT function on MAC1 and MAC4, a PURGE function on MAC2, a RENAME function on MAC3, and a JOIN function on LIB02. The following statement sequence is the most efficient to perform these functions.

```
LIB 'LIB01'
OUTPUT 'MAC1'
PURGE 'MAC2'
RENAME 'MAC3','GET'
OUTPUT 'MAC4'
JOIN 'LIB02'
ENDUP
```

The concatenate function does not cause library positioning, and thus, its position within a MUP run is not important.

A Sample Macro Update Program

The following sample program illustrates a sample Macro Update Program run.
**MACRO UPDATE**

1. The library named in a **BUILD** statement must have been defined previously with an "DFILE control statement.

2. The **NAME** statement is required and may specify parameter names to be used in a subsequent definition.

3. **"LIBRARY END"** is printed each time the Macro Update Program encounters the end of the library. The two numbers indicate (1) the logical drive number and sector address of the last sector of the library currently used, and (2) the relative address of the last word on that sector which is used.

   By inspecting these two words, you can determine how much of your library file has been filled.

4. The name of the macro being operated upon is printed. The two numbers preceding the macro name indicate where on the disk the subject macro was built. Word one indicates the logical drive number and sector address of the definition and word two indicates the relative location of where that definition begins on that sector.

5. An X is printed with each statement that is added to a library or macro.

6. Sequence numbers are printed whenever a macro text is printed.

7. **"MACRO END"** is printed whenever the Macro Update Program is through processing a definition.

8. If the library being worked upon is concatenated to another library, this number indicates the logical drive number and sector address of the concatenated library. If no concatenation exists, this number is zero.

9. The Macro Update Program prints the macro names it encounters when scanning through a library. Two numbers in front of the macro name indicate information as described in 4.

10. The library named in a **LIB, JOIN, or CONCAT** statement must have been defined previously with an "DFILE control statement and initialized with a **BUILD** statement.

11. This message is printed when the Macro Update run has been completed. It does not imply successful completion; messages prior to this one may indicate error conditions.
ADD A MACRO TO MOVE THE CONTENTS OF A SPECIFIED \nINDEX REGISTER TO ANOTHER SPECIFIED INDEX REGISTER \nCALLING FORMAT IS \nLABEL LOADX FROMX,TOX \nWHERE 'FROMX' IS THE INDEX REGISTER WHOSE CONTENTS ARE MOVED TO THE \nINDEX REGISTER 'TOX'.

NAME LABEL,XY ADD 'LOADX' 00003 $ MACRO END $ \n1487 0004 LOADX 00003 LABEL STX X 00002 LOX XY 00001 \nPHYSICALLY JOIN LIBRARY 'LIB03' TO LIBRARY 'LIB01'.

LIB 'LIB01' 00000 00006 00004 00005 00000 00007 00003 \nJOIN 'LIB03' 00005 00000 00005 00003 00006 00005 00006 \n1480 0029 MOVE 1480 0043 ACCNX 1480 0058 LOADX \nUPDATE THE 'MOVE' MACRO TO SAVE AND RESTORE INDEX REGISTER 1
UPDATE 'MOVE' 1480 0048 ACCNX 1480 0058 LOADX \nNAME LABEL,XY FROMX,TO,COUNT \nDELETE 1 00001 LABEL STX 1 0 00002 LOX LI COUNT \nINSERT 5 00003 LOAD FROM1 00004 STD LI TO1 00000 00005 MDX 1 -1 \n00006 MDX 1 -1 00007 LOX LI 0 0 \nRENAME THE 'MOVE', MACRO AS 'MOVEL' 0000000 $ MACRO END $ \nRENAME 'MOVE', 'MOVEL' 00480 0053 ACCNX \n1480 0063 LOADX 1480 0073 $ LIBRARY END $ 0000000 00005 00000 00000 \n1480 0055 ACCNX 1480 0060 MDX 00006 00007 00008 \n1480 0029 MOVE \nONLY THE NAME 'MOVEL' CAN NOW BE USED TO REFERENCE THE MACRO \nFORMERLY NAMED 'MOVE'.

PUNCH THE MACRO NOW CALLED 'MOVEL' OUTPUT 'MOVEL' 00480 0053 ACCNX \n1480 0063 LOADX 1480 0073 $ LIBRARY END $ 0000000 00000 00000 \n1480 0055 ACCNX 1480 0060 MDX 1480 0029 MOVE \n1480 0053 ACCNX 1480 0060 MDX 1480 0073 $ LIBRARY END $ 0000000
ENDUP 00000 $ UPDATE COMPLETED $
During the assembly process, the Macro Assembler checks for source program errors. If an error is detected, an error flag or an error code and message will be printed. If a questionable instruction is encountered, it is flagged with a warning flag, Q. Errors in the Macro Update Program are detected by the DM2 Disk Utility Program or the MPX Disk Management Program.

Macro Assembler Sign-Off Message

At the end of each assembly, the Macro Assembler indicates the number of errors and warnings it encountered during that assembly. The message reads:

XXX ERROR(S) AND XXX WARNING(S) IN ABOVE ASSEMBLY.

XXX represents a three-digit decimal number.

Macro Assembler Warning Flag

If the source program contains certain questionable instructions, the Macro Assembler will interpret and process them and flag them with a Q. For example, the statement MDX L PLACE is assembled as MDX L PLACE,0 and is flagged with the warning indicator. Warning flags are not counted as errors and will not prevent the execution of the object program. It is suggested that you check to make sure that the Macro Assembler has performed the task that you really wanted it to do on each statement flagged with a Q.

Macro Assembler Error Detection Codes

During the assembly process, the Macro Assembler checks the source program for errors. Error and warning flags are printed to the left of the label field of each source statement that is in error or is questionable. For a complete description of the listing format of an assembly, see the 1130 DM2 Programming and Operator's Guide, Order Number GC26-3717, or the 1800 Error Messages manual, Order Number GC26-3727.

See Table 1 for an explanation of the error flags and Table 2 for a listing of the Macro Assembler error messages and their meanings.

Macro Update Program Error Messages

When the DM2 Disk Utility Program or the MPX Disk Management Program encounters an error in the Macro Update Program, one of the error messages in Table 3 is printed.
### Table 1. Error Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Cause and Macro Assembler Action</th>
</tr>
</thead>
</table>
| A    | - Address Error. Attempt made to specify, directly or indirectly, a displacement field outside range of -128 to +127. Displacement is set to zero.  
- Invalid reference to CE Core in a long form instruction. Address field is set to zero. (1800 only) |
| C    | Condition Code Error. Character other than +, −, Z, E, C, or D detected in first operand of short branch or second operand of long BSC, BOSC, or BSI statement. Displacement is set to zero. |
| F    | Format Code Error. Character other than L, I, X, or blank detected in column 32, or L or I format specified for an instruction valid only in short form, or I format specified when not allowed. Instruction processed as if L format were specified, unless that instruction is valid only in short form, in which case it is processed as if the X format were specified. |
| L    | Label Error. Invalid symbol detected in label field. Label is ignored. |
| M    | Multiply Defined Label Error. Same symbol encountered in label fields of two or more statements. First occurrence of symbol in label field defines its value; subsequent occurrences of symbol in label field are ignored and cause an M error flag to be printed. |
- Invalid op code. Statement is ignored and address counter is incremented by 2.  
- ABS used when +COMMON is used to define FORTRAN common table. Statement is ignored. (1800 only)  
- ISS, ILS, ENT, LIBR, SPR, EPR, or ABS incorrectly placed. Statement is ignored. |
| Q    | Warning. A statement whose syntax is questionable was encountered. |
| R    | Relocation Error.  
- Expression does not have a valid relocation. Expression is set to zero.  
- Non-absolute displacement specified. Displacement is set to zero.  
- Absolute origin specified in relocatable program. Origin is ignored.  
- Non-relocatable operand in END statement of relocatable mainline program. Columns 9-12 are left blank; entry is assumed to be relative zero.  
- Non-absolute operand specified in BSS or BES. Operand is assumed to be zero.  
- ENT operand is non-relocatable. Statement is ignored.  
- Invalid reference to CE Core. Address field is set to zero. (1800 only)  
- Invalid reference to a symbol defined in a COMMON area. Address field is set to zero. (1800 only) |
| S    | Syntax Error.  
- Invalid expression, for example, invalid symbol, adjacent operators, or illegal constant. Expression is set to zero.  
- Illegal character in record. If illegal character appears in expression, label, op code, format, or tag field, additional errors may be caused.  
- Main program entry point not specified in END operand. Columns 9-12 are left blank; entry is assumed to be relative zero.  
- Incorrect syntax in EBC statement (such as no delimiter in column 35, or zero character count). Columns 9-12 are not punched; address counter is incremented by 17.  
- Invalid label in ENT or ISS operand. Statement is ignored. |
| T    | Tag Error. Column 33 contains character other than blank, 0, 1, 2, or 3 in instruction statement. Tag of zero is assumed. |
| U    | Undefined symbol. Undefined symbol in expression. Expression is set to absolute zero. |

**1130 ONLY**

| W    | X or Y coordinate, or both, not within the specified range, or invalid operand. Operand set to zero. |
| X    | Character other than R or I in column 32, or character other than D or N in column 33. Field set to zero. |
| Z    | Invalid condition in a conditional branch or interrupt order. Condition bits in first word set to zero. |
### Table 2. Macro Assembler Error Codes and Messages

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>A01</td>
<td>MINIMUM W.S. NOT AVAILABLE</td>
<td>Working Storage (for DM2) or Batch Processing Working Storage (for MPX) available is less than the number of overflow sectors specified plus one.</td>
</tr>
<tr>
<td>A01</td>
<td>MINIMUM W.S. NOT AVAILABLE...</td>
<td>Number of sectors of symbol table overflow is greater than the number of overflow sectors specified.</td>
</tr>
<tr>
<td>A02</td>
<td>SYMBOL TABLE OVERFLOW</td>
<td>Intermediate output in pass 1 or final disk system format output in pass 2 is larger than Working Storage (for DM2) or Batch Processing Working Storage (for MPX) minus the number of overflow sectors specified.</td>
</tr>
</tbody>
</table>
| A03 | DISK OUTPUT EXCEEDS W.S. | With *SAVE SYMBO L TABLE option specified:  
1. Program in relocatable assembly.  
2. Program contains an assembly error.  
3. Source program causes more than 100 symbols to be present in the System Symbol Table. |
| A04 | SAVE SYMBOL TABLE INHIBITED | XXX number of ORG, BSS, BES, and/or EQU statements that were undefined in pass 1. At the end of pass 1, these erroneous statements are printed on the list printer. If the error was due to forward referencing, it will not be detected during pass 2. |
| A05 | XXX ERRONEOUS ORG, BSS, OR EQU STATEMENTS IN ABOVE ASSEMBLY | Errors were detected during pass one. This warning message is printed at the end of pass one. |
| A06 | LOAD BLANK CARDS | A card containing a nonblank column in columns 1-71 has been read while punching the symbol table (as a result of an *PUNCH SYMBOL TABLE control statement). |
| A07 | ABOVE CONTROL STATEMENT INVALID | An invalid control statement has been read by the Macro Assembler. The control statement is ignored and the assembly is continued. |
| A08 | MACLIB UNDEFINED | The Macro Assembler has been asked to process a SMAC statement and either no *MACLIB control statement was previously read, or the name on the *MACLIB statement is not found in a disk area search. The library named in the *MACLIB statement has a library concatenated to it and this library could not be found. |
| A09 | PARAMETER LIST OVERFLOW | A call to a macro exceeds the space specified in the N2 field of the *OVERFLOW SECTORS control statement used with this assembly. If the *OVERFLOW SECTORS control statement was not used or if the N2 field was not specified, the Macro Assembler assumes the value of the N2 field to be zero. |
| A10 | MACRO AREA OVERFLOW | An attempt was made to define a temporary macro and either the N3 field of the *OVERFLOW SECTORS control statement was not specified, or the space specified by N3 was not large enough, or the macro library was exceeded. |
| A12 | NEST LEVEL EXCEEDS 20 ASSEMBLY TERMINATED | A macro call exceeded the allowable nest level limit of 20. |
| A21 | *LEVEL CONTROL STATEMENT MISSING | The program listed above was assembled as an ISS subroutine without the required *LEVEL control card. (1130 only) |
| A22 | INVALID LIST DECK OPTION ASSEMBLY TERMINATED | An attempt was made to punch macro statements in two pass mode. (1130 only) |
| A40 | MAIN PROG NO NAME | Mainline program being assembled has no name specified on a // ASM statement. (1800 only) |
| A41 | // CARD READ | A Supervisor control statement has been read by the Macro Assembler. The Macro Assembler has passed this statement along to the Supervisor before terminating the assembly. Loading and DMP operations are inhibited. (1800 only) |
| A42 | ABSOLUTE REENTRANT PROG | A non-relocatable program has been specified as reentrant. (1800 only) |
| A43 | INVALID SFRL FILE | When loading the source deck via *SFRL, the file was truncated due to insufficient sector allocation. (1800 only) |
| A44 | LIT TBL OVFL | The size of the literal table as specified on the fourth parameter of the *OVERFLOW SECTORS control statement was too small. (1800 only) |
| A46 | XREF DATA OVFL | VCORE was too small to sort and merge the data for building the cross-reference table. (1800 only) |
Table 3. Macro Update Program Error Messages

<table>
<thead>
<tr>
<th>Error Code and Message</th>
<th>Cause and Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>D100 LIBRARY NOT FOUND</td>
<td>Library named on a LIB, BUILD, JOIN, or CONCAT statement could not be found on drives currently in use. If LIB, BUILD, or JOIN statement, all statements are ignored until the next LIB, BUILD, or ENDUP statement is encountered. If CONCAT statement, processing continues with the next control statement. Correct the name field in the statement in error, or change the // JOB statement to include the drive on which the named library resides, or define the macro library using an *DFILE or *STOREDATA control statement.</td>
</tr>
<tr>
<td>D101 INVALID SUBFIELD COL XX</td>
<td>If on an INSERT or DELETE statement, the sequence number was incorrectly specified, that is, it was negative, or non-numeric, or sequence numbers were reversed. If on a SELECT statement, an incorrect parameter was specified. Processing continues with the next control statement. If on a NAME statement, an invalid parameter was detected. Processing continues with next LIB, BUILD, or ENDUP statement. XX indicates the column in which the error was found. Correct the error and rerun the portion of the job affected.</td>
</tr>
<tr>
<td>D102 ILLEGAL REQUEST</td>
<td>An invalid control statement was detected, an INSERT or DELETE statement was not preceded by an UPDATE or RENAME statement, or on an 1130, request was made for output to paper tape or to a pack configured for paper tape. Processing continues with the next control statement. Correct the error and rerun the portion of the job affected.</td>
</tr>
<tr>
<td>D103 LIBRARY OVERFLOW</td>
<td>The library last specified by a LIB or BUILD statement does not have enough room to perform the operation. If on a JOIN or an ADD statement, the operation is suppressed and the library is restored to its previous state. If on an INSERT statement, the statements listed prior to the message were the only ones that could be included. Processing continues with the next LIB, BUILD, or ENDUP statement. Additional space can be obtained in the current library by purging unneeded macros or deleting unneeded statements. If this is not possible, define a larger library using an *DFILE or *STOREDATA control statement, join the old library to a new one, and delete the old library. Once the additional space has been obtained, rerun the portion of the job affected. If on an INSERT, it may be necessary to alter your INSERT control statement as the statements in the macro library may have been resequenced.</td>
</tr>
<tr>
<td>D104 MACRO NOT FOUND</td>
<td>The macro name specified on an OUTPUT, PURGE, RENAME, or UPDATE statement could not be found in the library being processed. Processing continues with the next control statement. Correct the macro name on the statement in error, or specify the correct macro library and rerun the portion of the job affected.</td>
</tr>
<tr>
<td>D105 SEQUENCE NUMBER NOT FOUND</td>
<td>The sequence number on an INSERT or DELETE statement was out of range of the macro and could not be found or the sequence numbers on multiple INSERT and/or DELETE statements for the same macro were out of order. Processing continues with the next control statement. Place a correct sequence number on the statement in error and rerun the portion of the job affected.</td>
</tr>
<tr>
<td>D106 LIBRARY NOT SPECIFIED</td>
<td>An attempt was made to operate on a macro without specifying a macro library. Processing continues with the next LIB, BUILD, or ENDUP statement. Place a LIB or BUILD statement before the statement in error and rerun the portion of the job affected.</td>
</tr>
<tr>
<td>D107 SPILL OVERFLOW</td>
<td>Macro text insertions have caused the capacity of Working Storage spill to be exceeded. Processing continues with the next LIB, BUILD, or ENDUP statement. Correct the sequence numbers in the unprocessed INSERT statements, if necessary, and rerun these statements. It may be necessary to define additional disk drives to provide adequate Working Storage.</td>
</tr>
<tr>
<td>D108 CONTROL STATEMENT READ</td>
<td>An * or // statement has been read and the MUP run is terminated. On the 1800, control is passed to the Supervisor which will begin processing with the next // JOB statement. On the 1130 control is returned to the Supervisor for a // statement or to DUP for an * statement.</td>
</tr>
<tr>
<td>D109 NAME STATEMENT NOT FOUND</td>
<td>The operation attempted requires a NAME statement and one has not been processed following the last LIB or BUILD statement. Processing continues with the next LIB, BUILD, or ENDUP statement. Insert a NAME statement and rerun the part of the job that was affected.</td>
</tr>
<tr>
<td>Error Code and Message</td>
<td>Cause and Corrective Action</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>D110 INVALID NAME</td>
<td>The name field on a LIB, BUILD, JOIN, CONCAT, UPDATE, ADD, PURGE, RENAME, or OUTPUT statement was left blank, the name specified was invalid, or the apostrophes are improperly placed. If on a LIB, BUILD, or JOIN statement, processing continues with the next LIB, BUILD, or ENDUP statement. If on a CONCAT, UPDATE, ADD, PURGE, RENAME, or OUTPUT statement, processing continues with the next control statement.</td>
</tr>
<tr>
<td>D112 NONBLANK CARD READ. ENTER BLANK CARDS</td>
<td>A nonblank card has been read by a 1442-6 or -7 during a punch operation. Remove stacked input from hopper, NPRO nonblank cards, place blank cards followed by NPRO cards and stacked input in the hopper, and press reader START and for the 1130 press program START.</td>
</tr>
<tr>
<td>D116 LIBRARY NOT INITIALIZED</td>
<td>Library named on a LIB, JOIN, or CONCAT statement has not been initialized previously. If on a LIB, or JOIN statement, processing continues with the next LIB, BUILD, or ENDUP statement. If on a CONCAT statement, processing continues with the next control statement. Initialize the library with a BUILD statement, and rerun the portion of the job affected.</td>
</tr>
<tr>
<td>D117 INVALID PARAMETER</td>
<td>On a NAME statement, more than 20 parameters were specified. If during the processing of a macro, a parameter has been detected which was not defined in the NAME statement or a parameter greater than one character was used in the format or tag field. If during an OUTPUT operation, the operation is aborted and processing continues with the next control statement. If during a listing operation, this is a warning message and the invalid parameter is printed as /N where N is 1-20. (Note: N may be truncated if the field size is exceeded.) If on OUTPUT, correct the NAME statement and rerun the portion of the job affected.</td>
</tr>
</tbody>
</table>
Appendix A: General Examples of Macros and 1130 DM2
Macro Assembler Features

The following group of macros and the examples of their use are intended to demonstrate how macro instructions can be used to simplify assembler-language programming. If all these macros were defined in your system, then you could use two new statements—the READ statement and the WRITE statement—to accomplish all the programming normally required to effect input and output on an 1130 system having a disk, a 1442 card read punch, and a 1403 printer. When you issue a READ or WRITE macro, you need specify only the name (DISK, CARD, or PRINT) of the device you want to use, the name of the I/O area, and, if you want, the name of your error-handling program. This system of macros then issues calls to the appropriate I/O control subroutines, handles data conversion and blocking, and, at your option, handles error checking and retries.

You could, of course, expand this set of macros to include all I/O devices supported by the 1130 system; you could also write a similar set of macros to simplify I/O programming on the 1800 system.

Refer to the 1130 Subroutine Library manual, Order Number GC26-5929, for a complete description of the 1130 I/O control subroutines (DISKN, CARD1, and PRNT3) referred to in these macros.

The first part of this sample program (until the *MACRO UPDATE statement) is a FORTRAN program that builds a one sector file of one-word integers having the value 1 through 320. This is necessary to handle the data file in the sample program that follows it.

```fortran
// JOB
LOG DRIVE CART SPEC CART AVAIL PHY DRIVE 0000 0578 0578 0000
V2 M06 ACTUAL 32K CONFIG 32K

// * CREATE A DISK DATA FILE NAMED 'FILE1' AND FILL IT WITH * SMAC0020
// * 320 INTEGER VALUES 1 TO 320 VIA FORTRAN. * SMAC0040

// DUP
*DELETE FILE1
D 26 NAME NOT FOUND IN LET/FLET

*STOREDATA WS UA FILE1 1 CART ID 0578 DB ADDR 3220 DB CNT 0010

// FOR
*IDCS(DISK)
*LIST ALL
```

SMAC0010

SMAC0020

SMAC0030

SMAC0040

SMAC0050

SMAC0060

SMAC0070

SMAC0080

SMAC0090

SMAC0100

SMAC0110
*ONE WORD INTEGERS
DEFINE FILE 1(320,1,U,K)
  K = 1
  IVAL = 1
  DU 100 I = 1,320
  WRITE (1,K) IVAL
  IVAL = IVAL + 1
100 CONTINUE
  CALL EXIT
END

VARIABLE ALLOCATIONS
  K(I) = 0008  IVAL(I) = 0009  I(I) = 000A

STATEMENT ALLOCATIONS
  100 = 0028

FEATURES SUPPORTED
  ONE WORD INTEGERS
  IOCS

CALLED SUBPROGRAMS
  SDFIO  SDWRT  SDCOM  SDI

INTEGER CONSTANTS
  I = 000C  320 = 000D

CORE REQUIREMENTS FOR
  COMMON 0 VARIABLES 12 PROGRAM 38

END OF COMPILATION

// XEQ  L 1

*FILES(1,FILE1)
FILES ALLOCATION
  1 0322  0001  0578 FILE1
STORAGE ALLOCATION
  R 41 7AFC (HEX) WDS UNUSED BY CORE LOAD
LIBF TRANSFER VECTOR
  PAUSE 0408
  SDCOM 02A5
  SDI 025E
  SDWRT 02DA
  SDFIO 02DF
SYSTEM SUBROUTINES
  ILS04 00C4
  ILS02 00B3
  020C (HEX) IS THE EXECUTION ADDR

// JOB

LOG DRIVE  CART SPEC  CART AVAIL  PHY DRIVE
  0000  1111  1111  0002

V2 M07  ACTUAL 32K CONFIG 32K

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
// * DEFINE A DISK FILE AND INITIALIZE IT FOR A MACRO LIBRARY. *
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
// *DELETE* PURGE
*CART ID 1111 DB ADDR 2620 DB CNT 0050
*DFILE* UA PURGE 0005
*CART ID 1111 DB ADDR 2620 DB CNT 0050
*MACRO UPDATE*
BUILD 'PURGE'
0262 0005 ** LIBRARY END **
0000

ENDUP

UPDATE COMPLETED

// DEFINE 11 MACROS FOR THE MACRO LIBRARY NAMED 'PURGE'

// ASM

MACLIB PURGE

LIST

* ILLRQ MACRO
* PRINTS ILLEGAL REQUEST MESSAGE WHEN CALLED
* SMAC ILLRQ
* LIST ON ILLEGAL REQUEST LIST
00001 LIST ON ILLEGAL REQUEST LIST
00002 LIST ON ILLEGAL REQUEST LIST
00003 LIST ON ILLEGAL REQUEST LIST

* AUTO ERROR
* Generates a default error routine when ERRDR parameter is 0 or not passed.
* SMAC AUTOE AUTOMATIC ERROR GENERATOR
00001 DC ++1 CALL SEQ-ERROR ENTRY ADDR
00002 MDX ++3 SKIP AROUND ERROR SUBR
00003 DC ++4 ENTER HERE ON ERROR
00004 BSC I ++5 RETURN TO RETRY OPERATION

* READ MACRO
* Sets function code and calls specific device
* SMAC GENERAL READ MACRO
00001 LABEL READ DEVC, AREA, ERROR

* WRITE MACRO
* SMAC GENERAL WRITE MACRO
00001 LABEL DEVc 1, AREA, ERROR

Appendix A: General Examples of Macros and Macro Assembler Features
GENERAL WRITE MACRO

0001 LABEL WRITE DEVC,AREA,ERROR
MEND

0001 LABEL DEVC 3,AREA,ERROR
MEND

DISK MACRO

* Generates a LIBF call to DISKN.
* To test for DISKN busy you must call the DISK
* Macro directly with a func code of 0.

* Label = Label
* Func = 1 for read, 3 for write and 0 for test
* Area = I/O area address or label
* Error = optional users error routine

SMAC DISK CALL GENERATOR

0001 LABEL DISK FUNC,AREA,ERROR
0002 LABEL LIBF DISK CALL DISK SUBR
0003 AIF (FUNC EQ 1),READ TEST FOR READ FUNC
0004 AIF (FUNC EQ 3),WRITE TEST FOR WRITE FUNC
0005 ILLRQ ILLEGAL REQUEST,ABORT CALL
0006 AGO END TERMINATE MACRO
0007 READ ANOP DC /1000 READ FUNC CODE
0008 AGO AREA GO ASSEMBLE I/O AREA ADDR
0009 WRITE ANOP DC /3000 WRITE FUNC CODE
0010 AGO AREA GO ASSEMBLE I/O AREA ADDR
0011 TEST ANOP DC /0000 TEST FUNC CODE
0012 DC **2 I/O AREA ADDR
0013 MDX **2 BRANCH TO CONTINUE BUSY TEST
0014 MDX **2 BRANCH AROUND I/O AREA
0015 BSS 2 DUMMY I/O AREA
0016 AGO END EXIT MACRO
0017 AREA ANOP
0018 DC AREA-1 I/O AREA ADDRESS
0019 QERR SET DEFAULT ERROR CK FOR DEFAULT ERROR
0020 AIF (QERR EQ 0),AUTOE
0021 DC ERROR USER SPECIFIED ERROR PARAM
0022 AGO END EXIT MACRO
0023 AUTOE ANOP
0024 AUTOE GENERATE ERROR SUBR
0025 END ANOP MEND

CARD MACRO

* Generates a LIBF call to CARD1 subroutine.
* To test for CARD1 busy you must call the CARD
* Macro directly with a func code of 0.

* Label = Label
* Func = 1 for read, 3 for write and 0 for test
* Area = I/O area address or label
* Error = optional users error routine

SMAC CARD CALL GENERATOR

0001 LABEL CARD FUNC,AREA,ERROR
0002 LABEL LIBF CARD1 CALL CARD SUBR
0003 AIF (FUNC EQ 1),READ TEST FOR READ FUNC
0004 AIF (FUNC EQ 3),WRITE TEST FOR WRITE FUNC
0005 ILLRQ ILLEGAL REQUEST,ABORT CALL
0006 AGO END TERMINATE MACRO
0007 READ ANOP DC /1000 READ FUNC CODE
0008 AGO AREA GO ASSEMBLE I/O AREA ADDR
0009 WRITE ANOP

72 1130/1800 Macro Assembler Programming
# Macros and Macro Assembler Features

**APPENDIX A: GENERAL EXAMPLES OF MACROS AND MACRO ASSEMBLER FEATURES**

### PRINT MACRO
- Generates a LIBF call to PRNT3 subroutine.
- To test for PRNT3 busy, you must call the print macro directly with a FUNC code of 0.
- **LABEL** = LABEL
- **FUNC** = 3 for PRINT and 0 for TEST
- **AREA** = I/O area address or label
- **ERROR** = Optional users error routine

### CONVERT MACRO
- Handles Hollerith to printer code via 'HOLPR'.
- 1 BINARY TO 6 HOLLERITH CHARS via 'BINDC'.
- 6 HOLLERITH TO 1 BINARY CHAR via 'DCBIN'.
- **WHERE**
  - **AREA** = INPUT AREA (OUTPUT FOR BINDC)
  - **RT** = CONVERSION RT, HOLPR, BINDC OR DCBIN
  - **+PRNTR** = OUTPUT CODE FOR HOLPR
  - **0** for console, 1 for 1403 printer
  - **OUTPT** = OUTPUT AREA FOR HOLPR
  - **CHCT** = CHARACTER COUNT FOR HOLPR

### SMAC CODE CONVERSION CALL

---

**LABEL** CNVRT AREA, RT, +PRNTR, OUTPT, CHCT

---

### PRINT MACRO

```assembly
DC +/2000 WRITE FUNC CODE SMAC1610
AGO AREA GO ASSEM I/O AREA ADDR SMAC1620
SMAC1630
TEST ANOP SMAC1640
DC +/0000 TEST FUNC CODE SMAC1650
MDX +3 BRANCH TO CONTINUE BUSY TEST SMAC1650
SMAC1660
END DC SMAC1670
END ANOP SMAC1680
SMAC1690
DC AREA+ AREA ADDRESS SMAC1700
QERR SET ERROR CK FOR DEFAULT ERROR SMAC1710
SMAC1720
AIF (QERR EQ 0) AUTOE SMAC1730
SMAC1740
DC ERROR USER SPECIFIED ERROR PARAM SMAC1750
SMAC1760
AUTOE ANOP SMAC1770
SMAC1780
AUTOE GENERATE ERROR SUBR SMAC1790
SMAC1800
END ANOP SMAC1810
MEND SMAC1820
```

---

### CONVERT MACRO

```assembly
00001 LABEL PRINT FUNC, AREA, ERROR SMAC1830
SMAC1840
00002 LABEL LIBF PRNT3 CALL PRINT SUBR SMAC1850
SMAC1900
00003 AIF (FUNC EQ 3), WRITE TEST FOR WRITE FUNC SMAC1910
SMAC1920
00004 AIF (FUNC EQ 0), TEST TEST FOR TEST FUNC SMAC1930
SMAC1940
00005 AGO END TERMINATE MACRO SMAC1950
SMAC1960
00006 WRITE ANOP SMAC1970
SMAC1980
00007 DC +/2000 WRITE FUNC CODE SMAC1990
SMAC2000
00008 AGO AREA GO ASSEM I/O AREA ADDR SMAC2010
SMAC2020
00009 TEST ANOP SMAC2030
SMAC2040
00010 DC +/0000 TEST FUNC CODE SMAC2050
SMAC2060
00011 MDX +3 BRANCH TO CONTINUE BUSY TEST SMAC2070
SMAC2080
00012 AGO END EXIT MACRO SMAC2090
SMAC2100
SMAC2110
END ANOP SMAC2120
SMAC2130
SMAC2140
SMAC2150
SMAC2160
SMAC2170
SMAC2180
END ANOP SMAC2190
SMAC2200
SMAC2210
SMAC2220
SMAC2230
SMAC2240
SMAC2250
SMAC2260
SMAC2270
SMAC2280
```

---

### SMAC CODE CONVERSION CALL

```assembly
00001 LABEL CNVRT AREA, RT, +PRNTR, OUTPT, CHCT SMAC2290
SMAC2300
00002 AGO RT GEN APPROPRIATE CODE SMAC2310
SMAC2320
SMAC2330
HOLPR ANOP SMAC2340
SMAC2350
DC +/PRNTR CONVERSION CODE SMAC2360
SMAC2370
SMAC2380
DC AREA INPUT SMAC2390
SMAC2400
```

---

### Appendi x A: General Examples of Macros and Macro Assembler Features

73
DC OUTPT OUTPUT                 SMAC2340
DC CHCT CHAR COUNT              SMAC2350
AGO END                         SMAC2360
RT ANOP STOP HERE FOR BINDC AND DCBIN SMAC2370
DC AREA I/O AREA                SMAC2380
END MEND                        SMAC2390

* BLOCK MACRO                   SMAC2410
* GENERATE A BLOCK OF CONSTANTS. SMAC2420
* A = CONSTANT TO FILL BLOCK WITH SMAC2430
* B = NUMBER OF CONSTANTS TO GENERATE SMAC2440
* SMAC CONSTANT DATA BLOCK       SMAC2450
LABEL BLOCK A,B GENERATOR       SMAC2460

LIST OFF                        SMAC2480
COUNT SET B                     SMAC2500
DATA ANOP                       SMAC2510
LIST                            SMAC2520
DC A                            SMAC2530
LIST OFF                        SMAC2540
COUNT SET COUNT-1               SMAC2550
AIFB (COUNT GT 0),DATA          SMAC2560
LIST                            SMAC2570
MEND                            SMAC2580

* INREG MACRO                   SMAC2610
* LOAR AN INDEX REGISTER WITH A VALUE SMAC2620
* A = THE VALUE, B = THE REGISTER NUMBER SMAC2630
* SMAC SET REGISTER B TO VALUE A SMAC2640
A INREG B                       SMAC2650
AIF (A GT 127),LONG             SMAC2660
AIF (A LT -128),LONG            SMAC2670
LDX B A                         SMAC2680
AGO END                         SMAC2700
LONG ANOP                       SMAC2710
LDX LB A                        SMAC2720
END ANOP MEND                   SMAC2740

* DECK MACRO                    SMAC2760
* DECREMENT A COUNTER FOR LOOP CONTROL. SMAC2770
* A = STORAGE ADDRESS OF COUNTER, IF ANY SMAC2780
* B = REGISTER NUMBER            SMAC2800
* C = THE DECREMENT VALUE        SMAC2810
* SMAC DECREMENT COUNTER         SMAC2820
A DECK B,C                      SMAC2830
AIF (B EQ 0),LONG TEST FOR INDEXING SMAC2850
MDX B -C INDEXED COUNTER        SMAC2860
AGO END                         SMAC2870
LONG ANOP                       SMAC2880
MDX L A,-C NON-INDEXED COUNTER  SMAC2890
END ANOP MEND                   SMAC2910
SAMPLE PROGRAM

PROGRAM WHICH

A - READS ONE SECTOR DATA FILE WHICH CONTAINS INTEGER DATA FROM DISK

B - READS EVERY 5TH ENTRY FROM CARD

C - PUNCHES EVERY 3RD ENTRY TO CARD WITH SECOND ENTRY

D - ADDS ONE TO EVERY OTHER ENTRY BEGINNING WITH SECOND ENTRY

E - WRITES MODIFIED FILE TO PRINTER, 5 ENTRIES PER LINE

PROGRAM WHICH

A - READ THE DISK DATA FILE

B - READ DATA CARDS AND STORE IN EVERY 5TH ENTRY

C - PUNCHES EVERY 3RD ENTRY TO CARD

D - ADDS ONE TO EVERY OTHER ENTRY BEGINNING WITH SECOND ENTRY

E - WRITES MODIFIED FILE TO PRINTER, 5 ENTRIES PER LINE

Appendix A: General Examples of Macros and Macro Assembler Features 75
A I F  (0  EQ  0),TEST
+TEST
ANOP
0013 0 0000
+ DC  /0000
+ M D X  *-3
+ A G O  END
+END
ANOP
CNVRT DATA,DCBIN CONVERT 1ST 5 CARD COLUMN SMAC3200
0015 20 040C2255
+ LIBF DCBIN
+ A G O  DCBIN
+DCBIN
ANOP
STOP
0016 1 01A5
+ DC  DATA
0017 01 D6000062
STO L2 FILE SAVE AS 5TH CHARACTER SMAC3210
DECR 2,5 DECREMENT LOOP CONTROL SMAC3220
0019 0 72FB
+ MDX 2 -5
+ A G O  END
+END
ANOP
001A 0 70EF
* B  STEP8 CONTINUE STEP B SMAC3230
+ READ CARD,DATA PRIME 1442 FOR PUNCHING SMAC3240
+ CARD L1,DATA READ ONE BLANK CARD SMAC3250
001B 20 03059131
+ A I F  CARD1
+ A I F  (1  EQ  1),READ
001C 0 1000
+ DC  /1000
+ A G O  DATA
+DATA
ANOP
001D 1 01A4
+ QERR
SET
+ A I F  (QERR EQ 0),AUTOE
+A U T O E  ANOP
+A U T O E
001E 1 0020
+ DC  **1
001F 0 7003
+ M D X  *+3
0020 0 0000
+ DC
0021 01 4C800020
+ B S C  I *+3
+END
ANOP
319 INREG 2 SET LOOP CONTROL SMAC3260
+ A I F  (319 GT 127),LONG
+LONG
ANOP
+ LDX L2 319
+END
ANOP

* C - CONVERT AND PUNCH EVERY 3RD ENTRY SMAC3280
* SMAC3290
+ SMAC3300
+ SMAC3310
+ SMAC3320
0025 01 C6000061
STEPC LD L2 FILE-1 CONVERT INTEGER TO CARD 0 BUSY TEST SMAC3320
0027 20 03059131
+ LIBF CARD1
+ A I F  (0  EQ  1),READ
+ A I F  (0  EQ  3),WRITE
+ A I F  (0  EQ  0),TEST
+TEST
ANOP
0028 0 0000
+ DC  /0000
+ M D X  *-3
+ A G O  END
+END
ANOP
CNVRT DATA,BINDC CARD CODE SMAC3330
002A 20 02255103
+ LIBF BINDC
+ A G O  BINDC
+BINDC
ANOP
STOP
002B 1 01A5
+ DC  DATA
WRITE CARD,DATA PUNCH DATA WORD SMAC3340
002C 20 03059131
+ LIBF CARD1
+ A I F  (3  EQ  1),READ
+ A I F  (3  EQ  3),WRITE
+WRITE
ANOP
002D 0 2000
+ DC  /2000
+ A G O  DATA
+DATA
ANOP
002E 1 01A4
+ DC  DATA-1
0000
+QERR
SET
+ A I F  (QERR EQ 0),AUTOE

76 1130/1800 Macro Assembler Programming
Appendix A: General Examples of Macros and Macro Assembler Features 77

+AUTOE ANOP
+ AUTOE
+ DC  ++1
0030 0 7003
+ MDX  **3
0031 0 0000
+ DC  **-2
0032 01 4C800031
+ BSC 1 * -3
+END ANOP
DECR 2,3 DECR LOOP CONTROL SMAC3350
+ AIF (2 EQ 0),LONG
0034 0 72FD
+ MDX 2 -3
+ AGO END
+END ANOP
0035 0 70EF
B STEPC CONTINUE STEP C SMAC3360
319 INREG 2 SET LOOP CONTROL SMAC3370
+ AIF (319 GT 127),LONG
+LONG ANOP
0036 00 6600013F
+ LDX L2 319
+END ANOP

* D - ADD ONE TO EVERY OTHER ENTRY * SMAC3390
* SMAC3400
+ SMAC3410
0038 01 C6000062
STEPD LD L2 FILE ADD 1 TO EVERY OTHER SMAC3420
003A 0 8025
A ONE DATA ENTRY STARTING WITH SMAC3430
0038 01 D6000062
STO L2 FILE THE 2ND ENTRY SMAC3440
003D 0 72EF
+ MDX 2 -2
+ AGO END
+END ANOP
003E 0 70F9
B STEPD CONTINUE STEP D SMAC3450
-320 INREG 1 SET OUTER LOOP COUNT SMAC3460
+ AIF (-320 GT 127),LONG
+ AIF (-320 LT -128),LONG
+LONG ANOP
003F 00 6500FEC0
+ LDX L1 -320
+END ANOP
5 INREG 2 SET A TO 5 SMAC3470
+ AIF (5 GT 127),LONG
+ AIF (5 LT -128),LONG
0041 0 6205
+ LDX 2 5
+ AGO END
+END ANOP

* E - WRITE FILE TO 1403 * SMAC3500
* SMAC3510
0042 01 C50001A3
STEP2 LD L1 FILE+321 FETCH DATA WORD SMAC3520
0044 20 02255103
CNVRT DATA+7,BINDC CONVERT DATA TO 1403 SMAC3530
0045 1 01AC
+ LIBF BINDC
+ AGO BINDC
+BINDC ANOP STOP
0046 20 085935D9
+ CNVRT DATA+7,HOLPR,1,PRINT,6 CODE SMAC3540
+ LIBF HOLPR
+ AGO HOLPR
+HOLPR ANOP
0047 0 0001
+ DC /1
0048 1 01AC
+ DC DATA+7
0049 1 01F6
+ DC PRINT
004A 0 0006
+ DC 6
+ AGO END
+ AGO END
0049
OUTPT EQU 4 -2 SMAC3560
004A 0 74040049
MDX L OUTPT,4 ADJUST BUFFER ADDR SMAC3570
STEPF DECR 2,1 DECR LINE COUNT SMAC3580
+ AIF (2 EQ 0),LONG
004D 0 72FF
+ MDX 2 -1
+ AGO END
+END ANOP
004E 0 700E
B STEPG CONTINUE TO FILL BUFFER SMAC3590
OUTPT DECR 0,20 RESET BUFFER ADDR,NEW LINE SMAC3600
+ AIF (0 EQ 0),LONG
+LONG ANOP
004F 01 74EC0049 + MDX L OUTPT,-20
+END
+ANOP
+PRINT 0 BUSY TEST SMAC3610
0051 20 176558F3 + LIBF PRNT3
+ AIF (0 EQ 3),WRITE
+ AIF (0 EQ 0),TEST
+END
+ANOP
+TEST
+WRITE
+PRINT
+PRINT
+PRINT LINE
0054 20 176558F3 + LIBF PRNT3
+ AIF (3 EQ 3),WRITE
+END
+ANOP
+WRITE
+ANOP
0055 0 2000 + DC /2000
+ AGO PRINT
0056 1 01F5 + DC PRINT-1
0000 +QERR SET
+ AIF (QERR EQ 0),AUTOE
+AUTCE ANOP
+AUTOE
0057 1 0059 + DC ++1
0058 0 7003 + MDX ++3
0059 0 000D + DC **9
005A 01 4C000059 + BSC 1 **3
+END
+ANOP
% INREG 2 RESET LINE COUNT SMAC3630
+ AIF (5 GT 127),LONG
+ AIF (5 LT -128),LONG
005C 0 6205 + LDX 2 5
+ AGO END
+END
ANOP
005D 0 7101 STEP MDX 1 1 INCR DATA POINTER SMAC3640
005E 0 70E3 MDX STEP2 CONTINUE TO PRINT SMAC3650
005F 0 6038 EXIT EXIT SMAC3660
**CONSTANTS AND BUFFERS** SMAC3670
**CONSTANTS AND BUFFERS** SMAC3680
+ ONE DC 1 CONSTANT 1 SMAC3690
0060 0 0001 FILE EQU *+1 DATA FILE SECTOR ADDRESS SMAC3700
0062 31 00140 BSS 320 DATA AREA SMAC3710
0064 0 0050 DC 80 SMAC3720
0065 0 0050 DATA BSS 80 CARD BUFFER SMAC3730
01F5 0 0014 DC 20 SMAC3740
01F6 0 0000 END STEPA SMAC3750
01F6 0 7F7F + DC /7F7F
01F7 0 7F7F + DC /7F7F
01F8 0 7F7F + DC /7F7F
01FA 0 7F7F + DC /7F7F
01FB 0 7F7F + DC /7F7F
01FC 0 7F7F + DC /7F7F
01FD 0 7F7F + DC /7F7F
01FF 0 7F7F + DC /7F7F
0200 0 7F7F + DC /7F7F
0201 0 7F7F + DC /7F7F
0202 0 7F7F + DC /7F7F
0203 0 7F7F + DC /7F7F
0204 0 7F7F + DC /7F7F
0205 0 7F7F + DC /7F7F
0206 0 7F7F + DC /7F7F
0207 0 7F7F + DC /7F7F
0208 0 7F7F + DC /7F7F
0209 0 7F7F + DC /7F7F
020A 0 0000 END STEPA SMAC3790

000 OVERFLOW SECTORS SPECIFIED
000 OVERFLOW SECTORS REQUIRED

78 1130/1800 Macro Assembler Programming
014 SYMBOLS DEFINED
NO ERROR(S) AND NO WARNING(S) FLAGGED IN ABOVE ASSEMBLY

// XEQ L 1 N

*FILES1, FILE1
R 41 761E (HEX) WDS UNUSED BY CORE LOAD
LIBF TRANSFER VECTOR
PRTY 095C
HOLL 090C
PRNT3 0708
HOLPR 0774
BINDC 072C
DCBIN 0604
CARD1 05CA
DISK N 00F9

SYSTEM SUBROUTINES
ILS04 00C4
ILS02 00B3
ILS00 09AD

03C0 (HEX) IS THE EXECUTION ADDR
-00319 +00319 +00319 +00317 +00317
-00315 +00315 +00313 +00313 +00311
-00309 +00309 +00309 +00307 +00307
-00305 +00305 +00303 +00303 +00301
-00299 +00299 +00299 +00297 +00297
-00295 +00295 +00293 +00293 +00291
-00289 +00289 +00289 +00287 +00287
-00285 +00285 +00283 +00283 +00281
-00279 +00279 +00279 +00277 +00277
-00275 +00275 +00273 +00273 +00271
-00269 +00269 +00269 +00267 +00267
-00265 +00265 +00263 +00263 +00261
-00259 +00259 +00259 +00257 +00257
-00255 +00255 +00253 +00253 +00251
-00249 +00249 +00249 +00247 +00247
-00245 +00245 +00243 +00243 +00241
-00239 +00239 +00239 +00237 +00237
-00235 +00235 +00233 +00233 +00231
-00229 +00229 +00229 +00227 +00227
-00225 +00225 +00223 +00223 +00221
-00219 +00219 +00219 +00217 +00217
-00215 +00215 +00213 +00213 +00211
-00209 +00209 +00209 +00207 +00207
-00205 +00205 +00203 +00203 +00201
-00199 +00199 +00199 +00197 +00197
-00195 +00195 +00193 +00193 +00191
-00189 +00189 +00189 +00187 +00187
-00185 +00185 +00183 +00183 +00181
-00179 +00179 +00179 +00177 +00177
-00175 +00175 +00173 +00173 +00171
-00169 +00169 +00169 +00167 +00167
-00165 +00165 +00163 +00163 +00161
-00159 +00159 +00159 +00157 +00157
-00155 +00155 +00153 +00153 +00151
-00149 +00149 +00149 +00147 +00147
-00145 +00145 +00143 +00143 +00141
-00139 +00139 +00139 +00137 +00137
-00135 +00135 +00133 +00133 +00131
-00129 +00129 +00129 +00127 +00127
-00125 +00125 +00123 +00123 +00121
-00119 +00119 +00119 +00117 +00117
-00115 +00115 +00113 +00113 +00111
-00109 +00109 +00109 +00107 +00107
-00105 +00105 +00103 +00103 +00101
-00099 +00099 +00099 +00097 +00097
-00095 +00095 +00093 +00093 +00091
-00089 +00089 +00089 +00087 +00087
-00085 +00085 +00083 +00083 +00081
-00079 +00079 +00079 +00077 +00077
-00075 +00075 +00073 +00073 +00071
-00069 +00069 +00069 +00067 +00067

Appendix A: General Examples of Macros and Macro Assembler Features 79
*DFILE Statement 47-48, 61-62
*MACLIB Control Statement 2, 22
*MACRO UPDATE Control Statement 47
*OVERFLOW SECTORS Control Statement 1-2
*STOREDDATA Statement 47-48
*, Used to Designate Comment Statements 5
**LIBRARY END** 61-62
**MACRO END** 61-62
@ Sign 48
$ Sign 48
+ Sign
used as a macro expansion indicator, 7
used to indicate a positive number, 27
# Sign 48

ADD Statement 51, 52-53
Additional Records, Continuing
Calls to 12-13
AGO Pseudo-Op
use in label and blank parameter
checking, 30
AGO Pseudo-Op 17-18
special considerations in use, 18-19
AIF Pseudo-Op 15-16, 17
restriction of use, 29-30
AIFB Pseudo-Op 15-16
restriction of use, 29-30
special considerations in use, 18-19
Ampersand, Used as a Concatenator 23
equation, 24-25
ANOP Pseudo-Op 19-20
Apostrophe
restriction of use in the name of a
stored macro, 5
used in automatic name
generation, 22-23
Assemble if Back Pseudo-Op
Assemble if Pseudo-Op
Assembler Language, A symbolic
programming language.
Assembler-Language Statement, An
assembler-language instruction or a
pseudo-operation.
Assembler-Language Instruction, An
instruction that the Macro Assembler
can translate into exactly one
machine-language instruction.
Automatic Label Generation
See Automatic Name Generation.
Automatic Name Generation 1, 22-23.
The method by which different labels
can be generated during each
expansion of a macro instruction in
the same assembly.

Glossary—Index

special requirements on its use in
nested definitions, 57-58

Blank and Label Parameter Checking
Using AGO 30

BUILD Statement 48

Calling
macros, 7
Macro Update Program, 47
Calls
continuation of to additional
records, 12-23
macro, 7, 12

Character String
in message generation, 11-12
substitution for a parameter, 10-11

Characters, Special 6, 10, 47

Checking
for blank parameters, 28-29
for label and blank parameters
using AGO, 30-31
for unspecified parameters, 17-18
for unspecified parameters, a
macro, 34

Comma
lack of use in continuation of NAME
statement, 52
use in a macro instruction, 7-8

Comments
designating, 5, 58
listed within the expansion, 25
See also Remarks.
CONCAT Statement 49-50

Concatenation 2. The process by which
two things (such as two parts of an
instruction, or two macro libraries)
are logically joined together.
of macro instructions, 23-25
of macro libraries, 49-50

Concatenating a Multiplicity of
Libraries, 49-50

Concatenators
ampersand, 23
CONCAT statement, 49-50
period, 23

Conditional Assembly Pseudo-Ops 15-18
Continuing Calls to Additional
Records 12-13

Control Statement 47. A statement that
provides instructions to some part of
the Disk Management Program or Disk
Utility Program.
continuation of, 47

Copying an Existing Library 49
Core Storage
See Main Storage.
Creating a Language 1,37-45

DC-Generating Macro 34-35

Defining
a language, 1
a macro, 1,6
a macro during a Macro Update Run, 51-53
a macro instruction, 5

Definition Prototype Statement 5-6,53.
The statement in a macro definition that specifies the
op code and parameters of the macro.

printing of, 6
DELETE Statement 55-56
in automatic name generation in nested
definitions, 57-58
in renaming a library, 51
in updating a library, 50

Deleting a Macro from a Library 53
Deleting Statements from a Macro 55-56

Designating Comments 5,58

Disconnecting Concatenated Libraries 50

Disk Management Program (DMP) 47. A group
of 1800 MPX disk utility and
maintenance programs that operate
under control of the Batch-Processing
Monitor Supervisor.

Disk Monitor System, Version 2 (DM2), The
second version of an operating and
programming system that provides for
the continuous batch-processing
operation of the 1130.
estimating size required, 1-2
initializing, 2,47-48

Disk Utility Program (DUP) 47. A group of
1130 disk utility and maintenance
programs that operate under control of
the Supervisor.

Division Operator 27

DMP
See Disk Management Program.

DM2

DUP
See Disk Utility Program.

EJCT Pseudo-Op 15,17,18
END Statement 15,17,18
ENDUP Statement 58-59

EQU Statement 20
symbolic tag field in, 27

Error Flags 4,63-64

Error Messages 4,63,65-67

Estimating
size of N2, 2
size of N3, 2

Expansion 7. The coding generated when
the Macro Assembler encounters a macro
instruction; also, the process of
generating this coding.

Field Specifications on Macro Update
Control Statements 47

84 1130/1800 Macro Assembler Programming

Flags
error, 4,63-64
warning, 4,63

HDNG Pseudo-Op 15,17,18

Indirect Parameter Substitution 26,32-33.
The feature of the Macro Assembler
that allows different parameters on
the macro call statement to be
substituted for a specific
parameter in the macro expansion,
depending on some condition to be
inspected during assembly.

Initializing Disk Space 2,47-48

INSERT Statement 54,55
in automatic name generation in nested
definitions, 57-58
in renaming a library, 51
in updating a library, 50

Inserting Statements in a Macro 54-55,56
Restrictions on, 55,56

Insufficient Parameter Names 52

JOIN Statement 49-50

Joining Macro Libraries
logically, 2,22,49-50
physically, 49-50

Label Field Parameters 8

Label and Blank Parameter Checking
Using AGO 30

Label Generation
See Automatic Name Generation.

Language Creation 1,37-45

Library
See Macro Library.

LIB Statement 48

List Control Pseudo-Ops 6
LIST Pseudo-Op 15,17,18

Listing of
comment statements, 6
macro calls, 25
macro libraries, 56-57

LNAME 3,48

Logically Joining Macro
Libraries 2,22,49-50

MAC Statement 5,25,51,53

Macro Assembler, The translating program
that accepts as input
assembler-language instructions,
pseudo-operations, and macro
instructions.
error flags, 63-64
error messages, 63-65
language, 37-45
main-storage requirements, 3,4
performance, 4
purpose, 1
sample programs, 33-35,69-84
sign-off message, 63

warning flag, 63
Macro Calls 7, 12
continuation to additional records, 12
listing of, 6
nested, 3, 10, 23, 26, 32

Macro Definition 1, 2. A sequence of
instructions that define the op code
and parameters of a macro instruction
and the coding to be generated when
the macro is assembled.
during a macro update run, 51-53
nested, 3

Macro Expansion 7

Macro instruction 1. A source program
statement that, when encountered by
the Macro Assembler, causes a
predefined sequence of statements to
be assembled.
defining, 5
using, 7

Macro Library 3. A collection of macro
definitions, saved on disk, that can
be used by any program that references
that library.
copying, 49
disconnecting, 50
initializing, 3
joining logically, 49-50
joining physically, 48-49
maintaining, 3
naming, 47-48
specifying, 48

Macro Name 1, 5, 22

Macro Parameter Substitution 32-33

Macro Update Program 4, 47-62. A DMP or
DM2 program that allows you to
initialize and maintain macro
libraries.
calling, 47
control statement sequencing, 59
error messages, 65-67
making efficient use of, 59-60
sample program, 61-62

Macros
stored, 1, 2-3
temporary, 1-2
Main Storage Requirements
1130, 3-4
1800, 4

Making Efficient Use of the Macro
Update Program 59-60

MEND Statement 5, 51, 53
termination of AGO and AGOB
search, 17
termination of AIFB search, 16
Message Generation 11-12
Message Macro 34
Messages, Error 4, 63, 65-67
Move Macro 34

See Multiprogramming Executive
Operating System.

Multiple Concatenation 49-50

Multiprogramming Executive Operating
System (MPX), An operating system
for the 1800 that can control
processes and provide
multiprogramming and background
processing.

MUP
See Macro Update Program.

N2 1, 2
N3 1, 2
Name
of a library, 47-48
of a macro, 1, 5, 22
of a parameter, 5-6, 7, 52

Name Generation, Automatic 1, 22-23, 57-58

NAME Statement 47, 51-52
continuation of, 51-52
used in the sample program, 61-62

Name Subfield Left Blank on AIF and
AIFB Statements 16

Nested Macro Calls 3
in automatic name generation, 23
in continued calls, 32
in definitions defined during a macro
update run, 53
in indirect parameter
substitution, 26, 32
passing a set of parameters, 10
restrictions on, 3

Nested Macro Definitions 3. Macro
definitions that are defined
so that a call to one occurs
within the expansion of the other.
SELECT N considerations, 57-58

Obtaining a Listing of Macro
Libraries 56-57

Omitting
name subfields on AIF and AIFB
statements, 16
operand on AGO and AGOB statements, 17
parameters, 8-9

Op Code 5. That field of an
assembler-language statement that
specifies the operation to be carried
out by the CPU.

Operand Field 6

Operand Left Blank on AGO and AGOB
Statements 17

Optional Remarks 25

OUTPUT Statement 54

Parameters

blank, checking for, 28-29
in a macro instruction, 1
indirect substitution, 26, 32-33
insufficient names, 52
label field, 8
macro to check for unspecified
parameters, 34
name, 5-6, 7
omitting, 8-9
substituting a character string
for, 10-11
substitution, 6, 32-33
unspecified, checking for, 17-18

Performance of the Macro Assembler 4

Glossary-Index 85
Period
example of use as a concatenator, 24
in DEC and XPLC statements and
concatenation, 25
used as a concatenator, 23
used to designate comments, 58
Physically Joining Macro
Libraries 48-49
Plus Sign,
used as a macro expansion indicator, 7
used to indicate a positive number, 27
Printing
definition prototype statements, 6
inhibition of, 6,56-57
sequence numbers, 6,55,61-62
Pseudo-Operation 3. An assembler-language
statement that provides information
for the Macro Assembler rather than
generating executable code.
AGO, 17-18
AGOB, 17-18
AIF, 15-16,17
AIFB, 15-16
ANOP, 19-20
conditional assembly, 15-17
EJCT, 15,17,18
END, 15,17,18
HDNG, 15,17,18
LIST, 15,17,18
MEND, 15,17,18
PURG, 15,17,18
SET, 20-22
SPAC, 15,17,18
Pseudo-Ops
See Pseudo-Operation.
Punching Source Statements 54
PURG Statement 22
PURG vs. PURGE 54
PURGE Statement 53-54
Reclaiming Library Space 53-54
Records, Continuing Calls to
Additional 12-13
Remarks
optional, 25
retaining, 5
RENAME Statement 51
Renaming a Macro in a Library 51
Restrictions
on AIF, AIFB, and SET pseudo-ops, 29-30
on inserting statements, 55,56
on nested macro calls, 3
on nested macro definitions, 3
on parameter names, 6,11,51
Retaining Remarks 5
Sample Programs
creating a language, 37-45
Macro Assembler, 33-35
Macro Update Program, 61-62
1130 Macro Assembler, 69-84
SELECT Statement 56-57
SELECT N as Used with Automatic Name
Generation in a Nested
Definition 57-58
SELECT P as Used with NAME
Statement 59
Semicolon, Used in Indirect Parameter
Substitution 26-32
Sense Switch 2 6,56-57
Sequencing MUP Control Statements 59
Sequence Numbers, Printing of 6,55,61-62
SET Pseudo-Op 20-22
restriction of use, 29-30
symbolic tag field in, 27
Sign-Off Message 63
Signs,
@, 48
$, 48
+, 7,27
#, 48
SMAC Statement 5,25,51,53
Source Statements, Punching
of 54
Space, Reclaiming 53-54
SPAC Pseudo-Op 15,17,18
Special Characters 6,10,47
Special Requirements on the Use of
Automatic Name Generation in
Nested Definitions 57-58
Specifying the Macro Library 48
Statements
deleting from a macro, 55-56
inserting into a macro, 54-55,56
See also the name of a particular
statement or the function it performs.
Stored Macro 1,2-3,5. A macro
instruction that is saved on disk
and can be used by any program.
Substituting a Character String for a
Parameter 10-11
in message generation, 1-12
Substitution
indirect parameters, 26,32-33
parameter, 6,32-33
Symbol Table, for 1130 DM2 3-4
Symbolic Tag Field 27-28
Tag Field, Symbolic 27-28
Temporary Macro 1-2,5. A macro
instruction that is defined for use
only during one specific assembly.
Terminating a Macro Update Run 58-59
Text 6,7. A list of assembler-language
instructions, calls to other macros,
and/or pseudo-ops to be generated
when a macro call is encountered.
Truncation of Information Following
Column 71 6,10,25

86 1130/1800 Macro Assembler Programming
Unspecified Parameter Checking 17-18
UPDATE COMPLETED 59, 62
UPDATE Statement 50
Updating a Macro in a Library 50
See also DELETE Statement and INSERT Statement.
Using a Macro Instruction 7

Warning Flag 4, 63
X, Used to Designate an Addition of a Statement to a Macro or Library 55, 61-62
• Please comment on the usefulness and readability of this book, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you want a reply, be sure to give your name and address.

Name ___________________________________ Occupation ______________________________
Address ________________________________________________________________

• Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
YOUR COMMENTS, PLEASE...

This publication is one of a series which serves as reference sources for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.
• Please comment on the usefulness and readability of this book, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you want a reply, be sure to give your name and address.

Name __________________________ Occupation __________________________
Address __________________________

• Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
YOUR COMMENTS, PLEASE...

This publication is one of a series which serves as reference sources for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.
Please comment on the usefulness and readability of this book, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you want a reply, be sure to give your name and address.

Name ___________________________ Occupation ___________________________

Address ___________________________

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
YOUR COMMENTS, PLEASE...

This publication is one of a series which serves as reference sources for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

IBM
International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]
This technical newsletter provides pages for IBM 1130/1800 Macro Assembler Programming (Order Number GC26-3733-0). Pages to be replaced are listed below:

iii-iv
63-66

A change to the text is indicated by a vertical line in the left margin. A revised illustration is indicated by a bullet to the left of the caption.

Summary of Amendments

The Macro Assembler Error Flags table and Error Codes and Messages table have been updated to reflect the changes made in Version 3 of the 1800 Multiprogramming Executive Operating System.

Please put this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Programming Publications. Dept. D78, San Jose, Calif. 95114

PRINTED IN U.S.A.