Microsoft MACRO-80 ASSEMBLER

CP/M® Version

Software Reference Manual

for HEATH/ZENITH 8-bit digital computer systems

HEATH COMPANY
BENTON HARBOR, MICHIGAN 49022

CP/M is a registered trademark of Digital Research

595-2666-02
Printed in the
United States of America

Technical consultation is available for any problems you may encounter in verifying proper operation of this product. We are sorry, but we are not able to evaluate or assist in the debugging of any programs you may develop with this product. For technical assistance, call:

(616) 982-3884 Application Software
(616) 982-3860 Operating Systems/Language Software

Consultation is available between 8:00 am and 4:30 pm (Eastern), on normal business days.

Zenith Data Systems
Software Consultation
Hilltop Road
St. Joseph, MI 49085

Portions of this Manual have been adapted from Microsoft publications or documents.

Table of Contents

Chapter One — Using the MACRO-80 Assembler

Chapter Two — Expression Evaluation

Chapter Three — Pseudo-Opcodes/Assembler Directives

Chapter Four — Macros and Block Pseudo-Operations

Chapter Five — Index

Chapter One

Using the MACRO-80 Assembler

OVERVIEW

The MACRO-80 Assembler is an 8080/Z80 Assembler with complete facilities for macro development.

In order to use the Assembler, a source program must first be written using an editor, such as ED. A MACRO-80 source program is composed of a series of statements. The format of each statement must follow a predefined format.

After the source program has been written, it must be assembled using the Macro Assembler. The result of this process will be a relocatable module. This module must then be linked using the Linking Loader. (See Section 3, "LINK-80", for information on the Linking Loader.) After the relocatable module has been linked, it can be executed.

In order to provide the Assembler with the information it needs to successfully assemble a source program, a command string must be input. This command string tells the Assembler where to find the source program, where to put the relocatable module and where to write the listing.

There are also several switches which can be set in the command string. Some of these switches are used to control the format of the listing file. A switch can also be set to allow the Assembler to assemble Z80 mnemonics.

FORMAT OF MACRO-80 SOURCE FILES

In general, MACRO-80 accepts a source file that is almost identical to source files for INTEL-compatible assemblers. Input source lines up to 132 characters in length are allowed.

MACRO-80 preserves lower-case letters in quoted strings and comments. All symbols, opcodes and pseudo-opcodes typed in as lower-case will be converted to upper-case.

Statements

Source files input to MACRO-80 consist of statements of the form:

[**label:** [:]] [**operator**] [**arguments**] [;**comment**]

It is not necessary that statements begin in column one. Multiple blanks or tabs may be used to improve readability.

If a label is present, it is the first item in the statement and is immediately followed by a colon (:). If it is followed by two colons, it is declared as PUBLIC. Therefore:

```
FOO:: RET
```

is equivalent to:

```
PUBLIC FOO
FOO: RET
```

The next item after the label (or the first item on the line if no label is present) is an operator. An operator may be an opcode (8080 or Z80 mnenomic), pseudo-op, macro call, or expression.

The evaluation order is as follows:

1. Macro call
2. Opcode/Pseudo-operation
3. Expression

Instead of flagging an expression as an error, the Assembler treats it as if it were a DB statement. The arguments following the operator will, of course, vary in form according to the operator.

A comment always begins with a semicolon and ends with a carriage return. A comment may be a line by itself or it may be appended to a line that contains a statement. Extended comments can be entered using the .COMMENT operation.

Symbols

MACRO-80 symbols may be of any length. However, only the first six characters are significant. The following characters are legal in a symbol:

```
A-Z    0-9    $      .      ?    @
```

The underline character is also legal in a symbol. A symbol may not start with a numeric digit. Lower case symbols are translated to upper case. If a symbol reference is followed by ## it is declared external.

Numeric Constants

The default base for numeric constants is decimal. This may be changed by the .RADIX pseudo-op. Any base from 2 (binary) to 16 (hexadecimal) may be selected. When the base is greater than 10, A-F are the digits following 9. If the first digit of the number is not numeric (i.e. A-F), the number must be preceded by a zero.

Numbers are 16-bit unsigned quantities. A number is always evaluated in the current radix unless one of the following special notations is used:

```
nnnnB       Binary
nnnnD       Decimal
nnnnO       Octal
nnnnQ       Octal
nnnnH       Hexadecimal
X'nnnn'     Hexadecimal
```

Overflow of a number beyond two bytes is ignored and the result is the low order 16-bits.

Strings

A string is comprised of zero or more characters delimited by quotation marks. Either single or double quotes may be used as string delimiters. The delimiter quotes may be used as characters if they appear twice for every character occurance desired. If there are zero characters between the delimiters, the string is a null string.

FORMAT OF COMMANDS

To run MACRO-80, type M80 followed by a carriage return. MACRO-80 will return the prompt "*", indicating it is ready to accept commands. The format of a MACRO-80 command string is:

objprog-dev:filename.ext,list-dev:filename.ext=source-dev:filename.ext

Where:

objprog-dev: The device on which the object program is to be written.

list-dev: The device on which the program listing is written.

source-dev: The device from which the source-program input to MACRO-80 is obtained. If a device name is omitted, it defaults to the currently selected drive.

filename.ext The file name and file name extension of the object program file, the listing file, and the source file. If the file name extensions are omitted, the the operating system will insert the default extensions.

The default file name extensions are:

source file	.MAC
relocatable object file	.REL
listing file	.PRN
cross reference file	.CRF

Either the object file or the listing file or both may be omitted. If neither a listing file nor an object file is desired, place only a comma to the left of the equal sign. If the names of the object file and the listing file are omitted, the default is the name of the source file.

Examples:

(NOTE: The asterisk represents the prompt from the Assembler.)

`*EXP.REL,EXP.PRN=EXP.MAC`	Assemble the program EXP.MAC and place the object file in EXP.REL and the list file in EXP.PRN.
`*=EXP`	Assemble the program EXP.MAC and place the object file in EXP.REL.
`*,LST:=EXP`	Assemble the program EXP.MAC, place the object file in EXP.REL and list on the device LST:.
`*SMALL,TTY:=TEST`	Assemble the program TEST.MAC, place the object file in SMALL.REL and list on TTY:.

MACRO-80 Switches

A number of different switches may be given in the MACRO-80 command string that will affect the format of the listing file. Each switch must be preceded by a slash (/):

Switch	Action
O	Print all listing addresses, etc. in octal.
H	Print all listing addresses, etc. in hexadecimal. (Default)
R	Force generation of an object file.
L	Force generation of a listing file.
C	Force generation of a cross reference file. (See Page 1-11, "Cross-Reference Facility".)
Z	Assemble Z80 (Zilog format) mnemonics.
I	Assemble 8080 mnemonics. (Default)
P	Each /P allocates an extra 256 bytes of stack space for use during assembly. Use /P if stack overflow errors occur during assembly. Otherwise, it is not needed.

Switch	Action
/M	Initialize Block Data Areas. If the programmer wants the area that is defined by the DS (Define Space) pseudo-op initialized to zeros, then the programmer should use the /M switch in the command line. Otherwise, the space is not guaranteed to contain zeros. That is, DS does not automatically initialize the space to zeros.
/X	The presence or absence of /X in the command line sets the initial current mode and the initial value of the default for listing or suppressing lines in false conditional blocks. /X sets the current mode and initial value of default to not-to-list. No /X sets current mode and initial value of default to list. Current mode determines whether false conditionals will be listed or suppressed. The initial value of the default is used with the .TFCOND pseudo-op so that .TFCOND is independent of .SFCOND and .LFCOND. If the program contains .SFCOND or .LFCOND, /X has no effect after .SFCOND or .LFCOND is encountered until a .TFCOND is encountered in the file. So /X has an effect only when used with a file that contains no conditional listing pseudo-ops or when used with .TFCOND.

The following chart illustrates the effects of the three pseudo-ops when encountered under /X and under no /X.

PSEUDO-OP	NO /X	/X
(none)	ON	OFF
.	.	.
.	.	.
.	.	.
.SFCOND	OFF	OFF
.	.	.
.	.	.
.	.	.
.LFCOND	ON	ON
.	.	.
.	.	.
.	.	.
.TFCOND	OFF	ON
.	.	.
.	.	.
.	.	.
.TFCOND	ON	OFF
.	.	.
.	.	.
.	.	.
.SFCOND	OFF	OFF
.	.	.
.	.	.
.	.	.
.TFCOND	OFF	ON
.TFCOND	ON	OFF
.	.	.
.	.	.
.	.	.
.TFCOND	OFF	ON

Examples:

(NOTE: The asterisk represents the prompt from the Assembler.)

`*=TEST/L`	Compile TEST.MAC with object file TEST.REL and listing file TEST.PRN
`*LAST,LAST/C=MOD1`	Compile MOD1.MAC with object file LAST.REL and cross reference file LAST.CRF for use with CREF-80

Symbol Table Listing

In the symbol table listing, all the macro names in the program are listed alphabetically, followed by all the symbols in the program, listed alphabetically. After each symbol, a tab is printed, followed by the value of the symbol. If the symbol is Public, an I is printed immediately after the value. The next character printed will be one of the following:

Character	Definition
U	Undefined symbol.
C	COMMON block name. (The "value" of the COMMON block is its length (number of bytes) in hexadecimal or octal.)
*	External symbol.
<space>	Absolute value.
'	Program Relative value.
"	Data Relative value.
!	COMMON Relative value.

MACRO-80 ERROR MESSAGES

MACRO-80 errors are indicated by a one-character flag in column one of the listing file. If a listing file is not being printed on the terminal, each erroneous line is also printed or displayed on the terminal.

Error Codes

A Argument error —
Argument to pseudo-op is not in correct format or is out of range (.PAGE 1; .RADIX 1; PUBLIC 1; STAX H; MOV M,N; INX C).

C Conditional nesting error —
ELSE without IF, ENDIF without IF, two ELSEs on one IF.

D Double Defined symbol —
Reference to a symbol which is multiply defined.

E External error —
Use of an external illegal in context (e.g., FOO SET NAME ; MVI A,2-NAME).

M Multiply Defined symbol —
Definition of a symbol which is multiply defined.

N Number error —
Error in a number, usually a bad digit (e.g., 8Q).

O Bad opcode or objectionable syntax —
ENDM, LOCAL outside a block; SET, EQU or MACRO without a name; bad syntax in an opcode (MOV A:); or bad syntax in an expression (mismatched parenthesis, quotes, consecutive operators, etc.).

P Phase error —
Value of a label or EQU name is different on pass 2.

Q Questionable —
Usually means a line is not terminated properly. This is a warning error (e.g. MOV A,B,).

R Relocation —
Illegal use of relocation in expression, such as abs-rel. Data, code and COMMON areas are relocatable.

U Undefined symbol —
A symbol referenced in an expression is not defined. (For certain pseudo-ops, a V error is printed on pass 1 and a U on pass 2.)

V Value error —
On pass 1 a pseudo-op which must have its value known on pass 1 (e.g., .RADIX, .PAGE, DS, IF, IFE, etc.), has a value which is undefined later in the program, a U error will not appear on the pass 2 listing.

Error Messages:

'No end statement encountered on input file'
No END statement: either it is missing or it is not parsed due to being in a false conditional, unterminated IRP/IRPC/REPT block or terminated macro.

'Unterminated conditional'
At least one conditional is unterminated at the end of the file.

'Unterminated REPT/IRP/IRPC/MACRO'
At least one block is unterminated.

[xx] [No] Fatal error(s) [,xx warnings]
The number of fatal errors and warnings. The message is listed on the console and in the list file.

Chapter Two

Expression Evaluation

OVERVIEW

In most cases, the operand field of a given opcode may be coded as an operand expression. Such expression is a string of integers, symbols and characters. This character string is combined using certain operators.

The symbols used in the expression can be expressed in several modes. A symbol can also be classified as either external or not external.

Additionally, 8080 opcodes can be used as valid one-byte operands.

ARITHMETIC AND LOGICAL OPERATORS

The following operators are allowed in expressions and are listed in descending order of precedence.

NUL

LOW, HIGH

***, /, MOD, SHR, SHL**

Unary Minus

+,

EQ, NE, LT, LE, GT, GE

NOT

AND

OR, XOR

Parentheses are used to change the order of precedence. During evaluation of an expression, as soon as a new operator is encountered that has precedence less than or equal to the last operator encountered, all operations up to the new operator are performed. That is, subexpressions involving operators of higher precedence are computed first.

All operators except "+", "−", "*", "/" must be separated from their operands by at least one space.

The byte isolation operators (HIGH, LOW) isolate the high- or low-order eight bits of an Absolute 16-bit value. If a relocatable value is supplied as an operand, HIGH and LOW will treat it as if it were relative to location zero.

MODES

All symbols used as operands in expressions are in one of the following modes:

Absolute
Data Relative
Program (Code) Relative
COMMON

Symbols assembled under the ASEG, CSEG (default), or DSEG pseudo-ops are in Absolute, Code Relative or Data Relative mode respectively.

The number of COMMON modes in a program is determined by the number of COMMON blocks that have been named with the COMMON pseudo-op. Two COMMON symbols are not in the same mode unless they are in the same COMMON block.

In any operation other than addition or subtraction, the mode of both operands must be Absolute.

If the operation is addition, the following rules apply:

1. At least one of the operands must be Absolute.

2. Absolute + <mode> = <mode>

If the operation is subtraction, the following rules apply:

1. <mode> − Absolute = <mode>

2. <mode> − <mode> = Absolute

   where the two <mode>s are the same.

Each intermediate step in the evaluation of an expression must conform to the above rules for modes, or an error will be generated. For example, if FOO, BAZ and ZAZ are three Program Relative symbols, the expression:

```
FOO + BAZ - ZAZ
```

will generate an R error because the first step (FOO + BAZ) adds two relocatable values. (One of the values must be Absolute.)

This problem can always be fixed by inserting parentheses.

```
FOO + (BAZ - ZAZ)
```

is legal because the first step (BAZ − ZAZ) generates an Absolute value that is then added to the Program Relative value, FOO.

Externals

Aside from its classification by mode, a symbol is either External or not External. An External value must be assembled into a two-byte field. (Single-byte Externals are not supported.)

The following rules apply to the use of Externals in expressions:

1. Externals are legal only in addition and subtraction.
2. If an External symbol is used in an expression, the result of the expression is always External.
3. When the operation is addition, either operand (but not both) may be External.
4. When the operation is subtraction, only the first operand may be External.

Opcodes as Operands

8080 opcodes are valid one-byte operands. Note that only the first byte is a valid operand.

For example:

```
MVI   A,(JMP)
MVI   B,(RNZ)
MVI   C,MOV A,B
```

Errors will be generated if more than one byte is included in the operand — such as (CPI 5), (LXI B,LABEL1) or (JMP LABEL2).

Opcodes used as one-byte operands need not be enclosed in parentheses.

NOTE: Opcodes are not valid operands in Z80 mode.

Chapter Three

Pseudo-Opcodes/Assembler Directives

Overview

Within the Macro-80 Assembler there exists a set of instructions known as pseudo-opcodes or assembler directives. These instructions represent commands to the Assembler. They are called pseudo because although they are coded into the source program, they are not translated as instructions.

The following chapter explains the form and usage of the available pseudo-opcodes.

PSEUDO-OPCODES

ASEG

ASEG

ASEG sets the location counter to an absolute segment of memory. The location of the absolute counter will be that of the last ASEG (default is 0), unless an ORG is done after the ASEG to change the location. The effect of ASEG is also achieved by using the code segment (CSEG) pseudo operation and the /P switch in LINK-80.

COMMON

COMMON /<block name>/

COMMON sets the location counter to the selected common block in memory. The location is always the initial address of the common area so that compatibility with the FORTRAN COMMON statement is maintained. If <block name> is omitted or consists of spaces, it is considered to be blank common.

CSEG

CSEG

CSEG sets the location counter to the code relative segment of memory. The location will be that of the last CSEG (default is 0), unless an ORG is done after the CSEG to change the location. CSEG is the default condition of the assembler.

Define Byte

DB <exp>[,<exp>. . .]

DB <string>[<string>. . .]

The arguments to DB are either expressions or strings. DB stores the values of the expressions or the characters of the strings in successive memory locations beginning with the current location counter.

Expressions must evaluate to one byte. (If the high byte of the result is 0 or 255, no error is given; otherwise, an A error results.)

Strings of three or more characters may not be used in expressions (i.e., they must be immediately followed by a comma or the end of the line). The characters in a string are stored in the order of appearance, each as a one-byte value with the high order bit set to zero.

Example:

```
0000'   41 42          DB      'AB'
0002'   42             DB      'AB' AND 0FFH
0003'   41 42 43       DB      'ABC'
```

Define Character

DC <string>

DC stores the characters in <string> in successive memory locations beginning with the current location counter. As with DB, characters are stored in order of appearance, each as a one-byte value with the high order bit set to zero. However, DC stores the last character of the string with the high order bit set to one.

An error will result if the argument to DC is a null string.

Define Space

DS <exp>

DS reserves an area of memory. The value of <exp> gives the number of bytes to be allocated. All names used in <exp> must be previously defined (i.e., all names known at that point on pass 1).

Otherwise, a V error is generated during pass 1 and a U error may be generated during pass 2. If a U error is not generated during pass 2, a phase error will probably be generated because the DS generated no code on pass 1.

DSEG

DSEG

DSEG sets the location counter to the Data Relative segment of memory. The location of the data relative counter will be that of the last DSEG (default is 0), unless an ORG is done after the DSEG to change the location.

Define Word

DW <exp>[,<exp>. . .]

DW stores the values of the expressions in successive memory locations beginning with the current location counter. Expressions are evaluated as 2-byte (word) values.

END

END [<exp>]

The END statement specifies the end of the program. If <exp> is present, it is the start address of the program. If <exp> is not present, then no start address is passed to LINK-80 for that program.

ENTRY/PUBLIC

ENTRY <name>[,<name>. . .]

PUBLIC <name>[,<name>. . .]

ENTRY or PUBLIC declares each name in the list as internal and therefore available for use by this program and other programs to be loaded concurrently. All of the names in the list must be defined in the current program or a U error results. An M error is generated if the name is an external name or common-blockname.

EQU

<name> EQU <exp>

EQU assigns the value of <exp> to <name>. If <exp> is external, an error is generated. If <name> already has a value other than <exp>, an M error is generated.

EXT/EXTRN

EXT <name>[,<name>. . .]

EXTRN <name>[,<name>. . .]

EXT or EXTRN declares that the name(s) in the list are external (i.e., defined in a different program). If any item in the list references a name that is defined in the current program, an M error results. A reference to a name where the name is followed immediately by two pound signs (e.g., NAME##) also declares the name as external.

INCLUDE

INCLUDE <filename>

The INCLUDE pseudo-op assembles source statements from an alternate source file into the current source file. Use of INCLUDE eliminates the need to repeat an often-used sequence of statements in the current source file. The pseudo-ops INCLUDE, $INCLUDE and MACLIB are synonymous.

<filename> is any valid specification, as determined by the operating system. Defaults for filename extensions and device names are the same as those in a MACRO-80 command line.

The INCLUDE file is opened and assembled into the current source file immediately following the INCLUDE statement. When end-of-file is reached, assembly resumes with the statement following INCLUDE.

On a MACRO-80 listing, a plus sign is printed between the assembled code and the source line on each line assembled from an INCLUDE file.

Nested INCLUDEs are not allowed. If encountered, they will result in an objectionable syntax error 'O'.

The file specified in the operand field must exist. If the file is not found, the error 'V' (value error) is given, and the INCLUDE is ignored.

NAME

NAME ('modname')

NAME defines a name for the module. Only the first six characters are significant in a module name. A module name may also be defined with the TITLE pseudo-op. In the absence of both the NAME and TITLE pseudo-ops, the module name is created from the source file name.

Define Origin

ORG <exp>

The location counter is set to the value of <exp> and the Assembler assigns generated code starting with that value. All names used in <exp> must be known on pass 1, and the value must either be absolute or in the same area as the location counter.

PAGE

PAGE [<exp>]

PAGE causes the Assembler to start a new output page. The value of <exp>, if included, becomes the new page size (measured in lines per page) and must be in the range 10 to 255. The default page size is 50 lines per page. The Assembler puts a form feed character in the listing file at the end of a page.

SET

<name> SET <exp>

SET is the same as EQU, except no error is generated if <name> is already defined.

SUBTTL

SUBTTL <text>

SUBTTL specifies a subtitle to be listed on the line after the title on each page heading. <text> is truncated after 60 characters. Any number of SUBTTLs may be given in a program.

TITLE

TITLE <text>

TITLE specifies a title to be listed on the first line of each page. If more than one TITLE is given, a Q error results. The first six characters of the title are used as the module name unless a NAME pseudo operation is used. If neither a NAME or TITLE pseudo-op is used, the module name is created from the source file name.

.COMMENT

.COMMENT <delim><text><delim>

The first non-blank character encountered after .COMMENT is the delimiter. The following <text> comprises a comment block which continues until the next occurrence of <delimiter> is encountered. For example, using an asterisk as the delimiter, the format of the comment block would be:

```
.COMMENT  *
any amount of text entered
here as the comment block
    .
    .
    .
          *
;return to normal mode
```

.PRINTX

.PRINTX <delim><text><delim>

The first non-blank character encountered after .PRINTX is the delimiter. The following text is listed on the terminal during assembly until another occurrence of the delimiter is encountered.

.PRINTX is useful for displaying progress through a long assembly or for displaying the value of conditional assembly switches.

For example:

```
IF CP/M
.PRINTX /CP/M version
ENDIF
```

.PRINTX will output on both passes. If only one printout is desired, use the IF1 or IF2 pseudo-op.

.RADIX

.RADIX <exp>

The default base (or radix) for all constants is decimal. The .RADIX statement allows the default radix to be changed to any base in the range 2 to 16.

For example:

```
LXI H,0FFH
.RADIX 16
LXI H,0FF
```

The two LXIs in the example are identical. The <exp> in a RADIX statement is always in decimal radix, regardless of the current radix.

.REQUEST

.REQUEST <filename>[,<filename>. . .]

.REQUEST sends a request to the LINK-80 Loader to search the file names in the list for undefined globals before searching the FORTRAN library. The file names in the list should be in the form of legal MACRO-80 symbols. They should not include file name extensions or disk specifications. The LINK-80 loader will supply the default extension .REL and will assume the default drive.

.Z80

.Z80 enables the Assembler to accept Z80 opcodes. Z80 mode may also be set by appending the /Z switch to the MACRO-80 command string.

.8080

.8080 enables the Assembler to accept 8080 opcodes. This is the default condition. 8080 mode may also be set by appending the /I switch to the MACRO-80 command string.

CONDITIONAL PSEUDO-OPERATIONS

The conditional pseudo-operations are:

IF/IFT <exp>	True if <exp> is not 0.
IFE/IFF <exp>	True if <exp> is 0.
IF1	True if pass 1.
IF2	True if pass 2.
IFDEF <symbol>	True if <symbol> is defined or has been declared External.
IFNDEF <symbol>	True if <symbol> is undefined or not declared External.
IFB <arg>	True if <arg> is blank. The angle brackets around <arg> are required.
IFNB <arg>	True if <arg> is not blank. Used for testing when dummy parameters are supplied. The angle brackets around <arg> are required.
IFIDN <arg1>,<arg2>	True if the string <arg1> is IDeNtical to the string <arg2>. The angle brackets around <arg1> and <arg2> are required.
IFDIF <arg1>,<arg2>	True if the string <arg1> is DIFferent from the string <arg2>. The angle brackets around <arg1> and <arg2> are required.

All conditionals use the following format:

```
IFxx  [argument]
.
.
.
[ELSE
.
.
.  ]
ENDIF
```

Conditionals may be nested to any level.

Any argument to a conditional must be known on pass 1 to avoid V errors and incorrect evaluation. For IF, IFT, IFF, and IFE the expression must involve values which were previously defined and the expression must be absolute. If the name is defined after an IFDEF or IFNDEF, pass 1 considers the name to be undefined, but it will be defined on pass 2.

ELSE

Each conditional pseudo-operation may optionally be used with the ELSE pseudo-opcode which allows alternate code to be generated when the opposite condition exists. Only one ELSE is permitted for a given IF, and an ELSE is always bound to the most recently opened IF. A conditional with more than one ELSE or an ELSE without a conditional will cause a C error.

ENDIF

Each IF must have a matching ENDIF to terminate the conditional. Otherwise, an 'Unterminated conditional' message is generated at the end of each pass. An ENDIF without a matching IF causes a C error.

Listing Control Pseudo-Operations

There are five listing control pseudo-ops. Output to the listing file can be controlled by the following pseudo-ops:

.LIST, .XLIST, .SFCOND, .LFCOND, .TFCOND

If a listing is not being made, these pseudo-ops have no effect.

.LIST is the default condition. When a .XLIST is encountered, source and object code will not be listed until a .LIST is encountered.

The latter three pseudo-ops control the listing of conditional pseudo-op blocks which evaluate as false. These pseudo-ops give the programmer control over four cases.

1. **Normally list false conditionals** — For this case, the programmer simply allows the default mode to control the listing. The default mode is list false conditionals. If the programmer decides to suppress false conditionals, the /X switch can be issued in the command line instead of editing the source file.

2. **Normally suppress false conditionals** — For this case, the programmer issues the .TFCOND pseudo-op in the program file. .TFCOND reverses (toggles) the default, causing false conditionals to be suppressed. If the programmer decides to list false conditionals, the /X switch can be issued in the command line instead of editing the source file.

3. **Always suppress/list false conditionals** — For these cases, the programmer issues either the .SFCOND pseudo-op to suppress false conditionals, or the .LFCOND pseudo-op to list all false conditionals.

4. **Suppress/list some false conditionals** — For this case, the programmer has decided for most false conditionals whether to list or suppress; but for some false conditionals, the programmer has not yet decided. For the false conditionals decided about, use .SFCOND or .LFCOND. For those not yet decided, use .TFCOND. .TFCOND sets the current and default settings to the opposite of the default. Initially, the default is set by giving /X or no /X in the command line. Two subcases exist:

   A. The programmer wants some false conditionals not to list unless /X is given. The programmer uses the .SFCOND and .LFCOND pseudo-ops to control which areas always suppress or list false conditionals. To selectively suppress some false conditionals, the programmer issues .TFCOND at the beginning of the conditional block and again at the end of the conditional block. (NOTE: The second .TFCOND should be so that the default setting will be the same as the initial setting. Leaving the default equal to the initial setting makes it easier to keep track of the default mode if there are many such areas.) If the conditional block evaluates as false, the lines will be suppressed. In this subcase, issuing the /X switch in the command line causes the conditional block affected by .TFCOND to list even if it evaluates as false.

   B. The programmer wants some false conditionals to list unless /X is given of the file. Two consecutive .TFCONDs places the conditional listing setting in an initial state which is determined by the presence or absense of the /X switch (the first .TFCOND sets the default to not initial; the second to initial). The selected conditional block then responds to the /X switch: if a /X switch is issued in the command line, the conditional block is suppressed if false; if no /X switch is issued in the command line, the conditional block is listed even if false.

The programmer then must reissue the .SFCOND or .LFCOND conditional listing pseudo-op to restore the suppress or list mode. Simply issuing another .TFCOND will not restore the prior mode, but will toggle the default setting. Since in this subcase, the next area of code is supposed to list or suppress false conditionals always, the programmer must issue .SFCOND or .LFCOND.

The three conditional listing pseudo-ops are summarized below.

PSEUDO-OP	DEFINITION
.SFCOND	Suppresses the listing of conditional blocks that evaluate as false.
.LFCOND	Restores the listing of conditional blocks that evaluate as false.
.TFCOND	Toggles the current setting which controls the listing of false conditionals. .TFCOND sets the current and default setting to not default. If a /X switch is given in the MACRO-80 run command line for a file which contains .TFCOND, /X reverses the effect of .TFCOND.

The output of MACRO/REPT/IRP/IRPC expansions is controlled by three pseudo-ops:

.LALL, .SALL, and .XALL.

Where:

.LALL lists the complete macro text for all expansions.

.SALL lists only the object code produced by a macro and not its text.

.XALL is the default condition; it is similar to .SALL, except a source line is listed only it generates object code.

RELOCATION PSEUDO-OPERATIONS

The ability to create relocatable modules is one of the major features of MACRO-80. Relocatable modules offer the advantages of easier coding and faster testing, debugging and modifying. In addition, it is possible to specify segments of assembled code that will later be loaded into RAM (the Data Relative segment) and ROM/PROM (the Code Relative segment).

The pseudo-operations that select relocatable areas are CSEG and DSEG. The ASEG pseudo-op is used to generate non-relocatable (absolute) code. The COMMON pseudo-op creates a common data area for every COMMON block that is named in the program.

The default mode for the Assembler is Code Relative. That is, assembly begins with a CSEG automatically executed and the location counter in the Code Relative mode, pointing to location 0 in the Code Relative segment of memory. All subsequent instructions will be assembled into the Code Relative segment of memory until an ASEG or DSEG or COMMON pseudo-op is executed.

For example, the first DSEG encountered sets the location counter to location zero in the Data Relative segment of memory. The following code is assembled in the Data Relative mode, where, it is assigned to the Data Relative segment of memory. If a subsequent CSEG is encountered, the location counter will return to the next free location in the Code Relative segment and so on.

The ASEG, DSEG, CSEG pseudo-ops never have operands. If you wish to alter the current value of the location counter, use the ORG pseudo-op.

ORG Pseudo-Op

At any time, the value of the location counter may be changed by use of the the ORG pseudo-op.

The form of the ORG statement is:

ORG <exp>

where the value of <exp> will be the new value of the location counter in the current mode. All names used in <exp> must be known on pass 1 and the value of <exp> must be either Absolute or in the current mode of the location counter.

For example, the statements

```
DSEG
ORG 50
```

set the Data Relative location counter to 50, relative to the start of the Data Relative segment of memory.

LINK-80

The LINK-80 Linking Loader (see Section C) combines the segments and creates each relocatable module in memory when the program is loaded. The origins of the relocatable segments are not fixed until the program is loaded and the origins are assigned by LINK-80. The command to LINK-80 may contain user-specified origins through the use of the /P (for Code Relative) and /D (for Data and COMMON segments) switches.

For example, a program that begins with the statements:

```
ASEG
ORG 800H
```

and is assembled entirely in Absolute mode will always load beginning at 800 unless the ORG statement is changed in the source file. However, the same program, assembled in Code Relative mode with no ORG statement, may be loaded at any specified address by appending the /P:<address> switch to the LINK-80 command string.

Relocation Before Loading

Two pseudo-ops, .PHASE and .DEPHASE, allow code to be located in one area, but executed only at a different, specified area.

For example:

```
                                .PHASE   100H
0100    CD 0106     F00:        CALL     BAZ
0103    C3 0007'                JMP      Z00
0106    C9          BAZ:        RET
                                .DEPHASE
0007'   C3 0005     Z00:        JMP      5
```

All labels within a .PHASE block are defined as the absolute value from the origin of the phase area. The code, however, is loaded in the current area (i.e., from 0' in this example). The code within the block can later be moved to 100H and executed.

Chapter Four

Macros and Block Pseudo-Operations

OVERVIEW

The Macro-80 Assembler provides complete facilities for constructing macros within the source program. Three repeat pseudo-operations as well as the macro definition operation are included. The following chapter explains the construction and use of the macro facilities.

MACROS AND BLOCK PSEUDO OPERATIONS

The macro facilities provided by MACRO-80 include three repeat pseudo-operations: repeat (REPT), indefinite repeat (IRP), and indefinite repeat character (IRPC). A macro definition operation (MACRO) is also provided. Each of these four macro operations is terminated by the ENDM pseudo-operation.

Terms

For the purposes of discussion of macros and block operations, the following terms will be used:

1. <**dummy**> is used to represent a dummy parameter. All dummy parameters are legal symbols that appear in the body of a macro expansion.

2. <**dummylist**> is a list of <dummy>s separated by commas.

3. <**arglist**> is a list of arguments separated by commas. <arglist> must be delimited by angle brackets. Two angle brackets with no intervening characters (< >) or two commas with no intervening characters (, ,) enter a null argument in the list. Otherwise an argument is a character or series of characters terminated by a comma or >.

   With angle brackets that are nested inside an <arglist>, one level of brackets is removed each time the bracketed argument is used in an <arglist>.

   A "**quoted string**" is an acceptable argument and is passed as such. Unless enclosed in <brackets> or a "quoted string", leading and trailing spaces are deleted from arguments.

4. <**paramlist**> is used to represent a list of actual parameters separated by commas. No delimiters are required (the list is terminated by the end of line or a comment), but the rules for entering null parameters and nesting brackets are the same as described for <arglist>.

Block Pseudo Op-Codes

REPT-ENDM

```
REPT <exp>
.
.
.
ENDM
```

The block of statements between REPT and ENDM is repeated <exp> times. <exp> is evaluated as a 16-bit unsigned number. If <exp> contains any external or undefined terms, an error is generated.

Example:

```
X SET 0
  REPT   10 ;generates bytes 01-0A
X SET X+1
  DB X
  ENDM
```

IRP-ENDM

```
IRP  <dummy>,<arglist>
.
.
.
ENDM
```

The <arglist> must be enclosed in angle brackets. The number of arguments in the <arglist> determines the number of times the block of statements is repeated. Each repetition substitutes the next item in the <arglist> for every occurrence of <dummy> in the block. If the <arglist> is null (i.e., < >), the block is processed once with each occurrence of <dummy> removed.

For example:

```
IRP X,<1,2,3,4,5,6,7,8,9,10>
DB X
ENDM
```

generates the same bytes as the REPT example.

IRPC-ENDM

```
IRPC  <dummy>,string (or <string>)
.
.
.
ENDM
```

IRPC is similar to IRP but the arglist is replaced by a string of text and the angle brackets around the string are optional. The statements in the block are repeated once for each character in the string. Each repetition substitutes the next character in the string for every occurrence of <dummy> in the block.

For example:

```
IRPC X,0123456789
DB X+1
ENDM
```

generates the same code as the two previous examples.

MACRO

Often it is convenient to be able to generate a given sequence of statements from various places in a program, even though different parameters may be required each time the sequence is used.

This capability is provided by the MACRO statement.

```
<name> MACRO <dummylist>
.
.
.
ENDM
```

where <name> conforms to the rules for forming symbols. <name> is the name that will be used to invoke the macro. The <dummy>s in <dummylist> are the parameters that will be changed (replaced) each time the MACRO is invoked. The statements before the ENDM comprise the body of the macro.

During assembly, the macro is expanded everytime it is invoked but, unlike REPT/IRP/IRPC, the macro is not expanded when it is encountered.

In the listing, the expansion of the macro will be marked with a plus (+).

The form of a macro call is:

<name> <paramlist>

where <name> is the name supplied in the MACRO definition, and the parameters in <paramlist> will replace the <dummy>s in the MACRO <dummylist> on a one-to-one basis. The number of items in <dummylist> and <paramlist> is limited only by the length of a line.

The number of parameters used when the macro is called need not be the same as the number of <dummy>s in <dummylist>. If there are more parameters than <dummy>s, the extras are ignored. If there are fewer, the extra <dummy>s will be made null. The assembled code will contain the macro expansion code after each macro call.

NOTE: A dummy parameter in a MACRO/REPT/IRP/IRPC is always recognized exclusively as a dummy parameter. Register names such as A and B will be changed in the expansion if they were used as dummy parameters.

Here is an example of a MACRO definition that defines a macro called FOO:

```
FOO     MACRO   X
Y       SET     0
        REPT    X
Y       SET     Y+1
        DB      Y
        ENDM
        ENDM
```

This macro generates the same code as the previous three examples when the call:

```
FOO 10
```

is executed.

Another example, which generates the same code, illustrates the removal of one level of brackets when an argument is used as an arglist:

```
FOO   MACRO   X
IRP   Y,<X>
DB Y
ENDM
ENDM
```

When the call

```
FOO   <1,2,3,4,5,6,7,8,9,10>
```

is made, the macro expansion looks like this:

```
IRP     Y,<1,2,3,4,5,6,7,8,9,10>
DB Y
ENDM
```

ENDM

Every REPT, IRP, IRPC and MACRO pseudo-op must be terminated with the ENDM pseudo-op. Otherwise, the 'Unterminated REPT/IRP/IRPC/MACRO' message is generated at the end of each pass. An unmatched ENDM causes an O error.

EXITM

The EXITM pseudo-op is used to terminate a REPT/IRP/IRPC or MACRO call. When an EXITM is executed, the expansion is exited immediately and any remaining expansion or repetition is not generated. If the block containing the EXITM is nested within another block, the outer level continues to be expanded.

LOCAL

LOCAL <dummylist>

The LOCAL pseudo-op is allowed only inside a MACRO definition.

When LOCAL is executed, the Assembler creates a unique symbol for each <dummy> in <dummylist> and substitutes that symbol for each occurrence of the <dummy> in the expansion. These unique symbols are usually used to define a label within a macro, thus eliminating multiply-defined labels on successive expansions of the macro. The symbols created by the Assembler range from ..0001 to ..FFFF. Users will therefore want to avoid the form ..nnnn for their own symbols. If LOCAL statements are used, they must be the first statements in the macro definition.

Special Macro Operators and Forms

& The ampersand "&" is used in a macro expansion to concatenate text or symbols. A dummy parameter that is in a quoted string will not be substituted in the expansion unless it is immediately preceded by an ampersand. To form a symbol from text and a dummy, put an "&" between them.

For example:

```
ERRGEN   MACRO    X
ERROR&X: PUSH     B
         MVI      B,'&X'
         JMP      ERROR
         ENDM
```

In this example, the call ERRGEN A will generate:

```
ERROR&A: PUSH     B
         MVI      B,'A'
         JMP      ERROR
```

;; In a block operation, a comment preceded by two semicolons is not saved as part of the expansion (i.e., it will not appear on the listing even under .LALL). A comment preceded by one semicolon, however, will be preserved and appear in the expansion.

! When an exclamation point is used in an argument, the next character is entered literally (i.e., !; and <;> are equivalent).

NUL NUL is an operator that returns true if its argument (a parameter) is null. The remainder of a line after NUL is considered to be the argument to NUL.

The conditional:

IF NUL argument

is false if, during the expansion, the first character of the argument is anything other than a semicolon or carriage return. It is recommended that testing for null parameters be done using the IFB and IFNB conditionals.

% The percent sign is used only in a macro argument. % converts the expression that follows it (usually a symbol) to a number in the current radix. During macro expansion, the number derived from converting the expression is substituted for the dummy. Using the % special operator allows a macro call by value. (Usually, a macro call is a call by reference with the text of the macro argument substituting exactly for the dummy.)

The expression following the % must conform to the same rules as the DS (Define Space) pseudo-op. A valid expression returning a nonrelocatable constant is required.

For Example:

Normally, LB, the argument to MAKLAB, would be substituted for Y, the argument to MACRO, as a string. The % causes LB to be converted to a nonrelocatable constant which is then substituted for Y. Without the % special operator, the result of assembly would be 'Error LB' rather than 'Error 1', etc.

```
MAKLAB   MACRO    Y
ERR&Y:   DB       'Error &Y',0
         ENDM
MAKERR   MACRO    X
LB       SET      0
         REPT     X
LB       SET      LB+1
         MAKLAB   %LB
         ENDM
         ENDM
```

When called by MAKERR 3, the assembler will generate:

```
ERR&1:  DB    'Error 1',0
ERR&2:  DB    'Error 2',0
ERR&3:  DB    'Error 3',0
```

TYPE The TYPE operator returns a byte that describes two characteristics of its argument: 1) the mode, and 2) whether it is External or not. The argument to TYPE may be any expression (string, numeric, logical). If the expression is invalid, TYPE returns zero.

The byte that is returned is configured as follows:

The lower two bits are the mode. If the lower two bits are:

0 the mode is Absolute
1 the mode is Program Relative
2 the mode is Data Relative
3 the mode is Common Relative

The high bit (80H) is the External bit. If the high bit is on, the expression contains an External. If the high bit is off, the expression is local (not External).

The Defined bit is 20H. This bit is on if the expression is locally defined, and it is off if the expression is undefined or external. If neither bit is on, the expression is invalid.

TYPE is usually used inside macros, where an argument type may need to be tested to make a decision regarding program flow.

Using Z80 Pseudo-Ops

When using the 8080/Z80 assembler, the following Z80 pseudo-ops are valid. The function of each pseudo-op is equivalent to that of its 8080 counterpart.

Z80 pseudo-op	Equivalent 8080 pseudo-op
COND	IFT
ENDC	ENDIF
*EJECT	PAGE
DEFB	DB
DEFS	DS
DEFW	DW
DEFM	DB
DEFL	SET
GLOBAL	PUBLIC
EXTERNAL	EXTRN

The formats, where different, conform to the 8080 format. That is, DEFB and DEFW are permitted a list of arguments (as are DB and DW), and DEFM is permitted a string or numeric argument (as is DB).

MACRO-80 Reference Manual Index