MICROSOFT

utility software manual

Microsoft
Utility Software Manual

CONTENTS

SECTION 1

MACRO-80 Assembler

1.1 Format of MACRO-80 Commands

1.1.1 MACRO-80 Command Strings

To run MACRO-80, type M80 followed by a carriage return. MACRO-80 will return the prompt "*" (with the DTC operating system, the prompt is ">"), 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
```

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 filename and filename extension of the object program file, the listing file, and the source file. Filename extensions may be omitted. See Section 4 for the default extension supplied by your operating system.

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:

*=SOURCE.MAC	Assemble the program SOURCE.MAC and place the object in SOURCE.REL
*,LST:=TEST	Assemble the program TEST.MAC and list on device LST

*SMALL,TTY:=TEST	Assemble the program TEST.MAC, place the object in SMALL.REL and list on TTY

1.1.2 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. (Default for Altair DOS)
H	Print all listing addresses, etc. in hexadecimal. (Default for non-Altair versions)
R	Force generation of an object file.
L	Force generation of a listing file.
C	Force generation of a cross reference file.
Z	Assemble Z80 (Zilog format) mnemonics. (Default for Z80 operating systems)
I	Assemble 8080 mnemonics. (Default for 8080 operating systems)

Examples:

*=TEST/L	Compile TEST.MAC with object file TEST.REL and listing file TEST.LST
*LAST,LAST/C=MOD1	Compile MOD1.MAC with object file LAST.REL and cross reference file LAST.CRF for use with CREF-80 (See Section 1.12)

1.2 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 of up to 132 characters in length are acceptable.

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

NOTE

If the source file includes line numbers from an editor, each byte of the line number must have the high bit on. Line numbers from Microsoft's EDIT-80 Editor are acceptable.

1.2.1 Statements

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

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

With the exception of the ISIS assembler $ controls (see Section 1.10), it is not necessary that statements begin in column 1. 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 (see ENTRY/PUBLIC, Section 1.5.10). For exmple:

```
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 mnemonic), 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

(see Section 1.5.4).

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 pseudo operation (see Section 1.5.19).

1.2.2 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 $. ? @

With the 8080/Z80 assembler, the underline character is also legal in a symbol. A symbol may not start with a digit. When a symbol is read, lower case is translated into upper case. If a symbol reference is followed by ## it is declared external (see also the EXT/EXTRN pseudo-op, Section 1.5.12).

1.2.3 Numeric Constants

The default base for numeric constants is decimal. This may be changed by the .RADIX pseudo-op (see Section 1.5.21). 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. This eliminates the use of zero as a leading digit for octal constants, as in previous versions of MACRO-80.

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.

A character constant is a string comprised of zero, one or two ASCII characters, delimited by quotation marks, and used in a non-simple expression. For example, in the statement

```
DB      'A' + 1
```

'A' is a character constant. But the statement

```
DB      'A'
```

uses 'A' as a string because it is in a simple expression. The rules for character constant delimiters are the same as for strings.

A character constant comprised of one character has as its value the ASCII value of that character. That is, the high order byte of the value is zero, and the low order byte is the ASCII value of the character. For example, the value of the constant 'A' is 41H.

A character constant comprised of two characters has as its value the ASCII value of the first character in the high order byte and the ASCII value of the second character in the low order byte. For example, the value of the character constant "AB" is 41H*256+42H.

1.2.4 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 occurrence desired. For example, the statement

```
DB      "I am ""great"" today"
```

stores the string

```
I am "great" today
```

If there are zero characters between the delimiters, the string is a null string.

1.3 Expression Evaluation

1.3.1 Arithmetic and Logical Operators

The following operators are allowed in expressions. The operators are listed in 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 8 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.

1.3.2 Modes

All symbols used as operands in expressions are in one of the following modes: Absolute, Data Relative, Program (Code) Relative or COMMON. (See Section 1.5 for the ASEG, CSEG, DSEG and COMMON pseudo-ops.) 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. So that

```
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.

1.3.3 Externals

Aside from its classification by mode, a symbol is either External or not External. (See EXT/EXTRN, Section 1.5.12.) 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.

1.4 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)
ADI     (CPI)
MVI     B,(RNZ)
CPI     (INX H)
ACI     (LXI B)
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.

1.5 Pseudo Operations

1.5.1 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. See also Section 1.5.27.

1.5.2 COMMON

```
COMMON /<block name>/
```

COMMON sets the location counter to the selected common block in memory. The location is always the beginning of the 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. See also Section 1.5.27.

1.5.3 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 (the INTEL assembler defaults to ASEG). See also Section 1.5.27.

1.5.4 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'   4142            DB      'AB'
0002'   42              DB      'AB' AND 0FFH
0003'   41 42 43        DB      'ABC'
```

1.5.5 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.

1.5.6 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.

1.5.7 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. See also Section 1.5.27.

1.5.8 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.

1.5.9 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.

1.5.10 ENTRY/PUBLIC

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

1.5.11 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.

1.5.12 EXT/EXTRN

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

1.5.13 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.

1.5.14 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.

1.5.15 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.

1.5.16 SET

```
<name> SET <exp>
```

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

1.5.17 SUBTTL

```
SUBTTL <text>
```

SUBTTL specifies a subtitle to be listed on the line after the title (see TITLE, Section 1.5.18) on each page heading. <text> is truncated after 60 characters. Any number of SUBTTLs may be given in a program.

1.5.18 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 filename.

1.5.19 .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
```

1.5.20 .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      CPM
.PRINTX /CPM version/
ENDIF
```

NOTE

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

1.5.21 .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.

1.5.22 .REQUEST

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

.REQUEST sends a request to the LINK-80 loader to search the filenames in the list for undefined globals before searching the FORTRAN library. The filenames in the list should be in the form of legal MACRO-80 symbols. They should not include filename extensions or disk specifications. The LINK-80 loader will scpply its default extension and will assume the currently selected disk drive.

1.5.23 .Z80

.Z80 enables the assembler to accept Z80 opcodes. This is the default condition when the assembler is running on a Z80 operating system. Z80 mode may also be set by appending the Z switch to the MACRO-80 command string -- see Section 1.1.2.

1.5.24 .8080

.8080 enables the assembler to accept 8080 opcodes. This is the default condition when the assembler is running on an 8080 operating system. 8080 mode may also be set by appending the I switch to the MACRO-80 command string -- see Section 1.1.2.

1.5.25 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.

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 operation 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 recent, open 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.

1.5.26 Listing Control Pseudo Operations

Output to the listing file can be controlled by two pseudo-ops:

.LIST and .XLIST

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 output of cross reference information is controlled by .CREF and .XCREF. If the cross reference facility (see Section 1.12) has not been invoked, .CREF and .XCREF have no effect. The default condition is .CREF. When a .XCREF is encountered, no cross reference information is output until .CREF is encountered.

The output of MACRO/REPT/IRP/IRPC expansions is controlled by three pseudo-ops: .LALL, .SALL, and .XALL. .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 if it generates object code.

1.5.27 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 asembled in the Data Relative mode, that is, 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 2 of this manual) 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.

1.5.28 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:

```
0000'                          .PHASE  100H
0100    CD 0106      FOO:      CALL    BAZ
0103    C3 0007'               JMP     ZOO
0106    C9           BAZ:      RET
                               .DEPHASE
0007'   C3 0005      ZOO:      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.

1.6 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.

1.6.1 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>. (See example, Section 1.6.5.) 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>. (See example, Section 1.6.5.)

1.6.2 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:

```
SET    0
REPT   10     ;generates DB1-DB10
SET    X+1
DB     X
ENDM
```

1.6.3 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.

1.6.4 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.

1.6.5 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. The form is

```
<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.

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 <dummmy>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 dummmy 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
```

1.6.6 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.

1.6.7 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.

1.6.8 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.

1.6.9 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 &. To form a symbol from text and a dummy, put & 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:

```
ERRORA: 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.

1.7 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).

1.8 Sample Assembly

```
A>M80

*EXMPL1,TTY:=EXMPL1

        MAC80 3.2       PAGE    1


                                00100   ;CSL3(P1,P2)
                                00200   ;SHIFT P1 LEFT CIRCULARLY 3 BITS
                                00300   ;RETURN RESULT IN P2
                                00400           ENTRY   CSL3
                                00450   ;GET VALUE OF FIRST PARAMETER
                                00500   CSL3:
  0000'   7E                    00600           MOV     A,M
  0001'   23                    00700           INX     H
  0002'   66                    00800           MOV     H,M
  0003'   6F                    00900           MOV     L,A
                                01000   ;SHIFT COUNT
  0004'   06 03                 01100           MVI     B,3
  0006'   AF                    01200   LOOP:   XRA     A
                                01300   ;SHIFT LEFT
  0007'   29                    01400           DAD     H
                                01500   ;ROTATE IN CY BIT
  0008'   17                    01600           RAL
  0009'   85                    01700           ADD     L
  000A'   6F                    01800           MOV     L,A
                                01900   ;DECREMENT COUNT
  000B'   05                    02000           DCR     B
                                02100   ;ONE MORE TIME
  000C'   C2 0006'              02200           JNZ     LOOP
  000F'   EB                    02300           XCHG
                                02400   ;SAVE RESULT IN SECOND PARAMETER
  0010'   73                    02500           MOV     M,E
  0011'   23                    02600           INX     H
  0012'   72                    02700           MOV     M,D
  0013'   C9                    02800           RET
                                02900           END


        MAC80 3.2       PAGE    S


CSL3    0000I'  LOOP    0006'


No  Fatal error(s)
```

1.9 MACRO-80 Errors

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. Below is a list of the MACRO-80 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,M; 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. If the symbol is defined 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 CRT and in the list file.

1.10 Compatibility with Other Assemblers

The $EJECT and $TITLE controls are provided for compatability with INTEL's ISIS assembler. The dollar sign must appear in column 1 only if spaces or tabs separate the dollar sign from the control word. The control

```
$EJECT
```

is the same as the MACRO-80 PAGE pseudo-op.
The control

```
$TITLE('text')
```

is the same as the MACRO-80 SUBTTL <text>
pseudo-op.

The INTEL operands PAGE and INPAGE generate Q errors when used with the MACRO-80 CSEG or DSEG

pseudo-ops. These errors are warnings; the assembler ignores the operands.

When MACRO-80 is entered, the default for the origin is Code Relative 0. With the INTEL ISIS assembler, the default is Absolute 0.

With MACRO-80, the dollar sign ($) is a defined constant that indicates the value of the location counter at the start of the statement. Other assemblers may use a decimal point or an asterisk. Other constants are defined by MACRO-80 to have the following values:

```
A=7    B=0    C=1    D=2    E=3
H=4    L=5    M=6    SP=6   PSW=6
```

1.11 Format of Listings

On each page of a MACRO-80 listing, the first two lines have the form:

```
[TITLE text]    MAC80 3.2    PAGE x[-y]
[SUBTTL text]
```

where:

1. TITLE text is the text supplied with the TITLE pseudo-op, if one was given in the source program.

2. x is the major page number, which is incremented only when a form feed is encountered in the source file. (When using Microsoft's EDIT-80 text editor, a form feed is inserted whenever a page mark is done.) When the symbol table is being printed, x = 'S'.

3. y is the minor page number, which is incremented whenever the .PAGE pseudo-op is encountered in the source file, or whenever the current page size has been filled.

4. SUBTTL text is the text supplied with the SUBTTL pseudo-op, if one was given in the source program.

Next, a blank line is printed, followed by the first line of output.

A line of output on a MACRO-80 listing has the following form:

```
[crf#]    [error] loc#m    xx    xxxx ...    source
```

If cross reference information is being output, the first item on the line is the cross reference number, followed by a tab.

A one-letter error code followed by a space appears next on the line, if the line contains an error. If there is no error, a space is printed. If there is no cross reference number, the error code column is the first column on the listing.

The value of the location counter appears next on the line. It is a 4-digit hexadecimal number or 6-digit octal number, depending on whether the /O or /H switch was given in the MACRO-80 command string.

The character at the end of the location counter value is the mode indicator. It will be one of the following symbols:

'	Code Relative
"	Data Relative
!	COMMON Relative
<space>	Absolute
*	External

Next, three spaces are printed followed by the assembled code. One-byte values are followed by a space. Two-byte values are followed by a mode indicator. Two-byte values are printed in the opposite order they are stored in, i.e., the high order byte is printed first. Externals are either the offset or the value of the pointer to the next External in the chain.

The remainder of the line contains the line of source code, as it was input.

1.11.1 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:

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.

1.12 Cross Reference Facility

The Cross Reference Facility is invoked by typing CREF80. In order to generate a cross reference listing, the assembler must output a special listing file with embedded control characters. The MACRO-80 command string tells the assembler to output this special listing file. (See Section 1.5.26 for the .CREF and .XCREF pseudo-ops.) /C is the cross reference switch. When the /C switch is encountered in a MACRO-80 command string, the assembler opens a .CRF file instead of a .LST file.

Examples:

*=TEST/C	Assemble file TEST.MAC and create object file TEST.REL and cross reference file TEST.CRF.
*T,U=TEST/C	Assemble file TEST.MAC and create object file T.REL and cross reference file U.CRF.

When the assembler is finished, it is necessary to call the cross reference facility by typing CREF80. The command string is:

```
*listing file=source file
```

Possible command strings are: The default extension for the source file is .CRF. The /L switch is ignored, and any other switch will cause an error message to be sent to the terminal. Possible command strings are:

*=TEST	Examine file TEST.CRF and generate a cross reference listing file TEST.LST.
*T=TEST	Examine file TEST.CRF and generate a cross reference listing file T.LST.

Cross reference listing files differ from ordinary listing files in that:

1. Each source statement is numbered with a cross reference number.

2. At the end of the listing, variable names appear in alphabetic order along with the numbers of the lines on which they are referenced or defined. Line numbers on which the symbol is defined are flagged with '#'.

SECTION 2

LINK-80 Linking Loader

2.1 Format of LINK-80 Commands

2.1.1 LINK-80 Command Strings

To run LINK-80, type L80 followed by a carriage return. LINK-80 will return the prompt "*" (with the DTC operating system, the prompt is ">"), indicating it is ready to accept commands. Each command to LINK-80 consists of a string of filenames and switches separated by commas:

objdev1:filename.ext/switch1,objdev2:filename.ext,...

If the input device for a file is omitted, the default is the currently logged disk. If the extension of a file is omitted, the default is .REL. After each line is typed, LINK will load or search (see /S below) the specified files. After LINK finishes this process, it will list all symbols that remained undefined followed by an asterisk.

Example:

```
*MAIN

 DATA    0100    0200

 SUBR1*      (SUBR1 is undefined)

 DATA    0100    0300

*SUBR1
*/G          (Starts Execution - see below)
```

Typically, to execute a FORTRAN and/or COBOL program and subroutines, the user types the list of filenames followed by /G (begin execution). Before execution begins, LINK-80 will always search the system library (FORLIB.REL or COBLIB.REL) to satisfy any unresolved external references. If the user wishes to first search libraries of his own, he should append the filenames that are followed by /S to the end of the loader command string.

2.1.2 LINK-80 Switches

A number of switches may be given in the LINK-80 command string to specify actions affecting the loading process. Each switch must be preceded by a slash (/). These switches are:

Switch	Action
R	Reset. Put loader back in its initial state. Use /R if you loaded the wrong file by mistake and want to restart. /R takes effect as soon as it is encountered in a command string.
E or E:Name	Exit LINK-80 and return to the Operating System. The system library will be searched on the current disk to satisfy any existing undefined globals. The optional form E:Name (where Name is a global symbol previously defined in one of the modules) uses Name for the start address of the program. Use /E to load a program and exit back to the monitor.
G or G:Name	Start execution of the program as soon as the current command line has been interpreted. The system library will be searched on the current disk to satisfy any existing undefined globals if they exist. Before execution actually begins, LINK-80 prints three numbers and a BEGIN EXECUTION message. The three numbers are the start address, the address of the next available byte, and the number of 256-byte pages used. The optional form G:Name (where Name is a global symbol previously defined in one of the modules) uses Name for the start address of the program.
N	If a <filename>/N is specified, the program will be saved on disk under the selected name (with a default extension of .COM for CP/M) when a /E or /G is done. A jump to the start of the program is inserted if needed so the program can run properly (at 100H for CP/M).

P and D

/P and /D allow the origin(s) to be set for the next program loaded. /P and /D take effect when seen (not deferred), and they have no effect on programs already loaded. The form is /P:<address> or /D:<address>, where <address> is the desired origin in the current typeout radix. (Default radix for non-MITS versions is hex. /O sets radix to octal; /H to hex.) LINK-80 does a default /P:<link origin>+3 (i.e., 103H for CP/M and 4003H for ISIS) to leave room for the jump to the start address.

NOTE: Do not use /P or /D to load programs or data into the locations of the loader's jump to the start address (100H to 102H for CPM and 2800H to 2802H for DTC), unless it is to load the start of the program there. If programs or data are loaded into these locations, the jump will not be generated.

If no /D is given, data areas are loaded before program areas for each module. If a /D is given, all Data and Common areas are loaded starting at the data origin and the program area at the program origin. Example:

```
*/P:200,FOO
Data    200    300
*/R
*/P:200 /D:400,FOO
Data    400    480
Program 200    280
```

U

List the origin and end of the program and data area and all undefined globals as soon as the current command line has been interpreted. The program information is only printed if a /D has been done. Otherwise, the program is stored in the data area.

M

List the origin and end of the program and data area, all defined globals and their values, and all undefined globals followed by an asterisk. The program information

is only printed if a /D has been done. Otherwise, the program is stored in the data area.

S — Search the filename immediately preceding the /S in the command string to satisfy any undefined globals.

Examples:

```
*/M                 List all globals

*MYPROG,SUBROT,MYLIB/S
                    Load MYPROG.REL and SUBROT.REL and
                    then search MYLIB.REL to satisfy
                    any remaining undefined globals.

*/G                 Begin execution of main program
```

2.2 Sample Link

```
A>L80
*EXAMPL,EXMPL1/G
DATA    3000    30AC
[304F   30AC    49]
[BEGIN EXECUTION]

         1792           14336
        14336          -16383
       -16383              14
           14             112
          112             896
A>
```

2.3 Format of LINK Compatible Object Files

NOTE

Section 2.3 is reference material for users who wish to know the load format of LINK-80 relocatable object files. Most users will want to skip this section, as it does not contain material necessary to the operation of the package.

LINK-compatible object files consist of a bit stream. Individual fields within the bit stream are not aligned on byte boundaries, except as noted below. Use of a bit stream for relocatable object files keeps the size of object files to a minimum, thereby decreasing the number of disk reads/writes.

There are two basic types of load items: Absolute and Relocatable. The first bit of an item indicates one of these two types. If the first bit is a 0, the following 8 bits are loaded as an absolute byte. If the first bit is a 1, the next 2 bits are used to indicate one of four types of relocatable items:

00	Special LINK item (see below).
01	Program Relative. Load the following 16 bits after adding the current Program base.
10	Data Relative. Load the following 16 bits after adding the current Data base.
11	Common Relative. Load the following 16 bits after adding the current Common base.

Special LINK items consist of the bit stream 100 followed by:

a four-bit control field

an optional A field consisting of a two-bit address type that is the same as the two-bit field above except 00 specifies absolute address

an optional B field consisting of 3 bits that give a symbol length and up to 8 bits for each character of the symbol

A general representation of a special LINK item is:

```
1 00 xxxx  yy 00    zzz + characters of symbol name
           --------  ------------------------------
           A field             B field
```

xxxx	Four-bit control field (0-15 below)
yy	Two-bit address type field
00	Sixteen-bit value
zzz	Three-bit symbol length field

The following special types have a B-field only:

0	Entry symbol (name for search)
1	Select COMMON block
2	Program name
3	Request library search

4 Reserved for future expansion

The following special LINK items have both an A field and a B field:

5 Define COMMON size
6 Chain external (A is head of address chain, B is name of external symbol)
7 Define entry point (A is address, B is name)
8 Reserved for future expansion

The following special LINK items have an A field only:

9 External + offset. The A value will be added to the two bytes starting at the current location counter immediately before execution.
10 Define size of Data area (A is size)
11 Set loading location counter to A
12 Chain address. A is head of chain, replace all entries in chain with current location counter. The last entry in the chain has an address field of absolute zero.
13 Define program size (A is size)
14 End program (forces to byte boundary)

The following special Link item has neither an A nor a B field:

15 End file

2.4 LINK-80 Error Messages

LINK-80 has the following error messages:

?No Start Address	A /G switch was issued, but no main program had been loaded.
?Loading Error	The last file given for input was not a properly formatted LINK-80 object file.
?Out of Memory	Not enough memory to load program.
?Command Error	Unrecognizable LINK-80 command.
?<file> Not Found	<file>, as given in the command string, did not exist.

%2nd COMMON Larger /XXXXXX/

The first definition of COMMON block /XXXXXX/ was not the largest definition. Re-order module loading sequence or change COMMON block definitions.

%Mult. Def. Global YYYYYY

More than one definition for the global (internal) symbol YYYYYY was encountered during the loading process.

%Overlaying [Program / Data] Area [,Start = xxxx / ,Public = <symbol name>(xxxx) / ,External = <symbol name>(xxxx)]

A /D or /P will cause already loaded data to be destroyed.

?Intersecting [Program / Data] Area

The program and data area intersect and an address or external chain entry is in this intersection. The final value cannot be converted to a current value since it is in the area intersection.

?Start Symbol - <name> - Undefined

After a /E: or /G: is given, the symbol specified was not defined.

Origin [Above / Below] Loader Memory, Move Anyway (Y or N)?

After a /E or /G was given, either the data or program area has an origin or top which lies outside loader memory (i.e., loader origin to top of memory). If a Y <cr> is given, LINK-80 will move the area and continue. If anything else is given, LINK-80 will exit. In either case, if a /N was given, the image will already have been saved.

?Can't Save Object File

A disk error occurred when the file was being saved.

2.5 Program Break Information

LINK-80 stores the address of the first free location in a global symbol called $MEMRY if that symbol has been defined by a program loaded. $MEMRY is set to the top of the data area +1.

NOTE

If /D is given and the data origin is less than the program area, the user must be sure there is enough room to keep the program from being destroyed. This is particularly true with the disk driver for FORTRAN-80 which uses $MEMRY to allocate disk buffers and FCB's.

SECTION 3

LIB-80 Library Manager (CP/M Versions Only)

LIB-80 is the object time library manager for CP/M versions of FORTRAN-80 and COBOL-80. LIB-80 will be interfaced to other operating systems in future releases of FORTRAN-80 and COBOL-80.

3.1 LIB-80 Commands

To run LIB-80, type LIB followed by a carriage return. LIB-80 will return the prompt "*" (with the DTC operating system, the prompt is ">"), indicating it is ready to accept commands. Each command in LIB-80 either lists information about a library or adds new modules to the library under construction.

Commands to LIB-80 consists of an optional destination filename which sets the name of the library being created, followed by an equal sign, followed by module names separated by commas. The default destination filename is FORLIB.LIB. Examples:

```
*NEWLIB=FILE1 <MOD2>, FILE3,TEST

*SIN,COS,TAN,ATAN
```

Any command specifying a set of modules concatenates the modules selected onto the end of the last destination filename given. Therefore,

```
*FILE1,FILE2 <BIGSUB>, TEST
```

is equivalent to

```
*FILE1
*FILE2 <BIGSUB>
*TEST
```

3.1.1 Modules

A module is typically a FORTRAN or COBOL subprogram, main program or a MACRO-80 assembly that contains ENTRY statements.

The primary function of LIB-80 is to concatenate modules in .REL files to form a new library. In

order to extract modules from previous libraries or .REL files, a powerful syntax has been devised to specify ranges of modules within a .REL file.

The simplest way to specify a module within a file is simply to use the name of the module. For example:

```
SIN
```

But a relative quantity plus or minus 255 may also be used. For example:

```
SIN+1
```

specifies the module after SIN and

```
SIN-1
```

specifies the one before it.

Ranges of modules may also be specified by using two dots:

..SIN means all modules up to and including SIN.

SIN.. means all modules from SIN to the end of the file.

SIN..COS means SIN and COS and all the modules in between.

Ranges of modules and relative offsets may also be used in combination:

```
SIN+1..COS-1
```

To select a given module from a file, use the name of the file followed by the module(s) specified enclosed in angle brackets and separated by commas:

```
FORLIB <SIN..COS>

 or

MYLIB.REL <TEST>

 or

BIGLIB.REL <FIRST,MIDDLE,LAST>

 etc.
```

If no modules are selected from a file, then all

the modules in the file are selected:

```
TESTLIB.REL
```

3.2 LIB-80 Switches

A number of switches are used to control LIB-80 operation. These switches are always preceded by a slash:

/O Octal - set Octal typeout mode for /L command.

/H Hex - set Hex typeout mode for /L command (default).

/U List the symbols which would remain undefined on a search through the file specified.

/L List the modules in the files specified and symbol definitions they contain.

/C (Create) Throw away the library under construction and start over.

/E Exit to CP/M. The library under construction (.LIB) is revised to .REL and any previous copy is deleted.

/R Rename - same as /E but does not exit to CP/M on completion.

3.3 LIB-80 Listings

To list the contents of a file in cross reference format, use /L:

```
*FORLIB/L
```

When building libraries, it is important to order the modules such that any intermodule references are "forward." That is, the module containing the global reference should physically appear ahead of the module containing the entry point. Otherwise, LINK-80 may not satisfy all global references on a single pass through the library.

Use /U to list the symbols which could be undefined in a single pass through a library. If a module in the library makes a backward reference to a symbol in another module, /U will list that symbol. Example:

```
*SYSLIB/U
```

NOTE: Since certain modules in the standard FORTRAN and COBOL systems are always force-loaded, they will be listed as undefined by /U but will not cause a problem when loading FORTRAN or COBOL programs.

Listings are currently always sent to the terminal; use control-P to send the listing to the printer.

3.4 Sample LIB Session

```
A>LIB

*TRANLIB=SIN,COS,TAN,ATAN,ALOG
*EXP
*TRANLIB.LIB/U
*TRANLIB.LIB/L
  .
  .
  .
(List of symbols in TRANLIB.LIB)
  .
  .
  .
*/E
A>
```

3.5 Summary of Switches and Syntax

```
/O  Octal - set listing radix
/H  Hex - set listing radix
/U  List undefineds
/L  List cross reference
/C  Create - start LIB over
/E  Exit - Rename .LIB to .REL and exit
/R  Rename - Rename .LIB to .REL
```

module::=module name {+ or - number}

module sequence ::=

module | ..module | module.. | module1..module2

file specification::=filename {<module sequence>{,<module sequence>}

command::= {library filename=} {list of file specifications}
{list of switches}

SECTION 4

Operating Systems

This section describes the use of MACRO-80 and LINK-80 under the different disk operating systems. The examples shown in this section assume that the FORTRAN-80 compiler is in use. If you are using the COBOL-80 compiler, substitute "COBOL" wherever "F80" appears, and substitute the extension ".COB" wherever ".FOR" appears.

4.1 CPM

Create a Source File

Create a source file using the CPM editor. Filenames are up to eight characters long, with 3-character extensions. FORTRAN-80 source filenames should have the extension FOR, COBOL-80 source filenames should have the extension COB, and MACRO-80 source filenames should have the extension MAC.

Compile the Source File

Before attempting to compile the program and produce object code for the first time, it is advisable to do a simple syntax check. Removing syntax errors will eliminate the necessity of recompiling later. To perform the syntax check on a source file called MAX1.FOR, type

```
A>F80 ,=MAX1
```

This command compiles the source file MAX1.FOR without producing an object or listing file. If necessary, return to the editor and correct any syntax errors.

To compile the source file and produce an object and listing file, type

```
A>F80 MAX1,MAX1=MAX1
```

or

```
A>F80 =MAX1/L
```

The compiler will create a REL (relocatable) file called MAX1.REL and a listing file called MAX1.PRN.

Loading, Executing and Saving the Program (Using LINK-80)

To load the program into memory and execute it, type

```
A>L80 MAX1/G
```

To exit LINK-80 and save the memory image (object code), type

```
A>L80 MAX1/E,MAX1/N
```

When LINK-80 exits, three numbers will be printed: the starting address for execution of the program, the end address of the program and the number of 256-byte pages used. For example

```
[210C 401A 48]
```

If you wish to use the CPM SAVE command to save a memory image, the number of pages used is the argument for SAVE. For example

```
A>SAVE 48 MAX1.COM
```

NOTE

> CP/M always saves memory starting at 100H and jumps to 100H to begin execution. Do not use /P or /D to set the origin of the program or data area to 100H, unless program execution will actually begin at 100H.

An object code file has now been saved on the disk under the name specified with /N or SAVE (in this case MAX1). To execute the program simply type the program name

```
A>MAX1
```

CPM - Available Devices

```
A:, B:, C:, D:  disk drives
HSR:    high speed reader
LST:    line printer
TTY:    Teletype or CRT
```

CPM Disk Filename Standard Extensions

```
FOR     FORTRAN-80 source file
COB     COBOL-80 source file
MAC     MACRO-80 object file
REL     relocatable object file
PRN     listing file
COM     absolute file
```

CPM Command Lines
CPM command lines and files are supported; i.e., a COBOL-80, FORTRAN-80, MACRO-80 or LINK-80 command line may be placed in the same line with the CPM run command. For example, the command

```
A>F80 =TEST
```

causes CPM to load and run the FORTRAN-80 compiler, which then compiles the program TEST.FOR and creates the file TEST.REL. This is equivalent to the following series of commands:

```
A>F80
*=TEST
*^C
A>
```

4.2 DTC Microfile

Create a Source File
Create a source file using the DTC editor. Filenames are up to five characters long, with 1-character extensions. COBOL-80, FORTRAN-80 and MACRO-80 source filenames should have the extension T.

Compile the Source File
Before attempting to compile the program and produce object code for the first time, it is advisable to do a simple syntax check. Removing syntax errors will eliminate the necessity of recompiling later. To perform the syntax check on the source file called MAX1, type

```
*F80 ,=MAX1
```

This command compiles the source file MAX1 without producing an object or listing file. If necessary, return to the editor and correct any syntax errors.

To compile the source file MAX1 and produce an object and listing file, type

```
*F80 MAX1,MAX1=MAX1
```

or

```
*F80 =MAX1/L/R
```

The compiler will create a relocatable file called MAX1.O and a listing file called MAX1.L.

Loading, Executing and Saving the Program (Using LINK-80)
To load the program into memory and execute it,

type

```
*L80 MAX1/G
```

To save the memory image (object code), type

```
*L80 MAX1/E
```

which will exit from LINK-80, return to the DOS monitor and print three numbers: the starting addressfor execution of the program, the end address of the program, and the number of 256-byte pages used. For example

```
[210C 401A 48]
```

Use the DTC SAVE command to save a memory image. For example

```
*SA MAX1 2800 401A 2800
```

2800H (24000Q) is the load address used by the DTC Operating System.

NOTE

> If a /P:<address> or /D:<address> has been included in the loader command to specify an origin other than the default (2800H), make sure the low address in the SAVE command is the same as the start address of the program.

An object code file has now been saved on the disk under the name specified in the SAVE command (in this case MAX1). To execute the program, simply type

```
*RUN MAX1
```

DTC Microfile - Available Devices

DO:, D1:, D2:, D3:	disk drives
TTY:	Teletype or CRT
LIN:	communications port

DTC Disk Filename Standard Extensions

T	COBOL-80, FORTRAN-80 or MACRO-80 source file
O	relocatable object file
L	listing file

DTC Command Lines
DTC command lines are supported as described in Section 4.1, CPM Command Lines.

4.3 Altair DOS

Create a Source File
Create a source file using the Altair DOS editor. The name of the file should have four characters, and the first character must be a letter. For example, to create a file called MAX1, initialize DOS and type

```
.EDIT MAX1
```

The editor will respond

```
CREATING FILE
00100
```

Enter the program. When you are finished entering and editing the program, exit the editor.

Compile the Source File
Load the compiler by typing

```
.F80
```

The compiler will return the prompt character "*".

Before attempting to compile the program and produce object code for the first time, it is advisable to do a simple syntax check. Removing syntax errors will eliminate the necessity of recompiling later. To perform the syntax check on the source file called MAX1, type

```
*,=&MAX1.
```

(The editor stored the program as &MAX1) Typing ,=&MAX1. compiles the source file MAX1 without producing an object or listing file. If necessary, return to the editor and correct any syntax errors.

To compile the source file MAX1 and produce an object and listing file, type

```
*MAX1R,&MAX1=&MAX1.
```

The compiler will create a REL (relocatable) file called MAX1RREL and a listing file called &MAX1LST. The REL filename must be entered as five characters instead of four, so it is convenient to use the source filename plus R.

After the source file has been compiled and a prompt has been printed, exit the compiler. If the computer uses interrupts with the terminal, type Control C. If not, actuate the RESET switch on the computer front panel. Either action will return control to the monitor.

Using LINK-80
Load LINK-80 by typing

```
.L80
```

LINK-80 will respond with a "*" prompt. Load the program into memory by entering the name of the program REL file

```
*MAX1R
```

Executing and Saving the Program
Now you are ready to either execute the program that is in memory or save a memory image (object code) of the program on disk. To execute the program, type

```
*/G
```

To save the memory image (object code), type

```
*/E
```

which will exit from LINK-80, return to the DOS monitor and print three numbers: the starting address for execution of the program, the end address of the program, and the number of 256-byte pages used. For example

```
[26301 44054 35]
```

Use the DOS SAVE command to save a memory image. Type

```
.SAV MAX1 0 17100 44054 26301
```

17100 is the load address used by Altair DOS for the floppy disk. (With the hard disk, use 44000.)

An object code file h[s now been saved on the disk under the name specified in the SAVE command (in this case MAX1). To execute the program, simply type the program name

```
.MAX1
```

Altair DOS - Available Devices

FO:, F1:, F2:, ...	disk drives
TTY:	Teletype or CRT

Altair DOS Disk Filename Standard Extensions

FOR	FORTRAN-80 source file
COB	COBOL-80 source file
MAC	MACRO-80 source file
REL	relocatable object file
LST	listing file

Command Lines
Command lines are not supported by Altair DOS.

4.4 ISIS-II

Create a Source File
Create a source file using the ISIS-II editor. Filenames are up to six characters long, with 3-character extensions. FORTRAN-80 source filenames should have the extension FOR and COBOL-80 source filenames should have the extension COB. MACRO-80 source filenames should have the extension MAC.

Compile the Source File
Before attempting to compile the program and produce object code for the first time, it is advisable to do a simple syntax check. Removing syntax errors will eliminate the necessity of recompiling later. To perform the syntax check on the source file called MAX1.FOR, type

```
-F80 ,=MAX1
```

This command compiles the source file MAX1.FOR without producing an object or listing file. If necessary, return to the editor and correct any syntax errors.

To compile the source file MAX1.FOR and produce an object and listing file, type

```
-F80 MAX1,MAX1=MAX1
```

or

```
-F80 =MAX1/L/R
```

The compiler will create a REL (relocatable) file called MAX1.REL and a listing file called MAX1.LST.

Loading, Saving and Executing the Program (Using LINK-80)

To load the program into memory and execute it, type

```
-L80 MAX1/G
```

To save the memory image (object code), type

```
-L80 MAX1/E,MAX1/N
```

which will exit from LINK-80, return to the ISIS-II monitor and print three numbers: the starting address for execution of the program, the end address of the program, and the number of 256-byte pages used. For example

```
[210C 401A 48]
```

An object code file has now been saved on the disk under the name specified with /N (in this case MAX1).

ISIS-II - Available Devices

:F0:, :F1:, :F2:, ...	disk drives
TTY:	Teletype or CRT
LST:	line printer

ISIS-II Disk Filename Standard Extensions

FOR	FORTRAN-80 source file
COB	COBOL-80 source file
MAC	MACRO-80 source file
REL	relocatable object file
LST	listing file

ISIS-II Command Lines

ISIS-II command lines are supported as described in Section 4.1, CPM Command Lines.

Index

FORTRAN-80 under TEKDOS

FORTRAN-80 and MACRO-80

The FORTRAN-80 compiler and MACRO-80 assembler accept commands of the same format as TEKDOS assembler commands; i.e., three filename or device name parameters plus optional switches.

F80 [object-output][list-output]{source-input}[sw1][sw2]...

The object and listing file parameters are optional. These files will not be created if the parameters are omitted, however any error messages will still be displayed on the console. The available switches are as described in the FORTRAN-80 User's Manual and Microsoft Utility Software Manual, except that the switches are delimited by commas or blanks instead of slashes.

LINK-80

The LINK-80 loader accepts interactive commands only. When LINK-80 is invoked, and whenever it is waiting for input, it will prompt with an asterisk. Commands are lists of filenames and/or devices separated by commas and optionally interspersed with switches. The input to LINK-80 must be Microsoft relocatable object code (not the same as TEKDOS loader format).

Switches to LINK-80 are delimited by hyphens under TEKDOS, instead of slashes. All LINK-80 switches (as documented in the Microsoft Utility Software Manual) are supported, except "G" and "N", which are not implemented at this time.

Examples:

1. Compile a Fortran program named FTEST, creating an object file called FREL and a listing file called FLST:

```
>F80 FREL FLST FTEST
```

2. Load FTEST, link in the required library routines, and save the loaded module:

```
>L80
*FREL-E
[04AD  22B8]
*DOS*ERROR 46
L80 TERMINATED
>M FMOD 400 22B8 04AD
```

Note that "-E" exits via an error message due to execution of a Halt instruction. The memory image is intact, however, and the "Module" command may be used to save it. Once a program is saved in module format, it may then be executed directly without going through LINK-80 again. "-E" searches the system library (FORLBREL), if necessary, before exiting.

The bracketed numbers printed by LINK-80 before exiting are the entry point address and the highest address loaded, respectively. The loader default is to begin loading at 400H. However, the loader also places a jump to the start address in location 0, thereby allowing execution to begin at 0.

The memory locations between 0003 and 0400H are reserved for SRB's and I/O buffers at runtime. If you wish to load a program below 400H, then the I/O drivers should be altered. The modules that must potentially be modified for custom I/O are:

DSKDRV, TEKIO, INIT, LUNTB, IOINIT, EXIT

These source modules are provided on the standard distribution disks and may be modified and assembled using MACRO-80. If the modified I/O routines are then force-loaded before the library search, the standard library routines will not be loaded.

Disk I/O and LUN Assignments

(See FORTRAN-80 Reference Manual, Section 8.3.)

Logical units 1-4 are assigned to the console and may be used for either input or output.

Logical units 5-10 go through DSKDRV. They default to sequential disk files with the names

FOR05DAT,...,FOR10DAT.

These units may be re-assigned to any filename or device using an OPEN call. The form of an OPEN call is:

```
CALL OPEN(LUN, filename)
```

where LUN is a logical unit number (Integer variable or constant between 5 and 10), and filename is a Hollerith or Literal constant or variable containing the ASCII filename and/or device. The filename cannot have more than 11 characters, and it must be terminated by a blank or null character.

Examples:

```
CALL OPEN(8,'TSTFIL/1 ')
```

opens TSTFIL on drive 1 and associates it with LUN8.

```
CALL OPEN(5,'REMO ')
```

opens LUN5 for device REMO.

ADDENDUM
FORTRAN-80 under TEKDOS

CREF80

The form of commands to CREF80 is:

```
C80 {list-output}{cref-input}[sw1][sw2]...
```

Both filename parameters are required; switches are optional.

Example:

Create a CREF file using MACRO-80:

```
M80 ,, TSTCRF TSTMAC C
```

Create a cross reference listing from the CREF file:

```
C80 TSTLST TSTCRF
```