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)
I	Assemble 8080 Mnemonics (Default for 8080 oper. systems)
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 oper. 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 1.8)

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.

The assembler outputs a module name to the loader. This module name consists of the first six characters of the title if a TITLE statement is included. If no TITLE statement is included, the module name is created from the source file name.

1.3 Assembler Features

The features of the MACRO-80 assembler are described briefly below.

1.3.1 Names

All names are 1-6 characters. The first character is an alpha character (A-Z) or $. The remaining characters may be alphanumeric (A-Z, 0-9) or . or $ or ? or @. Names followed immediately by two number signs with no intervening blanks (e.g. NAME##) are classified as external. This type of name is an alternative to the program statement

```
     EXT    NAME
 or
     EXTRN  NAME
```

1.3.2 Constants

a. Decimal: Numbers consisting of decimal digits and having no leading zero. The allowable range is 65535 to -65535.

b. Octal: Numbers consisting of octal digits and having a leading zero or a trailing Q or O. The allowable range is 0177777 to -0177777.

c. Hex: Numbers consisting of one to four hexadecimal digits and having the form x'hhhh'. One-digit or three-digit values are treated as though zero were to the left (i.e., X'A' and X'0A' are the same). The allowable range is X'FFFF' to -X'FFFF'. Numbers consisting of from one to four hexadecimal digits immediately followed by the suffix H (e.g., hhhhH) are also allowed.

d. Binary: Numbers consisting of a string of binary digits (0's and 1's) followed by a B. (e.g., 101011B)

e. Character: One or two ASCII characters preceded and followed by quotation marks (i.e., "a" or "BC" or 'BC'). The delimiters may be either single quotes (') or double quotes ("), but the starting and end delimiters must be identical. Whenever one type of quote is used as a delimiter, the other type of quote is allowed as a character. Two-character strings are stored in low order byte/high order byte sequence. See Section 1.4.4.

1.3.3 Labels

A label is a name that does not contain an imbedded space and is terminated by a colon (:). Labels alone on a line with no further opcode or pseudo-op are allowed.

1.3.4 Operators

An operator consists of an 8080 mnemonic or one of the pseudo-operations described in Section 1.4.

1.3.5 Address Expressions

An address expression uses the current assigned address of a name or the 16-bit value of a constant to form a 16-bit value which, after the expression is evaluated, is truncated to the field size required by the operator.

1.3.6 Remarks

A remark always begins with a semicolon (;) and ends with a carriage return. A remark may be a line by itself or it may be appended to a line that contains a statement.

1.3.7 Statement Form

A statement consists of an optional label followed by an operator, followed by as many address expressioks as the operator requires, followed by an optional remark, and terminated by a carriage return. It is not necessary that statements begin in column 1. Multiple blanks or tabs may be used to improve readability (except inside character constants or character strings).

1.3.8 Expression Evaluation

Operator precedence during expression evaluation is as follows:

Parenthesized expressions
HIGH, LOW
*, /, MOD, SHL, SHR
+, - (unary and binary)
Relational Operators EQ, LT, LE, GT, GE, NE
Logical NOT
Logical AND
Logical OR, XOR

The Relational, Logical and HIGH/LOW operators must be separated from their operands by at least one space.

Byte Isolation Operators

The byte isolation operators are as follows:

HIGH	Isolate the high order 8 bits of a 16-bit value
LOW	Isolate the low order 8 bits of a 16-bit value

Example:

```
IF HIGH VALUE EQ 0
```

The above IF pseudo-op determines whether the high order byte of VALUE is zero.

Relational Operators

The relational operators are as follows:

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

These operators yeild a true or false result. They are commonly used in conditional IF pseudo-ops. They must be separated from their operands by spaces. Example:

```
IF LABEL1 EQ LABEL2
```

The above pseudo-op tests the values of LABEL1 and LABEL2 for equality. If the result of the comparison is true, the assembly language code following the IF pseudo-op is assembled, otherwise the code is ignored.

1.3.9 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.

1.4 Pseudo Operations

1.4.1 Define Byte

```
        DB    E1,E2,...,En
   or
        DB    "Character String"
   or
        DB    'Character String'
```

Each of the address expressions E1, E2,...En is evaluated and stored in n successive bytes. One- and two-character strings can be used in any expression. A string that is longer than two characters may only be used as a string.

Either single or double quotes may be used as character string delimiters, but the starting and end delimiters must be identical. It is permissible to use the delimiter quotes as characters, but the quote marks must appear twice for every character occurrence desired. For example:

```
        DB    "I am ""great"" today"
   will store
              I am "great" today
```

Each character in the character string is stored as one byte with its high-order bit set to zero.

1.4.2 Define Character

```
DC    "Character String"
```

Only double quotes may be used as character string delimiters, and double quotes may not be used as characters.

Each character in the character string is stored as one byte with its high-order bit set to zero except for the last byte which has its high-order bit set to one.

1.4.3 Define Space

```
DS    E
```

The address expression E is evaluated and that many bytes of space are allocated. All names used in E must be defined prior to the DS statement.

1.4.4. Define Word

```
DW    E1, E2, ..., En
```

Each address expression is evaluated and stored as n successive words. Example:

```
DW    'AB'
```

Two-byte values are stored in memory in low order byte/high order byte sequence. The ASCII code representation for character B is stored, then the character A is stored.

On the object code listing however, the printout for all two-byte values is in high order byte/low order byte sequence, for easier reading.

1.4.5 Program Termination

```
END    E
```

This statement is the last statement of each program. The optional address expression E gives the program execution address. If E evaluates to absolute zero, it is equivalent to no execution address.

1.4.6 Terminated Conditional Assembly

```
        ENDIF
```

Terminates conditional assembly initiated by a previous IFF or IFT.

1.4.7 Define Entry Points

```
        ENTRY     N1, N2, ..., Nn
   or
        PUBLIC    N1, N2, ..., Nn
```

The names N1, N2, ..., Nn are entry points from external programs and act as names for the program being assembled. The names must appear in an ENTRY or PUBLIC statement prior to their appearance as a label.

1.4.8 Define Equivalence

```
        Label   EQU    E
```

The label of the EQU statement is assigned the address given by address expression E. The label is required and must not have previously appeared as a label. All names used in E must be defined prior to the EQU statement.

1.4.9 Define External

```
        EXT       N1, N2, ..., Nn
   or
        EXTRN     N1, N2, ..., Nn
```

The names N1, N2, ..., Nn are defined to be external references and may not have been used as a label. Names may also be defined as external by using NAME##. See Section 1.3.1.

1.4.10 False Conditional Assembly

```
        IFF    E
```

The address expression E is evaluated and if it is False (=0), all statements down to the next ENDIF are assembled. If E is True (not =0), the statements are not assembled.

1.4.11 True Conditional Assembly

```
        IFT     E
   or
        IF      E
```

The address expression E is evaluated and if it is True (not =0), all statements down to the next ENDIF are assembled. If E is False (=0), the statements are not assembled. Unlimited nesting of conditionals is allowed.

1.4.12 Define Origin

```
        ORG     E
```

The address expression E is evaluated and the assembler assigns generated code starting with that value. All names used in E must be defined prior to the ORG statement, and the mode of E must not be external.

1.4.13 Page Break

```
        PAGE
```

A page break will occur on the listing. The PAGE statement will not list and code is not generated. If a TITLE statement has been included, the title (up to 125 characters) will be printed at the top of the page.

1.4.14 Set

```
        Label     SET     E
```

The label of the SET statement is assigned the address given by expression E. The label is required and must not have previously appeared as a label. All names used in E must be defined prior to the SET statement.

The difference between the SET and EQU statements is that SET allows redefinition of label values. Redefinition of a label by an EQU statement will result in an error.

1.4.15 Title

```
TITLE ICOMP INTEGER COMPARE ROUTINE
```

TITLE followed by a title of up to 125 characters is allowed. This title will appear at the top of each page. The title must be terminated by a carriage return. The module name that the assembler outputs to the loader is taken from the first six characters that follow the TITLE statement. If no TITLE statement is included, the assembler outputs to the loader a module name that is taken from the file name.

1.4.16 Memory Segment Specification

It is possible to specify that sections of a program be loaded in absolute, code relative or data relative segments of memory. The pseudo-ops are:

ASEG	For loading in an absolute segment of memory
DSEG	For loading in a data relative segment of memory
CSEG	For loading in a code relative segment of memory

One of the possible uses of these pseudo-ops is to specify RAM and ROM segments of memory. The data relative segment would be RAM, and the code relative segment would be ROM.

After an ASEG, CSEG, or DSEG pseudo-op is encountered, all following code is loaded in that area until a subsequent ASEG, CSEG or DSEG pseudo-op is encountered.

If none of these three pseudo-ops is specified, the default condition is to load everything code relative.

Additional flexibility in relocating code is possible through use of the ORG pseudo-op, which sets the value of the appropriate program counter. For example:

```
DSEG      Sets the data relative program
ORG 50    counter to a value of 50
```

NOTE

1. The Intel operands PAGE and INPAGE will generate expression errors when used with CSEG or DSEG pseudo-ops. These errors are warnings; the assembler ignores the operands.

2. In version 3.0 of the MACRO-80 Assembler, references to a particular external symbol may not be made in more than one memory segment. For example, an external symbol EXT1 might be referenced in the code relative segment, external symbols EXT3, EXT4 might be referenced in the data relative segment, but none could be referenced in more than one memory segment. (This restriction will be removed in a later release of the MACRO-80 Assembler.)

Refer to Section 2, LINK-80 Linking Loader, to determine how these segments are placed in specific areas of memory.

1.5 Notes

1. A dollar sign ($) indicates the value of the location counter at the start of the statement.

2. When the assembler is entered, the origin is assumed to be Relative-0.

3. Address expressions used in the conditional assembly pseudo-operations IFF and IFT must have all names defined prior to the use in the expression, and the expression must be Absolute.

4. Address expressions whose final mode is other than Absolute must generate assembly data that is stored as two bytes.

5. The following names are defined by the assembler to have the indicated Absolute values.

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

1.6 Sample Assembly

```
A>M80

*EXMPL1,TTY:=EXMPL1

       MAC80 3.0       PAGE     1

                       00100    ;CSL3(P1,P2)
                       00200    ;SHIFT P1 LEFT CIRCULARLY 3 BITS
                       00300    ;RETURN RESULT IN P2
  0000'                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
  0014'                02900            END

       MAC80 3.0       PAGE     2


CSL3      0000'    LOOP      0006'
```

1.7 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:

Code	Meaning
C	Too many ENDIFs
D	Bad octal or hex or binary digit
E	Expression error
L	No label in EQU
M	Label or symbol defined more than once
N	Name too long
O	Bad operator (opcode)
T	Illegal field termination
U	Undefined symbol
2	Missing second field for opcode
P	Phase error
Q	Missing or incorrect character string delimiter

1.8 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. /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
```

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.

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.

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     zzz + characters of symbol name
          --------  --------------------------------
          A field              B field
```

xxxx	Four-bit control field (0-15 below)
yy	Two-bit address type field
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	Reserved for future expansion

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. Reorder 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 $\begin{bmatrix}\text{Program}\\ \text{Data}\end{bmatrix}$ Area

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

?Intersecting $\begin{bmatrix}\text{Program}\\ \text{Data}\end{bmatrix}$ 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 $\begin{bmatrix}\text{Above}\\ \text{Below}\end{bmatrix}$ 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 <u>all</u>

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
```

<u>CPM - Available Devices</u>

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

<u>CPM Disk Filename Standard Extensions</u>

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
```

<u>DTC Microfile - Available Devices</u>

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

<u>DTC Disk Filename Standard Extensions</u>

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

FO:, 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.