MOSTEK®

Z80 MICROCOMPUTER SYSTEMS

Operations Manual

Z80 MACRO ASSEMBLER
VERSION 2.1
MACRO-80

MOSTEK MACRO-80

Z80 MACRO ASSEMBLER

VERSION 2.1

MK78165

MOSTEK MACRO-80

Z80 MACRO ASSEMBLER

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK.

MOSTEK MACRO-80

Z80 MACRO ASSEMBLER

VERSION 2.1

MANUAL REVISION 1.5

SECTION 1

OVERVIEW AND OPERATION

1-1. INTRODUCTION.

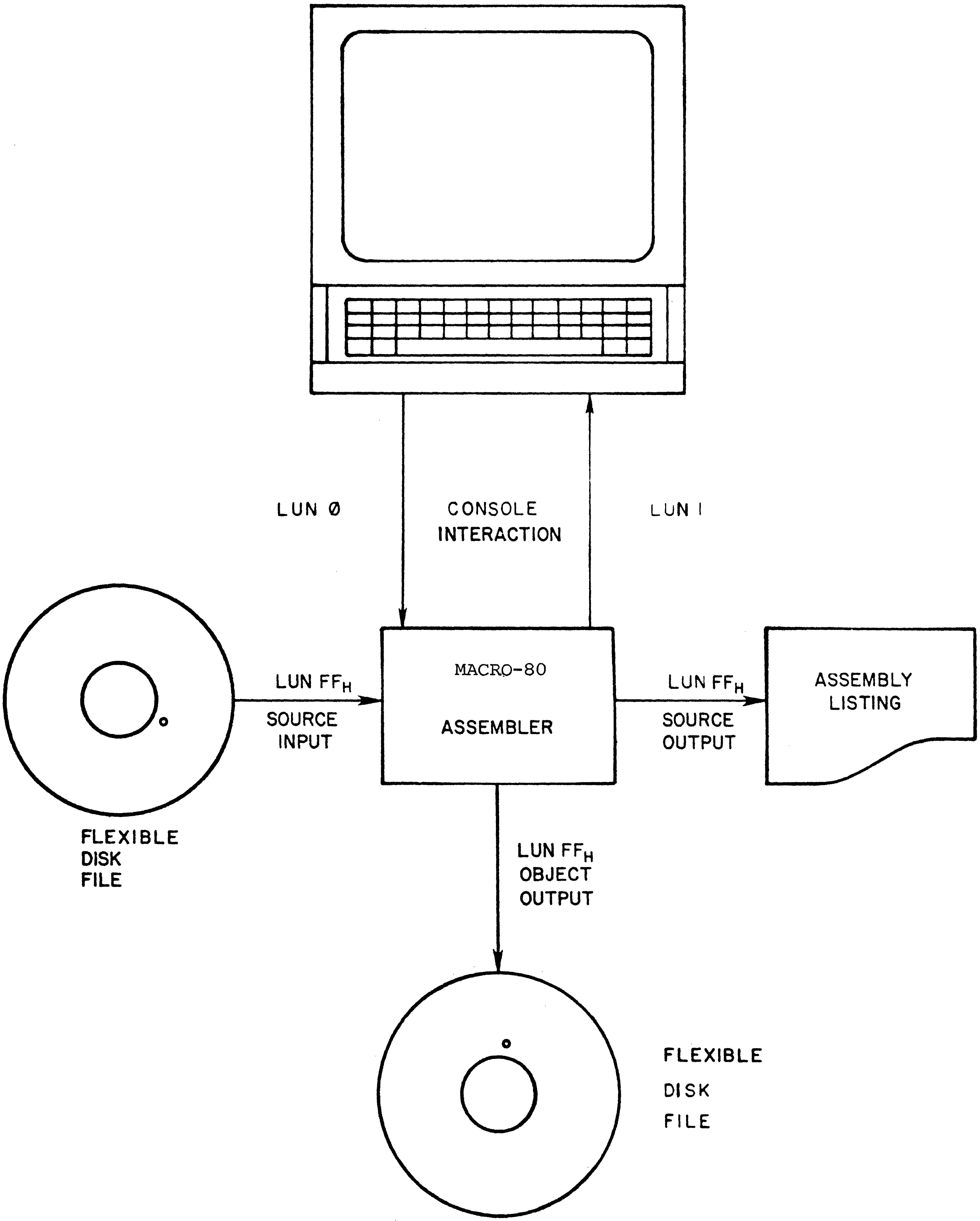
1-2. The MOSTEK Z80 Macro Assembler (MACRO-80) is designed to run under FLP-80DOS Version 2.0 or above with 32K or more of RAM. MACRO-80 is the most powerful macro assembler in the microcomputer market. It features:

1. optional arguments
2. default arguments
3. looping capability
4. global/local macro labels
5. nested/recursive expansions
6. integer/boolean variables
7. string manipulation
8. conditional expansion based on symbol definition
9. call by value facility
10. expansion of code producing statements only

1-3. MACRO-80 is an advanced upgrade from the FLP-80DOS Assembler (ASM). In addition to its macro capabilities, it provides for nested conditional assembly, and it allows symbol lengths of any number of characters. It supports global symbols, relocatable programs, a symbol cross reference listing, and an unused symbol reference table.

1-4. Figure 1-1. shows the Assembler with typical device usage. The source module is read from a disk file; the object output is directed to a disk file; the assembly listing is directed to a line printer. User interaction is via the console device. Note that the Assembler can interact with any dataset.

Figure 1-1. Typical Device Usage



1-5. REFERENCES.

AID-80F Operations Manual, MK78569
SYS-80F Operations Manual, MK78576
FLP-80DOS Operations Manual, MK78557

1-6. DEFINITIONS.

1-7. SOURCE MODULE - the user's source program. Each source module is assembled into one object module by the Assembler. The end of a source module is defined by an EOT character (ASCII 04) on input (standard end-of-file) or an END statement.

1-8. OBJECT MODULE - the object output of the Assembler for one source module. The object module contains linking information, address and relocating information, machine code, and checksum information for use by the FLP-80DOS Linker. The object module is in ASCII. A complete definition of the MOSTEK object format is given in Appendix B of the FLP-80DOS Operations Manual. The object module is typically output to a disk file with extension OBJ.

1-9. LOAD MODULE - the binary machine code of one complete program. The load module is defined in RAM as an executable program or on disk as a binary file (extension BIN). It is created by the Linker from one or more object modules.

1-10. LOCAL SYMBOL - a symbol in a source module which appears in the label field of a source statement.

1-11. INTERNAL SYMBOL - a symbol in a source (and object) module which is to be made known to all other modules which are linked with it by the Linker. An internal symbol is also called global, defined, public, or common. Internal symbols are defined by the GLOBAL pseudo-op. An internal symbol must appear in the label field of the same source module. Internal symbols are assumed to be addresses, not constants, and they will be relocated when linked by the Linker.

1-12. EXTERNAL SYMBOL - a symbol which is used in a source (and object) module but which is not a local symbol (does not appear in the label field of a statement). External symbols are defined by the GLOBAL pseudo-op. External symbols may not appear in an expression which uses operators. An external symbol is a reference to a symbol that exists and is defined as internal in another program module.

1-13. GLOBAL DEFINITION - both internal and external symbols are defined as GLOBAL in a source module. The Assembler determines which are internal and which are external.

1-14. POSITION INDEPENDENT - a program which can be placed anywhere in memory. It does not require relocating information in the object module.

1-15. ABSOLUTE - a program which has no relocating information in the object module. An absolute program which is not position independent can be loaded only in one place in memory in order to work properly.

1-16. RELOCATABLE - a program which has extra information in the object module which allows the Linker to place the program anywhere in memory.

1-17. LINKABLE - a program which has extra information in the object module which defines internal and external symbols. The Linker uses the information to connect, resolve, or link, external references to internal symbols.

1-18. CONVENTIONS USED IN THIS MANUAL.

1-19. All user input is underlined. Those items which must be entered exactly as shown are upper case. Those items which are variable are lower case. The symbol (CR) stands for carriage return.

1-20. USING THE ASSEMBLER.

1-21. The MACRO-80 Assembler is resident on a FLP-80DOS diskette. The user first prepares his source module using the FLP-80DOS Editor. Then the source file may be assembled via the following command:

```
        $MACRO dataset S [TO dataset L [,dataset O ]] (CR)
         ------------------------------------------------
                where dataset S = source input dataset
                      dataset L = assembly listing output dataset (optiona
                      dataset O = object output dataset (optional)
```

1-22. Dataset S is always a diskette file. Dataset L and dataset O are optional. If not given, dataset L defaults to the same disk unit and file name as dataset S, but the extension is LST. Dataset O, if not given, defaults to the same disk unit and file name as dataset L, but the extension is OBJ.

EXAMPLE

```
        $MACRO DK1:MYFILE TO CP:(CR)
         --------------------------
                - the user has selected to assemble file MYFILE on
```

disk unit 1. The listing is to be directed to the Centronics line printer device. The object will be directed to disk unit 1 on file MYFILE.OBJ.

1-23. ASSEMBLER OPTIONS

1-24. The Assembler allows the user to select the following options from the console when the Assembler outputs the message:

```
MOSTEK MACRO-80 ASSEMBLER V2.1. OPTIONS?
```

C - cross reference listing - prints a symbol cross reference table at the end of the assembly listing.

E - error exit - if any errors occur in pass 1 of the Assembler, they will be printed and pass 2 will not be done.

F - normal operation of pass 1 and pass 2 of the Assembler (default), switch off option E.

K - no listing - suppresses the assembly listing output. All errors will be output to the console device.

L - listing - the assembly listing will be output (default)

N - no object output - suppresses object output from the Assembler.

O - object output - the object output will be produced (default).

Q - quit - return to Monitor.

R - redefine opcodes - allows normal Z80 opcodes to be redefined by macros (default off).

U - unused symbols - a list of unused symbols will be printed at the start of the assembly listing.

V - switch off option U (default).

If no options are to be selected, the user enters a carriage return only.

EXAMPLE

```
        OPTIONS?NU(CR)
                ------
                - the user has selected no object output and
                an unused symbol listing.
```

1-25. ASSEMBLY LISTING OUTPUT

1-26. Figure 1-2. shows a sample Assembler listing output. The title (defined by the TITLE pseudo-op) is printed at the top of each page. The page number is in decimal notation. Three names appear in the second line at the top of each page. The first name is that of the source module; the second is the name of the object module; the third is that defined by the NAME pseudo-op. The key following the names is REL for a relocatable program and ABS for an absolute program.

1-27. Columns in the listing are automatically assigned by the Assembler. The LOC column defines the program address of the object code in hexadecimal. For relocatable programs, LOC is the relative offset from the start of the program. For absolute programs, LOC is the absolute address of the object code. The OBJ.CODE column defines the assembled Z80 opcode in hexadecimal. It is preceded by a quote (') if the statement contains a relocatable label. It is followed by a quote if the object code contains a relocatable address.

1-28. The STMT-NR heading defines two statement number columns. The column on the right defines a running statement number for all lines of the assembled program. The cross reference listing always refers to this number. The column on the left appears in programs with included files (INCLUDE pseudo-op) and/or macro expansions. Statement numbers are printed in decimal. The rest of each listing line is the source statement. If the line exceeds an 80 column width, then the source line is overflowed to the next line in the listing. The value of each equated symbol (EQU pseudo-op) is printed with an equal sign (=) next to it.

1-29. The number of lines printed per page of assembly listing is in address 0BH of the Assembler. The number of characters per line of listing is in address 0CH of the Assembler. Either of these values may be changed by the user. The default is 60 lines per page, 80 characters per line.

1-30. CROSS REFERENCE LISTING.

1-31. Figure 1-3. shows a cross reference listing, which is selected by option 'C'. The NAME column on the left hand side shows each symbol name used in the program in alphabetical order. The TYPE column indicates the type of the variable:

D	variable defined by DEFL pseudo-op
E	external variable
I	internal variable

```
URE 1-2. SAMPLE LISTING        MOSTEK MACRO-80 ASSEMBLER  V2.0 PAGE   2
  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 FIG1D2 FIG1D2 FIG1D2 REL

                         1          TITLE FIGURE 1-2. SAMPLE LISTING
                         2 SHIFT2   MACRO #REG #N #KIND ;GENERALIZED SHIFT MACRO
                    1    3          MLOCAL L1,L2,L3,L4,L5,L6,L7 ;LOCAL MACRO LABEL
                    2    4 N1       DEFL #N-1    ;GET NUMBER OF BITS TO SHIFT
                    3    5 NL       DEFL '%N1'[4,1]       ;PREPARE FOR CONDITIONAL J
                    4    6          AND  A         ;RESET CARRY BIT FOR SHIFT
                    5    7          MIF  ('%N1'<='0007').AND.('%N1'>='0001') THEN
                                                     L#NL
                    6    8          MERROR '  N>7 OR N<1  '
                    7    9          MEXIT
                    8   10 L7       #KIND #REG    ;SHIFT REGISTER NUMBER OF BITS
                    9   11 L6       #KIND #REG    ;SPECIFIED BY #N PARAMETER
                   10   12 L5       #KIND #REG
                   11   13 L4       #KIND #REG    ;THE TYPE OF SHIFT IS SHOWN
                   12   14 L3       #KIND #REG    ;BY THE #KIND PARAMETER
                   13   15 L2       #KIND #REG
                   14   16 L1       #KIND #REG
                   15   17          MEND
                           ;
     =0005              19 BB       EQU  5              ;DEFINE NBR OF BITS TO SHIFT
)0                      20          SHIFT2 A BB SRL
                    1   21          MLOCAL L1,L2,L3,L4,L5,L6,L7 ;LOCAL MACRO LABEL
     =0004          2   22 N1       DEFL BB-1           ;GET NUMBER OF BITS TO SHIFT
     =0034          3   23 NL       DEFL '0004'[4,1] ;PREPARE FOR CONDITIONAL JUMP
)0 A7               4   24          AND  A              ;RESET CARRY BIT FOR SHIFT
     =FFFF          5   25          MIF  ('0004'<='0007').AND.('0004'>='0001') THE
                                                     N L4
01 CB3F            11   26 L4       SRL  A              ;THE TYPE OF SHIFT IS SHOWN
03 CB3F            12   27 L3       SRL  A              ;BY THE SRL PARAMETEP
05 CB3F            13   28 L2       SRL  A
07 CB3F            14   29 L1       SRL  A
                   15   30          MEND
                           ;
09                 22   32          SHIFT2 A 'BB-2' RR
                    1   33          MLOCAL L1,L2,L3,L4,L5,L6,L7 ;LOCAL MACRO LABEI
     =0002          2   34 N1       DEFL BB-2-1         ;GET NUMBER OF BITS TO SHIFT
     =0032          3   35 NL       DEFL '0002'[4,1] ;PREPARE FOR CONDITIONAL JUMI
09 A7               4   36          AND  A              ;RESET CARRY BIT FOR SHIFT
     =FFFF          5   37          MIF  ('0002'<='0007').AND.('0002'>='0001') THI
                                                     N L2
0A CB1F            13   38 L2       RR   A
0C CB1F            14   39 L1       RR   A
                   15   40          MEND
                           ;
0E                 24   42          SHIFT2 L '2*BB' RL ;SHOULD GENERATE AN ERROR
                    1   43          MLOCAL L1,L2,L3,L4,L5,L6,L7 ;LOCAL MACRO LABE
     =0009          2   44 N1       DEFL 2*BB-1         ;GET NUMBER OF BITS TO SHIFT
     =0039          3   45 NL       DEFL '0009'[4,1] ;PREPARE FOR CONDITIONAL JUM
)0E A7              4   46          AND  A              ;RESET CARRY BIT FOR SHIFT
     =0000          5   47          MIF  ('0009'<='0007').AND.('0009'>='0001') TH
                                                     N L9
                    6   48          MERROR '  N>7 OR N<1  '
****ERR 5A **************           *
                    7   49          MEXIT
                           ;
)0F                26   51          END


ASS2 FIG1D2 FIG1D2 FIG1D2 REL  ERRORS :    1
```

```
 NAME TYP VALUE DEF   REFERENCES        PASS2 FIG1D3 FIG1D3 FIG1D3 REL

BB        0005   19     22   34   44
N1      D 0009   44     22*  23   25   25   34*  35   37   37   44*  45
                        47
NL      D 0039   45     23*  25   35*  37   45*  47
SHIFT2 M  1604    2     20   32   42
```

```
M        macro name
U        undefined symbol
blank    absolute value, not global
'        relocatable value, not global
2        multiply defined variable
```

1-32. The VALUE column shows the 16-bit value of the symbol. The DEF column shows the statement number in which the symbol is defined. REFERENCES defines each statement number in which the symbol is used. A reference marked with an asterisk means the variable is used as a 'target operand' in the statement. For example:

```
LD       (NN),A
SET      NBIT,B
         - the references of NN and NBIT are marked by an
         asterisk (*) in the cross reference listing.
```

1-33. OBJECT OUTPUT.

1-34. The object output of the Assembler can be loaded by an Intel hexadecimal loader for non-linkable programs. Extra information is inserted into the object output for linkable and relocatable programs for using the MOSTEK Linker. For a complete discussion of the object format, see Appendix B in the FLP-80DOS Operations Manual.

1-35. ERROR MESSAGES.

1-36. Any error which is found is denoted in the assembly listing. A message is printed immediately after the statement which is in error. An asterisk is printed under the location in the statement where the error was detected. All the error codes for this Assembler are defined in Appendix A of this manual.

EXAMPLE

```
                        H2:     LC      A,B
 *****ERR 41 BAD OPCODE         *
```

1-37. Several errors abort the Assembler when they are encountered. Abort errors are output only to the console device and control is immediately returned to the Monitor. Abort errors may occur during pass 1 or pass 2.

1-38. ADVANCED OPERATIONS.

1-39. Several source modules may be assembled together to form one object module. The INCLUDE pseudo-op may be used several times in one module to properly sequence a set of source modules.

EXAMPLE

```
NAME    MYFILE  ;name of final object module
INCLUDE FILE1
INCLUDE FILE2
INCLUDE FILE3
END

- the object module named MYFILE will be built by the
assembly from FILE1 + FILE2 + FILE3.
```

1-40. SAMPLE ASSEMBLY SESSION

1-41. Assume that the file to be assembled is named PROG1. The diskette on which PROG1 exists is in disk unit 1 (DK1). The object output of the Assembler is to be directed to file PROG1.OBJ on disk unit 1. The assembly listing is to be directed to a line printer (LP:). A cross reference table is to be printed.

EXAMPLE

```
$MACRO DK1:PROG1 TO LP:(CR)
 --------------------------
MOSTEK MACRO-80 ASSEMBLER V2.1.  OPTIONS? C(CR)
                                          -----
        - user selects a printed cross reference table

        .
        .
        .
$
        - indication that assembly is done and control is
        returned to the Monitor.
```

SECTION 2

ASSEMBLY LANGUAGE SYNTAX

2-1. INTRODUCTION.

2-2. An assembly language program (source module) consists of labels, opcodes, pseudo-ops, operands, and comments in a sequence which defines the user's program. The assembly language conventions for MACRO-80 are described below.

2-3. DELIMITERS.

2-4. Labels, opcodes, operands, and pseudo-ops must be separated from each other by one or more spaces or tab characters (ASCII 09). The operands must be separated from each other by commas. Operands in a macro call or macro definition statement may be separated from each other by one or more spaces or tab characters. The label may be separated from the opcode by a colon, only, if desired.

EXAMPLE

```
        label   opcode  operands        comment
        LAB1    LD      A,B             ;LOAD REGISTER A WITH B
```

2-5. LABELS.

2-6. A label may have any number of characters in it. The first six characters are decoded uniquely; any remaining characters are identified by a 'hash code'. This means that it is possible to use labels longer than 6 characters which appear different but are multiply defined by the Assembler. For example, 'ALABEL65' and 'ALABEL56' would be identified as the same label.

2-6A. The first character of a label must be alphabetic (A-Z). The remaining characters may be alphanumeric (A-Z, 0-9), question mark (?), or underline (_). Note that this is more restrictive than the FLP-80DOS ASM Assembler. A label may start in any column if immediately followed by a colon (:). It does not require a colon if started in column one.

EXAMPLE

```
        allowed         not allowed
```

-------	------------
LABEL1	1LAB4 (starts with a number)
HERE?	AD%DC (contains illegal character)

2-7. OPCODES.

2-8. There are 74 generic opcodes (such as LD), 25 operand key words (such as A), and 693 legitimate combinations of opcodes and operands in the Z80 instruction set. The full set of these opcodes is documented in the 'Z80 CPU Technical Manual'. The MACRO-80 Assembler allows one other opcode which is not explicitly shown in the Technical Manual:

```
        IN      F,(C)   ;SET CONDITION BITS ACCORDING TO THE CONTENTS
                        ;OF THE PORT DEFINED BY THE C-REGISTER
```

2-9. PSEUDO-OPS.

2-10. Pseudo-ops are used to define assembly time parameters. Pseudo-ops appear like Z80 opcodes in the source module. Several pseudo-ops require a label. The following pseudo-ops are recognized by the Assembler:

ORG nn - origin - sets the program counter to the value of the expression nn. Each origin statement in a program must be greater than the first origin of the program to assure proper linking.

label EQU nn - equate - sets the value of the label to nn in the progr where nn is an expression; it can occur only once for any label.

label DEFL nn - define label - sets the value of a label to nn in the program, where nn is an expression; it may be repeated in the program with different values for the same label. At any point in the program, the label assumes the last previously defined value. DEFL has certain other very useful properties associated with its use in macros. (See Section 3 of this manual).

DEFM m,m,m... - define message - defines the contents of successive bytes of memory according to m. m is composed of a sequence of either strings of characters surrounded by quotes or constants, each separated by one comma. Strings and constants may be mixed. The maximum length of the message is 63 bytes. The number of bytes allocated to a constant depends on its value. For example, the constant 0AF3H will have 2 bytes allocated to it, and OEFH will have one byte allocated. Symbols and expressions are not allowed in operands in the DEFM statement. The delimiting quote characters are required on a character string. A quote may be placed in a message by a sequence of 2 quotes (''). Example: DEFM 5H,'TEXT1',20414E4420H,'TEXT2'

DEFB n,n,n... - define byte - defines the contents of successive bytes starting at the current program counter address to be n, where n is any expression.

DEFW nn,nn,nn... - define word - defines the contents of successive two-byte words to be the value of expressions nn. The least significant byte of each expression is located at the current program counter address. The most significant byte is located at the program counter address plus one.

DEFS nn - define storage - reserves nn bytes of memory starting at the current program counter, where nn is an expression. When loaded, these bytes are not overwritten, i.e., they will contain what was previously in memory. This pseudo-op cannot be used at the start or end of a program to reserve storage.

END nn - end statement - defines the last statement of a program. The END statement is not required. The expression nn is optional and represents the transfer address (starting execution address) of the program. Note that for binary files the transfer address must be the same as the starting address.

GLOBAL symbol,symbol,... - define global symbol - any symbol which is to be made known among several separately assembled modules must appear in this type of statement. The Assembler determines if the symbol is internal (defined as a label in the program), or external (used in the program but not defined as a label).

NAME symbol - module name - This pseudo-op defines the name of the program (source and object). The name is placed in the heading of the assembly listing and is placed in the first record of the object module to identify it. This pseudo-op is designed primarily to facilitate future compiler design. The name of a module defaults to 6 blanks.

PSECT op - program section - may appear only once at the start of a source module. This pseudo-op defines the program module attributes for the following operands:

REL - relocatable program (default)
ABS - absolute program. No relocating information is generated in the object module. The module will be linked where it is origined.

IF nn or COND nn	- conditional assembly - if the expression nn is true (non-zero), the pseudo-op is ignored. If the expression is false (zero), the assembly of subsequent statements is disabled until an ENDIF statement is encountered. IF pseudo-ops can be nested to a level of 11.
ENDIF or ENDC	- end of conditional assembly - re-enables assembly of subsequent statements.
INCLUDE dataset	- include source from another dataset - allows source statements from another dataset to be included within the body of the given program. If a file name only is specified, then the file is searched for first on DK0:, then on DK1:. If the dataset cannot be opened properly, then assembly is aborted. The source module to be included must not end with an END pseudo-op (otherwise, assembly would be terminated). The source module must end with an EOT character (04H), which is true for all FLP-80DOS ASCII datasets. The INCLUDE pseudo-op cannot be nested, it cannot be followed by a comment on the same line, and it cannot appear in a macro definition.
LIST nn	- list all assembled statements (default on), where nn is an expression. If nn = 0 then the listing is turned off. Otherwise it is turned on.
ELIST nn	- list expanded statements from macro expansions - if the expression nn = 0, then only the macro call statements will appear in the assembly listing. Otherwise, all expanded statements from macro calls will appear in the assembly listing (default on).
CLIST nn	- list only code-producing statements from macro expansions - if the expression nn = 0, then only code-producing statements in the macro expansions will be listed. Otherwise all statements in each macro expansion will be listed in the assembly listing (default on).
NLIST	- turn off assembly listing. This is provided for compatibility with the FLP-80DOS ASM.
EJECT	- eject a page of the assembly listing.

TITLE s - print a title 's' at the top of each page of the listing. The title may be up to 32 characters in length.

2-11. OPERANDS.

2-12. There may be zero, one, or more operands in a statement depending upon the opcode or pseudo-op used. Operands in the Assembler may take the following forms:

2-13. GENERIC OPERAND. Table 2-1 summarizes the generic operands in the MACRO-80 Assembler.

2-14. CONSTANT. The constant must be in the range 0 thru OFFFFH. It may be in any of the following forms:

Decimal - this is the default mode of the Assembler. Any number may be denoted as decimal by following it with the letter 'D'. E.g., 35, 249D

Hexadecimal - must begin with a number (0-9) and end with the letter 'H'. E.g., 0AF1H

Octal - must end with the letter 'Q' or 'O'. E.g. 377Q, 277O

Binary - must end with the letter 'B'. E.g., 011011B

ASCII - letters enclosed in quote marks will be converted to their ASCII equivalent value. E.g., 'A' = 41H

2-16. LABEL. Labels cannot be defined by labels which have not yet appeared in the user program. This is an inherent limitation of a two pass assembler.

```
EXAMPLE not allowed             allowed
        -----------             -------
        L EQU H                 I EQU 7
        H EQU I                 H EQU I
        I EQU 7                 L EQU H
```

TABLE 2-1.

MACRO-80 GENERIC OPERANDS

A	A register (Accumulator)
B	B register
C	C register
D	D register
E	E register
F	F register (flags)
H	H register
L	L register
AF	AF register pair
AF'	AF' register pair
BC	BC register pair
DE	DE register pair
HL	HL register pair
SP	Stack Pointer register
$	Program Counter
I	I register (interrupt vector MS byte)
R	Refresh register
IX	IX index register
IY	IY index register
NZ	not zero
Z	zero
NC	not carry
C	carry
PO	parity odd/not overflow
PE	parity even/overflow
P	sign positive
M	sign negative

2-17. EXPRESSION. MACRO-80 recognizes a wide range of expressions in the operand field of a statement. All expressions are evaulated left to right constrained by the hierarchies shown in Table 2-2. Parentheses may be used to ensure correct expression evaluation. The symbol '$' is used to represent the value of the program counter of the current instruction. Note that enclosing an expression wholly in parentheses indicates a memory address. Integer two's complement arithmetic is used throughout. The negative (2's complement) of an expression or quantity may be formed by preceding it with a minus sign. The one's complement of an expression may be formed by preceding it with the '.NOT.' operator.

2-18. In doing relative addressing, the current value of the program counter may or may not be subtracted from the label, at the programmer's discretion:

```
        JR      LOOP
        JR      LOOP-$
                -will both jump relative to the label 'LOOP'.
```

2-19. The allowed range of an expression depends on the context of its use. An error message will be generated if this range is exceeded during its evaluation. In general, the limits on the range of an expression are 0 thru 0FFFFH. The range of a jump relative instruction (JR or DJNZ) is -126 bytes and +129 bytes. The Assembler monitors the number of items in an expression. If an expression is too long, an error message will be output. For relocatable programs the Assembler outputs relocation information in the object module for those addresses which are to be relocated by the Linker. Expressions are determined to be relocatable addresses or non-relocatable constants according to the rules shown in Table 2-3.

TABLE 2-2.

ALLOWED OPERATORS IN MACRO-80

OPERATOR	HIERARCHY	RELOCATE RULE	RANGE
.RES.	---	---	---
.DEF.	---	1	operand must be a symbol
unary +	1	1	
unary -			
**	1	2	
*	2	2	
/	2	2	operand 2 not = 0
+	3	3	
-	3	4	
.EQ. or =	4	5	string handling allowed
.LT. or <	4	5	
.GT. or >	4	5	
.LE. or <= or =<	4	5	
.GE. or >= or =>	4	5	
.NE. or <> or ><	4	5	
.ULT.	4	5	
.UGT.	4	5	
.AND.	5	2	
.OR.	6	2	
.XOR.	6	2	
.MOD.	6	2	
.NOT.	6	1	
.SHR.	6	2	operand 2 < 16
.SHL.	6	2	operand 2 < 16
[m,n]	---	---	operand must be a string

For relocate rules see Table 2-3.

TABLE 2-3.

RELOCATE RULES FOR OPERATORS

<operand 1> op	<operand 2>	Relocate rule 1 NOT	2 *	3 /	4 +	5 - >
		(rule number) / (mnemonic)				
relocatable	relocatable	ERR	ERR	ERR	ABS	ABS
relocatable	absolute	ABS	ERR	REL	REL	ABS
absolute	relocatable	ERR	ERR	REL	ERR	ABS
absolute	absolute	ABS	ABS	ABS	ABS	ABS

where ABS denotes absolute result
REL denotes relocatable result
ERR denotes error condition.

The following table shows the rules for global symbols used in relocatable and absolute programs.

	relocatable programs nn = rel	relocatable programs nn = abs	absolute programs nn = rel	absolute programs nn = abs
GS EQU nn	REL	ERR	REL	REL
LS EQU nn	REL	ABS	REL	ABS

where

GS denotes a global symbol
LS denotes a non-global symbol
nn is an expression
REL means relocatable result
ABS means absolute result
ERR denotes error condition

.RES. - reset overflow - appearance of this operator anywhere in an expression forces any overflow indication to be unconditionally reset.

.NOT. - one's complement.

** - exponentiation operator.

Relational operators (= > < etc.) can be used with character strings. This facility is useful when using macros to define a higher level language.

.ULT. - unsigned less than.

.UGT. - unsigned greater than.

.SHR. - shift first operand right by number of bits designated in second operand.

.SHL. - shift first operand left by number of bits designated by the second operand.

.DEF. - defined symbol operator - returns the value zero (false) if the symbol following the operator is not defined. Returns true (not zero) if the symbol is defined.

2-20. STRING EXPRESSIONS. The operator [,] extracts a substring from a given string. This is most useful in macros in which strings can be passed as arguments. Note that the Assembler does not support string variables. The general form of a string expression is:

```
        string[m,n]     or      string[m]

        where   string is any character string enclosed by quotes,
                [ and ] are delimiters,
                m is an integer which represents the starting
                  column number, and
                n is an integer which represents the number of
                  columns to be accessed.
```

2-21. If the integer n is not present, then n is assumed to be equal to the remaining number of columns in the given string.

EXAMPLE

```
        'ABCDEF'[3,2] is equivalent to 'CD'
        'ABCDEF'[3] is equivalent to 'CDEF'
```

2-22. COMMENTS.

2-23. A comment is defined as any set of characters following a semicolon in a statement. A semicolon which appears in quotes in an operand is treated as an expression rather than a comment starter. Comments are ignored by the Assembler, but they are printed in the assembly listing. Comments can begin in any column. Note that the Assembler also treats as comments any statements with an asterisk (*) in column one.

2-24. ABSOLUTE MODULE RULES.

2-25. The pseudo-op 'PSECT ABS' defines a module to be absolute. The program will be loaded in the exact addresses at which it is assembled. This is useful for defining constants, a common block of global symbols, or a software driver whose position must be known. This method can be used to define a list of global constants as follows:

EXAMPLE

```
                PSECT   ABS     ;ABSOLUTE ASSEMBLY
                GLOBAL  AA
        AA      EQU     0E3H
                GLOBAL  AX
        AX      EQU     0AF3H
                END
```

2-26. RELOCATABLE MODULE RULES.

2-27. Programs default to relocatable if the 'PSECT ABS' statement is not used or if 'PSECT REL' is used.

2-28. Only those values which are 16-bit address values will be relocated. 16-bit constants will not be relocated.

EXAMPLE

```
        AA      EQU     0A13H   ;ABSOLUTE VALUE
                LD      A,(AA)  ;AA NOT RELOCATED
        AR      EQU     $       ;RELOCATABLE VALUE
                LD      HL,(AR) ;AR WILL BE RELOCATED UPON LINKING
```

2-29. Relocatable quantities may not be used as 8-bit operands. This restriction exists because only 16-bit operands are relocated by the Linker.

EXAMPLE

```
        LAB     EQU     $       ;RELOCATABLE VALUE
```

```
                DEFB    LAB     ;NOT ALLOWED
                LD      A,(IX+LAB)      ;NOT ALLOWED
                LD      A,(LAB) ;ALLOWED
                LD      HL,LAB  ;ALLOWED
```

2-30. Labels equated to labels which are constants will be treated as constants. Labels equated to labels which are relocatable addresses will be relocated.

EXAMPLE

```
        B8      EQU     20H     ;CONSTANT
        C8      EQU     B8      ;CONSTANT
                LD      A,(C8)  ;C8 WILL NOT BE RELOCATED
        AR      EQU     $       ;RELOCATABLE ADDRESS
        BR      EQU     AR      ;RELOCATABLE
                LD      A,(BR)  ;BR WILL BE RELOCATED
```

2-31. External symbols in a relocatable program are marked relocatable, except for the first usage. The code for external symbols is actually a backward link list through the object code.

2-32. GLOBAL SYMBOL HANDLING.

2-33. A global symbol is a symbol which is known by more than one module. A global symbol has its value defined in one module. It can be used by that module and by any other module which is linked with it by the Linker. A global symbol is defined as such by the GLOBAL pseudo-op.

2-34. An internal symbol is one which is defined as global and also appears as a label in the same program. The symbol value is thus defined for all programs which use that symbol. An external symbol is one which is defined as global but does NOT appear as a label in the same program.

EXAMPLE

```
                GLOBAL  SYM1    ;DEFINE GLOBAL SYMBOL
                CALL    SYM1
                .
                .
                .
                END
                        - SYM1 is an external symbol
```

EXAMPLE

```
                GLOBAL  SYM1    ;DEFINE GLOBAL SYMBOL
        SYM1    EQU     $
                LD      A,(SYM1)
                .
```

```
                .
                .
                END
                        - SYM1 is an internal symbol.  Its value
                        is the address of the LD instruction.
```

2-35. If these two programs were assembled and then linked by the Linker, then all global symbol references from the first program would be 'resolved'. This means that each address in which an external symbol was used would be modified to the value of the corresponding internal symbol. The linked programs would be equivalent (using our example) to one program written as follows:

EXAMPLE

```
                CALL    SYM1
                .
                .
                .
        SYM1    EQU     $
                LD      A,(SYM1)
                .
                .
                .
                END
```

2-36. Global symbols are used to allow large programs to be broken up into smaller modules. The smaller modules are used to ease programming, facilitate changes, or allow programming by different members of the same team.

2-37. GLOBAL SYMBOL RULES.

2-38. An external symbol cannot appear in an expression which uses operators.

EXAMPLE

```
                GLOBAL  SYM1    ;EXTERNAL SYMBOL
                CALL    SYM1    ;OK
                LD      HL,(SYM1+2)     ;NOT ALLOWED
```

2-39. An external symbol is always considered to be a 16-bit address. Therefore, an external symbol cannot appear in an instruction requiring an 8-bit operand.

EXAMPLE

```
                GLOBAL  SYM1    ;EXTERNAL SYMBOL
                CALL    SYM1    ;OK
                LD      A,SYM1  ;NOT ALLOWED
```

2-40. An external symbol cannot appear in the operand field of an EQU or DEFL statement.

2-41. For a set of modules to be linked together, no duplication of internal symbol names is allowed. That is, an internal symbol can be defined only once in a set of modules to be linked together.

SECTION 3

MACRO CAPABILITY

3-1. INTRODUCTION.

3-2. MACRO-80 offers the most advanced macro handling capability in the microcomputer industry. Macros provide a means for the user to define his own opcodes or to redefine existing opcodes. A macro defines a body of text which will be inserted automatically into the source program at each occurrence of a macro call. Parameters associated with a macro provide a capability for making changes in the macro at each call. The following paragraphs describe how to use the macro facility.

3-3. MACRO DEFINITION.

3-4. The body of text to be used as a macro is given in the macro definition. Each definition begins with a MACRO pseudo-op and ends with an MEND pseudo-op. The general form is:

```
        label   opcode  operands                 comment

        name:   MACRO   #p1,#p2,...,#pn          ;comments (optional)
                .
                .       body of macro goes here
                .
        label:  MEND
```

3-5. The name is required, and it must obey all the usual rules for forming labels (recall that the colon is optional if the name starts in column one). If the name is a Z80 opcode (e.g., LD, EXX), then the 'R' option must be selected at the start of the Assembler to permit redefinition of opcodes by macros.

3-6. There can be any number of parameters from 0 to 99, each starting with the symbol '#'. The rest of the parameter name follows normal symbol rules. Parameter names are not entered into the symbol table. Parameters are separated from each other by single commas, or one or more blanks, or one or more tab characters.

3-7. The label on the MEND statement is optional, but if one is given it refers to the next program address upon expansion of the macro.

3-8. Each statement between the MACRO and MEND statements is entered into a temporary macro file. The only restriction on these statements is that they do not include another macro definition (nested definitions are not allowed) or an INCLUDE statement. They may include macro calls. The depth of nested calls is limited only by available memory space for buffering.

3-9. The statements of the macro body are not assembled at definition time, so they will not define labels, generate code, or cause errors. Exceptions are the Assembler commands such as LIST which are processed whenever they are encountered. Within the macro body text, the formal parameter names may occur anywhere that an expansion-time substitution is desired. This also applies to comments and quoted strings. However, no substitution of parameters is performed for comments defined by an asterisk in column one.

3-10. Macros must be defined before they are called. Once defined, a macro cannot be redefined within the same program. If a macro is called by another macro, then its definition must precede the calling macro's definition.

3-11. MACRO CALLS AND MACRO EXPANSION.

3-12. A macro is called by using its name as an opcode at any point after the definition. The general form is:

```
        label   opcode  operands                comment

        label   name    s1,s2,...,sn            ;comment (optional)
```

3-13. The label is optional and will be assigned to the current value of the program counter. The name must be a previously defined macro. There may be any number of argument strings s1 thru sn, separated by any number of blanks or tabs or single commas. The comma can be used as a place holder to pass null arguments to the macro expansion. All arguments are passed. If too few are passed, the remaining arguments assume the value of null (no characters in the argument string). If there are too many arguments, the extras may be accessed by the MNEXT pseudo-op (described below).

3-14. The position of each string in the list corresponds to the position of the macro parameter name it is to replace. Thus, the third string in a macro call statement will be substituted for each occurrence of the third parameter name.

3-15. Each string may be of any length and may contain any characters. Quotes around the string are optional; they are required if the string contains delimiters or the quote character itself. The quote character is represented by a sequence of two successive quote characters at the inner level. The outer level of quotes, if present, will not occur in the substitution, i.e., they are stripped from the argument. The null string, represented by two successive quote characters, may be used in any parameter position.

3-16. After processing the macro call statement, the Assembler switches its input from the source file to the macro file. Each statement of the macro body is scanned for occurrences of parameter names. For each occurrence found, the corresponding argument string from the macro call statement is substituted. After substitution, the statement is assembled normally.

3-17. Default arguments may be specified in the parameter list by use of an equal sign (=). The call to the macro must specify comma place holders for each default argument to be substituted (otherwise the null argument will be substituted).

EXAMPLE

```
        MAC1    MACRO   #A=DE,#B=HL,#C=BC
                .
                .
                .
                MEND

                MAC1            ;EXPANSION WITH NO ARGUMENTS
                .               ;ALL ARGUMENTS WILL DEFAULT TO NULL
                .
                .
                MEND

                MAC1    ,,,     ;EXPANSION TO USE DEFAULT ARGUMENTS
                .               ;DEFAULT ARGUMENTS WILL BE
                .               ; USED FOR PARAMETERS #A, #B, AND #C
                .
                MEND
```

3-18. RECURSION.

3-19. Macros may include calls to other macros, including themselves. The definition statements of a macro which calls other macros must follow the definition statements of those macros. A macro which directly calls itself (or indirectly by calling a second macro which calls the first macro) is said to be recursive. Each recursive call causes a new expansion of the macro, possibly with different parameters. In order to prevent the macro from being called endlessly, conditional assembly can be used to inhibit a recursive call when certain conditions are met. A recursion of greater than 255 calls will generate an error.

3-20. SUBSTITUTION BY VALUE (% OPERATOR).

3-21. Symbol values can be expanded within a macro by preceding the symbol name with a percent sign (%). The symbol must appear as the label of a DEFL statement. The value of the symbol is expanded to 4 decimal digits when the macro is called.

3-22. The value of an argument may be substituted by value by using the DEFL statement and the % operator. In this case, some symbol is equated to the parameter via the DEFL pseudo-op. The value of the symbol is then expanded to four decimal digits by using the % operator. This facility can be used only within a macro.

The DEFL statement within a macro also has the characteristic that it can be expanded just like a macro parameter. The symbol defined by the DEFL pseudo-op can be preceded by a # sign elsewhere in the macro definition to expand its value as ASCII characters. See the example below.

EXAMPLE

```
MAC1    MACRO   #N
N1      DEFL    #N-1
NL      DEFL    '%N1'[4,1]          ;GET ONE-DIGIT ASCII NUMBER
        JP      L#NL
L1      ...
L2      ...
L3      ...
L4      MEND

BB      EQU     4
        MAC1    BB                  ;EXPANSION
N1      DEFL    3
NL      DEFL    '0003'[4,1]
        JP      L3
L1      ...
L2      ...
L3      ...
L4      MEND
```

3-23. PREDEFINED ARGUMENTS.

3-14. The following predefined arguments are unique symbols and may be used anywhere in the macro definition.

%NEXP - expands to a four decimal digit representation of the number of the expansion of any macro. Thus, the first expansion of any macro yields %NEXP = 0001, the second yields %NEXP = 0002, etc.

EXAMPLE

```
        MAC1    MACRO
                DEFW    %NEXP
                MEND

                MAC1            ;1ST EXPANSION
                DEFW    0001
                MEND
                MAC1            ;2ND EXPANSION
                DEFW    0002
                MEND
```

%NARG - expands to a four decimal digit representation of the number of aguments passed to the macro expansion.

EXAMPLE

```
        MAC1    MACRO   #A,#B,#C
                LD      A,%NARG
                MEND

                MAC1    1,2     ;EXPANSION
                LD      A,0002
                MEND
```

#PRM - expands to the last used argument. Note that the first parameter of the macro must be expanded explicitly before #PRM is used. Alternatively, the MNEXT pseudo-op can be used to access the first parameter. See the discussion of MNEXT, below.

EXAMPLE

```
        MAC1    MACRO   #A,#B
                LD      HL,#A
                LD      DE,#PRM
                LD      BC,#B
                LD      IY,#PRM
                MEND

                MAC1    SYM1,SYM2       ;EXPANSION
                LD      HL,SYM1
                LD      DE,SYM1
                LD      BC,SYM2
                LD      IY,SYM2
                MEND
```

%NPRM - expands to a two decimal digit representation of the position number of the last used argument. This shows the position of an argument in the argument list.

EXAMPLE

```
        MAC1    MACRO   #A,#B
                LD      HL,#B
```

```
                LD      A,%NPRM
                MEND

                MAC1    SYM1,SYM2       ;EXPANSION
                LD      HL,SYM2
                LD      A,02
                MEND
```

%NCHAR - expands to a two decimal digit representation of the number of characters in the last used argument.

EXAMPLE

```
        MAC1    MACRO   #A #B
        P1      DEFL    $       ;#A
                DEFB    %NCHAR
                DEFM    '#A'
        P2      DEFL    $       ;#B
                DEFB    %NCHAR
                DEFM    '#B'
                MEND

                MAC1    A BCDE  ;EXPANSION
        P1      DEFL    $       ;A
                DEFB    01
                DEFM    'A'
        P2      DEFL    $       ;BCDE
                DEFB    04
                DEFM    'BCDE'
                MEND
```

3-25. FORMATION OF LABELS WITHIN A MACRO EXPANSION.

3-26. There are three ways of forming unique labels within a macro expansion.

3-27. PREDEFINED ARGUMENT %NEXP. The current expansion number will be expanded as four decimal digits, which may be appended to a character or set of characters to form a unique label.

EXAMPLE

```
        MAC1    MACRO   #A
        L%NEXP  LD      HL,#A
                MEND

                MAC1    SYM             ;EXPANSION 1
        L0001   LD      HL,SYM
                MEND
                MAC1    SYM2            ;EXPANSION 2
        L0002   LD      HL,SYM2
                MEND
```

3-28. SUBSTITUTION OF PARAMETER. Unique labels may be formed by using a parameter as part of the label. A passed argument then defines a label or set of unique labels for the given expansion.

EXAMPLE

```
        MAC1    MACRO #A
        L#A     DEFM    'A MESSAGE'
        M#A     DEFB    9
                MEND

                MAC1    FST     ;EXPANSION
        LFST    DEFM    'A MESSAGE'
        MFST    DEFB    9
                MEND
                MAC1    SND     ;EXPANSION 2
        LSND    DEFM    'A MESSAGE'
        MSND    DEFB    9
                MEND
```

3-29. DOT OPERATOR (.). Symbols in a macro definition may have a dot as the first character. The dot in every symbol will be replaced by the label specified in the macro call statement during macro expansion. Labels formed by the dot operator may also be used in MGOTO, MIF, and MNEXT statements.

EXAMPLE

```
        MAC1    MACRO                   ;MACRO DEFINITION
        .L1     LD      HL,.L2
                .
                .
                .
        .L2
        .LAB
                MEND

        M1      MAC1                    ;THE MACRO CALL
        M1L1    LD      HL,M1L2
                .
                .
                .
        M1L2
        M1LAB
                MEND
```

Note that the dot operator can be used with a parameter if the two items are separated by another character.

EXAMPLE

```
        MAC1    MACRO   #A              ;MACRO DEFINITION
                LD      HL,.L#A
                ...
        .L#A
```

```
                MEND

        M4      MAC1    25      ;MACRO CALL
                LD      HL,M4L25
                ...
        M4L25
                MEND
```

3-30. LOCAL MACRO LABELS.

3-31. Local macro labels are allowed only in the MGOTO, MIF, and MNEXT statements. Local macro labels must follow normal symbol rules. They may not be formed by use of predefined arguments, substitution of parameters, or by use of the dot operator. Each local macro label will be in effect only during the current expansion of the current macro. They are in effect from the time of declaration via the MLOCAL pseudo-op through the MEND pseudo-op. They may not be redefined or respecified within one macro. Local declarations of the same symbol in nested or recursive macro calls are allowed. Local macro labels are not placed in the symbol table; they are used merely as pointers for the MGOTO, MIF, and MNEXT statements. A local macro label must be declared before it is used. The format for declaring local macro labels is:

```
        MLOCAL  mlabel1,mlabel2,...
```

- where mlabel1, mlabel2, etc., are labels which only appear in the macro body. The MLOCAL statement may not have a label on it.

EXAMPLE

```
        MAC1    MACRO   #A,#B
                MLOCAL  L1,L2,L3
                MIF     '#A'='IF' THEN L1 ELSE L3
        L1      MIF     '#B'='' THEN L2 ELSE L3
        L2      MERROR  BAD IF STATEMENT
        L3      MNOP
                MEND
```

3-32. MACRO RELATED PSEUDO-OPS.

3-33. In the following discussion, mlabel, mlabel1, and mlabel2 refer to local macro labels or labels formed by using the dot operator (.). The symbol nn refers to any valid expression. Brackets [ ] refer to optional parameters.

3-34. MNEXT nn [THEN mlabel1] [ELSE mlabel2]

- moves the argument pointer according to the expression nn in the argument list. A move to the left can be achieved by a negative value, to the right by a positive value. The argument may then be accessed by the #PRM predefined argument. If the argument pointer leaves the argument list and if the ELSE clause is present, then a jump to mlabel2 is performed. Otherwise the next statement in sequence is processed.

EXAMPLE

```
        MAC1    MACRO   #A,#B
                MLOCAL  L1,L2
        L1      MNEXT   1 ELSE L2
                DEFB    #PRM
                MGOTO   L1
        L2      MEND

                MAC1    1,2,3   ;EXPANSION
                MLOCAL  L1,L2
        L1      MNEXT   1 ELSE L2
                DEFB    1
                MGOTO   L1
        L1      MNEXT   1 ELSE L2
                DEFB    2
                MGOTO   L1
        L1      MNEXT   1 ELSE L2
                DEFB    3
                MGOTO   L1
        L1      MNEXT   1 ELSE L2
        L2      MEND
```

3-35. MGOTO mlabel

- continues the expansion at the specified macro label.

EXAMPLE

See the EXAMPLE for the MNEXT pseudo-op.

3-36. MIF nn THEN mlabel1 [ELSE mlabel2)

- if the expression nn evaluates to true (non-zero), then expansion is continued at the mlabel1 macro label. If the expression is false (equals zero) and the ELSE clause is present, expansion continues at the mlabel2 macro label. Otherwise expansion continues at the next statement in the macro.

EXAMPLE

```
        MAC1    MACRO   #A
                MLOCAL  L1,L2
                MIF     '#A'='THEN' THEN L1 ELSE L2
        L1      DEFM    '#A'
        L2      MEND
```

```
                MAC1    THEN     ;FIRST EXPANSION
                MLOCAL  L1,L2
                MIF     'THEN'='THEN' THEN L1 ELSE L2
        L1      DEFM    'THEN'
        L2      MEND

                MAC1    ELSE
                MLOCAL  L1,L2
                MIF     'ELSE'='THEN' THEN L1 ELSE L2
        L2      MEND
```

3-37. MNOP

- no operation is performed. This pseudo-op can be used to define a local macro label at this point in the macro body. This is useful because the local macro labels will not appear in the assembly listing if the CLIST 0 pseudo-op is used.

3-38. MEXIT

- terminates the current macro expansion.

EXAMPLE

```
        MAC1    MACRO   #A
                MLOCAL  L1
                MIF     '#A'='THEN' THEN L1
                MEXIT
        L1      MNOP
                LD      A,1
                MEND

                MAC1    ELSE
                MLOCAL  L1
                MIF     'ELSE'='THEN' THEN L1
                MEXIT
```

3-39. MERROR text

- prints the line of text like an error message with error number 5A called out.

EXAMPLE

```
        MAC1    MACRO
                MLOCAL  L1,L2,L3
                MNEXT   1 ELSE L2
        L1      ...
                MGOTO   L3
        L2      MERROR  ARGUMENTS REQUIRED
        L3      MEND

                MAC1
                MLOCAL  L1,L2,L3
                MNEXT   1 ELSE L2
```

```
        L2       MERROR  ARGUMENTS REQUIRED
*****ERR 5A *************
        L3       MEND
```

3-40. MEND

- marks the end of a macro.

3-41. MLOCAL label1,label2,...

- defines local macro labels.

SECTION 4

APPLICATIONS OF MACROS

4-1. INTRODUCTION.

4-2.
The MACRO-80 Assembler provides a powerful tool for microcomputer systems development. Five areas of applications are discussed below to show how the macro facility can be used to simplify program development:

1. Use of macros in implementing special-purpose languages.
2. Emulation of non-standard machine architectures.
3. Development of cross-assemblers.
4. Implementation of additional control structures.
5. Operating systems interface macros.

4-3. As macros are developed by a team of programmers, it is important to document each macro and its usage for each member of the team. The examples below should be studied for both their procedural content and the method of documenting them.

4-4. SPECIAL PURPOSE LANGUAGES.

4-5. A wide variety of microcomputer designs can be broadly classed as 'controller' designs. In these designs, the microcomputer is the controlling element in sequencing and decision-making as real-time events are sampled and directed. An example of this is a traffic control system. In this situation, it is useful to define a 'language' via macros which suits the particular application. After the macros are defined, an application programmer can use them as primitive language elements. If properly defined, the application language is easily programmed and can allow considerable machine independence. Further, the macros can incorporate debugging facilities to aid the application programmer.

4-6. In the traffic system defined here, the following hardware elements are present:

1. central and corner traffic lights which display green, yellow, red, or are off completely.
2. pushbutton switches for pedestrian crosswalks.
3. road treadles for sensing the presence of an automobile at an intersection.
4. a central controller box.

4-7. The central controller box contains a microprocessor connected through external logic to relays which control the lights and to latches which hold sensor input information. The controller also contains a time-of-day clock which counts hours from 0 through 23. The program which is run on the microprocessor is contained in PROM and is tailored to each intersection for traffic control.

4-8. We first define a set of macros to perform simple traffic-control functions via the system. These are shown in Figure 4-1. The system is configured such that the central traffic light is controlled by the microprocessor port number 0 (given by LIGHT). The time-of-day clock is read from port 3 (given by CLOCK). The north-south direction of the traffic light is controlled by the high order 4 bits of output port 0, and the east-west direction is controlled by the low order 4 bits of port 0. When either of these fields is set to 0, 1, 2, or 3, then the light in that direction is turned off or set to red, yellow, or green, respectively. Thus, the SETLITE macro sets the specified direction to the appropriate color.

4-9. The TIMER macro uses the cycle time of the microprocessor (one cycle = 400 nanoseconds) to construct an inline timing loop, based on the number of seconds delay requested.

4-10. Additional macros are provided for automobile treadles and pedestrian pushbuttons. For treadles (macro TREAD?) the sensors are attached to port 1 of the microprocessor (TRINP). The treadles require a 'reset' operation which is performed via port 1 (TROUT). At any intersection, the treadles are numbered clockwise from north from 0 through a maximum of 7. Each sensor and reset position of the treadle port corresponds to one bit position of port 1. Thus treadle #0 sensor is read from bit 0 of port 1 and reset via bit 0 of port 1. The TREAD? macro is used to sense the presence of a latched value for treadle #TR and, if on, the sensor is reset with control transferring to the label given by #IFTRUE.

4-11. Latched pedestrian pushbuttons are processed by the macro PUSH?. A latched pushbutton is sensed on input port 0 (CWINP) as a sequence of 1's and 0's in the least significant positions, corresponding to the switches at the intersection. Thus, if there are four pedestrian pushbuttons, bits 0, 1, 2, and 3 corresponds to these switches. A set bit in any of these positions indicates that a button has been pushed. All the crosswalk latches are reset whenever input port 0 is read.

4-12. Figure 4-2 shows a program written in the macros for controlling a rather simple intersection. Here, the lights are merely sequenced in proper fashion for traffic control.

4-13. Figure 4-3 shows a more complex intersection control program. In this case, heavy traffic normally occurs in an East-West direction. Light traffic from a residential section occurs in a North-South direction. Here, the lights favor traffic in the East-West direction until an automobile treadle or a pedestrian pushbutton is activated.

```
;       FIGURE 4-1
        NLIST
;
**************************************************
; MACRO LIBRARY FOR TRAFFIC CONTROL APPLICATION
**************************************************
;
; THIS LIBRARY CONTAINS SEVERAL MACROS WHICH
; DEFINE A LANGUAGE FOR A TRAFFIC CONTROL APPLICATION.
; THE LANGUAGE IS DEFINED AS FOLLOWS:
;
; SETLITE  DIR,COLOR
;       - SET THE COLOR LIGHT IN THE DIRECTION SHOWN
;       WHERE COLOR IS OFF, RED, YELLOW, OR GREEN AND
;       DIRECTION IS 'NS' FOR NORTH-SOUTH OR 'EW' FOR
;       EAST-WEST.

; TIMER    SECONDS
;       - DELAY THE NUMBER OF SECONDS SHOWN
;
; CLOCK     LOW,HIGH,LABEL
;       - TRANSFER CONTROL TO THE 'LABEL' IF
;       THE CURRENT HOUR (0-23) IS BETWEEN 'LOW'
;       AND 'HIGH'.
;
; RETRY   LABEL
;       - TRANSFER CONTROL TO 'LABEL'.
;
; TREAD?  TR,LABEL
;       - INTERROGATE TREADLE NUMBER 'TR' AND
;       IF THE INPUT IS SET, RESET IT AND TRANSFER
;       CONTROL TO 'LABEL'.
;
; PUSH?   LABEL
;       - CHECK IF ANY PUSHBUTTON HAS BEEN PUSHED.
;       IF SO, TRANSFER CONTROL TO 'LABEL'.
;
;
; INPUT PORTS FOR LIGHT AND CLOCK
;
LIGHT   EQU     0       ;TRAFFIC LIGHT CONTROL
CLOCK   EQU     3       ;24 HOUR CLOCK (0-23)
;
; CONSTANTS FOR TRAFFIC LIGHT CONTROL
;
BITSNS  EQU     4       ;NORTH-SOUTH BITS
BITSEW  EQU     0       ;EAST-WEST BITS
;
OFF     EQU     0       ;TURN LIGHT OFF
RED     EQU     1       ;RED LIGHT
YELLOW  EQU     2       ;YELLOW LIGHT
GREEN   EQU     3       ;GREEN LIGHT
;
;
;
; SET LIGHT IN DIRECTION #DIR (NS, EW) TO #COLOR (OFF,
; RED, YELLOW, GREEN)
SETLITE MACRO   #DIR,#COLOR
        LD      A,#COLOR.SHL.BITS#DIR   ;READY COLOR BITS
        OUT     (LIGHT),A       ;OUTPUT TO LIGHT
```

```
        MEND
;
; TIMER FOR NUMBER OF SECONDS TO DELAY
TIMER   MACRO   #SECOND
        LD      BC,1000*#SECOND ;SECONDS TIMES MSECS
L%NEXP  PUSH    BC              ;SAVE IT
        LD      B,191           ;MILLISECOND COUNTER
K%NEXP  DJNZ    K%NEXP          ;LOOP FOR 1 MSEC
        POP     BC
        DEC     BC      ;DECREMENT MSEC COUNT
        LD      A,B             ;CHECK FOR END OF SECONDS
        OR      C
        JR      NZ,L%NEXP       ;LOOP FOR MORE
; ARRIVE HERE AFTER APPROXIMATE DELAY OF 'SECONDS'
        MEND
;
;
; CHECK CLOCK AND JUMP TO #IFTRUE IF TIME IS BETWEEN #LOW AND #HIGH
CLOCK?  MACRO   #LOW,#HIGH,#IFTRUE
        MLOCAL  L2
        IN      A,(CLOCK)       ;READ CLOCK
; IF UPPER LIMIT NOT INPUT, DON'T CHECK IT
        MIF '#HIGH'='' THEN L2
        CP      #HIGH   ;EQUAL OR GREATER?
        JR      NC,F%NEXP       ;IF SO, SKIP OUT
L2      MNOP
        CP      #LOW    ;LESS THAN LOW VALUE?
        JP      NC,#IFTRUE      ;IF SO, EXIT TO LABEL
F%NEXP
        MEND
;
;
; RETRY BY GOING TO '#LABEL'
RETRY   MACRO   #LABEL
        JP      #LABEL
        MEND
;
;
TRINP   EQU     1       ;TREADLE INPUT PORT
TROUT   EQU     1       ;TREADLE OUTPUT PORT
;
; CHECK IF TREADLE '#TR' HAS BEEN SENSED.  IF SO, RESET
; AND EXIT TO LABEL '#IFTRUE'.
TREAD?  MACRO   #TR,#IFTRUE
        IN      A,(TRINP)       ;CHECK FOR TREADLE SET
        AND     1.SHL.#TR       ;CHECK FOR THIS TREADLE
        JR      Z,F%NEXP        ;IF NOT, SKIP OUT
        LD      A,1.SHL.#TR     ;ELSE RESET THE BIT
        OUT     (TROUT),A       ;TO CLEAR IT
        JP      #IFTRUE         ;EXIT VIA LABEL
F%NEXP
        MEND
;
;
CWINP   EQU     0       ;PEDESTRIAN PUSHBUTTON PORT
;
;
; JUMP TO LABEL '#IFTRUE' IF ANY PUSHBUTTON PUSHED.
; READING THE PORT CLEARS ALL INPUT.
PUSH?   MACRO   #IFTRUE
```

```
        IN      A,(CWINP)         ;READ PUSHBUTTONS
        AND     (1.SHL.CWCNT)-1 ;BUILD MASK
        JP      NZ,#IFTRUE        ;IF ANY SET, EXIT VIA LABEL
; CONTINUE ON FALSE CONDITION
        MEND
**************************************************
; END OF MACRO LIBRARY
**************************************************
        LIST
```

```
FIGURE 4-2 TRAFFIC INTERSECTION   MOSTEK MACRO-80 ASSEMBLER  V2.0 PAGE
LOC  OBJ.CODE       STMT-NR SOURCE-STMT PASS2 FIG4D2 FIG4D2 FIG4D2 REL

                          1         TITLE FIGURE 4-2 TRAFFIC INTERSECTION
                            ;
                            ; SIMPLE INTERSECTION EXAMPLE WHERE THE TRAFFI
                            ; LIGHTS ARE MERELY SET AND RESET IN THE PROPE
                            ; SEQUENCE.
                            ;
                            ; INCLUDE THE MACRO LIBRARY IN THE ASSEMBLY
                            ;
0000                      9         INCLUDE FIG4D1
                            ;    FIGURE 4-1
                    129  138        LIST
                     10  139        ELIST 0          ;NO LIST EXPANSIONS
                            ;
                            ;
                            ; START OF CONTROL ....
                            ;
0000'                15  144 CYCLE  SETLITE NS,GREEN
0004                 16  148        SETLITE EW,RED
0008                 17  152        TIMER 20         ;DELAY 20 SECONDS
                            ;
                            ; CHANGE LIGHTS
                            ;
0016                 21  167        SETLITE NS,YELLOW
001A                 22  171        TIMER 3          ;DELAY 3 SECONDS
0028                 23  183        SETLITE NS,RED
002C                 24  187        SETLITE EW,GREEN
0030                 25  191        TIMER 15         ;DELAY 15 SECONDS
                            ;
                            ; CHANGE BACK
                            ;
003E                 29  206        SETLITE EW,YELLOW
0042                 30  210        TIMER 3          ;3 SECONDS
0050                 31  222        RETRY CYCLE      ;GO LOOP FOR MORE
0053                 32  225        END
```

```
  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 FIG4D3 FIG4D3 FIG4D3 REL

                      1         TITLE FIGURE 4-3 COMPLEX INTERSECTION
                        ;
       =0004          3 CWCNT   EQU  4              ;4 CROSSWALK SWITCHES
       =0000          4 LULL0   EQU  0              ;NAME FOR TREADLE ZERO
       =0001          5 LULL1   EQU  1              ;NAME FOR TREADLE ONE
                        ;
                        ; INCLUDE MACRO LIBRARY
                        ;
 0                    9         INCLUDE FIG4D1
                        ;    FIGURE 4-1
                129  138        LIST
                 10  139        ELIST 0             ;NO LIST EXPANSIONS
                        ;
                        ; START OF PROGRAM FOR CONTROL ....
                        ;
       =0000'    14  143 CYCLE                      ;ENTER HERE FOR EACH MAJOR CY
                                                    CLE OF THE LIGHTS
 0               15  144        CLOCK? 2,5,NIGHT ;BETWEEN 2 AND 5 AM?
                        ; NOT BETWEEN 2 AND 5 AM, SO PROCESS
                        ; EAST-WEST GETS MAJOR TRAFFIC FLOW
 B               18  158        SETLITE NS,RED
 F               19  162        SETLITE EW,GREEN
                        ;
       =0013'    21  167 SAMPLE                     ; SAMPLE THE BUTTONS AND TREA
                                                    DLES
 13              22  168        PUSH? SWITCH        ;ANYONE THERE?
 1A              23  174        TREAD? LULL0,SWITCH ;ANY CARS?
 27              24  183        TREAD? LULL1,SWITCH
 34              25  192        CLOCK? 2,,NIGHT  ;PAST 2AM?
 3B              26  202        RETRY SAMPLE        ;NO, LOOP FOR ANOTHER SAMPLE
                        ;
                        ;
       =003E'    29  207 SWITCH                     ;SOMEONE IS WAITING, CHANGE T
                                                    HE LIGHTS
 3E              30  208        SETLITE EW,YELLOW ;SLOW THEM DOWN
 42              31  212        TIMER 3             ;3 SECONDS
 50              32  224        SETLITE EW,RED      ; STOP THEM
 54              33  228        SETLITE NS,GREEN ;LET NORHT-SOUTH GO
 58              34  232        TIMER 23            ;FOR A WHILE
                        ;
       =0066'    36  245 DONE?                      ; IS ALL THE TRAFFIC THROUGH
                                                    ON NORHT-SOUTH?
 66              37  246        TREAD? LULL0,NOTDONE ;CHECK THE TREADLES
 73              38  255        TREAD? LULL1,NOTDONE
                        ; NEITHER TREADLE IS SET, CYCLE FOR ANOTHER LOOP
 80              40  265        RETRY CYCLE
                        ;
                        ;
       =0083'    43  270 NOTDONE                    ;WAIT 5 SECONDS AND TRY AGAI
 83              44  271        TIMER 5
 91              45  283        RETRY DONE?
                        ;
                        ;
       =0094'    48  288 NIGHT                      ;THIS IS NIGHTTIME, FLASH TH
                                                     LIGHTS
 94              49  289        SETLITE EW,OFF      ;TURN OFF
 98              50  293        SETLITE NS,OFF
```

```
009C                  51  297          TIMER 1             ;WAIT WITH OFF
00AA                  52  309          SETLITE EW,YELLOW ;CAUTION ON
00AE                  53  313          SETLITE NS,RED    ;STOP ON
00B2                  54  317          TIMER 1           ;DELAY
00C0                  55  329          RETRY CYCLE       ;GO AROUND AGAIN
00C3                  56  332          END
```

When the lights change to allow North-South flow, all traffic must be allowed to clear the lanes before a change to East-West can be done again. During early morning hours, the lights merely flash yellow in the East-West direction and red the in North-South direction. In the program shown, each major cycle of the traffic light enters as 'CYCLE' where the time of day is tested. If between 2 and 5AM, then control transfers to 'NIGHT' where the lights are merely flashed. Otherwise, the treadles and pedestrian pushbuttons are sampled until a change is required.

4-14. Macro-based languages of this sort can easily incorporate debugging facilities. In this example, a debugging flag (DEBUG) is set for use in the macro shown in Figure 4-4. The debug flag, when set, allows trace information to be output to the console device rather than code to activate the system. Here calls to MOSTEK's FLP-80DOS are shown to produce the trace output shown in Figure 4-5. After debugging is complete, the DEBUG flag can be reset and Assembly done once more for the final system. This idea can be extended to the other macros in the system to simulate operation of the system.

4-15. In this application of macros, a simple to use 'language' was developed for a specific use to ease programming and debugging of a final system employing the microprocessor.

4-16. MACHINE EMULATION.

4-17. A second application of macros is found in 'emulation' of a machine operation code set which is different from the given microprocessor. In this case, after the machine to be emulated is defined, a set of macros are written to emulate the opcodes. Each macro assumes the name of an opcode, and the macro body contains instructions which perform the same function as the opcode on the emulated machine. After the macros are defined, then a program can be written using these opcodes which expand to the given microprocessor instructions but which emulate the operation of the new machine.

4-18. In this example, a new machine is defined as an analog sensing and control element in a larger electronic environment. The new machine is based around a 16-bit word length and it is a 'stack machine', in which data can be loaded to the top of a 'stack' of data elements, automatically pushing existing elements deeper onto the stack. Arithmetic operations are performed on the topmost stack elements, automatically absorbing the stacked operands as the arithmetic is performed. The opcodes of the new machine are defined as follows:

SIZ n -reserves n 16-bit elements for the maximum size of the operand stack. This operation code must be provided at the beginning of the program.

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 FIG4D4 FIG4D4 FIG4D4 REL

                          ;    FIGURE 4-4 DEBUGGING MACRO
                          ;
                          ; THIS MACRO DEFINITION IS THE SAME AS FIGURE 4-
                          ; EXCEPT THAT A DEBUGGING FACILITY HAS BEEN ADDEI
                          ;
                          ; DEFINITIONS FOR DEBUG PROCESSING
        =FFFF           7 TRUE    EQU  OFFFFH       ;TRUE VALUE
        =0000           8 FALSE   EQU  .NOT.TRUE    ;FALSE VALUE
        =0000           9 DEBUG   DEFL FALSE        ;INITIALLY FALSE
                          ;
                          ;
                          ; INPUT/OUTPUT PORTS FOR TRAFFIC LIGHT CONTROL
                          ;
        =0000          14 LIGHT   EQU  0            ;TRAFFIC LIGHT
        =0003          15 CLOCK   EQU  3            ;24 HOUR CLOCK (0-23)
                          ;
                          ; BIT POSITIONS FOR TRAFFIC LIGHT CONTROL
        =0004          18 BITSNS  EQU  4            ;NORHT-SOUTH
        =0000          19 BITSEW  EQU  0            ;EAST-WEST
                          ;
                          ; CONSTANT VALUES FOR LIGHT CONTROL
        =0000          22 OFF     EQU  0
        =0001          23 RED     EQU  1
        =0002          24 YELLOW  EQU  2
        =0003          25 GREEN   EQU  3
                          ;
                          ;
                          ; SET LIGHT MACRO WITH DEBUGGING INFO
                          ;
                       30 SETLITE MACRO #DIR,#COLOR
                    1  31         MIF  .NOT.DEBUG THEN L1
                          ; DEBUGGING, PRINT INFO ON CONSOLE
                    3  33         LD   HL,MS%NEXP
                    4  34         LD   E,1
                    5  35         GLOBAL PTXT
                    6  36         CALL PTXT
                    7  37         JR   L%NEXP
                    8  38 MS%NEXP DEFM '#DIR CHANGING TO #COLOR',ODH,OAH,3
                    9  39 L%NEXP  MEXIT
                   10  40 L1      MNOP
                   11  41         LD   A,#COLOR.SHL.BITS#DIR   ;READY COLOR
                   12  42         OUT  (LIGHT),A       ;OUTPUT IT
                   13  43         MEND
```

FIGURE 4-5.
SAMPLE OUTPUT

```
NS CHANGING TO GREEN
EW CHANGING TO RED
NS CHANGING TO YELLOW
NS CHANGING TO RED
EW CHANGING TO GREEN
EW CHANGING TO YELLOW
NS CHANGING TO GREEN
EW CHANGING TO RED
```

RDM i -reads the analog signal from input port i (0, 1, 2, or 3) to the top of the stack, automatically pushing the stack down.

WRM i -writes the digital value from the top of the stack to the D-A output port given by i (0, 1, 2, or 3). The value at the top of the stack is removed.

DUP -duplicates the item at the top of the stack.

SUM -the top two elements of the stack are added, both operands are removed from the stack, and the resulting sum is placed on the top of the stack.

LSR n -performs a logical shift of the topmost stack element to the right by n bits (1, 2, ..., 15), replacing the original operand by the shifted result. Note that LSR n performs a division of the topmost stack value by the divisor 2 to the nth power.

JMP a -branches directly to the program address given by the label a.

4-19. Each of these opcodes can be emulated by using macros to define them in terms of the given microprocessor instructions. The complete definition of the macros is shown in Figure 4-6.

4-20. The SIZ macro sets the program origin (hence, it must be the first opcode used in a program), and the stack area is reserved. Double bytes of storage are reserved since a 16-bit word size is assumed.

4-21. In the following macros, the stack top is assumed to be in the HL register pair. Each operation which pushes the stack of the emulated machine causes the element in the HL register pair to be pushed onto the memory area designated as STACK.

4-22. The DUP opcode simply pushes the HL register pair to the memory stack. In the case of the SUM opcode, it is assumed that the programmer has loaded two values to the stack to be summed. Thus, the HL register pair contains the most recently loaded value, and the memory stack contains the next-to-most recently stacked value. The POP DE operation loads the second operand into the DE register pair, ready for adding to HL. The result goes into the HL register pair because the top of the stack of the emulated machine is located in the HL register pair.

4-23. The LSR macro generates a loop which shifts the HL register pair right the specified number of times.

4-24. The RDM and WRM opcodes are implemented by 'memory mapped' I/O

```
;               FIGURE 4-6
        NLIST
;
**************************************************
;       STACK MACHINE OPCODE MACRO LIBRARY
**************************************************
;
; SET THE PROGRAM ORIGIN AND CREATE A STACK
;
SIZ     MACRO   #SIZE
        ORG     0
        LD      SP,STACK        ;SET STACK POINTER
        JP      STACK   ;GET PAST STACK
        DEFS    2*#SIZE ;SET UP STACK AREA
STACK   MEND
;
;
; DUPLICATE TOP OF STACK
;
DUP     MACRO
        PUSH    HL
        MEND
;
;
; ADD THE TOP TWO STACK ELEMENTS
;
SUM     MACRO
        POP     DE      ;TOP OF STACK TO DE
        ADD     HL,DE   ;ADD AND PUT INTO HL
        MEND
;
;
; LOGICAL SHIFT RIGHT BY #LEN
;
LSR     MACRO   #LEN
        LD      B,#LEN  ;COUNT OF SHIFTS
L%NEXP  XOR     A       ;RESET CARRY
        RR      H       ;ROTATE H INTO CARRY
        RR      L       ;ROTATE L WITH CARRY
        DJNZ    L%NEXP  ;LOOP FOR TOTAL COUNT
        MEND
;
;
; JUMP TO A LABEL
;
JMP     MACRO   #A
        JP      #A
        MEND
;
;
;DEFINITION OF ADC INPUTS AND DAC OUTPUTS VIA
; MEMORY MAPPED I/O
;
ADC0    EQU     1080H   ;A-D CONVERTER 0
ADC1    EQU     1082H   ;A-D CONVERTER 1
ADC2    EQU     1084H   ;A-D CONVERTER 2
ADC3    EQU     1086H   ;A-D CONVERTER 3
;
DAC0    EQU     1090H   ;D-A CONVERTER 0
DAC1    EQU     1092H   ;D-A CONVERTER 1
```

```
DAC2    EQU     1094H   ;D-A CONVERTOR 2
DAC3    EQU     1096H   ;D-A CONVERTER 3
;
;
; READ A-D CONVERTER NUMBER #NUM
;
RDM     MACRO   #NUM
        PUSH    HL      ;CLEAR THE STACK
        LD      HL,(ADC#NUM)    ;READ VIA MEMORY MAP
        MEND
;
;
; WRITE D-A CONVERTER NUMBER #NUM
;
WRM     MACRO   #NUM
        LD      (DAC#NUM),HL    ;WRITE VIA MEMORY MAP
        POP     HL      ;RESTORE STACK
        MEND
**************************************************
; END OF MACRO LIBRARY
**************************************************
        LIST
```

operations. That is, locations 1080H through 1087H are intercepted external to the given microprocessor and treated as external read operations. Thus a load of HL from 1080H and 1081H is treated as a read from A-D device 0, rather than from RAM. This applies also to devices ADC1, ADC2, and ADC3. Similarly, the D-A output values are written to locations 1090H through 1097H for devices DAC0 through DAC3.

4-25. Figure 4-7 shows a sample program written for the emulated machine. In this case, the machine is connected to four temperature sensors via ADC0 through ADC3. The program continuously reads the four input values and computes their average value by summing and dividing by four. The average value is sent to DAC0 where it is used to set environmental controls.

4-26. The program begins by reserving 20 elements for the stack, which are more than enough. The program then cycles through 'LOOP', where the values are read and processed. The four RDM operations read the four temperature sensors, placing their data values on the top of the stack. The three SUM operations which follow perform pairwise addition of the temperature values, producing a single sum at the top of the stack. To obtain the average value, the LSR opcode is applied to perform a division by 4. The resulting average is then sent to DAC0 using the WRM opcode. Control then transfers back to 'LOOP' and the operation is repeated.

4-27. As in the previous example, debugging statements could be added to the macro to perform an emulation without the ADC and DAC hardware. These statements could take the form of additional macros used to print out values as the program is executed.

4-28. DEVELOPMENT OF CROSS-ASSEMBLERS.

4-29. Macros can be written to assemble another microprocessor's instruction set. The resultant object code may be used directly or may have to be translated to a different format by a utility program. Each opcode of the new machine is used as a macro name. Parameters are used if the opcode uses operands. The macro can decode the operands to produce the correct machine code. If any of the new machine's opcodes are the same as the Z80 opcodes, then the 'R' option must be used when the Assembler is executed.

4-30. Consider a portion of the 3870 microcomputer instruction set given in Figure 4-8. The corresponding macros to produce the correct object code are shown. Note that in this implementation, programs formed by the resultant cross-assembler must be non-linkable. This restriction exists because of the way in which the FLP-80DOS Linker processes external reference addresses. That is, such addresses are produced by the MACRO-80 Assembler with least significant byte first,

```
LOC   OBJ.CODE     STMT-NR SOURCE-STMT PASS2 FIG4D7 FIG4D7 FIG4D7 REL

                         1        TITLE FIGURE 4-7 A-D AVERAGING PROGRAM
                           ;
                           ; AVERAGE THE VALUES WHICH ARE READ FROM A-D CONVI
                                                          S
                           ; 0 THROUGH 3, WRITE THE RESULTING VALUE TO THE
                           ; D-A CONVERTER 0, THEN LOOP FOR MORE.
                           ;
                           ; INCLUDE MACRO LIBRARY
                           ;
0000                     9        INCLUDE FIG4D6
                           ;    FIGURE 4-6
                  82    91        LIST
                  10    92        ELIST 0          ;NO LIST EXPANSIONS
                           ;
0000              12    94        SIZ 20           ;RESERVE 20 LEVELS FOR ST
002E'             13   100 LOOP   RDM 0            ;READ ADC0
0032              14   104        RDM 1            ;READ ADC1
0036              15   108        RDM 2            ;READ ADC2
003A              16   112        RDM 3            ;READ ADC3
                           ;
                           ; ALL FOUR VALUES ARE STACKED, SUM THEM
                           ;
003E              20   119        SUM              ;ADC3+ADC2
0040              21   123        SUM              ;(ADC3+ADC2)+ADC1
0042              22   127        SUM              ;((ADC3+ADC2)+ADC1)+ADC0
                           ;
                           ; SUM IS AT TOP OF STACK, DIVIDE BY 4
0044              25   133        LSR 2            ;SHIFT RIGHT BY 2 = DIVIDI
                                                    4
004D              26   140        WRM 0            ;WRITE RESULT TO DAC0
0051              27   144        JMP LOOP         ;REPEAT THE PROCESS
0054              28   147        END
```

FIGURE 4-8

3870 CROSS ASSEMBLER MACROS

THESE MACROS ARE EXAMPLES WHICH COULD BE
EXTENDED TO PRODUCE A 3870 CROSS ASSEMBLER
RUNNING UNDER MACRO-80.

REGISTER DEFINITION

```
;       EQU     0CH
:       EQU     0DH
)       EQU     0EH
;
;
DCI     MACRO   #ADDR   ;LOAD DATA COUNTER
        DEFB    2AH,(#ADDR.SHR.8).AND.0FFH,#ADDR.AND.0FFH
        MEND
;
AS      MACRO   #R      ;ADD TO SCRATCHPAD
        MLOCAL  LERR
        MIF     #R.UGT.0EH THEN LERR
        DEFB    0C0H.OR.#R
        MEXIT
LERR    MERROR  *** OUT OF RANGE ***
        MEND
;
SL      MACRO   #N      ;SHIFT LEFT
        MLOCAL  L1,L2,L3
        MIF #N=4 THEN L1 ELSE L2        ;CHECK RANGE OF OPERAND
L1      MNOP
        DEFB    15H
        MEXIT
L2      MIF #N=1 THEN L3
        MERROR  *** OUT OF RANGE ***
L3      MNOP
        DEFB    13H
        MEND
;
LI      MACRO   #OP     ;LOAD IMMEDIATE
        DEFB    20H
        DEFB    #OP.AND.0FFH
        MEND
;
LISL    MACRO   #A
        MLOCAL  LERR
        MIF     #A.UGT.7 THEN LERR
        DEFB    68H.OR.#A
        MEXIT
LERR    MERROR  *** OUT OF RANGE ***
        MEND
;
BR7     MACRO   #AA
        MLOCAL  LERR
        DEFB    8FH
        MIF (#AA-$>128).OR.(#AA-$<0) THEN LERR ;CHECK RANGE
        DEFB    #AA-$
        MEXIT
LERR    MERROR  *** OUT OF RANGE ***
        MEND
```

```
;
BF      MACRO   #T,#AA
        MLOCAL  LERR
        MIF #T.UGT.OFH THEN LERR        ;CHECK RANGE
        DEFB    90H.OR.#T
A%NEXP  EQU     #AA-$
        MIF (A%NEXP>128).OR.A%NEXP<0) THEN LERR ;CHECK RANGE
        DEFB    A%NEXP
        MEXIT
LERR    MERROR  *** OUT OF RANGE ***
        MEND
```

while the 3870 requires most significant byte first. Note also that cross-assemblers developed under MACRO-80 must follow the Z80 conventions for forming constants and expressions.

4-31. PROGRAM CONTROL STRUCTURES.

4-32. Macros can be used to provide program-control statements which resemble those found in many high-level languages. Figure 4-9 shows a set of macros which define a simple language for performing 16-bit integer operations. The following paragraphs describe each type of statement allowed in a program written around these macros.

4-33. LET var1 = var2 or LET var1 = var2 <op> var3

The LET statement allows a variable to be set equal to another variable or to the result of an operation performed on two variables. The allowed operations are addition (<op> = +), subtraction (-), multiplication (*), and division (/). The blanks between the operands are required.

4-34. TEST var1 <relop> var2 THEN label1 ELSE label2

The TEST statement allows two variables to be compared as being equal (=), less than (<) or greater than (>). If the result is true, then a branch is made to label1. Otherwise a branch is made to label2. The ELSE-clause is optional. If it is not present and a false condition is encountered, then the next statement in sequence will be processed.

4-35. DCL var1 INIT n

The DCL statement declares variables used in the program. Note that all variables must be declared. The initial value n is optional and defaults to zero.

4-36. DO var1 = var2 TO var3

The DO statement, together with the ENDDO statement, allows writing of loops. The value of var1 is initially set to var2. Each pass through the loop increments var1 until it equals the value of var3. DO loops may be nested, but the program stack must always be balanced between the DO and ENDDO statements.

4-37. ENDDO

This signals the end of a DO loop.

4-38. READ var1,var2,...

This statement reads and converts to binary sequences of two

```
;               FIGURE 4-9
        NLIST
;
**************************************************
;       PROGRAM CONTROL STRUCTURES VIA MACROS
**************************************************
;
; PRINT message
;
**************************************************
PRINT   MACRO   #A
        GLOBAL  PTXT
        LD      E,CHNL+1        ;CHANNEL NBR
        LD      HL,MS%NEXP
        CALL    PTXT
        JR      L%NEXP
MS%NEXP DEFM    '#A',0DH,0AH,3H
L%NEXP
        MEND
;
**************************************************
;
; LET var1 = var2 <op> var3
;
**************************************************
LET     MACRO   #A #B #C #D #E
        MLOCAL  L1,L2,L3,L4,L5,LS,LERR
        MIF '#B'='=' THEN L1 ELSE LERR ;SYNTAX CHECK
L1      MNOP
        LD      HL,(#C) ;GET VAR2
        MIF '#D'='' THEN LS ;IF NO OPERATOR, DO ASSIGNMENT
        LD      DE,(#E) ;GET VAR3
        MIF     '#D'='+' THEN L2        ;CHECK OPERATOR
        MIF     '#D'='-' THEN L3
        MIF     '#D'='*' THEN L4
        MIF     '#D'='/' THEN L5
        MERROR  ***** ILLEGAL OPERATOR *****
        MEXIT
;
L2      MNOP
        ADD     HL,DE
        MGOTO   LS
L3      MNOP
        OR      A
        SBC     HL,DE
        MGOTO   LS
L4      MNOP            ;MULTIPLY BY SEVERAL ADDITIONS
        LD      A,D     ;CHECK FOR MULT BY ZERO
        OR      E
        JR      NZ,I%NEXP
        LD      HL,0    ;IF SO, ZERO RESULT
        JP      K%NEXP
I%NEXP  DEC     DE      ;CHECK FOR MULT BY ONE
        LD      A,D
        OR      E
        JR      Z,K%NEXP        ;YES, JUST PUT IN VALUE
        LD      BC,(#C) ;GET VAR2
L%NEXP  ADD     HL,BC
        DEC     DE
        LD      A,D     ;CHECK FOR END
```

```
        OR      E
        JR      NZ,L%NEXP
 %NEXP
        MGOTO   LS

 ERR    MERROR  ***** BAD SYNTAX *****
        MEXIT

 5      MNOP
        LD      A,D     ;CHECK FOR DIVIDE BY ZERO
        OR      E
        JR      NZ,C%NEXP
        PRINT   '*** OVERFLOW ERROR'
        JR      Z%NEXP
 %NEXP  LD      BC,0    ;RESULT
 %NEXP  OR      A       ;RESET CARRY
        SBC     HL,DE   ;SUBTRACT UNTIL DONE
        INC     BC
        JR      NC,D%NEXP       ;LOOP UNTIL NEGATIVE
        DEC     BC      ;CORRECT THE RESULT
        LD      L,C     ;PUT INTO HL
        LD      H,B
.S      MNOP
%NEXP   LD      (#A),HL ;SAVE IN VAR1
        MEND
;
;************************************************
;
; TEST var1 <op> var2 THEN label1 [ ELSE label2 ]
;
;************************************************
TEST    MACRO   #A #B #C #D #E #F #G
        MLOCAL  L1,L2,L3,L4,L5,L6,L7,L8,LERR,LCONT
        MIF     '#D'='THEN' THEN L1 ELSE LERR   ;SYNTAX CHECK
L1      MNOP
        LD      HL,(#A) ;GET VAR1
        LD      DE,(#C) ;GET VAR2
        OR      A
        SBC     HL,DE   ;SUBTRACT FOR COMPARE
        MIF     '#B'='=' THEN L2 ELSE L3        ;CHECK OPERATOR
L2      JP      Z,#E    ;IF EQUAL (TRUE), DO JUMP
        MGOTO   LCONT
L3      MIF     '#B'='<' THEN L4 ELSE L5
L4      MNOP
        JP      C,#E    ;IF LESS THAN, JUMP
        MGOTO   LCONT
L5      MIF     '#B'='>' THEN L6 ELSE LERR
L6      MNOP
        JR      Z,L%NEXP        ;IF EQUAL TO THEN FALSE
        JP      NC,#E   ;IF GREATER THAN, JUMP
        MGOTO   LCONT
;
LERR    MERROR  ***** BAD SYNTAX *****
        MEXIT
;
LCONT   MNOP
L%NEXP
        MIF     '#F'='ELSE' THEN L7 ELSE L8     ;CHECK FOR IF CLAUSE
L7      MNOP
        JP      #G      ;JUMP TO FALSE LABEL
```

```
        MEXIT
L8      MNOP
        MEND
;
**************************************************
;
; DCL var INIT n
;
**************************************************
DCL     MACRO   #A #B #C
        MLOCAL  L1,L2,L3
        MIF     '#B'='INIT' THEN L1 ELSE L2
L1      MIF     '#C'='' THEN L2
#A      DEFW    #C       ;DECLARE VARIABLE
        MEXIT
L2      MNOP
#A      DEFW    0        ;DEFAULT TO ZERO
        MEND
;
**************************************************
;
; DO var1 = var2 TO var3
;
**************************************************
DO      MACRO   #A #B #C #D #E
        MLOCAL  L1,L2,LERR
        MIF     '#B'='=' THEN L1 ELSE LERR      ;SYNTAX CHECK
L1      MIF     '#D'='TO' THEN L2
LERR    MERROR  ***** BAD SYNTAX *****
        MEXIT
;
L2      MNOP
        LD      HL,(#C) ;GET VAR2
        LD      DE,(#E) ;GET VAR3
        LD      IX,L%NEXP       ;GET LOOP BACK LABEL
L%NEXP  LD      (#A),HL         ;SET VAR1
        PUSH    HL              ;PUSH VALUES ONTO STACK
        PUSH    DE
        PUSH    IX
        MEND
;
**************************************************
;
; ENDDO
;
**************************************************
ENDDO   MACRO
        POP     IX       ;LOOP ADDRESS
        POP     DE       ;FINAL VALUE
        POP     HL       ;CURRENT VALUE
        INC     HL       ;INCREMENT VAR1
        PUSH    HL
        OR      A        ;CHECK IT
        SBC     HL,DE
        POP     HL
        JR      Z,KK%NEXP       ;LAST TIME THRU
        JR      NC,L%NEXP       ;IF DONE, SKIP OUT
KK%NEXP JP      (IX)     ;ELSE LOOP
L%NEXP
        MEND
```

```
 *************************************************

  READ var1,var2,...

 *************************************************
READ    MACRO   #A
        MLOCAL  L1,L2
; #A FIRST TIME USAGE OF PARAMETER
        GLOBAL  ECHO,ASBIN
        LD      E,CHNL
L1      MNOP
        CALL    ECHO    ;READ A CHARACTER
        LD      A,D     ;PREPARE TO CONVERT
        CALL    ASBIN   ;CONVERT
        AND     0FH
        RLCA
        RLCA
        RLCA
        RLCA
        PUSH    AF
        CALL    ECHO    ;GET NEXT ONE
        LD      A,D
        CALL    ASBIN
        AND     0FH
        LD      L,A     ;SAVE IT
        POP     AF
        OR      L
        LD      L,A
        LD      H,0
        LD      (#PRM),HL       ;SAVE RESULT
LN%NEXP CALL    ECHO    ;GET NEXT INPUT CHAR
        LD      A,D     ;CHECK CHARACTER
        CP      0DH     ;CARRIAGE RETURN?
        JP      Z,P%NEXP        ;YES, SKIP OUT
        CP      ','     ;COMMA?
        JR      NZ,LN%NEXP      ;NO, LOOP FOR ANOTHER
        MNEXT   1 THEN L1 ELSE L2       ;CHECK FOR MORE ARGS
L2      MNOP
P%NEXP
        CALL    CRLF
        MEND
;
*************************************************
;
; WRITE var1,var2,...
;
*************************************************
WRITE   MACRO   #A,#B
; #A FIRST TIME USAGE OF PARAMETER
        MLOCAL  L1
        GLOBAL  PTXT,CRLF,PADDO
        LD      E,CHNL+1        ;OUTPUT CHANNEL
L1      MNOP
        LD      HL,MS#PRM       ;OUTPUT MESSAGE
        CALL    PTXT
        LD      HL,(#PRM)
        CALL    PADDO   ;WRITE OUT IN HEX
        JR      L#PRM
MS#PRM  DEFM    '#PRM = '
```

```
        DEFB    3
L#PRM
        MNEXT   1 THEN L1
        CALL    CRLF
        MEND
;
**************************************************
;
; GOTO label
;
**************************************************
GOTO    MACRO   #A
        JP      #A
        MEND
;
**************************************************
;
; EXIT
;
**************************************************
EXIT    MACRO
        GLOBAL  JTASK
        LD      A,1
        JP      JTASK
        MEND
**************************************************
; END OF MACRO LIBRARY
**************************************************
        LIST
```

hexadecimal characters, placing them into the variables var1, var2, etc.

4-39. WRITE var1,var2,...

This statement writes each variable in the list in the form 'name = value', where name is the name of the variable and value is its value in four hexadecimal digits.

4-40. PRINT 'message'

This macro prints a message of any length on the console.

4-41. GOTO label

This macro transfers control to the specified label.

4-42. EXIT

This macro transfers control back to the FLP-80DOS Monitor.

4-43. Figure 4-10 shows two simple programs which demonstrate use of these macros. The first program calculates n numbers in a Fibonacci series where n is a number input from the console keyboard. The second program generates n x n combinations of addition, subtraction, multiplication, and division, where n is read from the console keyboard. Figure 4-11 shows sample output from the programs.

4-44. OPERATING SYSTEM INTERFACE.

4-45. The fifth area where macros are useful is in providing systematic and simplified mechanisms for access to operating system functions. These macros can allow easy use of the operating system's I/O facilities, service routines, and system support routines.

4-47. In this example, a set of macros are shown which provide access to FLP-80DOS I/O facilities. Use of these macros can eliminate a large portion of the drudgery of assembly language programming. Furthermore, the macros reduce programming errors and provide for some checking of parameters associated with the operating system calls. It is assumed in this discussion that the user is acquainted with Section 9 of the FLP-80DOS manual (IOCS).

4-47. Figure 4-12 shows a file which has definitions of each IOCS related parameter. This file is included in programs which use IOCS to provide a set of standard symbols for use in the macros and in the program itself. (The file is called IODEF).

4-48. The set of macros shown in Figure 4-13 allows a simplified

```
FIGURE 4-10.                            MOSTEK MACRO-80 ASSEMBLER  V2.0 PAGE
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 FIG410 FIG410 FIG410 REL

                      1          TITLE FIGURE 4-10.
                         ;
                         ; SAMPLE USAGE OF CONTROL STRUCTURES
                         ;
                         ; INCLUDE MACRO DEFINITIONS
                         ;
0000                  7          INCLUDE FIG4D9
                         ;      FIGURE 4-9
                 269  276          LIST
                         ;
        =0000      9  278 CHNL    EQU  0
                         ;
                         ; PROGRAM 1 ... GENERATE UP TO N FIBONACCI NUMBI
                         ; WHERE N IS READ FROM THE CONSOLE KEYBOARD
                         ;
0000              14  283          PRINT 'ENTER 2 HEX DIGITS'
                   1  284          GLOBAL PTXT
0000 1E01          2  285          LD   E,CHNL+1     ;CHANNEL NBR
0002 210A00'       3  286          LD   HL,MS0001
0005 CDFFFF        4  287          CALL PTXT
0008 1815          5  288          JR   L0001
000A'454E5445      6  289 MS0001  DEFM 'ENTER 2 HEX DIGITS',0DH,0AH,3H
     52203220
     48455820
     44494749
     54530D0A
     03
        =001F'     7  290 L0001
                   8  291          MEND
001F              15  292          READ N
                   1  293          MLOCAL L1,L2
                         ; N FIRST TIME USAGE OF PARAMETER
                   3  295          GLOBAL ECHO,ASBIN
001F 1E00          4  296          LD   E,CHNL
                   5  297 L1       MNOP
0021 CDFFFF        6  298          CALL ECHO          ;READ A CHARACTER
0024 7A            7  299          LD   A,D           ;PREPARE TO CONVERT
0025 CDFFFF        8  300          CALL ASBIN         ;CONVERT
0028 E60F          9  301          AND  0FH
002A 07           10  302          RLCA
002B 07           11  303          RLCA
002C 07           12  304          RLCA
002D 07           13  305          RLCA
002E F5           14  306          PUSH AF
002F CD2200'      15  307          CALL ECHO          ;GET NEXT ONE
0032 7A           16  308          LD   A,D
0033 CD2600'      17  309          CALL ASBIN
0036 E60F         18  310          AND  0FH
0038 6F           19  311          LD   L,A           ;SAVE IT
0039 F1           20  312          POP  AF
003A B5           21  313          OR   L
003B 6F           22  314          LD   L,A
003C 2600         23  315          LD   H,0
003E 22EB00'      24  316          LD   (N),HL        ;SAVE RESULT
0041'CD3000'      25  317 LN0002  CALL ECHO          ;GET NEXT INPUT CHAR
0044 7A           26  318          LD   A,D           ;CHECK CHARACTER
0045 FE0D         27  319          CP   0DH           ;CARRIAGE RETURN?
```

```
 OBJ.CODE      STMT-NR SOURCE-STMT PASS2 FIG410 FIG410 FIG410 REL

 CA4E00'       28  320          JP   Z,P0002      ;YES, SKIP OUT
 FE2C          29  321          CP   ','          ;COMMA?
 20F3          30  322          JR   NZ,LN0002    ;NO, LOOP FOR ANOTHER
               31  323          MNEXT 1 THEN L1 ELSE L2       ;CHECK FOR MORE AR
               32  324 L2       MNOP
    =004E'     33  325 P0002
 CDFFFF        34  326          CALL CRLF
               35  327          MEND
               16  328          LET COUNT = ONE
                1  329          MLOCAL L1,L2,L3,L4,L5,LS,LERR
    =FFFF       2  330          MIF  '='='=' THEN L1 ELSE LERR ;SYNTAX CHECK
                3  331 L1       MNOP
 2ADF00'        4  332          LD   HL,(ONE)     ;GET VAR2
    =FFFF       5  333          MIF  ''='' THEN LS ;IF NO OPERATOR, DO ASSIGNM
                                                  ENT
               57  334 LS       MNOP
'22E900'       58  335 Z0003    LD   (COUNT),HL   ;SAVE IN VAR1
               59  336          MEND
7              17  337          LET A = ONE
                1  338          MLOCAL L1,L2,L3,L4,L5,LS,LERR
    =FFFF       2  339          MIF  '='='=' THEN L1 ELSE LERR ;SYNTAX CHECK
                3  340 L1       MNOP
7 2ADF00'       4  341          LD   HL,(ONE)     ;GET VAR2
    =FFFF       5  342          MIF  ''='' THEN LS ;IF NO OPERATOR, DO ASSIGNM
                                                  ENT
               57  343 LS       MNOP
A'22E300'      58  344 Z0004    LD   (A),HL       ;SAVE IN VAR1
               59  345          MEND
D              18  346          LET B = TWO
                1  347          MLOCAL L1,L2,L3,L4,L5,LS,LERR
    =FFFF       2  348          MIF  '='='=' THEN L1 ELSE LERR ;SYNTAX CHECK
                3  349 L1       MNOP
D 2AE100'       4  350          LD   HL,(TWO)     ;GET VAR2
    =FFFF       5  351          MIF  ''='' THEN LS ;IF NO OPERATOR, DO ASSIGNM
                                                  ENT
               57  352 LS       MNOP
0'22E500'      58  353 Z0005    LD   (B),HL       ;SAVE IN VAR1
               59  354          MEND
3              19  355          WRITE A,B
                       ; A FIRST TIME USAGE OF PARAMETER
                2  357          MLOCAL L1
                3  358          GLOBAL PTXT,CRLF,PADDO
3 1E01          4  359          LD   E,CHNL+1     ;OUTPUT CHANNEL
                5  360 L1       MNOP
55 217300'      6  361          LD   HL,MSA       ;OUTPUT MESSAGE
58 CD0600'      7  362          CALL PTXT
5B 2AE300'      8  363          LD   HL,(A)
6E CDFFFF       9  364          CALL PADDO        ;WRITE OUT IN HEX
71 1805        10  365          JR   LA
73'41203D20    11  366 MSA      DEFM 'A = '
77 03          12  367          DEFB 3
    =0078'     13  368 LA
               14  369          MNEXT 1 THEN L1
                5  370 L1       MNOP
78 218600'      6  371          LD   HL,MSB       ;OUTPUT MESSAGE
7B CD6900'      7  372          CALL PTXT
7E 2AE500'      8  373          LD   HL,(B)
```

```
LOC  OBJ.CODE         STMT-NR SOURCE-STMT PASS2 FIG410 FIG410 FIG410 REL

0081 CD6F00'           9  374            CALL PADDO        ;WRITE OUT IN HEX
0084 1805             10  375            JR   LB
0086'42203D20         11  376 MSB        DEFM 'B = '
008A 03               12  377            DEFB 3
        =008B'        13  378 LB
                      14  379            MNEXT 1 THEN L1
008B CD4F00'          15  380            CALL CRLF
                      16  381            MEND
                              ;
008E'                 21  383 LAB1       LET C = A + B
                       1  384            MLOCAL L1,L2,L3,L4,L5,LS,LERR
        =FFFF          2  385            MIF  '='='=' THEN L1 ELSE LERR ;SYNTAX CH
                       3  386 L1         MNOP
008E 2AE300'           4  387            LD   HL,(A)       ;GET VAR2
        =0000          5  388            MIF  '+'='' THEN LS ;IF NO OPERATOR, DO A
                                                           MENT
0091 ED5BE500'         6  389            LD   DE,(B)       ;GET VAR3
        =FFFF          7  390            MIF  '+'='+' THEN L2 ;CHECK OPERATOR
                      14  391 L2         MNOP
0095 19               15  392            ADD  HL,DE
                      16  393            MGOTO LS
                      57  394 LS         MNOP
0096'22E700'          58  395 Z0007      LD   (C),HL       ;SAVE IN VAR1
                      59  396            MEND
0099                  22  397            TEST COUNT > N THEN DONE
                       1  398            MLOCAL L1,L2,L3,L4,L5,L6,L7,L8,LERR,LCONT
        =FFFF          2  399            MIF  'THEN'='THEN' THEN L1 ELSE LERR ;SYN'
                                                           HECK
                       3  400 L1         MNOP
0099 2AE900'           4  401            LD   HL,(COUNT)   ;GET VAR1
009C ED5BEB00'         5  402            LD   DE,(N)       ;GET VAR2
00A0 B7                6  403            OR   A
00A1 ED52              7  404            SBC  HL,DE        ;SUBTRACT FOR COMPARE
        =0000          8  405            MIF  '>'='=' THEN L2 ELSE L3 ;CHECK OPERAI
        =0000         11  406 L3         MIF  '>'='<' THEN L4 ELSE L5
        =FFFF         15  407 L5         MIF  '>'='>' THEN L6 ELSE LERR
                      16  408 L6         MNOP
00A3 2803             17  409            JR   Z,L0008      ;IF EQUAL TO THEN FALSE
00A5 D2DA00'          18  410            JP   NC,DONE      ;IF GREATER THAN, JUMP
                      19  411            MGOTO LCONT
                      24  412 LCONT      MNOP
        =00A8'        25  413 L0008
        =0000         26  414            MIF  ''='ELSE' THEN L7 ELSE L8       ;CHEC
                                                           LAUSE
                      30  415 L8         MNOP
                      31  416            MEND
00A8                  23  417            WRITE C
                              ; C FIRST TIME USAGE OF PARAMETER
                       2  419            MLOCAL L1
                       3  420            GLOBAL PTXT,CRLF,PADDO
00A8 1E01              4  421            LD   E,CHNL+1     ;OUTPUT CHANNEL
                       5  422 L1         MNOP
00AA 21B800'           6  423            LD   HL,MSC       ;OUTPUT MESSAGE
00AD CD7C00'           7  424            CALL PTXT
00B0 2AE700'           8  425            LD   HL,(C)
00B3 CD8200'           9  426            CALL PADDO        ;WRITE OUT IN HEX
00B6 1805             10  427            JR   LC
```

```
  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 FIG410 FIG410 FIG410 REL

3'43203D20      11   428 MSC       DEFM 'C = '
: 03            12   429           DEFB 3
     =00BD'     13   430 LC
                14   431           MNEXT 1 THEN L1
) CD8C00'       15   432           CALL CRLF
                16   433           MEND
 0              24   434           LET COUNT = COUNT + ONE
                 1   435           MLOCAL L1,L2,L3,L4,L5,LS,LERR
     =FFFF       2   436           MIF  '='='=' THEN L1 ELSE LERR ;SYNTAX CHECK
                 3   437 L1        MNOP
 0 2AE900'       4   438           LD   HL,(COUNT)  ;GET VAR2
     =0000       5   439           MIF  '+'='' THEN LS ;IF NO OPERATOR, DO ASSIGN
                                                      MENT
 3 ED5BDF00'     6   440           LD   DE,(ONE)    ;GET VAR3
     =FFFF       7   441           MIF  '+'='+' THEN L2 ;CHECK OPERATOR
                14   442 L2        MNOP
 7 19           15   443           ADD  HL,DE
                16   444           MGOTO LS
                57   445 LS        MNOP
!8'22E900'      58   446 Z0010     LD   (COUNT),HL  ;SAVE IN VAR1
                59   447           MEND
:B              25   448           LET A = B
                 1   449           MLOCAL L1,L2,L3,L4,L5,LS,LERR
     =FFFF       2   450           MIF  '='='=' THEN L1 ELSE LERR ;SYNTAX CHECK
                 3   451 L1        MNOP
:B 2AE500'       4   452           LD   HL,(B)      ;GET VAR2
     =FFFF       5   453           MIF  ''='' THEN LS ;IF NO OPERATOR, DO ASSIGNM
                                                      ENT
                57   454 LS        MNOP
:E'22E300'      58   455 Z0011     LD   (A),HL      ;SAVE IN VAR1
                59   456           MEND
)1              26   457           LET B = C
                 1   458           MLOCAL L1,L2,L3,L4,L5,LS,LERR
     =FFFF       2   459           MIF  '='='=' THEN L1 ELSE LERR ;SYNTAX CHECK
                 3   460 L1        MNOP
D1 2AE700'       4   461           LD   HL,(C)      ;GET VAR2
     =FFFF       5   462           MIF  ''='' THEN LS ;IF NO OPERATOR, DO ASSIGNM
                                                      ENT
                57   463 LS        MNOP
D4'22E500'      58   464 Z0012     LD   (B),HL      ;SAVE IN VAR1
                59   465           MEND
D7              27   466           GOTO LAB1
D7 C38E00'       1   467           JP   LAB1
                 2   468           MEND
                         ;
DA'             29   470 DONE      EXIT
                 1   471           GLOBAL JTASK
DA 3E01          2   472           LD   A,1
DC C3FFFF        3   473           JP   JTASK
                 4   474           MEND
                         ;
)DF             31   476           DCL ONE INIT 1
                 1   477           MLOCAL L1,L2,L3
     =FFFF       2   478           MIF  'INIT'='INIT' THEN L1 ELSE L2
     =0000       3   479 L1        MIF  '1'='' THEN L2
)DF'0100         4   480 ONE       DEFW 1               ;DECLARE VARIABLE
                 5   481           MEXIT
```

```
00E1                  32  482            DCL TWO INIT 2
                       1  483            MLOCAL L1,L2,L3
        =FFFF          2  484            MIF  'INIT'='INIT' THEN L1 ELSE L2
        =0000          3  485 L1         MIF  '2'='' THEN L2
00E1'0200              4  486 TWO        DEFW 2             ;DECLARE VARIABLE
                       5  487            MEXIT
00E3                  33  488            DCL A
                       1  489            MLOCAL L1,L2,L3
        =0000          2  490            MIF  ''='INIT' THEN L1 ELSE L2
                       6  491 L2         MNOP
00E3'0000              7  492 A          DEFW 0             ;DEFAULT TO ZERO
                       8  493            MEND
00E5                  34  494            DCL B
                       1  495            MLOCAL L1,L2,L3
        =0000          2  496            MIF  ''='INIT' THEN L1 ELSE L2
                       6  497 L2         MNOP
00E5'0000              7  498 B          DEFW 0             ;DEFAULT TO ZERO
                       8  499            MEND
00E7                  35  500            DCL C
                       1  501            MLOCAL L1,L2,L3
        =0000          2  502            MIF  ''='INIT' THEN L1 ELSE L2
                       6  503 L2         MNOP
00E7'0000              7  504 C          DEFW 0             ;DEFAULT TO ZERO
                       8  505            MEND
00E9                  36  506            DCL COUNT
                       1  507            MLOCAL L1,L2,L3
        =0000          2  508            MIF  ''='INIT' THEN L1 ELSE L2
                       6  509 L2         MNOP
00E9'0000              7  510 COUNT      DEFW 0             ;DEFAULT TO ZERO
                       8  511            MEND
00EB                  37  512            DCL N
                       1  513            MLOCAL L1,L2,L3
        =0000          2  514            MIF  ''='INIT' THEN L1 ELSE L2
                       6  515 L2         MNOP
00EB'0000              7  516 N          DEFW 0             ;DEFAULT TO ZERO
                       8  517            MEND
```

```
 OBJ.CODE      STMT-NR SOURCE-STMT PASS2 FIG410 FIG410 FIG410 REL

                              ;
                              ; PROGRAM 2 ... GENERATE N BY N CALCULATIONS FOR
                              ; ADDITION, SUBTRACTION, MULTIPLICATION, AND DIVISION
                              ; WHERE N IS INPUT FROM THE CONSOLE KEYBOARD.
                              ;
 )'                 44   524 LOOP    PRINT 'ENTER TWO HEX DIGITS'
                     1   525         GLOBAL PTXT
 ) 1E01              2   526         LD   E,CHNL+1     ;CHANNEL NBR
 ' 21F700'           3   527         LD   HL,MS0022
 2 CDAE00'           4   528         CALL PTXT
 5 1817              5   529         JR   L0022
 7'454E5445          6   530 MS0022  DEFM 'ENTER TWO HEX DIGITS',0DH,0AH,3H
   52205457
   4F204845
   58204449
   47495453
   0D0A03
      =010E'         7   531 L0022
                     8   532         MEND
 E                  45   533         READ N
                     1   534         MLOCAL L1,L2
                              ; N FIRST TIME USAGE OF PARAMETER
                     3   536         GLOBAL ECHO,ASBIN
 E 1E00              4   537         LD   E,CHNL
                     5   538 L1      MNOP
 0 CD4200'           6   539         CALL ECHO         ;READ A CHARACTER
 3 7A                7   540         LD   A,D          ;PREPARE TO CONVERT
 4 CD3400'           8   541         CALL ASBIN        ;CONVERT
 7 E60F              9   542         AND  0FH
 9 07               10   543         RLCA
 A 07               11   544         RLCA
 B 07               12   545         RLCA
 C 07               13   546         RLCA
 D F5               14   547         PUSH AF
 E CD1101'          15   548         CALL ECHO         ;GET NEXT ONE
21 7A               16   549         LD   A,D
22 CD1501'          17   550         CALL ASBIN
25 E60F             18   551         AND  0FH
27 6F               19   552         LD   L,A          ;SAVE IT
28 F1               20   553         POP  AF
29 B5               21   554         OR   L
2A 6F               22   555         LD   L,A
2B 2600             23   556         LD   H,0
2D 22EB00'          24   557         LD   (N),HL       ;SAVE RESULT
30'CD1F01'          25   558 LN0023  CALL ECHO         ;GET NEXT INPUT CHAR
33 7A               26   559         LD   A,D          ;CHECK CHARACTER
34 FE0D             27   560         CP   0DH          ;CARRIAGE RETURN?
36 CA3D01'          28   561         JP   Z,P0023      ;YES, SKIP OUT
39 FE2C             29   562         CP   ','          ;COMMA?
3B 20F3             30   563         JR   NZ,LN0023    ;NO, LOOP FOR ANOTHER
                    31   564         MNEXT 1 THEN L1 ELSE L2      ;CHECK FOR MORE
                    32   565 L2      MNOP
     =013D'         33   566 P0023
3D CDBE00'          34   567         CALL CRLF
                    35   568         MEND
40                  46   569         TEST N = ZERO THEN LOOP
                     1   570         MLOCAL L1,L2,L3,L4,L5,L6,L7,L8,LERR,LCONT
```

```
       =FFFF       2  571          MIF  'THEN'='THEN' THEN L1 ELSE LERR ;SY
                                                  HECK
                   3  572 L1       MNOP
0140 2AEB00'       4  573          LD   HL,(N)        ;GET VAR1
0143 ED5B6502'     5  574          LD   DE,(ZERO)     ;GET VAR2
0147 B7            6  575          OR   A
0148 ED52          7  576          SBC  HL,DE         ;SUBTRACT FOR COMPARE
       =FFFF       8  577          MIF  '='='=' THEN L2 ELSE L3 ;CHECK OPER
014A CAED00'       9  578 L2       JP   Z,LOOP        ;IF EQUAL (TRUE), DO JU
                  10  579          MGOTO LCONT
                  24  580 LCONT    MNOP
       =014D'     25  581 L0024
       =0000      26  582          MIF  ''='ELSE' THEN L7 ELSE L8        ;CH
                                                  LAUSE
                  30  583 L8       MNOP
                  31  584          MEND
014D              47  585          DO I = ONE TO N
                   1  586          MLOCAL L1,L2,LERR
       =FFFF       2  587          MIF  '='='=' THEN L1 ELSE LERR        ;SYN
       =FFFF       3  588 L1       MIF  'TO'='TO' THEN L2
                   7  589 L2       MNOP
014D 2ADF00'       8  590          LD   HL,(ONE)      ;GET VAR2
0150 ED5BEB00'     9  591          LD   DE,(N)        ;GET VAR3
0154 DD215801'    10  592          LD   IX,L0025      ;GET LOOP BACK LABEL
0158'226702'      11  593 L0025    LD   (I),HL        ;SET VAR1
015B E5           12  594          PUSH HL            ;PUSH VALUES ONTO STACK
015C D5           13  595          PUSH DE
015D DDE5         14  596          PUSH IX
                  15  597          MEND
015F              48  598          DO J = ONE TO N
                   1  599          MLOCAL L1,L2,LERR
       =FFFF       2  600          MIF  '='='=' THEN L1 ELSE LERR        ;SYN
       =FFFF       3  601 L1       MIF  'TO'='TO' THEN L2
                   7  602 L2       MNOP
015F 2ADF00'       8  603          LD   HL,(ONE)      ;GET VAR2
0162 ED5BEB00'     9  604          LD   DE,(N)        ;GET VAR3
0166 DD216A01'    10  605          LD   IX,L0026      ;GET LOOP BACK LABEL
016A'226902'      11  606 L0026    LD   (J),HL        ;SET VAR1
016D E5           12  607          PUSH HL            ;PUSH VALUES ONTO STACK
016E D5           13  608          PUSH DE
016F DDE5         14  609          PUSH IX
                  15  610          MEND
0171              49  611          LET ADD = I + J
                   1  612          MLOCAL L1,L2,L3,L4,L5,LS,LERR
       =FFFF       2  613          MIF  '='='=' THEN L1 ELSE LERR ;SYNTAX CHE
                   3  614 L1       MNOP
0171 2A6702'       4  615          LD   HL,(I)        ;GET VAR2
       =0000       5  616          MIF  '+'='' THEN LS ;IF NO OPERATOR, DO AS
                                                  MENT
0174 ED5B6902'     6  617          LD   DE,(J)        ;GET VAR3
       =FFFF       7  618          MIF  '+'='+' THEN L2 ;CHECK OPERATOR
                  14  619 L2       MNOP
0178 19           15  620          ADD  HL,DE
                  16  621          MGOTO LS
                  57  622 LS       MNOP
0179'226B02'      58  623 Z0027    LD   (ADD),HL      ;SAVE IN VAR1
                  59  624          MEND
```

```
 OBJ.CODE       STMT-NR SOURCE-STMT PASS2 FIG410 FIG410 FIG410 REL

:               50  625            LET SUB = I - J
                 1  626            MLOCAL L1,L2,L3,L4,L5,LS,LERR
    =FFFF        2  627            MIF  '='='=' THEN L1 ELSE LERR ;SYNTAX CHECK
                 3  628 L1         MNOP
: 2A6702'        4  629            LD   HL,(I)        ;GET VAR2
    =0000        5  630            MIF  '-'='' THEN LS ;IF NO OPERATOR, DO ASSIGN
                                                       MENT
F ED5B6902'      6  631            LD   DE,(J)        ;GET VAR3
    =0000        7  632            MIF  '-'='+' THEN L2 ;CHECK OPERATOR
    =FFFF        8  633            MIF  '-'='-' THEN L3
                17  634 L3         MNOP
3 B7            18  635            OR   A
4 ED52          19  636            SBC  HL,DE
                20  637            MGOTO LS
                57  638 LS         MNOP
6'226D02'       58  639 Z0028      LD   (SUB),HL      ;SAVE IN VAR1
                59  640            MEND
9               51  641            LET MUL = I * J
                 1  642            MLOCAL L1,L2,L3,L4,L5,LS,LERR
    =FFFF        2  643            MIF  '='='=' THEN L1 ELSE LERR ;SYNTAX CHECK
                 3  644 L1         MNOP
9 2A6702'        4  645            LD   HL,(I)        ;GET VAR2
    =0000        5  646            MIF  '*'='' THEN LS ;IF NO OPERATOR, DO ASSIGN
                                                       MENT
C ED5B6902'      6  647            LD   DE,(J)        ;GET VAR3
    =0000        7  648            MIF  '*'='+' THEN L2 ;CHECK OPERATOR
    =0000        8  649            MIF  '*'='-' THEN L3
    =FFFF        9  650            MIF  '*'='*' THEN L4
                21  651 L4         MNOP               ;MULTIPLY BY SEVERAL ADDITION
                                                       S
)0 7A           22  652            LD   A,D           ;CHECK FOR MULT BY ZERO
)1 B3           23  653            OR   E
)2 2006         24  654            JR   NZ,I0029
)4 210000       25  655            LD   HL,0          ;IF SO, ZERO RESULT
)7 C3A901'      26  656            JP   K0029
)A'1B           27  657 I0029      DEC  DE            ;CHECK FOR MULT BY ONE
)B 7A           28  658            LD   A,D
9C B3           29  659            OR   E
9D 280A         30  660            JR   Z,K0029       ;YES, JUST PUT IN VALUE
9F ED4B6702'    31  661            LD   BC,(I)        ;GET VAR2
A3'09           32  662 L0029      ADD  HL,BC
A4 1B           33  663            DEC  DE
A5 7A           34  664            LD   A,D           ;CHECK FOR END
A6 B3           35  665            OR   E
A7 20FA         36  666            JR   NZ,L0029
    =01A9'      37  667 K0029
                38  668            MGOTO LS
                57  669 LS         MNOP
A9'226F02'      58  670 Z0029      LD   (MUL),HL      ;SAVE IN VAR1
                59  671            MEND
AC              52  672            LET DIV = I / J
                 1  673            MLOCAL L1,L2,L3,L4,L5,LS,LERR
    =FFFF        2  674            MIF  '='='=' THEN L1 ELSE LERR ;SYNTAX CHECK
                 3  675 L1         MNOP
AC 2A6702'       4  676            LD   HL,(I)        ;GET VAR2
    =0000        5  677            MIF  '/'='' THEN LS ;IF NO OPERATOR, DO ASSIG
                                                       MENT
```

```
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 FIG410 FIG410 FIG410 REL

01AF ED5B6902'      6  678         LD   DE,(J)        ;GET VAR3
         =0000      7  679         MIF  '/'='+' THEN L2 ;CHECK OPERATOR
         =0000      8  680         MIF  '/'='-' THEN L3
         =0000      9  681         MIF  '/'='*' THEN L4
         =FFFF     10  682         MIF  '/'='/' THEN L5
                   43  683 L5      MNOP
01B3 7A            44  684         LD   A,D           ;CHECK FOR DIVIDE BY ZE
01B4 B3            45  685         OR   E
01B5 2021          46  686         JR   NZ,C0030
01B7               47  687         PRINT '*** OVERFLOW ERROR'
                    1  688         GLOBAL PTXT
01B7 1E01           2  689         LD   E,CHNL+1      ;CHANNEL NBR
01B9 21C101'        3  690         LD   HL,MS0031
01BC CDF300'        4  691         CALL PTXT
01BF 1815           5  692         JR   L0031
01C1'2A2A2A20       6  693 MS0031  DEFM '*** OVERFLOW ERROR',0DH,0AH,3H
     4F564552
     464C4F57
     20455252
     4F520D0A
     03
        =01D6'      7  694 L0031
                    8  695         MEND
01D6 180C          48  696         JR   Z0030
01D8'010000        49  697 C0030   LD   BC,0          ;RESULT
01DB'B7            50  698 D0030   OR   A             ;RESET CARRY
01DC ED52          51  699         SBC  HL,DE         ;SUBTRACT UNTIL DONE
01DE 03            52  700         INC  BC
01DF 30FA          53  701         JR   NC,D0030      ;LOOP UNTIL NEGATIVE
01E1 0B            54  702         DEC  BC            ;CORRECT THE RESULT
01E2 69            55  703         LD   L,C           ;PUT INTO HL
01E3 60            56  704         LD   H,B
                   57  705 LS      MNOP
01E4'227102'       58  706 Z0030   LD   (DIV),HL      ;SAVE IN VAR1
                   59  707         MEND
01E7               53  708         WRITE ADD,SUB,MUL,DIV
                           ; ADD FIRST TIME USAGE OF PARAMETER
                    2  710         MLOCAL L1
                    3  711         GLOBAL PTXT,CRLF,PADDO
01E7 1E01           4  712         LD   E,CHNL+1      ;OUTPUT CHANNEL
                    5  713 L1      MNOP
01E9 21F701'        6  714         LD   HL,MSADD      ;OUTPUT MESSAGE
01EC CDBD01'        7  715         CALL PTXT
01EF 2A6B02'        8  716         LD   HL,(ADD)
01F2 CDB400'        9  717         CALL PADDO         ;WRITE OUT IN HEX
01F5 1807          10  718         JR   LADD
01F7'41444420      11  719 MSADD   DEFM 'ADD = '
     3D20
01FD 03            12  720         DEFB 3
        =01FE'     13  721 LADD
                   14  722         MNEXT 1 THEN L1
                    5  723 L1      MNOP
01FE 210C02'        6  724         LD   HL,MSSUB      ;OUTPUT MESSAGE
0201 CDED01'        7  725         CALL PTXT
0204 2A6D02'        8  726         LD   HL,(SUB)
0207 CDF301'        9  727         CALL PADDO         ;WRITE OUT IN HEX
020A 1807          10  728         JR   LSUB
```

```
)C  OBJ.CODE       STMT-NR SOURCE-STMT PASS2 FIG410 FIG410 FIG410 REL

20C'53554220       11   729 MSSUB    DEFM 'SUB = '
    3D20
212 03             12   730          DEFB 3
       =0213'      13   731 LSUB
                   14   732          MNEXT 1 THEN L1
                    5   733 L1       MNOP
213 212102'         6   734          LD   HL,MSMUL      ;OUTPUT MESSAGE
216 CD0202'         7   735          CALL PTXT
219 2A6F02'         8   736          LD   HL,(MUL)
21C CD0802'         9   737          CALL PADDO         ;WRITE OUT IN HEX
21F 1807           10   738          JR   LMUL
221'4D554C20       11   739 MSMUL    DEFM 'MUL = '
    3D20
227 03             12   740          DEFB 3
       =0228'      13   741 LMUL
                   14   742          MNEXT 1 THEN L1
                    5   743 L1       MNOP
)228 213602'        6   744          LD   HL,MSDIV      ;OUTPUT MESSAGE
)22B CD1702'        7   745          CALL PTXT
)22E 2A7102'        8   746          LD   HL,(DIV)
)231 CD1D02'        9   747          CALL PADDO         ;WRITE OUT IN HEX
)234 1807          10   748          JR   LDIV
)236'44495620      11   749 MSDIV    DEFM 'DIV = '
    3D20
)23C 03            12   750          DEFB 3
       =023D'      13   751 LDIV
                   14   752          MNEXT 1 THEN L1
023D CD3E01'       15   753          CALL CRLF
                   16   754          MEND
0240               54   755          ENDDO
0240 DDE1           1   756          POP  IX            ;LOOP ADDRESS
0242 D1             2   757          POP  DE            ;FINAL VALUE
0243 E1             3   758          POP  HL            ;CURRENT VALUE
0244 23             4   759          INC  HL            ;INCREMENT VAR1
0245 E5             5   760          PUSH HL
0246 B7             6   761          OR   A             ;CHECK IT
0247 ED52           7   762          SBC  HL,DE
0249 E1             8   763          POP  HL
024A 2802           9   764          JR   Z,KK0033      ;LAST TIME THRU
024C 3002          10   765          JR   NC,L0033      ;IF DONE, SKIP OUT
024E'DDE9          11   766 KK0033   JP   (IX)          ;ELSE LOOP
       =0250'      12   767 L0033
                   13   768          MEND
0250               55   769          ENDDO
0250 DDE1           1   770          POP  IX            ;LOOP ADDRESS
0252 D1             2   771          POP  DE            ;FINAL VALUE
0253 E1             3   772          POP  HL            ;CURRENT VALUE
0254 23             4   773          INC  HL            ;INCREMENT VAR1
0255 E5             5   774          PUSH HL
0256 B7             6   775          OR   A             ;CHECK IT
0257 ED52           7   776          SBC  HL,DE
0259 E1             8   777          POP  HL
025A 2802           9   778          JR   Z,KK0034      ;LAST TIME THRU
025C 3002          10   779          JR   NC,L0034      ;IF DONE, SKIP OUT
025E'DDE9          11   780 KK0034   JP   (IX)          ;ELSE LOOP
       =0260'      12   781 L0034
                   13   782          MEND
```

```
LOC  OBJ.CODE        STMT-NR SOURCE-STMT PASS2 FIG410 FIG410 FIG410 REL

0260                 56  783          EXIT
                      1  784          GLOBAL JTASK
0260 3E01             2  785          LD   A,1
0262 C3DD00'          3  786          JP   JTASK
                      4  787          MEND
                             ;
0265                 58  789          DCL ZERO
                      1  790          MLOCAL L1,L2,L3
        =0000         2  791          MIF  ''='INIT' THEN L1 ELSE L2
                      6  792 L2       MNOP
0265'0000             7  793 ZERO     DEFW 0           ;DEFAULT TO ZERO
                      8  794          MEND
0267                 59  795          DCL I
                      1  796          MLOCAL L1,L2,L3
        =0000         2  797          MIF  ''='INIT' THEN L1 ELSE L2
                      6  798 L2       MNOP
0267'0000             7  799 I        DEFW 0           ;DEFAULT TO ZERO
                      8  800          MEND
0269                 60  801          DCL J
                      1  802          MLOCAL L1,L2,L3
        =0000         2  803          MIF  ''='INIT' THEN L1 ELSE L2
                      6  804 L2       MNOP
0269'0000             7  805 J        DEFW 0           ;DEFAULT TO ZERO
                      8  806          MEND
026B                 61  807          DCL ADD
                      1  808          MLOCAL L1,L2,L3
        =0000         2  809          MIF  ''='INIT' THEN L1 ELSE L2
                      6  810 L2       MNOP
026B'0000             7  811 ADD      DEFW 0           ;DEFAULT TO ZERO
                      8  812          MEND
026D                 62  813          DCL SUB
                      1  814          MLOCAL L1,L2,L3
        =0000         2  815          MIF  ''='INIT' THEN L1 ELSE L2
                      6  816 L2       MNOP
026D'0000             7  817 SUB      DEFW 0           ;DEFAULT TO ZERO
                      8  818          MEND
026F                 63  819          DCL MUL
                      1  820          MLOCAL L1,L2,L3
        =0000         2  821          MIF  ''='INIT' THEN L1 ELSE L2
                      6  822 L2       MNOP
026F'0000             7  823 MUL      DEFW 0           ;DEFAULT TO ZERO
                      8  824          MEND
0271                 64  825          DCL DIV
                      1  826          MLOCAL L1,L2,L3
        =0000         2  827          MIF  ''='INIT' THEN L1 ELSE L2
                      6  828 L2       MNOP
0271'0000             7  829 DIV      DEFW 0           ;DEFAULT TO ZERO
                      8  830          MEND
0273                 65  831          END
```

FIGURE 4-11.

SAMPLE RUNS

FIBONACCI SERIES:

```
ENTER 2 HEX DIGITS
07
A = 0001 B = 0002
C = 0003
C = 0005
C = 0008
C = 000D
C = 0015
C = 0022
C = 0037
```

COMBINATIONS:

```
ENTER TWO HEX DIGITS
04
ADD = 0002 SUB = 0000 MUL = 0001 DIV = 0001
ADD = 0003 SUB = FFFF MUL = 0002 DIV = 0000
ADD = 0004 SUB = FFFE MUL = 0003 DIV = 0000
ADD = 0005 SUB = FFFD MUL = 0004 DIV = 0000
ADD = 0003 SUB = 0001 MUL = 0002 DIV = 0002
ADD = 0004 SUB = 0000 MUL = 0004 DIV = 0001
ADD = 0005 SUB = FFFF MUL = 0006 DIV = 0000
ADD = 0006 SUB = FFFE MUL = 0008 DIV = 0000
ADD = 0004 SUB = 0002 MUL = 0003 DIV = 0003
ADD = 0005 SUB = 0001 MUL = 0006 DIV = 0001
ADD = 0006 SUB = 0000 MUL = 0009 DIV = 0001
ADD = 0007 SUB = FFFF MUL = 000C DIV = 0000
ADD = 0005 SUB = 0003 MUL = 0004 DIV = 0004
ADD = 0006 SUB = 0002 MUL = 0008 DIV = 0002
ADD = 0007 SUB = 0001 MUL = 000C DIV = 0001
ADD = 0008 SUB = 0000 MUL = 0010 DIV = 0001
```

```
;                 FIGURE 4-12.
        NLIST
;
; THESE DEFINITIONS ARE FOR THE CONVENIENCE OF THE USER WRITING
; IOCS-BASED PROGRAMS.  THESE DEFINITIONS MAY BE CHANGED TO SUIT
; THE USER, BUT BEWARE OF POSSIBLE CONFLICT WITH SYSTEM PROGRAMS
; AND ROUTINES INCLUDING THIS FILE.  THE USER MAY ALSO ADD ADDITIONAL
; DEFINITIONS, ESPECIALLY IN THE ERROR CODE SECTION (ERRC)
;
; THIS FILE IS GENERALLY USED AS AN INCLUDED FILE:
;       INCLUDE IODEF
;
; I/O SYSTEM DEFINITIONS
;
; VECTOR DISPLACEMENTS
;
LUNIT   EQU     0       ;DEFB 1  BYTE
DVCE    EQU     1       ;DEFM 2  BYTE
UNIT    EQU     2       ;DEFM 1  BYTE
FNAM    EQU     4       ;DEFM 6  BYTE
FEXT    EQU     10      ;DEFM 3  BYTE
VERS    EQU     13      ;DEFB 1  BYTE
USER    EQU     14      ;DEFB 1  BYTE
RQST    EQU     15      ;DEFB 1  BYTE
FMAT    EQU     16      ;DEFB 1  BYTE
;HADDR  EQU     17      ;DEFW 2  BYTE
ERRA    EQU     19      ;DEFW 2  BYTE
CFLGS   EQU     21      ;DEFB 1  BYTE
SFLGS   EQU     22      ;DEFB 1  BYTE
ERRC    EQU     23      ;DEFB 1  BYTE
;PBFFR  EQU     24      ;DEFB 1  BYTE
UBFFR   EQU     25      ;DEFW 2  BYTE
USIZE   EQU     27      ;DEFW 2  BYTE
;NREC   EQU     29      ;DEFB 1  BYTE
;HSCR   EQU     30      ;DEFS 10 BYTE
;ISCR   EQU     40      ;DEFS 8  BYTE
;
;
; REQUEST CODES
;
OPRRQ   EQU     0       ;OPEN READ
OPWRQ   EQU     1       ;OPEN WRITE
CLRQ    EQU     2       ;CLOSE
RDRQ    EQU     3       ;READ
WRRQ    EQU     4       ;WRITE
RWRQ    EQU     5       ;REWIND
INRQ    EQU     6       ;INITIALIZE
ERRQ    EQU     7       ;ERASE
;
;
; FORMAT CODES
;
BYTE    EQU     00H     ;BYTE I/O THRU ACCUMULATOR
LINE    EQU     10H     ;ASCII LINE I/O, TERMINATED BY CR/LF
LBUF    EQU     20H     ;LOGICAL BUFFER, LENGTH IN USIZE
BIN     EQU     30H     ;BINARY RAM IMAGE
;
;
; CFLGS CODES
;
```

```
OUNT    EQU     1       ;MOUNT/DISMOUNT
CHO     EQU     2       ;AUTO ECHO FOR CONSOLE DEVICES
RET     EQU     4       ;IMMEDIATE RETURN REQUESTED
DRW     EQU     8       ;READ AFTER WRITE
RRPR    EQU     16      ;ERROR PRINT
PAR     EQU     32      ;STRIP PARITY


 SFLGS CODES

NOP     EQU     1       ;UNIT OPEN
NOPW    EQU     2       ;UNIT OPEN FOR WRITE
NON     EQU     4       ;UNIT ON
OF      EQU     8       ;END OF FILE DETECTED
;
;
; ERROR CODES FOR ERRC
;
;
INVOP   EQU     1       ;INVALID OPERATION
DUPFIL  EQU     2       ;DUPLICATE FILE
FNF     EQU     4       ;FILE NOT FOUND
IOTIME  EQU     7       ;IO TIME OUT
NOPEN   EQU     8       ;FILE NOT OPEN
EOFERR  EQU     9       ;ATTEMPT TO READ PAST END OF FILE
;
;
; ASCII SPECIAL CHARACTERS
;
ETX     EQU     03H
EOT     EQU     04H
BEL     EQU     07H
HT      EQU     09H
LF      EQU     0AH
FF      EQU     0CH
CR      EQU     0DH
DEL     EQU     7FH
;
;
        LIST
```

FIGURE 4-13.

```
        NLIST
**************************************************
  IOMAC

  MACRO DEFINITIONS FOR I/O FUNCTIONS
**************************************************
VECTOR  MACRO   #LUN,#DEV='DK0',#NAME='        ,#EXT='   ',#FMAT,#CFLGS,#UB
        MLOCAL  L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12
        DEFB    #LUN
        DEFM    '#DEV'
        DEFM    '#NAME'
        DEFM    '#EXT'
        DEFB    0,0,0
        MIF     '#FMAT'='' THEN L1 ELSE L2
L1      DEFB    BYTE+4
        MGOTO   L3
L2      DEFB    #FMAT
L3      DEFW    0,0
        MIF     '#CFLGS'='' THEN L4 ELSE L5
L4      DEFB    0
        MGOTO   L6
L5      DEFB    #CFLGS
L6      DEFB    0,0,0
        MIF     '#UBFFR'='' THEN L7 ELSE L8
L7      DEFW    0
        MGOTO   L9
L8      DEFW    #UBFFR
L9      MIF     '#USIZE'='' THEN L10 ELSE L11
L10     DEFW    0
        MGOTO   L12
L11     DEFW    #USIZE
L12     DEFB    0
        DEFW    0,0,0,0,0,0,0,0,0
        MEND
;
;
OPENR   MACRO   #VECTOR,#ERR,#ERRPR
        GLOBAL  JIOCS,JTASK
        MLOCAL  L1,L2,L3,L4,L5,L6,L7
        MIF     '#VECTOR'='' THEN L6 ELSE L7
L7      LD      IY,#VECTOR
L6      LD      (IY+RQST),OPRRQ
        MIF     '#ERRPR'='' THEN L3 ELSE L4
L3      LD      (IY+CFLGS),0
        MGOTO   L5
L4      LD      (IY+CFLGS),#ERRPR
L5      CALL    JIOCS
        LD      A,(IY+ERRC)
        AND     A
        MIF     '#ERR'='' THEN L1 ELSE L2
L2      JP      NZ,#ERR
L1      LD      A,1
        JP      NZ,JTASK
        MEND
;
;
OPENW   MACRO   #VECTOR,#ERR,#ERRPR
        GLOBAL  JIOCS,JTASK
        MLOCAL  L1,L2,L3,L4,L5,L6,L7
```

```
        MIF     '#VECTOR'='' THEN L6 ELSE L7
 7      LD      IY,#VECTOR
 5      LD      (IY+RQST),OPWRQ
        MIF     '#ERRPR'='' THEN L3 ELSE L4
 3      LD      (IY+CFLGS),0
        MGOTO   L5
 4      LD      (IY+CFLGS),#ERRPR
 5      CALL    JIOCS
        LD      A,(IY+ERRC)
        AND     A
        MIF     '#ERR'='' THEN L1 ELSE L2
 2      JP      NZ,#ERR
        MEXIT
 1      LD      A,1
        JP      NZ,JTASK
        MEND



:LOSE   MACRO   #VECTOR,#ERR,#ERRPR,#EOT
        MLOCAL  L1,L2,L3,L4,L5,L6,L7,L8,L9
        MIF     '#VECTOR'='' THEN L8 ELSE L9
L9      LD      IY,#VECTOR
L8      MIF     '#EOT'='' THEN L6 ELSE L7
L7      LD      (IY+RQST),WRRQ
        LD      (IY+FMAT),BYTE
        LD      A,EOT
        CALL    JIOCS
L6      LD      (IY+RQST),CLRQ
        MIF     '#ERRPR'='' THEN L3 ELSE L4
L3      LD      (IY+CFLGS),0
        MGOTO   L5
L4      LD      (IY+CFLGS),#ERRPR
L5      CALL    JIOCS
        LD      A,(IY+ERRC)
        AND     A
        MIF     '#ERR'='' THEN L1 ELSE L2
L2      JP      NZ,#ERR
L1      LD      A,1
        JP      NZ,JTASK
        MEND
;
;
PARSE   MACRO   #VECTOR,#ERR
        GLOBAL  JTASK,PTXT
        MLOCAL  L1,L2,L3
        LD      IY,#VECTOR
        LD      A,6      ;CSIPAR
        CALL    JTASK    ;CALL VIA TASK
        MIF     '#ERR'='' THEN L1 ELSE L2
L1      MNOP
        JR      Z,I%NEXP         ;IF NO ERRORS, SKIP
        LD      HL,MS%NEXP       ;GET SYNTAX ERROR MESSAGE
        LD      E,1      ;PRINT ON LUN 1
        CALL    PTXT
        LD      A,1      ;RETURN TO MONITOR
        JP      JTASK
MS%NEXP DEFM    'SYNTAX ERROR'
I%NEXP
        MGOTO   L3
```

```
L2      JP      NZ,#ERR
L3      MNOP
        LD      A,(IY+DVCE)
        CP      ' '
        JR      NZ,L%NEXP
        LD      (IY+DVCE),'D'
        LD      (IY+DVCE+1),'K'
L%NEXP  EQU     $
        MEND
;
;
READ    MACRO   #VECTOR,#ERR,#ERRPR     ;READ BYTE AT A TIME
        MLOCAL  L1,L2,L3,L4,L5,L6,L7
        MIF     '#VECTOR'='' THEN L7
        LD      IY,#VECTOR
L7      LD      (IY+RQST),RDRQ  ;READ REQUEST
        MIF     '#ERRPR'='' THEN L3 ELSE L4
L3      LD      (IY+CFLGS),0
        MGOTO   L5
L4      LD      (IY+CFLGS),#ERRPR
L5      CALL    JIOCS
        LD      D,A     ;SAVE CHARACTER FOR BYTE MODE
        LD      A,(IY+ERRC)     ;CHECK FOR ERROR
        AND     A
        MIF     '#ERR'='' THEN L1 ELSE L2
L2      JP      NZ,#ERR         ;RETURN VIA ERROR EXIT
L1      LD      A,1
        JP      NZ,JTASK        ;RETURN TO MONITOR
        LD      A,D     ;RESTORE BYTE FOR BYTE I/O
        MEND
;
;
WRITE   MACRO   #VECTOR,#ERR,#ERRPR     ;WRITE
        MLOCAL  L1,L2,L3,L4,L5,L6,L7
        MIF     '#VECTOR'='' THEN L7
        LD      IY,#VECTOR
L7      LD      (IY+RQST),WRRQ  ;WRITE REQUEST
        MIF     '#ERRPR'='' THEN L3 ELSE L4
L3      LD      (IY+CFLGS),0
        MGOTO   L5
L4      LD      (IY+CFLGS),#ERRPR
L5      CALL    JIOCS
        LD      A,(IY+ERRC)     ;CHECK FOR ERROR
        AND     A
        MIF     '#ERR'='' THEN L1 ELSE L2
L2      JP      NZ,#ERR         ;RETURN VIA ERROR EXIT
L1      LD      A,1
        JP      NZ,JTASK        ;RETURN TO MONITOR
        MEND
;
;
        LIST
```

approach to creating and calling IOCS related functions. Each is described below.

4-49. VECTOR lun,device,filename,file extension,format,cflgs,ubffr,usize

This macro creates an IOCS parameter vector with several default parameters supplied. Use of this macro eliminates the need to write out a complete parameter vector definition using DEFB, DEFW, and DEFM pseudo-ops in the program. The user calls the macro and specifies the logical unit number (LUN), device mnemonic and unit number (DEV), file name (NAME), and file extension (EXT). Optionally, the user may specify the format (FMAT), control flags (CFLGS), user buffer address (UBFFR), and user buffer size (USIZE). The following defaults are applied:

```
LUN = 0FFH
DEV = DK1:
NAME = blanks
EXT = blanks
FMAT = 0 (byte I/O)
CFLGS = 0
UBFFR = 0
USIZE = 0
```

All of the required bytes for the parameter vector are allocated when the macro is expanded.

4-50. OPENR vector name,error abort address,error print flag

This macro performs an open-for-read request via the vector specified in the first parameter. If the vector is not specified, then it is assumed that the IY register is pointing to the proper vector. If any errors were encountered, then exit is made via the error-abort address (second parameter), which is optional. If the error-exit address is not specified, then the macro returns control to the Monitor in case of an error. The third parameter, error-print flag, defaults to zero but can be set to 16H to force error printing via IOCS (this is the CFLGS parameter).

4-51. OPENW vector name,error-abort address,error-print flag

This macro performs an open for write request via the vector specified in the first parameter. All other operations are identical to OPENR.

4-52. CLOSE vector name,error abort address,error print flag

This macro performs a close function via the vector specified in the first parameter. All other operation is identical to OPENR.

4-53. PARSE vector name,error abort address

This macro provides a call to CSISYN and CSIPAR via the system routine

JTASK. Entry is with the HL register pair pointing to the dataset specification to be checked and parsed. The validity of the dataset specification is first checked, then it is parsed into the vector specified by the first parameter of the call to the macro. If any errors are found, then return is made via the second parameter. If this parameter is not given, then a message is printed (SYNTAX ERROR) and control is returned to the Monitor. If no errors are found and the device type is not given, then the device is defaulted to DK0.

4-54. EXIT

This macro returns control to the Monitor.

4-55. Figure 4-14 shows a typical program written using these macros. This program reads a dataset and prints it on the console output device (TT:). The dataset is specified in the Monitor command line which calls up this program. Upon entry to the program, the DE register pair points to the dataset specification. After initializing the stack pointer and interrupt mode, the dataset specification pointer is placed into the HL register pair. The dataset is parsed into INPUT, the input vector. The dataset is then opened. The output dataset is opened for write. This dataset is specified in the vector OUTPUT, which appears later in the program. Then a series of read/write operations are performed in byte I/O mode. The end of the data is specified by an ASCII 04H (end-of-file). When this character is read, the input dataset is closed and the program is terminated. (Closing the output dataset, the console device, is not necessary here).

```
 OBJ.CODE     STMT-NR SOURCE-STMT PASS2 FIG414 FIG414 FIG414 REL

                    1          TITLE FIGURE 4-14.
                      ;
                      ; APPLICATION OF I/O MACROS
                      ;
                      ; THIS PROGRAM READS A DATASET IN BYTE I/O
                      ; AND COPIES IT TO THE CONSOLE DEVICE (TT:).
                      ; TO EXECUTE THE PROGRAM:
                      ;
                      ;    $VIEW DATASET(CR)
                      ;     ----------------
                      ;
                      ; INCLUDE IOCS DEFINITIONS
                      ;
 0                 14          INCLUDE IODEF
                      ;     FIGURE 4-12.
              100  114          LIST
                      ;
                      ; INCLUDE I/O MACROS
                      ;
)0             18  118          INCLUDE IOMAC
                      ;     FIGURE 4-13.
              172  290          LIST
                      ;
                      ;
               21  293          CLIST 0          ;CODE LISTING ONLY
                      ;
                      ;
                      ; START OF PROGRAM
                      ;
 00 312101'    26  298          LD   SP,STACK    ;SET STACK POINTER
 03 ED5E       27  299          IM   2           ;INTERRUPT MODE FOR Z80
 05 FB         28  300          EI               ;ENABLE INTERRUPTS
 06 EB         29  301          EX   DE,HL       ;HL POINTS TO DATASET SPEC
                      ; PARSE THE DATASET INTO THE INPUT VECTOR
 07            31  303          PARSE INPUT
 07 FD212101'   3  306          LD   IY,INPUT
 0B 3E06        4  307          LD   A,6         ;CSIPAR
 0D CDFFFF      5  308          CALL JTASK       ;CALL VIA TASK
 10 2819        8  311          JR   Z,I0001     ;IF NO ERRORS, SKIP
 12 211F00'     9  312          LD   HL,MS0001   ;GET SYNTAX ERROR MESSAGE
 15 1E01       10  313          LD   E,1         ;PRINT ON LUN 1
 17 CDFFFF     11  314          CALL PTXT
 1A 3E01       12  315          LD   A,1         ;RETURN TO MONITOR
)1C C30E00'    13  316          JP   JTASK
)1F'53594E54   14  317 MS0001   DEFM 'SYNTAX ERROR'
    41582045
    52524F52
)2B FD7E01     19  321          LD   A,(IY+DVCE)
)2E FE20       20  322          CP   ' '
)30 2008       21  323          JR   NZ,L0001
)32 FD360144   22  324          LD   (IY+DVCE),'D'
)36 FD36024B   23  325          LD   (IY+DVCE+1),'K'
                      ; OPEN THE INPUT DATASET.  ANY ERRORS ABORT THE PROGR
                                                 M.
03A            33  329          OPENR INPUT,,ERRPR
03A FD212101'   4  333 L7       LD   IY,INPUT
03E FD360F00    5  334 L6       LD   (IY+RQST),OPRRQ
```

```
LOC  OBJ.CODE        STMT-NR SOURCE-STMT PASS2 FIG414 FIG414 FIG414 REL

0042 FD361510         9   336 L4       LD   (IY+CFLGS),ERRPR
0046 CDFFFF          10   337 L5       CALL JIOCS
0049 FD7E17          11   338          LD   A,(IY+ERRC)
004C A7              12   339          AND  A
004D 3E01            15   341 L1       LD   A,1
004F C21D00'         16   342          JP   NZ,JTASK
                              ; OPEN CONSOLE OUTPUT DRIVER.  IGNORE ANY ERRORS
0052                 35   345          OPENW OUTPUT,CONTINUE
0052 FD215101'        4   349 L7       LD   IY,OUTPUT
0056 FD360F01         5   350 L6       LD   (IY+RQST),OPWRQ
005A FD361500         7   352 L3       LD   (IY+CFLGS),0
005E CD4700'         10   354 L5       CALL JIOCS
0061 FD7E17          11   355          LD   A,(IY+ERRC)
0064 A7              12   356          AND  A
0065 C26800'         14   358 L2       JP   NZ,CONTINUE
        =0068'       36   360 CONTINUE
                              ;
                              ; READ BYTES FROM INPUT DATASET.  ABORT IF ERRORS
        =0068'       39   363 LOOP
0068                 40   364          READ INPUT,,ERRPR
0068 FD212101'        3   367          LD   IY,INPUT
006C FD360F03         4   368 L7       LD   (IY+RQST),RDRQ ;READ REQUEST
0070 FD361510         8   370 L4       LD   (IY+CFLGS),ERRPR
0074 CD5F00'          9   371 L5       CALL JIOCS
0077 57              10   372          LD   D,A            ;SAVE CHARACTER FOR BYTE
0078 FD7E17          11   373          LD   A,(IY+ERRC) ;CHECK FOR ERROR
007B A7              12   374          AND  A
007C 3E01            15   376 L1       LD   A,1
007E C25000'         16   377          JP   NZ,JTASK       ;RETURN TO MONITOR
0081 7A              17   378          LD   A,D            ;RESTORE BYTE FOR BYTE I/
                              ; CHECK FOR END OF FILE BYTE
0082 FE04            42   381          CP   04H
0084 281A            43   382          JR   Z,DONE         ;IF SO, DONE
                              ; WRITE BYTE TO THE CONSOLE DEVICE
0086                 45   384          WRITE OUTPUT
0086 FD215101'        3   387          LD   IY,OUTPUT
008A FD360F04         4   388 L7       LD   (IY+RQST),WRRQ ;WRITE REQUEST
008E FD361500         6   390 L3       LD   (IY+CFLGS),0
0092 CD7500'          9   392 L5       CALL JIOCS
0095 FD7E17          10   393          LD   A,(IY+ERRC) ;CHECK FOR ERROR
0098 A7              11   394          AND  A
0099 3E01            14   396 L1       LD   A,1
009B C27F00'         15   397          JP   NZ,JTASK       ;RETURN TO MONITOR
009E 18C8            46   399          JR   LOOP           ;LOOP FOR MORE BYTES
                              ;
                              ; END OF FILE FOUND, CLOSE THE INPUT DATASET
                              ;
00A0'                50   403 DONE     CLOSE INPUT
00A0 FD212101'        3   406 L9       LD   IY,INPUT
00A4 FD360F02         9   408 L6       LD   (IY+RQST),CLRQ
00A8 FD361500        11   410 L3       LD   (IY+CFLGS),0
00AC CD9300'         14   412 L5       CALL JIOCS
00AF FD7E17          15   413          LD   A,(IY+ERRC)
00B2 A7              16   414          AND  A
00B3 3E01            19   416 L1       LD   A,1
00B5 C29C00'         20   417          JP   NZ,JTASK
00B8 3E01            51   419          LD   A,1
```

```
:  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 FIG414 FIG414 FIG414 REL

3A C3B600'        52  420          JP   JTASK        ;RETURN TO MONITOR
                          ;
                          ;
                          ; DEFINE STACK AREA
3D                56  424          DEFS 100
      =0121'      57  425 STACK
                          ;
                          ; DEFINE I/O VECTORS
                          ;
 21'              61  429 INPUT    VECTOR OFFH,,,,04H
 21 FF             2  431          DEFB OFFH
 22 444B30         3  432          DEFM 'DKO'
 25 20202020       4  433          DEFM '
    2020
 2B 202020         5  434          DEFM '
 2E 000000         6  435          DEFB 0,0,0
 31 04            10  437 L2       DEFB 04H
 32 00000000      11  438 L3       DEFW 0,0
 36 00            13  440 L4       DEFB 0
 37 000000        16  442 L6       DEFB 0,0,0
 3A 0000          18  444 L7       DEFW 0
 3C 0000          22  447 L10      DEFW 0
 3E 00            25  449 L12      DEFB 0
 3F 00000000      26  450          DEFW 0,0,0,0,0,0,0,0,0
    00000000
    00000000
    00000000
    0000
                          ; (FMAT IS BYTE I/O WITH 4 SECTORS PER DISK ACCESS)
                          ;
151'              64  454 OUTPUT   VECTOR OFFH,TTO,,,,
151 FF             2  456          DEFB OFFH
152 545430         3  457          DEFM 'TTO'
155 20202020       4  458          DEFM '
    2020
15B 202020         5  459          DEFM '   '
15E 000000         6  460          DEFB 0,0,0
161 04             8  462 L1       DEFB BYTE+4
162 00000000      11  464 L3       DEFW 0,0
166 00            13  466 L4       DEFB 0
167 000000        16  468 L6       DEFB 0,0,0
16A 0000          18  470 L7       DEFW 0
16C 0000          22  473 L10      DEFW 0
16E 00            25  475 L12      DEFB 0
16F 00000000      26  476          DEFW 0,0,0,0,0,0,0,0,0
    00000000
    00000000
    00000000
    0000
                          ; (THE EXTRA COMMAS ARE REQUIRED TO DEFAULT THE
                          ; FILENAME AND EXTENSION TO BLANKS)
                          ;
0181              68  481          END
```

APPENDIX A

MACRO-80 ERROR CODES

3F RELOCATABLE USE - A relocatable value was used in an 8-bit operand. The user should assure that relocatable quantities are used only for 16-bit operand values (addresses).

40 BAD LABEL - An invalid label was specified. A label must start with an alphabetic character (A-Z) and may contain only alphanumeric characters (A-Z, 0-9) or question mark (?) or underline (_). A label may start in any column if followed by a colon. It does not require a colon if started in column one.

41 BAD OPCODE - An invalid Z80 opcode or pseudo-op or an undefined macro name was specified.

42 BAD OPERAND - An invalid operand or combination of operands was specified for a given opcode.

43 BAD SYNTAX - The specification of an operand or expression was invalid.

44 UNDEFINED - A symbol was used in an operand which was not defined in the program, either locally or as an external symbol.

45 MULTIPLE DEF - A symbol was defined more than once in the same program.

46 MULTIPLE PSECT - A PSECT pseudo-op was used more than once or was defined after the first code-producing statement of the program. The PSECT pseudo-op should be used only once at the beginning of a program.

47 MEMORY OVERFLO - This means that not enough memory exists in the system to assemble the given program. This can occur because the program contains too many symbols, macro parameters, or macro expansion arguments.

48 EXTERNAL USAGE - An external symbol was used in an expression or the operand of an EQU or DEFL pseudo-op. The user should assure that an external symbol is not used in these situations.

49 not used.

4A UNBAL QUOTES - An uneven number of quote characters (') occurred in an operand.

4B LABEL REQUIRED - A label was not used in a statement that required it. A label is required for EQU, DEFL, and MACRO statements.

4C OVERFLOW - In evaluating an expression, the value of the expression exceeded 65536 (OFFFFH). The user should check the expression for validity. Alternatively, the .RES. operation may be used to ignore the overflow condition and only the least significant 16 bits of the expression will be used.

4D OUT OF RANGE - The final value of an operand was found to be out of the range allowed for the given opcode. For example, the valid range of the JR instruction is -126 through +129.

4E BAD DIGIT - An invalid digit was found in a number.

4F not used.

50 not used.

51 not used.

52 MULTIPLE NAME - The NAME pseudo-op was used more than once in the same program.

53 NESTED INCLUDE - An included file contained another INCLUDE pseudo-op. The user should assure that the INCLUDE pseudo-op is not used in the body of an included module.

54 EXPR TOO BIG - The expression evaluator stack reached its limit. The user should reduce the complexity of the expression in the statement which caused the error.

55 not used.

56 NUMBER TOO LARGE - A constant in an operand was too large in value for the given operation.

57 OUT OF RANGE - The value of either operand in the string operand [,] was found to be out or range. The limits are 1 and 63.

58 TOO MANY IFS - The nesting of conditional assembly pseudo-ops (IF and ENDIF, or COND and ENDC) was too large or unmatched. The maximum level of nesting is 11, and each IF (COND) statement must be matched by an ENDIF (ENDC) statement.

59 STRING TOO BIG - The size of the substring in a sequence of substring operations exceeded the available space. The user should reduce the number of substring expressions within the statement or macro body.

5A MERROR INDICATION - This error code is output when an MERROR statement is expanded in a macro.

5B BAD THEN/ELSE - A THEN-clause or ELSE-clause operand was incorrectly specified. The operand must be a local macro label defined by an MLOCAL pseudo-op.

5C TOO MANY PARMS - The maximum number of parameter substitutions in calling a macro was exceeded. Maximum is 99.

5D BAD MACRO STMT - A macro pseudo-op was used outside of a macro body.

5E INCLUDE IN MAC - An INCLUDE statement was used inside a macro body.

5F LABEL USAGE - The usage of a label in a macro expansion was not allowed.

60 NO MEND STMT - A macro was defined without an MEND statement.

MOSTEK®
Z80•F8 Covering the full spectrum of
3870 microcomputer applications.

1215 W. Crosby Rd. • Carrollton, Texas 75006 • 214/242-0444
In Europe, Contact: MOSTEK Brussels
150 Chaussee de la Hulpe, B1170, Belgium;
Telephone: (32) 02/660-2568/4713

PRINTED IN USA January 1979
Publication No. MK79635