IMSAI 8080
Self-Contained System
Assembler
Revision 1

The IMSAI 8080 Monitor, Assembler, and Text Editor, supplied by IMS Associates, Incorporated free of charge, is a modified version of software written by Microtec of Sunnyvale, California for Processor Technology of Berkeley, California who distributed the package free of charge.

IMSAI 8080 SELF-CONTAINED SYSTEM

OPERATING SYSTEM

The IMSAI 8080 Self-Contained System is a software system designed to run on the IMSAI 8080 computer. Included in the package is an Executive to handle memory files, an Assembler, and a line oriented Editor.

To use the system 6K of memory must be available for use by the system. This memory is allocated as follows:

0050 - OEFF	Operating Program
1000 - 1123	Special System RAM
1000 - 17FF	Symbol Table (Assembler Only)

In addition other memory must be available for source and object files necessary for the user's program.

I/O within the program interacts with I/O ports addressed as follows:

PORT	FUNCTION
3	Status Input Bit 6 indicates DAV Bit 7 indicates TBE
2	TTY Input
FF	Sense Switch Input ADDRESS - PROGRAMMED INPUT switch seven is used to control file listing.

IMSAI 8080
Self-Contained System
Operating System
Revision 1

Executive Commands

CONTROL X	System reset and CR/LF
ENTR	Enter data to memory
DUMP	Display memory data
FILE	Create, assign or display file information
EXEC	Execute a program
ASSM	Assemble a source file to object code
LIST	List file
DELT	Delete lines of file.
1111	Any four numeric digits enters editor
PAGE	Move a page of data
BREK	Set or clear break points
PROC	Proceed from break point
CUST	Optional user command at location 2000

To initialize the system, start it at 0050. To restart the system without initializing it, start at 0053.

The executive has one error message.....WHAT?.....indicating an improper command or an error on parameters following the command.

Command Format

ENTR AAAA----Enter data to memory

This command is used to enter data to memory starting at address AAAA and continuing until a return command (/) is given. Data is entered in hexadecimal format.

Example:

```
ENTR 500
0 0A 30 44 FF FE/
```

DUMP AAAA BBBB----Dump Contents of Memory

This command is used to examine the contents of memory. The values contained in memory from locations AAAA to BBBB are displayed in hexadecimal. Each line of display consists of an address followed by the contents of the next 16 memory locations. If BBBB is not specified, only location AAAA will be displayed.

FILE /NAME/ AAAA

This command is used to enter, examine or modify parameters of files created in the system. Up to six files can exist simultaneously with any one of the files called as "current". Depending on the form of the command, the following parameters the following functions are performed.

FILE /NAME/ ADDR. Create a file with the name, NAME starting at address ADDR and make it current. If a file with the same name already exists, output error message NO NO.

FILE /NAME/ 0 Delete file with name NAME and make no file current. Note: No file can start at ADDR 0.

FILE /NAME/ Get file NAME and make it current. Save all parameters of existing current file.

FILE Display parameters of the "current" file in the following format with AAAA and BBBB being the beginning of file and end of file addresses:

```
NAME  AAAA  BBBB
```

FILES Display the parameters of all files currently saved by the system.

EXEC AAAA----Execute a program

This command is used to execute a program at address AAAA.

LIST N----List file

This command is used to display the lines entered by the user into the file. The output consists of the lines in the file starting at line number N. If N is not specified, the display starts at the beginning of the file. The user can terminate the display by using ADDRESS-PROGRAMMED INPUT switch 7.

DELT L1 L2----Delete line(s) from file

This command is used to delete lines entered by the user from the file. All lines starting at line L1 and continuing up to and including L2 are deleted from the file. If L2 is not specified, only L1 is deleted.

PAGE AAAA BBBB----Move page of data

This command is used to move one page (256 bytes) of data from address AAAA to BBBB.

CUST----Optional user command at location 2000

This command allows any routine to be placed at location 2000 by the user. If the command is terminated by a RET and proper stack operations are used, the system will return in an orderly manner.

BREK or BREK AAAA

This command is used to set or clear break points. If called without the argument AAAA, all break points are cleared.

If called with the argument AAAA, a break point is set at location AAAA. When the break point is encountered in the course of execution, the break point is cleared, all registers are saved, the A register is displayed in the PROGRAMMED OUTPUT on the front panel, the message "AAAA BREAK" is typed and control returns to the executive. The registers are saved in the following locations, and may be examined or modified using the DUMP or ENTR commands.

Location	Register
1000	PSW
1001	A
1002	C
1003	B
1004	E
1005	D
1006	SP (low)
1007	SP (high)
1008	L
1009	H
100A	PC (low)
100B	PC (high)

Restrictions: (1) A maximum of 8 break points may be set.

(2) Break points may not be set below location 000B.

(3) Setting a break point causes information to be stored into locations 0008-000A, destroying any information already there.

PROC or PROC AAAA

This command is used to proceed from a break point. All registers are restored from the locations specified above, and execution continues from the location specified by the PC, unless the argument AAAA is given, in which case execution begins at location AAAA.

ASSM (E) AAAA BBBB ---- Assemble a source file to object code.

This command is used to assemble a source program written by the user and located in the file area. The assembler performs the assembly, assigning addresses to the object code starting at AAAA. On the second pass the object code is placed in memory starting at location BBBB. If BBBB is not specified, it assumes the same value as AAAA. During pass one certain errors are displayed, and during pass two a complete listing is produced. If the optional E is specified in the command, only those lines which contain errors are listed.

TEXT EDITOR

Editor

The editor is a line oriented editor which enables the user to easily create program files in the system. Each line is prefaced by a fixed line number which provides for stable line referencing. Since line numbers can range from 0000 to 9999 (decimal) there are 10,000 lines that can exist in each file. (If enough storage exists.) As the user types lines on the input device, they are entered into the file area. The editor places all line numbers in sequence and automatically over-writes an existing line in the file, if a new line with the same line number is entered by the user. A feature of the editor is that the file area never contains any wasted space.

Note: The Editor ALWAYS operates on the current file.

The editor does not automatically assign line numbers. The user must first, when entering a line of data, enter a decimal number which will be interpreted as being the line number. Valid line numbers must contain four digits... preceding zeros must be included. An entry to the editor is terminated by the carriage return key. No more than 80 characters may be input for one line.

All lines are ordered by the ascending numeric sequence of their line numbers. If the user wishes to insert lines after the initial entry is made, it is suggested that he input the original lines with line numbers at least five units difference.

ASSEMBLER

When the Assembler is given control by the executive, it proceeds to translate the Symbolic 8080 Assembly Language (Source) program into 8080 machine (object) code. The Assembler is a two pass assembler which operates on the "current" file. Features of the Assembler include:

- free format source input.
- symbolic addressing, including forward references and relative symbolic references.
- complex expressions may be used as arguments.
- self defining constants.
- multiple constant forms.
- up to 256 five character symbols.
- reserved names for 8080 registers
- ASCII character code generation
- 6 Pseudo Operations (assembler directives)

The assembler translates those lines contained in the current file into object code. The second character following the line number, is considered to be the first source code character position. Hence, the character immediately following the line number should normally be blank. Line numbers are not processed by the assembler; they are merely reproduced on the listing.

The assembler will assemble a source program file composed of STATEMENTS, COMMENTS, and PSEUDO OPERATIONS.

During Pass 1, the assembler allocates all storage necessary for the translated program and defines the values of all symbols used, by creating a symbol table. The storage allocated for the object code will begin at the first byte dictated by the 1st parameter in the original Executive ASSM command.

During Pass 2, all expressions, symbols and ASCII constants are evaluated to absolute values and are placed in allocated memory in the appropriate locations. The listing, also produced during Pass 2, indicates exactly what data is in each location of memory.

Statements

Statements may contain either symbolic 8080 machine instructions or pseudo-ops. The structure of such a statement is:

```
     NAME     OPERATION     OPERAND     COMMENT
```

The name-field, if present, must begin in assembler character position one. The symbol in the name field can contain as many characters as the user wants; however, only the first 5 characters are used in the symbol table to uniquely define a symbol. All symbols in this field must begin with an alphabetic character and may contain no special characters.

The operation field contains either a 8080 operation mnemonic or a system pseudo-operation code.

The operand field contains parameters pertaining to the operation in the operation field. If two arguments are present, they must be separated by a comma. Example:

```
0015 FLOP  MOV M,B  COMMENT
0020 * COMMENT
0025       JMP  BEG
0030       CALL FLOP
0035 BEG   ADI  8+6-4
0040       MOV  A,B
```

All fields are separated and distinguished from one another by the presence of one or more blank characters (spaces).

The comment field is for explanatory remarks. It is reproduced on the listing without processing. See example 0015. Comment lines must start with an asterisk (*) in character position 1. See example 0020.

Symbolic Names

To assign a symbolic name to a statement, one merely places the symbol in the name field. To leave off the name field, the user skips two or more spaces after the line number and begins the operation field. If a name is attached to a statement, the assembler assigns it the value of the current Location Counter. The Location Counter always holds the address of the next byte to be assembled. The only exception to this is the EQU pseudo-op. In this case a symbol in the name field is assigned a value which is contained in the operand field of the EQU pseudo-op statement. Example:

```
0057 POTTS EQU 128
```

assigns the value 128 to the name POTTS. This data can then be used elsewhere in the program as: eg ADI POTTS.

Names are defined when they appear in the name field. All defined names may be used as symbolic arguments in the argument field. See examples 0015, 0025, 0030, 0035.

In addition to user defined names, the assembler has reserved several symbols, the value of which is predetermined. These names may not be used by the user except in the operand field. They are (with their value in parenthesis):

A-	the accumulator	(7)
B-	Register B	(0)
C-	Register C	(1)
D-	Register D	(2)
E-	Register E	(3)
H-	Register H	(4)
L-	Register L	(5)
M-	Memory (through H,L)	(6)

In addition to the above reserved symbols, there is the single special character symbol ($). This symbol changes in value as the assembly progresses. It is alway equated with the value of the program counter after the current instruction is assembled. It may only be used in the operand field. Examples:

```
JMP   $       means jump to the next location.
MOV   A,B     after this instruction; i.e., the
              MOV instruction.
LDA   $+5
DB     0
DB     1      means load the data at the fifth location
DB     2      after this location.  In this case, the
DB     3      data has the value 5.
DB     4
DB     5
```

Relative Symbolic Addressing

If the name of a particular location is known, a nearby location may be specified using the known name and a numeric offset. Example:

```
     JMP   BEG
     JPE   BEG+4
     CC    SUB
     CALL  $+48
 BEG MOV   A,B
     HALT
     MVI   C, 'B'
     INR   B
```

In this example the instruction JMP BEG refers to the MOV A,B instruction. The instruction JPE BEG+4 refers to the INR B instruction. BEG+4 means the address BEG plus four bytes. This form of addressing can be used to locate several bytes before or after a named location.

Constants

The Assembler allows the user to write positive or negative numbers directly in a statement. They will be regarded as decimal constants and their binary equivalents will be used appropriately. All unsigned numbers are considered positive. Decimal constants can be defined using the descriptor "D" after the numeric value. (This is not required, as the default is decimal.)

Hexadecimal constants may be defined using the descriptor "H" after a numeric value. IE. +10H, 10H, 3AH, 0F4H.

Note that a hexadecimal constant cannot start with the digits A-F. In this case, a leading 0 must be included. This enables the assembler to differentiate between a numeric value and a symbol.

ASCII constants may be defined by enclosing the ASCII character within single quote marks, i.e., 'C'. For double word constants, two characters may be defined within one quote string.

Expressions

An expression is a sequence of one or more symbols, constants or other expressions separated by the arithmetic operators plus or minus.

```
PAM +3
ISAB-'A'+52
LOOP+32H-5
```

Expressions are calculated using 16 bit arithmetic. All arithmetic is done modulo 65536. Single byte data cannot contain a value greater than 255 or less than -256. Any value outside this range will result in an assembler error.

Pseudo-Operations

The pseudo-operations are written as ordinary statements, but they direct the assembler to perform certain functions which do not always develop 8080 machine code. The following section describes the pseudo-ops.

ORG----Set Program Origin

Format is

```
label ORG expression
```

where the label is optional but if present will be equaled to the given expression.

END----End of Assembly

The pseudo-op informs the assembler that the last source statement has been read. The assembler will then start on pass 2 and terminate the assembly and pass control back to the executive. This pseudo-op is not needed when assembling from a memory file since the assembler will stop when an end of file indicator has been reached.

EQU----Equal Symbolic Value

Format is

label EQU expression

where label is a symbol the value of which will be determined from the expression, and expression is an expression which when evaluated will be assigned to the symbol given in the name field.

DS----Define Storage

Format is

label DS expression.

The DS causes the assembler to advance the Assembly Program Counter, effectively skipping past a given number of memory bytes.

DB----Define Byte

Format is

label DB expression.

This pseudo-op is used to reserve one byte of storage. The content of the byte is specified in the argument field.

DW----Define Word

This pseudo-op is used to define two bytes of storage. The evaluated argument will be placed in the two bytes; high order 8 bits in the low order byte, and the low order 8 bits in the high order byte. This conforms to the Intel format for two byte addresses.

Assembler Errors

The following error flags are output on the assembler listing when the error occurs. Some of the errors are only output during pass 1.

O	Opcode Error
L	Label Error
D	Duplicate Label Error
M	Missing Label Error
V	Value Error
U	Undefined Symbol
S	Snytax Error
R	Register Error
A	Argument Error.

OBJECT TAPE FORMAT

The IMSAI Self-Contained System is supplied on paper tape in a blocked hexadecimal format. The data on the tape is blocked into discrete records, each record containing record length, record type, memory address and checksum information in addition to data. A frame-by-frame description is as follows:

Frame 0

Record Mark. Signals the start of a record. The ASCII character colon (":" HEX 3A) is used as the record mark.

Frames 1,2
(0-9,A-F)

Record Length. Two ASCII characters representing a hexadecimal number in the range 0 to 'FF' (0 to 255). This is the count of actual data bytes in the record type or checksum. A record length of 0 indicates end of file.

Frames 3 to 6

Load Address. Four ASCII characters that represent the initial memory location where the data following will be loaded. The first data byte is stored in the location pointed to by the load address; succeeding data bytes are loaded into ascending addresses.

Frames 7, 8

Record Type. Two ASCII characters. Currently all records are type 0. This field is reserved for future expansion.

Frames 9 to 9+2*
(Record Length) -1

Data. Each 8 bit memory word is represented by two frames containing the ASCII characters (0 to 9, A to F) to represent a hexadecimal value 0 to 'FF'H (0 to 255).

Frames 9+2* (Record Length) to 9+2*(Record Length) +1

Checksum. The checksum is the negative of the sum of all 8 bit bytes in the record since the record mark (":") evaluated modulus 256. That is, if you add together all the 8 bit bytes, ignoring all carries out of an 8-bit sum, then add the checksum, the result is zero.

Example: If memory locations 1 through 3 contain 53F8EC, the format of the hex file produced when these locations are punched is:

```
:0300010053F8ECC5
```

SAVING AND RESTORING PROGRAMS

While the system has no explicit provision for saving and restoring programs, it is possible to do so with an ASR style teletype. The procedure is as follows:

1. Make the file you want to save the current file.

2. Type 'LIST', but don't type the carriage return.

3. Turn on the paper tape punch.

4. Type carriage return. The program will be listed on the teletype and simultaneously punched on the paper tape punch.

5. When the 'LIST' is completed, turn off the punch.

The procedure for restoring the file is as follows:

1. Make the file you want to restore into the current file.

2. Mount the tape in the paper tape reader.

3. Start the paper tape reader. The program will be automatically read in.