**** SLAC COMPUTATION RESEARCH GROUP ****

```
********************************************
*                                          *
*    INTEL 8080  MICROCOMPUTER ASSEMBLER   *
*                                          *
********************************************
```

This document describes an assembler for the INTEL 8080 microcomputer which is based on the IBM H-level assembler for the 360/370.

The language implemented is a minor variation of that described in Intel documentation ("Programming Manual for the 8080 Microcomputer System") and available through the MAC80 Assembler. Most of the differences are minor syntactical changes imposed by the macro language of the IBM assembler; see Table II for details.

SUMMARY OF CHANGES

The following additions were made to the language as defined by Intel:

1. LOD has been added as a synomym for MOV. MOV A,B encourages one to say "move A to B", which is the wrong interpretation. LOD A,B encourages "load A from B", and is therefore less likely to lead to errors.

2. The JMP, CALL, and RET instructions have been generalized to allow an optional first argument to specify a condition. This allows the short and possibly confusing opcodes like CM to be avoided.

3. Double register pairs can be clearly indicated; one may write BC instead of B when the pair is needed.

4. Instructions have been generalized, or general versions introduced, to minimize the number of special-case opcodes required. For example, the single opcode INC can be used to increment both single registers and register pairs. Thus one may
```
write    INC B     instead of  INR B
  and    INC BC    instead of  INX B
```

5. Immediate-operand mnemonics which are derived from the register version by the addition of "I" have been added. Thus further reduces the number of unique opcodes that must be learned.

6. Standard symbols for ASCII special characters have been added (e.g. @CR for carriage return). See table III.

Although all the opcodes described in Intel documentation have been implemented, the additional features just described allow the selection of a shorter but still complete subset. Table I lists the format of a consistent and easier-to-remember set of mnemonics which makes available the entire instruction repertoire of the 8080.

STATEMENT FORMS

1. Comment Statement

Any input card with an asterisk (*) in column 1 is a comment and may contain any information in columns 1-71.

2. Source statements

Source statements consist of up to four field: label, opcode, operand, and comment. Each field should contain no blanks, and the fields are separated from each other by one or more blanks. Only the opcode field is required; the others are optional.

A) Label

The label field, if present, must begin with a letter in column 1 and may continue for up to 15 more letters or digits. If the label field is to be omitted, column 1 must be a blank.

B) Opcode

The opcode is separated from the label (or column 1) by at least one blank. The opcode is required, and may be a machine instruction mnemonic or an Assembler pseudo-operation.

C) Operands

The operand field is optional, and may consist of a number of items separated by commas. There should be no imbedded blanks, except within quoted strings.

D) Comments

The comment field is optional and may extend out to column 71. If no operand field is required and you wish to include a comment, you must put a single comma in the operand field so that the comment will not be mistaken for the operand.

Although free-form input is allowed, program readability is almost always improved by starting the opcode, operand, and comment fields in fixed columns, such as 10 16 36.

TABLE I - 8080 INSTRUCTIONS

Notation: Upper case letters must be written as shown.
Lower case letters indicate variable fields to be substituted; see the KEY at the end.
{x,y,z} means x or y or z
[x] means x is optional

Opcode	Operands	Description
LOD	rrr,rrr	Register-to-register loads
LOD	SP,HL	
LODI	r,data	Load register from immediate data
LODI	r,addr,{<,>}	
LODI	rp[1],addr	
PUSH	rp[2]	Push onto stack pointed to by SP
POP	rp[2]	Pop from stack pointed to by SP
LD	A,gaddr	Load A from memory
ST	A,gaddr	Store A into memory
LD	HL,addr	Load HL from memory
ST	HL,addr	Store HL into memory
XCH	HL,DE	Exchange HL with DE
XCH	HL,(SP)	Exchange HL with the top-of-stack
opr	r	register-to-A arithmetic
ADD	HL,rp[1]	16-bit addition to HL
oprI	data	Immediate data to A arithmetic
oprI	addr,{<,>}	
ROT	rop[,n]	Accumlator rotate/shift
INC	{r,rp[1]}[,n]	Increment register or register pair
DEC	{r,rp[1]}[,n]	Decrement register or register pair
JMP	[cc,]addr	Jump
JMP	(HL)	Jump indirect
CALL	[cc,]addr	Call subroutine
RET	[cc]	Return from subroutine
RST	m	Restart (CALL 8*m, $0 \leq m \leq 7$)
IN	dev	Input to A from I/O device
OUT	dev	Output from A to I/O device
NOP	[n]	Null operation n times
HLT		Halt (actually Wait-for-Interrupt)
STC		Set carry bit on
CMC		Complement carry bit
CMA		Complement A
DAA		Decimal adjust A
EI		Enable Interrupts
DI		Disable Interrupts

KEY

rrr	one or more registers {A,B,C,D,E,H,L,M} LOD B,C means "load B from C" LOD BC,HL is equivalent to LOD B,H then LOD C,L M represents the memory location pointed to by HL.
r	one register {A,B,C,D,E,H,L,M}
rp¹	a register pair {BC,DE,HL,SP}
rp²	a register pair {BC,DE,HL,FA,PSW} (FA=PSW=flags and A)
data	is an 8-bit constant or expression
addr	is a 16-bit constant or expression
{<,>}	is the character < if the high-order 8 bits of the address are to be used, or the character > if the low-order 8 bits of the address are to be used. Mnemonic: view them as left- or right-pointing arrows.
gaddr	a "generalized" address {addr,(BC),(DE),(HL)}
opr	an arithmetic operation {ADD,ADC,SUB,SBB,ANA,XRA,ORA,CMP}
rop	a rotate operation {R,L} for 8-bit rotates {RC,LC} for 9-bit rotates
cc	a condition code {C,NC,Z,NZ,P,M,NS,S,PE,PO} (S=M=sign bit on, NS=P=sign bit off)
n	a positive integer exression or constant indicating how many such instructions should be generated.
dev	is an 8-bit I/O device number or expression.

ASSEMBLER PSEUDO-OPERATIONS

	TITLE	'...TEXT ...'	Title the listing
	SPACE	n	Space n lines
	EJECT		Eject to a new page
	PRINT	{OFF,ON}	Stop or restart listing
var var	EQU SET	expr expr	Define assembly-time variable
op	OPSYN	op	(re-)define opcode
	ORG	expr	Change location counter
	ALIGN	n,m	Align the location counter to n bytes past an m-byte boundary.
	PAGE	n	Adjust the location counter so the following n bytes will fit on one page.
[lab]	DB	op,op...	Define byte constants Each operand can be a constant, expression, or quoted character string.
[lab]	DW	op,op...	Define word constant(s) Each operand can be a constant or an address expression.
[lab]	DS	[n]	Define storage (i.e. skip locations without assigning their values). Reserves n bytes of memory (or 1 if n is omitted).

TABLE II - DIFFERENCES FROM MAC80

1. Numbers in hexadecimal are written X'A2' rather than A2H
2. Numbers in binary are written B'11001100' rather than 11001100B
3. There are no octal constants.
4. Labels should not be followed by a colon (:) but may be up to 16 characters long. They must begin in column 1. Opcodes may be used as labels.
5. There should be no blanks in the operand field. Comment fields need not begin with a semicolon (;), and all-comment cards should begin with an asterisk (*).
6. The functions MOD, NOT, AND, OR, XOR, SHR, and SHL are not available.
7. An instruction in parenthesis is not a legal expression.
8. Macros are defined in a slightly different format, but are much more powerful. See an IBM Assembler Language manual.
9. Conditional assembly using IF and ENDIF is done slightly differently. See an IBM Assembler Language manual.
10. Register symbols may not be redefined.
11. SET symbols may not be redefined; use the IBM assembler SETA, SETB, or SETC symbols instead.

TABLE III - SPECIAL ASCII SYMBOLS

0-7	8-15	16-23	24-31
@NUL	@BS	@DLE	@CAN
@SOH	@HT	@DC1	@EM
@STX	@LF	@DC2	@SUB
@ETX	@VT	@DC3	@ESC
@EOT	@FF	@DC4	@FS
@ENQ	@CR	@NAK	@GS
@ACK	@SO	@SYN	@RS
@BEL	@SI	@ETB	@US

127
@DEL

INTEL-STYLE OPCODES

The following shows the correspondence between the additional instructions of the Intel programming manual and those shown above. Note however, that the above list is complete in the sense that all valid machine instuctions can be generated with it.

MOV	LOD
MVI	LODI
INR	INC
INX	INC
DCR	DEC
DCX	DEC
ADI	ADDI
ACI	ADCI
SUI	SUBI
SBI	SBCI
ANI	ANDI
XRI	XRAI
ORI	ORAI
CPI	CMPI
JC	JMP,C
JNC	JMP,NC
JZ	JMP,Z
JP	JMP,P
JM	JMP,M
JPE	JMP,PE
JPO	JMP,PO
CC	CALL,C
CNC	CALL,NC
CZ	CALL,Z
CNZ	CALL,NZ
CP	CALL,P
CM	CALL,M
CPE	CALL,PE
CPO	CALL,PO
RC	RET,C
RNC	RET,NC
RZ	RET,Z
RNZ	RET,NZ
RP	RET,P
RM	RET,M
RPE	RET,PE
RPO	RET,PO
LXI	LODI
STA	ST A
LDA	LD A

XCHG	XCH HL,DE
XTHL	XCH HL,(SP)
SPHL	LOD SP,HL
PCHL	JMP (HL)
DAD	ADD HL
STAX	ST A
LDAX	LD A
SHLD	ST HL
LHLD	LD HL
RLC	ROT L
RRC	ROT R
RAL	ROT LC
RAR	ROT RC

HOW THE USE THE ASSEMBLER

There is a WYLBUR exec file available which will generate all the necesary JCL to use the 8080 assembler. To use it, you should

1) Get the source program in the active file.

2) EXE FRO #ACALL USER MCS GRO CG ON CAT
   This execfile will insert the JCL necessary to run the Assembler at the appropriate points in the active file. It asks some questions to which you may reply "?" to get information.

The Assembler produces a single print output file, which may be FETCHed at a WYLB UR terminal, or printed. The object file will be saved on disk SCFEV5 under your account, and will be called "OBJ". If an "OBJ" already exists from a previous assembly, it will be replaced.

Lenny Shustek   8/19/74
                 1/12/76