4- MACHINE LANGUAGE CODING

4-1<u>THE INSTRUCTION SET:</u>

The microprocessor's instruction set defines the basic operations that a programmer can specify to the device to perform. Table 4-1 contains list basic instructions for the 8086. For the purpose of discussion, these instructions are organized into groups of functionally related instructions. In Table 4-1, we see that these groups consist of the data transfer instructions, arithmetic instructions, logic instructions, string manipulation instructions, control transfer instructions, and processor control instructions.

4-2 CONVERTING ASSEMBLY LANGUAGE INSTRUCTIONS TO MACHINE CODE

To convert an assembly language program to machine code, we must convert each assembly language instruction to its equivalent machine code instruction. The machine code instructions of the 8086 vary in the number of bytes used to encode them. Some instructions can be encoded with just 1 byte, others can be done in 2 bytes, and many require even more. The maximum number of bytes of an instruction is 6. Single-byte instructions generally specify a simpler operation with a register or a flag bit.

The machine code for instructions can be obtained by following the formats used in encoding the instructions of the 8086 microprocessor. Most multi-byte instructions use the general instruction format shown in Fig. 4-1.

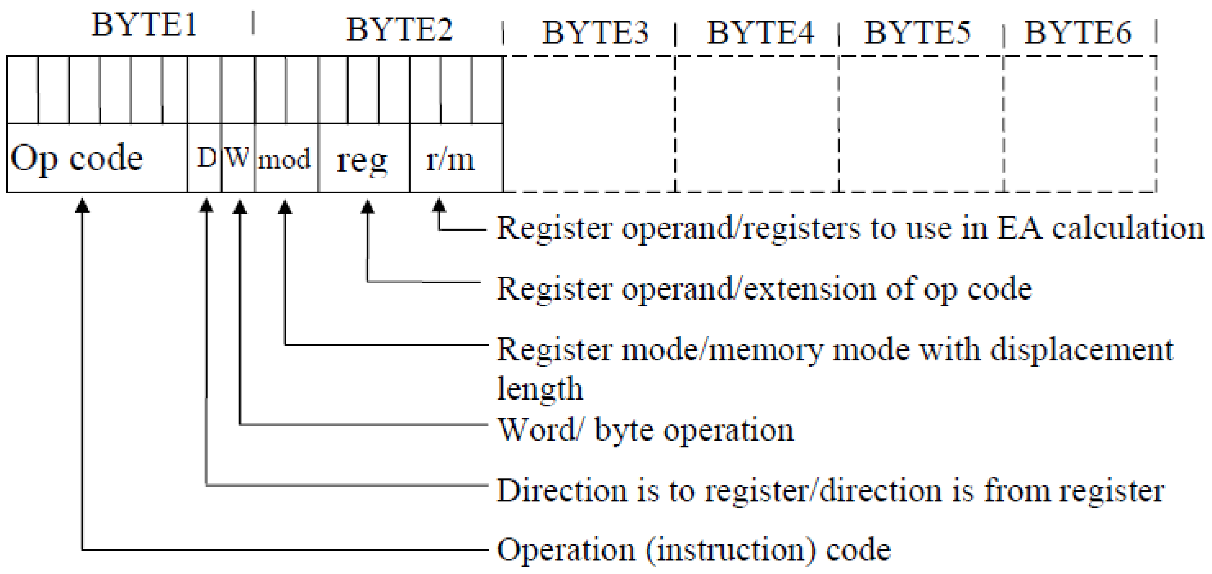

BYTE1
BYTE2
BYTE3
BYTE4
BYTE5
BYTE6
Op code
D
W
mod
reg
r/m
Register operand/registers to use in EA calculation
Register operand/extension of op code
Register mode/memory mode with displacement length
Word/ byte operation
Direction is to register/direction is from register
Operation (instruction) code

Fig 4-1 General instruction format.

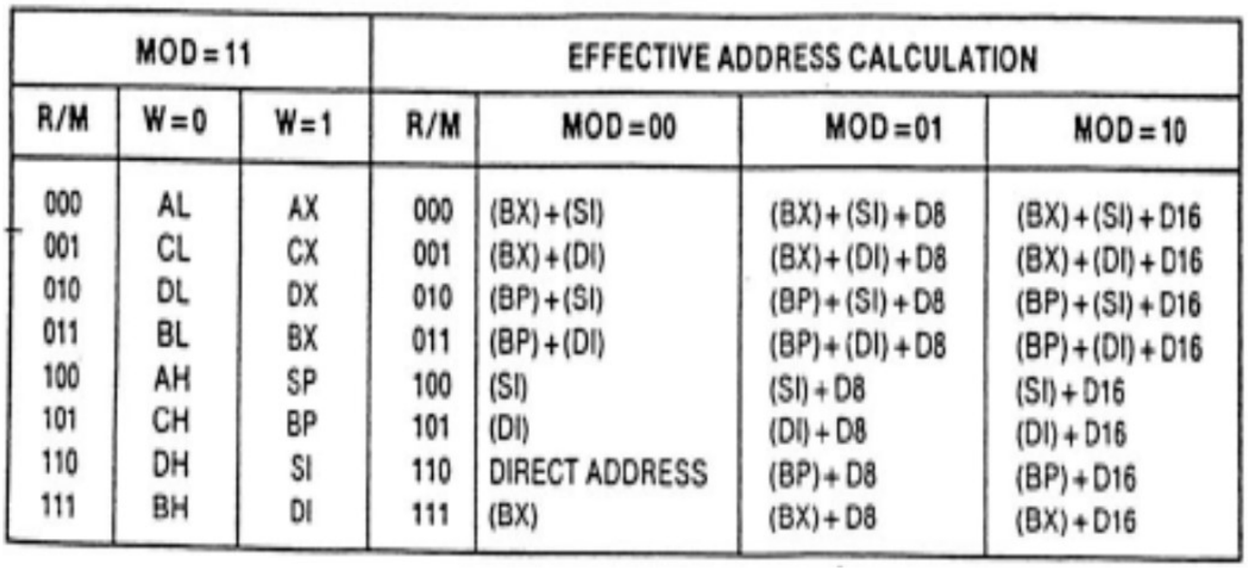
Looking at Fig. 4-1, we see that byte 1 contains three kinds of information:

1. Opcode field (6-bit): Specifies the operation, such as add, subtract, or move, that is to be performed.

2. Register direction bit (D bit): Tells whether the register operand specified by register in byte 2 is the source or destination operand. Logic 1 in this bit position indicates that the register operand is a destination operand, and logic 0 indicates that it is a source operand.
3. Data size bit (W bit): Specifies whether the operation will be performed on 8-bit or 16-bit data. Logic 0 selects 8 bits and 1 selects 16 bits as the data size.
The second byte in Fig. 4-1 has three fields:
The register filed (reg.) is 3-bit. It is used to identify the register for the first operand, which is the one that was defined as the source or destination by the D bit in byte 1. Table-2 shows the encoding for each of the registers in 8086 µp. Here we find that the 16-bit register AX and the 8-bit register AL are specified by the same binary code. Note that (W) bit in byte 1 determined whether AX or AL is used.
The 2-bit mod field and 3-bit r/m field together specify the second operand. Encoding for these two fields is shown in tables 4-3(a) and (b), respectively. Mod indicates whether the operand is in a register or memory. Note that in the case of a second operand in a register, the mod field is always 11. The r/m field, along with the W bit from byte1, selects the register.
Table of REG. Coding for W=0 and W=1:

REG	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Table of R/m field encoding



MOD=11			EFFECTIVE ADDRESS CALCULATION			
R/M	W=0	W=1	R/M	MOD=00	MOD=01	MOD=10
000	AL	AX	000	(BX)+(SI)	(BX)+(SI)+D8	(BX)+(SI)+D16
001	CL	CX	001	(BX)+(DI)	(BX)+(DI)+D8	(BX)+(DI)+D16
010	DL	DX	010	(BP)+(SI)	(BP)+(SI)+D8	(BP)+(SI)+D16
011	BL	BX	011	(BP)+(DI)	(BP)+(DI)+D8	(BP)+(DI)+D16
100	AH	SP	100	(SI)	(SI)+D8	(SI)+D16
101	CH	BP	101	(DI)	(DI)+D8	(DI)+D16
110	DH	SI	110	DIRECT ADDRESS	(BP)+D8	(BP)+D16
111	BH	DI	111	(BX)	(BX)+D8	(BX)+D16

Note: when r/m = 110 and MOD = 00 direct addressing occurred

Segment Override

Override byte:

001	segment code	110

Segment codes

Register	Code
ES	00
CS	01
SS	10
DS	11

Segment override prefix as shown use alternative segment once rather than the default segment.

Example: ES: MOV AX, [DI]200H

Physical address of data = ES*10H + (DI) + 200H

Rather than using the default segment DS.

Example 4-1: Encode the following instruction using the information in figure 4-1, tables 2, 3 and the op code for MOV is 100010.

MOV BL, AL

Solution:-

For byte 1:

The six most significant bits of first byte is 100010.

D =0 to specify that a register AL is the source operand.

W=0 to specify an 8-bit data operation.

$$\therefore byte\ 1 = (10001000)_2 = (88)_{16}$$

For byte 2: $(11000011)_2 = (C3)_{16}$

Thus, the hexadecimal machine code for instruction MOV BL,AL=88C3H

Example 4-2: Encode the following instruction using the information in figure 4-1, tables 2,3 and the op code for ADD is 000000.

ADD [BX][DI]+1234H, AX

Solution:-

For byte 1:

The six most significant bits of first byte is 000000.

D =0 to specify that a register AX is the source operand.

W=1 to specify a 16-bit data operation.

$\therefore Byte1 = (00000001)_2 = (01)_{16}$

For byte 2:

Mod = 10 (Memory mode with 16-bit displacement)
reg = 000 (from table 4-2 code of AX=000)
r/m = 011 (from table 4-2 (b))

$\therefore Byte2 = (10000011)_2 = (81)_{16}$

The displacement 1234_{16} is encoded in the next two bytes, with the Least Significant Byte (LSB) first. Therefore, the machine code is:

ADD [BX][DI]+1234H, AX=01813412

NOTES: The general form of figure (4-1) cannot be used to encode all instructions of 8086. We can note the following:

1- In some instructions, one or more additional single bit fields need to be added. Table-4 shows these 1-bit fields and there functions.

2- Instructions that involve a segment register need a 2-bit field to encode which register is to be affected. This field is called seg field. Table-5 shows the encoded code of segment register.

Example 4-3: Encode the following instruction:

MOV [BP][DI]+1234H, DS

Solution: Table -1 shows that this instruction is encoded as:

10001100 mod 0 seg r/m disp

$\therefore Byte1 = (10001100)_2 = (8C)_{16}$
For byte 2:

Mod = 10 (Memory mode with 16-bit displacement)
seg = 11 (from table-5)
r/m = 011 (from table-1 (b))

$\therefore Byte2 = (10011011)_2 = (9B)$
The machine code of MOV [BP][DI]+1234H, DS=8C9B3421

DATA TRANSFER AND STRING MANIPULATION GROUPS

5-1 Data transfer group:

THE STACK: The stack is implemented in the memory of 8086, and it is used for temporary storage.
Starting address of stack memory (top of the stack) obtained from the contents of the stack pointer (SP) and the stack segment (SS) (SS:SP). Figure 5-1 shows the stack region for SS=0400H and SP=A000H. Data transferred to and from the stack are **word-wide**, not byte-wide. Whenever a word of data is pushed onto the top of the stack, the high-order 8 bits are placed in the location addressed by SP-1. The low-order 8 bits are placed in the location addressed by SP-2. The SP is then decremented by 2.
Whenever data are popped from the stack, the low-order 8 bits are removed from the location addressed by SP. The high-order 8 bits are removed from the location addressed by SP+2. The SP is then incremented by 2.

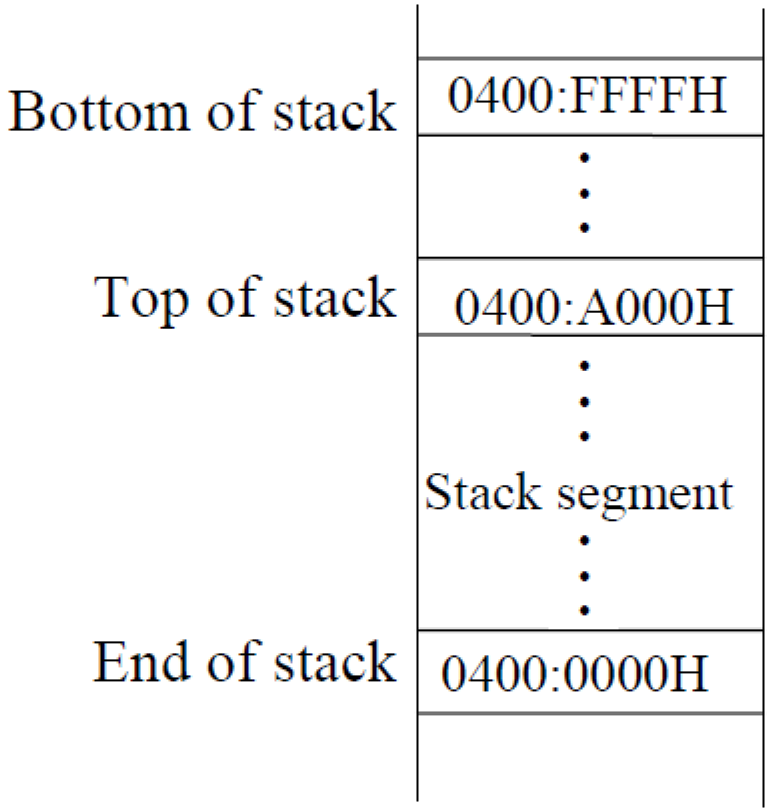


Figure 5-1 The stack region

The MOV instruction: The function of MOV instruction is to transfer a byte or word of data from a source location to a destination location. The general form of MOV instruction is as shown below:

Mnemonic	Meaning	Format	Operation	Flags affected
MOV	move	MOV D,S	(D) ← (S)	None

From table T1-a, we see that data can be moved between general purpose-registers, between a general purpose-register and a segment register, between a general purpose-register or segment register and memory, or between a memory location and the accumulator. Note that memory-to-memory transfers are note allowed.

• **PUSH/POP:** The PUSH and POP instructions are important instructions that store and retrieve data from the LIFO (Last In First Out) stack memory. The general forms of PUSH and POP instructions are as shown below:

Mnem.	meaning	Format	Operation	Flags affected
PUSH	Push word into stack	PUSH S	(SP) ← (SP) -2 ((SP)) ← (S)	None
POP	Pop word from stack	POP D	(D) ← ((SP)) (SP) ← (SP) +2	None

- **LEA, LDS, and LES (load-effective address) INSTRUCTIONS:**

These instructions load a segment and general purpose registers with an address directly from memory. The general forms of these instructions are as shown below:

Mnem.	meaning	Format	Operation	Flags
LEA	Load effective address	LEA reg16,EA	(reg16) ← EA	None
LDS	Load register and DS	LDS reg16,EA	(reg16)←[PA] (DS)← [PA+2]	None
LES	Load register and ES	LES reg16,EA	(reg16)←[PA] (ES)← [PA+2]	None

The LEA instruction is used to load a specified register with a 16-bit effective address (EA).

The LDA instruction is used to load a specified register with the contents of PA and PA+1 memory locations, and load DS with the contents of PA+2 and PA+3 memory locations.

The LES instruction is used to load a specified register with the contents of PA and PA+1 memory locations, and load ES with the contents of PA+2 and PA+3 memory locations.

EXAMPLE 5-1: Assuming that (BX) = 20H, DI = 1000H, DS = 1200H, and the following memory contents:

Memory	12200	12201	12202	12203	12204
Contents	11	AA	EE	FF	22

What result is produced in the destination operand by execution the following instructions?
a- LEA SI, [DI+BX+5] b-LDS SI, [200].

SOLUTION:

a- EA=1000+20+5=1025→(SI)=1025
b- PA=DS:EA=DS*10+EA=1200*10+200=12200

$\therefore (SI) = AA11H \ and \ (DS) = FFEEH$

• MISCELLANEOUS DATA TRANSFER INSTRUCTIONS:

XCHG: The XCHG (exchange) instruction exchanges the contents of a register with the contents of any other register or memory. The general form of this instruction is as shown below:

Destination	source
Accumulator	Reg16
Memory	Register
Register	Register
Register	Memory

Allowed operand

Mnem.	Meaning	Format	operation	Flags
XCHG	exchange	XCHG D,S	(S)↔(D)	None

XLAT: This instruction used to simplify implementation of the lookup table operation. The general form of this instruction is as shown below:

Mnem.	Meaning	Format	operation	Flags
XTAL	Translate	XTAL	(AL) ←((AL)+(BX)+(DS)*10	None

LAHF and SAHF: The LAHF and SAHF instructions are seldom used because they were designed as bridge instructions. These instructions allowed 8085 microprocessor software to be translated into 8086 software by a translation program.

IN and OUT: There are two different forms of IN and OUT instructions: the direct I/O instructions and variable I/O instructions. Either of these two types of instructions can be used to transfer a byte or a word of data. All data transfers take place between an I/O device and the MPU's accumulator register. The general form of this instruction is as shown below:

Mnem.	Meaning	Format	operation	Flags
IN	Input direct Input variable	IN Acc, port IN Acc, DX	(Acc) ← (port) (Acc) ← ((DX))	None
OUT	output direct Variable	OUT port, Acc OUT DX, Acc	(Acc) → (port) (Acc) → ((DX))	None