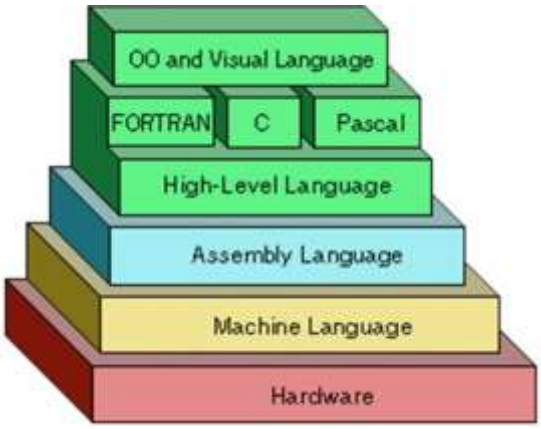
Software Systems-1

8086 INSTRUCTIONS ENCODING-PART01

Dr. Azam E. Al-Rawachy
COLLEGE OF COMPUTER SCIENCE AND MATHEMATICS

1 Introduction

Computer programming, consisting of machine language instruction, which is a low-level programming language used to directly control a computer's central processing unit (CPU). Each instruction causes the CPU to perform a very specific task, such as a load, a store, a jump, or an arithmetic logic unit (ALU) operation on one or more units of data in the CPU's registers or memory.



Machine code is a strictly numerical language which is intended to run as fast as possible, and it may be regarded as the lowest-level representation of a compiled or assembled computer program or as a primitive and hardware-dependent programming language. While it is possible to write programs directly in machine code, managing individual bits and calculating numerical addresses and constants manually is tedious and error-prone. For this reason, programs are very rarely written directly in machine code in modern contexts, but may be done for low level debugging, and assembly language. The majority of practical programs today are written in higher-level languages or assembly language. The source code is then translated to executable machine code by utilities such as compilers, assemblers, and linkers.

2 Classification of Instruction

Classification of Instruction depends on three main points as listed below:

- Addressing Mode
- Size { instructions in term of Byte size Or Word Size?
- Operation {Types of operation}
  - Data Transfer Instructiom
  - Arithmetic Instruction
  - Logical Instruction
  - Branching Instruction
  - Control Instructiom

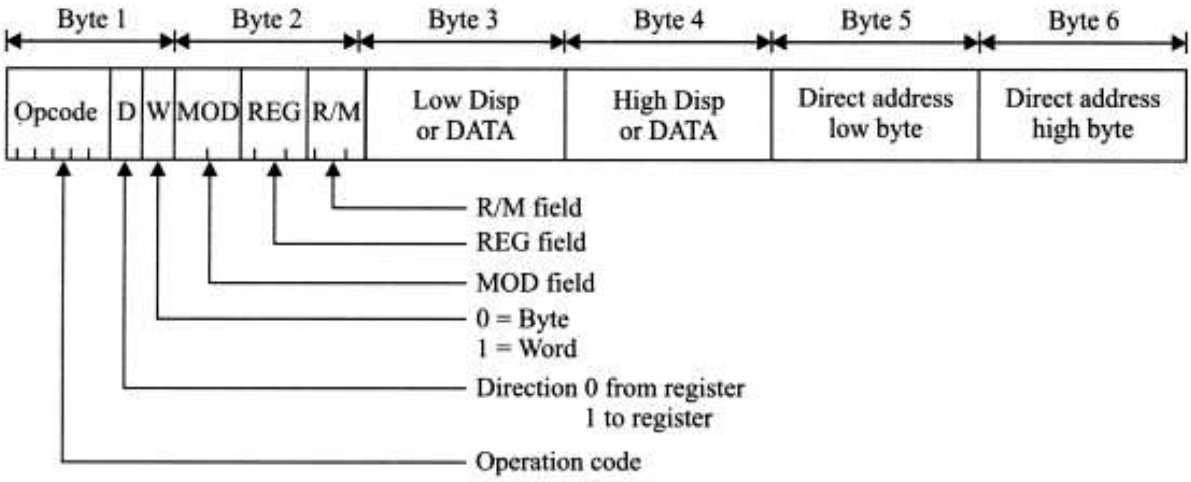
It is important to recap what we have learnt about addressing mode

Addressing Mode	Operand	Default Segment
Register	Reg.	None
Immediate	Data	None
Direct	[Offset]	DS
Register indirect	[BX] [SI] [DI]	DS DS DS
Based Relative	[BX] +disp [BP] +disp	DS SS
Index relative	[DI] +disp [SI] +disp	DS DS
Based Index Relative	[BX] [SI or DI] +disp [BP] [SI or DI] +disp	DS SS

3 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 bytes**. Single-byte instructions generally specify a simpler operation with a register or a flag bit.

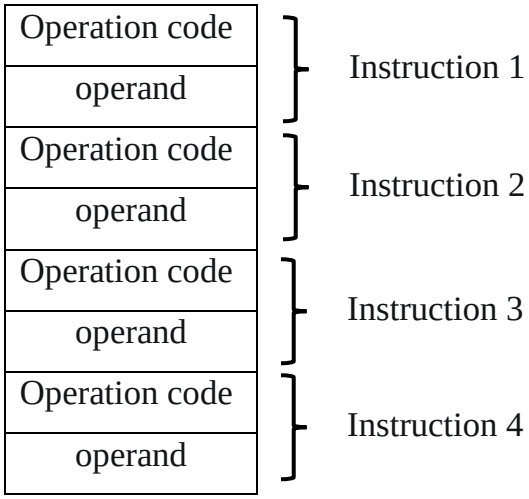
The machine code for instructions can be obtained by following the formats used in encoding the instructions of the 8086 microprocessor. Most multi-byte instructions use the general instruction format shown in the following figure.

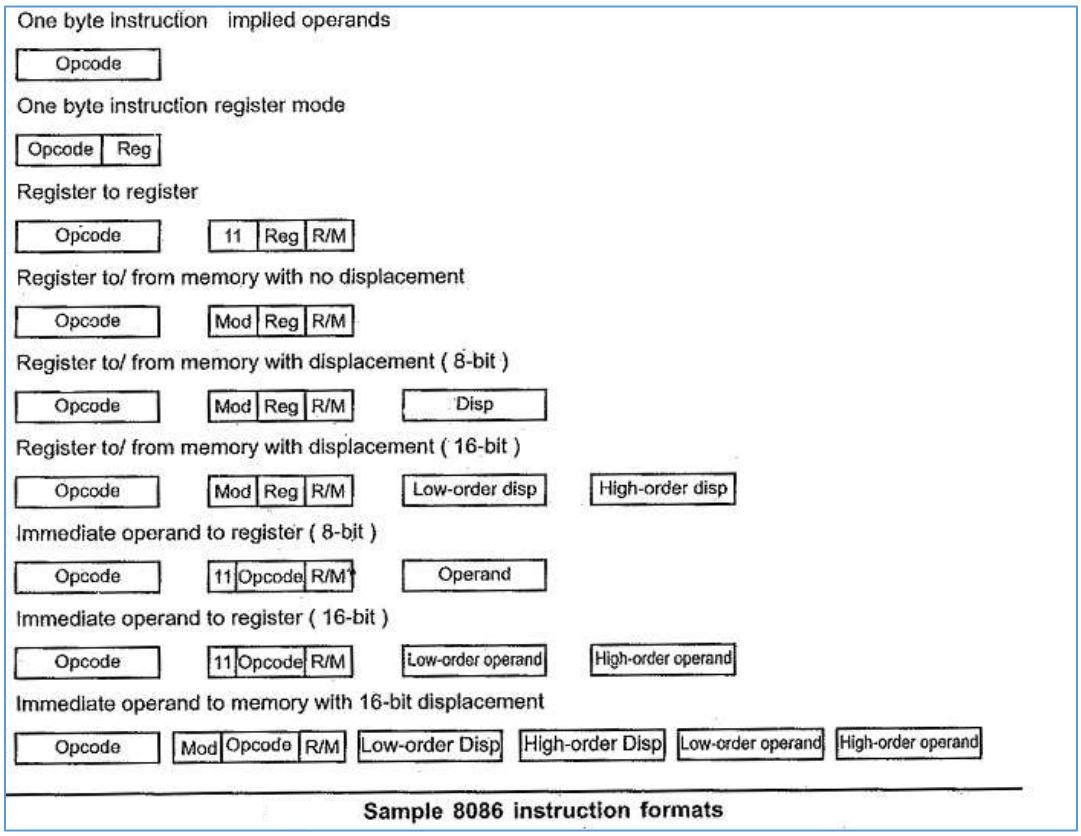


A few opcodes as listed here:

Name	Code	Decimal
Add	000000	(0)d
Sub	001010	(10)d
Comp	001110	(14)d
MOV	100010	(34)d

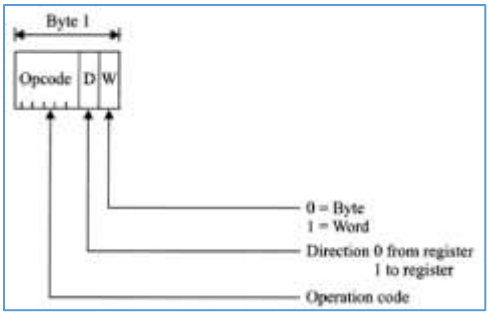
A machine code program consists of a sequence of Opcodes and Operands stored in the computer's memory (e.g. RAM)





Sample 8086 instruction formats

3.1 BYTE 1 Specification:



1. **OPCODE field (6-bits):**

   Specifies the operation to be performed such as MOV, ADD, SUB …. etc.

2. **Register direction bit (D-bit):**

   D = 1: the register operand specified by REG in byte 2 is a destination operand.

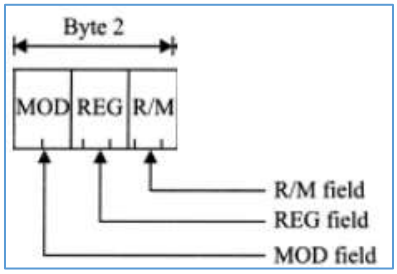
   D = 0: the register operand specified by REG in byte 2 is a source operand.

3. **Data size bit (W-bit):** Specifies whether the operation will be performed on 8-bit or 16-bit data.

   W = 1: 16-bit data size.

   W = 0: 8-bit data size.

3.2 BYTE 2 Specification: byte 2 has three fields:



1. **Mode (MOD) field (2-bits):** Indicates whether the operand is in register or memory.

MOD	Explanation
00	Memory Mode no displacement
01	Memory Mode 8-bit displacement
10	Memory Mode 16-bit displacement
11	Register Mode (no displacement)

2. **Register (REG) field (3-bit):** Identifies the register for the **first operand.**

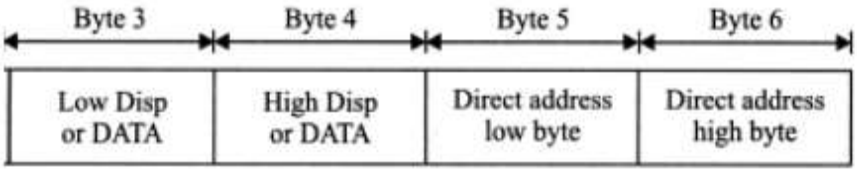
REG	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

3. **Register/Memory (R/M) field (3-bit):** Together with MOD field to specify the **second operand.**

MOD = 11			Effective Address (EA)Calculation			
Reg.	**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

3.3 Register to/from Memory with Displacement:

This type of instruction format contains 1 or 2 additional bytes for displacement along with 2-byte format of the register to/from memory without displacement.



Example1: Encode the following instruction:

MOV BL, AL [AL as the source operand]

Solution: -

7	2	1	0		7	6	5 4 3	2 1 0
Opcode		D	W		MOD		REG	R/M

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**

OR (BL as a destination)

Opcode	D	W	MOD	REG	R/M	Results	Note
100010	**1**	**0**	**11**	**011**	**000**	**8AD8h**	2nd operand 'AL' encoded in R/M field

Example2: Encode the following instruction using the information in the above figures and tables and the opcode 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.

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

byte	7	6	5	4	3	2	1	0	
1	opcode						d	w	Opcode byte
2	mod		reg			r/m			Addressing mode byte
3	[optional]								low disp, addr, or data
4	[optional]								high disp, addr, or data
5	[optional]								low data
6	[optional]								high data

For byte 2:

Mod = 10 (Memory mode with 16-bit displacement)

reg = 000

r/m = 001

$\therefore Byte2 = (10000001)_2 = (81)_{16}$

The displacement $\mathbf{1234_{16}}$ is encoded in the next two bytes, with the Least Significant Byte (LSB) first. Therefore, the machine code is: 01813412

Example3: MOV 1234 [BP], DX

Here we have specified DX using REG field, the D bit must be 0, indicating the DX is the source register. The REG field must be 010 to indicate DX register. The W bit must be **1** to indicate it is a word operation. 1234 [BP] is specified using MOD value of **10** and R/M value of **110** and a displacement of **1234H**. The 4-byte code for this instruction would be: 100010**0**1**10**0**1**0110**34**12= 89 96 34 12H.

Opcode	D	W	MOD	REG	R/M	LB displacement	HB displacement
100010	0	1	10	010	110	34H	12H

Example4: Encode the following instruction:

MOV DS: 2345 [BP], DX

Here we have to specify DX using REG field. The D bit **must** be 0, indicating that DX is the ***source register.*** The REG field **must** be 010 to indicate DX register. The W bit **must** be 1 to indicate it is a word operation. 2345 [BP] is specified with MOD=10 and R/M = 110 and displacement = 2345 H.

Whenever **BP** is used to generate the Effective Address (**EA**), the default segment would be **SS**. In this example, we want the segment register to be **DS**, we have to provide the **S**egment **O**verride **P**refix byte (**SOP byte**) to start with. The SOP byte is **001 xx 110**, where SR value is provided as shown on the right-hand side.

xx	Segment register
00	ES
01	CS
10	SS
11	DS

To specify DS register, the SOP byte would be 001**1 1**110 = 3E H. Thus the **5-byte** code for this instruction would be 3E 89 96 45 23 H.

SOP	Opcode	D	W	MOD	REG	R/M	LB displacement	HB displacement
3EH	100010	0	1	10	010	110	45	23

MOV = Move:				
Register/Memory to/from Register	1 0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w 1
Immediate to Register	1 0 1 1 w reg	data	data if w 1	
Memory to Accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high	
Accumulator to Memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/Memory to Segment Register ††	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment Register to Register/Memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		

Example5: Encode the following instruction:

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

Solution:

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; r/m = 011

$\therefore Byte2=(10011011)2=(9B)$

The machine code of MOV [BP][DI]+1234H, DS=**8C9B3412**

xx	Segment register
00	ES
01	CS
10	SS
11	DS

Example6: Encode the following instruction:

MOV AX, 1234H

MOV reg, imm	1011w reg data

1011w reg. data = 1011 **1000** 34 12= B**8** 34 12

Opcode:	1011	MOV (imm, reg)
W:	1	16 bit transfer
REG:	000	AX
Data Low:	34H	0011 0100
Data High:	12H	0001 0010

Example7: Encode the following instruction:

MOV [1236], AL ➔ 1010 0010 36 12➔ A23612H

Opcode:	1010001	MOV (acc, mem)
W:	0	8 bit transfer
Data Low:	36H	0011 0110
Data High:	12H	0001 0010

Memory addressing = **direct Memory**

MOV [BX], AX is not allowed

Accumulator = AX, AL

AH is **not** considered as an accumulator reg.

Try it by yourself!!

MOV AX, [1256] ➔ [ ]$_{16}$

Memory to Accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high
Accumulator to Memory	1 0 1 0 0 0 1 w	addr-low	addr-high
