12.2.5 I/O Addressing Mode in 8086

This addressing mode is basically used for IOs. Under IO mapping we have :

- Memory mapped I/O.
- I/O mapped I/O.

1) Memory mapped I/O

- If an I/O port is memory mapped, any of the memory operand addressing modes may be used to access the port.
- For example, a group of terminals can be accessed as an *array*. String instructions also can be used to transfer data to memory mapped I/O ports with in appropriate hardware interface.

2) I/O mapped I/O

- If I/Os are mapped in ***I/O map I/O***, then 8086 supports two different addressing modes :
  1) Direct port addressing. 2) Indirect port addressing.
- In direct port addressing, the port number is an 8 bit ***immediate*** operand. This allows fixed access to ports numbered 0-255.
- Indirect port addressing is similar to register indirect addressing of memory operands. The port number is taken forms register DX and can range from 0000H to FFFFH (0 to 65535 decimal). By previously adjusting the contents of register DX, one instruction can access any port in the IO space.
- A group of adjacent ports can be accessed using a simple software loop that adjusts the value in DX.

12.2.6 Implied Addressing Mode

The instructions which do not have operands come under implied addressing mode.

e.g. XLAT Translate or replace byte
CMA
STC set carry flag set direction flag
STD

These instructions do not have operands.

12.3 Instruction Encoding Format

- In 8086 we have variety in instructions, therefore all instructions will not be of same size.
- The instructions vary from 1 to 6 bytes in length.
- The obvious question is, ***what these bytes will contain and on what parameter the length of the instruction bytes is decided ?*** The length of instruction bytes is dependent upon addressing mode used by programmer i.e. immediate, register, register relative, based indexed, relative based indexed and so on.

Basically instruction bytes will contain information of :

1) OPCODE

52

2) **Addressing mode designations :**
   a) **2 byte Effective Address.**
   b) **1 or 2 byte displacement.**
   c) **1 or 2 byte immediate operand.**

To understand more clearly, let's observe Fig. 12.3.1 carefully.

1) Normally first byte is OPCODE byte.
2) 2nd byte normally specifies addressing mode. (Remember MOD and R/M). Sometime it may also contain OPCODE part.
3) After OPCODE and addressing mode bytes, we have following different cases :
   a) No additional bytes (Figs. 12.3.1(a), (b), (c) and (d)).
   b) A 2 byte EA (for direct addressing mode (Fig. 12.3.1(e)).
   c) A 1 or 2 byte immediate operand (Fig. 12.3.1(f)).
   d) A 1 or 2 byte displacement followed by 1 or 2 byte immediate operand (Fig. 12.3.1(g)).

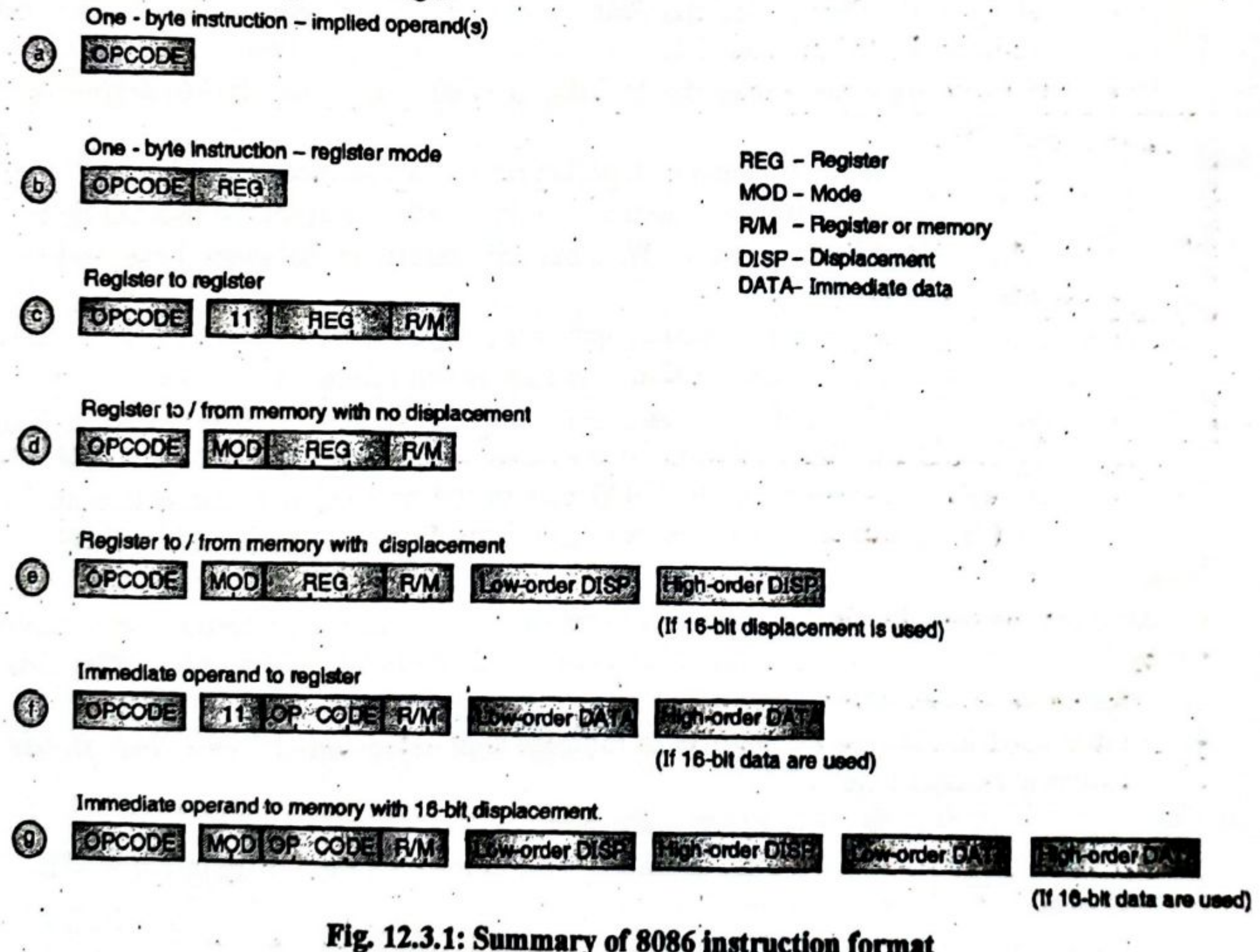
One - byte instruction – implied operand(s)
a
OPCODE
One - byte instruction – register mode
b
OPCODE REG
REG – Register
MOD – Mode
R/M – Register or memory
DISP – Displacement
DATA– Immediate data
Register to register
c
OPCODE 11 REG R/M
Register to / from memory with no displacement
d
OPCODE MOD REG R/M
Register to / from memory with displacement
e
OPCODE MOD REG R/M Low-order DISP High-order DISP
(If 16-bit displacement is used)
Immediate operand to register
f
OPCODE 11 OP CODE R/M Low-order DATA High-order DATA
(If 16-bit data are used)
Immediate operand to memory with 16-bit displacement.
g
OPCODE MOD OP CODE R/M Low-order DISP High-order DISP Low-order DATA High-order DATA
(If 16-bit data are used)

Fig. 12.3.1: Summary of 8086 instruction format

4) If a displacement or immediate operand is 2 bytes long, the low order byte always appears first, this is Intel standard (same was followed by 8085 also).

- To remember these formats, I will give you only a single format, from that we get these different formats refer Fig. 12.3.2.

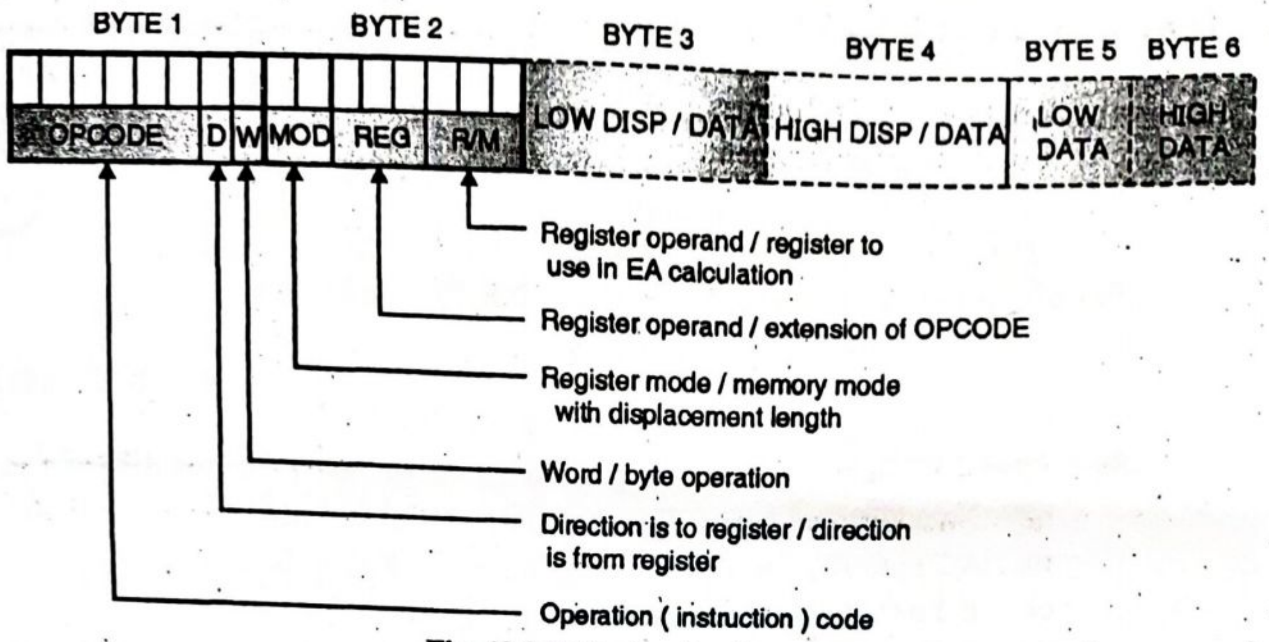
BYTE 1
BYTE 2
BYTE 3
BYTE 4
BYTE 5
BYTE 6
OPCODE
D
W
MOD
REG
R/M
LOW DISP / DATA
HIGH DISP / DATA
LOW DATA
HIGH DATA
Register operand / register to use in EA calculation
Register operand / extension of OPCODE
Register mode / memory mode with displacement length
Word / byte operation
Direction is to register / direction is from register
Operation (instruction) code

Fig. 12.3.2 : Instruction format

- As shown in Fig. 12.3.2, the first six bits of a multibyte instruction generally contains an opcode that identifies the basic instruction type i.e. ADD, XOR etc.
- The following bit, called the D field, generally specifies the ***direction*** of the operation.

  D = 1 means instruction source is specified in REG field.

  D = 0 means instruction destination is specified in REG field.

  The next following bit is *W*. This bit identifies between byte and word operation.

  W = 0 Instruction operates on byte data.

  = 1 Instruction operates on word data.
- Refer Fig. 12.3.2, if you observe in some case, in 2nd byte we have MOD, OPCODE and R/M, for some of the cases we have MOD, REG and R/M. First we will concentrate on OPCODE bits in 2nd byte of instruction format. This field is 3 bit wide. Under that we have three single bit fields, S, V and Z.

S bit

- An 8 bit, 2's complement number can be extended to a 16 bit 2's complement number by letting all of the bits in high order byte equal the MSB in low order byte. This is referred to as ***sign extension***.
- S bit is used in conjunction with W to indicate sign extension of ***Immediate fields*** in arithmetic instructions.

  S = 0 No sign extension

  = 1 Sign extended 8 bit immediate data to 16 bits if W = 1.

  Therefore for 8 bit operation : S = W = 0

  16 bit operation with a 16 bit immediate operand : S = 0, W = 1

  16 bit operation with a sign extended 8 bit immediate operand : S = W = 1

V bit

- Used by shift and rotate, to determine single and variable - bit shifts and rotate.

  V = 0 shift/rotate count is one

  = 1 shift/rotate count is specified in CL register.

Z bit

- This bit is used as a compare bit with zero flag in conditional repeat (REP) and loop instructions.

  Z = 0 Repeat/loop while zero flag is clear.
  = 1 Repeat/loop while zero flag is set.

Refer Table 12.3.1, it summarizes all 5 bits used in OPCODE field.

Table 12.3.1 : 5 bits used in OPCODE field

Field	Value	Function
S	0	No sign extension
	1	Sign extend 8-bit immediate data to 16 bits if W = 1
W	0	Instruction operates on byte data
	1	Instruction operates on word data
D	0	Instruction source is specified in REG field
	1	Instruction destination is specified in REG field.
V	0	Shift/rotate count is one
	1	Shift/rotate count is specified in CL register
Z	0	Repeat/loop while zero flag is clear
	1	Repeat/loop while zero flag is set

- Now concentrate on MOD, R/M and REG field in 2nd byte of instruction format. The second byte of the instruction usually identifies the instruction's operands.

MOD

- The mode (MOD) field indicates whether one of the operands is in memory or whether both operands are register. Table 12.3.2 shows MOD field encoding, this field is of size 2 bits.

Table 12.3.2 : MOD field ENCODING

CODE	EXPLANATION
0 0	Memory mode, no displacement follows *
0 1	Memory mode, 8 bit displacement follows
1 0	Memory mode, 16 bit displacement follows
1 1	Register mode (No displacement)

- * Except when R/M = 110, then 16 bit displacement follows. As seen MOD is basically concerned with displacement i.e. 8 bit or 16 bit or no displacement.

REG

- The Register (REG) field identifies a register that is one of the instruction operands. REG field depends upon W bit. Table 12.3.3 shows the selection of registers depending upon W bit.

Table 12.3.3 : REG (Register) field encoding

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

- When W = 0, all 8 bit registers are selected, whereas for W = 1 all 16 bit registers are selected. Thus in a number of instructions and mainly in immediate to memory variety, REG is used as an extension of the OPCODE to identify the type of operation i.e. 8 bit or 16 bit.
- **R/M : Register or memory :** This field is of 3 bits. The meaning of R/M bits changes depending upon mode (MOD) field.

At this stage we have general clear idea about these three fields, now, we will take some cases.

Case I : **Register to register transfer :**

- In this operation, data movement is within the register either 8 bit or 16 bit. As mentioned in this operation REG field identifies ONE of the instruction operands. ***What about another instruction operand ?*** It is specified by R/M, W and MOD bits. Refer Table 12.3.4.

Table 12.3.4 : R/M field encoding when MOD = 11 (binary)

MOD = 11 (binary)		
R/M	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

You will find that Table 12.3.3 matches with Table 12.3.4. Secondly when W = 0, you can select ONLY 8 bit source and destination operand. When W = 1, you can select ONLY 16 bit source and destination operand.

S3

- Following are the example instructions those are not correct or valid :

MOV AH, CX	;	Moving 16 bit to 8 bit (wrong)
MOV DX, BL	;	Moving 8 bit to 16 bit (wrong)

Thus any such combination with other register(s) is INVALID :

Case II : Memory MODE (8 bit/16 bit or No displacement).

- When MOD selects memory mode (MOD = 00 or 01 or 10), then data transfer is register to/from memory. In that case R/M field indicates how the effective address of the memory operand is to be calculated. Now refer Table 12.3.5, it depicts EA calculation.

Table 12.3.5 : R/M field encoding when MOD = 00/01/10

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

D8 = 8 bit displacement D16 = 16 bit displacement

REG field in this case, as usual identify the register that is one of the instruction operand.

12.4 Segment Override Prefix

Q. *What do you mean by segment override prefix?*

- Normally for each offset, segment is fixed. But using ***segment override prefix*** one can change segment registers.