

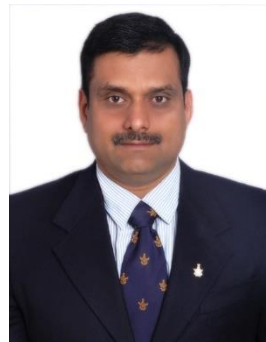


Microprocessors and Interfaces: 2021-22

Lecture 8

8086 Instructions Set : Part-2

By Dr. Sanjay Vidhyadharan



Data Transfer Instructions

Sanjay Vidhyadharan

Segment Override

- The **Segment Override Prefix** says that if we want to use some other segment register than the default segment for a particular code, then it is possible. It can simply be one by mentioning the segment that is to be used before the address location or the offset register containing that address. By doing so, the machine, i.e. the 8086 microprocessor, while calculating the effective address will consider the mentioned segment for calculating the effective address rather than opting for the default one.
- E.g. MOV AX, SS : [BX]
- Here, in this case, the Stack segment register is used as a prefix for the offset BX. So, instead of DS, which is the default segment register for BX, the SS will be used for finding the effective address location. Therefore, the effective address in the above-mentioned equation will be:
 - Effective address = $SS \times 10H + \text{content of BX register}$

Segment Override

Write the machine language equivalent code : **MOV DS: 2345 [BP], DX**

Solution:

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 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 segment override prefix byte (SOP byte) to start with. The SOP byte is 001 SR 110, where SR value is provided as per table shown below.

SR	Segment register
00	ES
01	CS
10	SS
11	DS

To specify DS register, the SOP byte would be 001 11 110 = 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 disp.	HD disp.
3EH	1000 10	0	1	10	010	110	45	23

Suppose we want to code MOV SS: 2345 (BP), DX. This generates only a 4 byte code, without SOP byte as SS is already the default segment register in this case.

Input / Output

- IN : Input byte or word
- OUT : Output byte or word

Sanjay Vidhyadharan

IN and OUT

- IN & OUT instructions perform I/O operations.
- Contents of AL, AX, or EAX are transferred only between I/O device and microprocessor.
 - an IN instruction transfers data from an external I/O device into AL, AX, or EAX
 - an OUT transfers data from AL, AX, or EAX to an external I/O device
- Only the 80386 and above contain EAX

IN

- IN transfers a byte or word from an input port to the AL register or AX register.
- IN instruction has two formats:
 - Fixed port: port number is specified directly in the instruction (port no: 0-255).
 - Variable port: port number is loaded into the DX register before IN instruction (port no : 0 – 65535).

IN acc , port no#

1110010w

port no #

IN acc , DX

1110110w

Machine code formats

OUT

- OUT transfers a byte or a word from AL register or AX register respectively, to an output port.
- OUT instruction has two formats:
 - Fixed port: port number is specified directly in the instruction (port no: 0-255).
 - Variable port: port number is loaded into the DX register before OUT instruction (port no : 0 – 65535).

OUT port no# , acc

1 1 1 0 0 1 0 w | port no #

OUT DX, acc

1 1 1 0 1 1 1 w

Machine code formats

IN/OUT

IN AL, 19 H

IN AX, 19 H

IN AL, DX

IN AX, DX

OUT 19H, AL

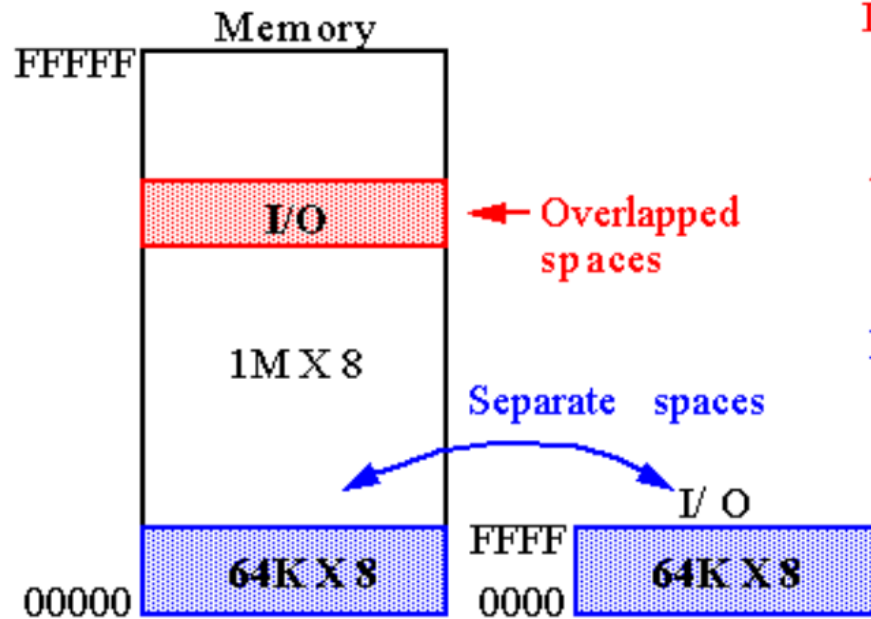
OUT 19H , AX

OUT DX, AL

OUT DX, AX

Isolated versus Memory-Mapped I/O

- In the Isolated scheme, IN, OUT, and IO/M are required.
- In the Memory-mapped scheme, any instruction that references memory can be used.



Disadvantage:

A portion of the memory space is used for I/O devices.

Advantage:

$\overline{\text{IORC}}$ and $\overline{\text{IOWC}}$ not required.
Any data transfer instruction.

Disadvantage:

Hardware using $\overline{\text{MIO}}$ and $\overline{\text{W/R}}$ needed to develop signals $\overline{\text{IORC}}$ and $\overline{\text{IOWC}}$.
Requires IN, OUT, INS and OUTS

Address Object data transfer

- LEA : Load effective address
- LDS : Load pointer using DS
- LES : Load pointer using ES
- LFS : Load pointer using FS
- LGS : Load pointer using GS
- LSS : Load pointer using SS

LEA (Load Effective Address)

- LEA transfers the offset of the source operand to a destination operand.
- The source operand must be a memory operand.
- The destination operand must be a 16-bit general purpose register.
- Does not effects flags.

LEA reg , mem

1 0 0 0 1 1 0 1

mod reg r/m

Machine code format

Example

LEA BX, [1234H]

MOV BX, [1234H]

Before execution BX = xxxh

Before execution BX = xxxh

After execution BX = ?

After execution BX = ?

LEA AX, [BP+SI+5] ; Compute address of value

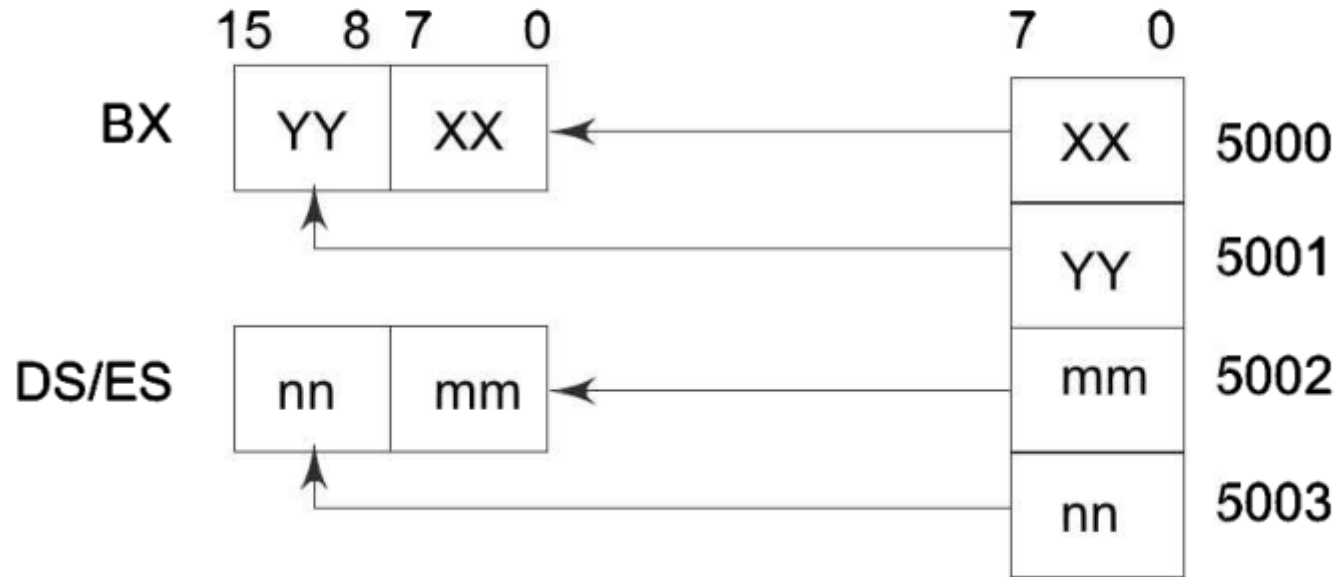
MOV AX, [BP+SI+5] ; Load value at that address

LDS (load pointer using DS)

- LDS transfers 32-bit pointer variable from source operand to destination operand and DS register.
- The source operand must be a memory operand.
- The destination operand may be any 16-bit general purpose register.
- The first word of the pointer variable is transferred into 16-bit general purpose register.
- The second word of the pointer variable transferred into DS.

LDS (load pointer using DS)

LDS BX, 5000H/LES BX, 5000H



Sc

Accessing array in data segment

```
A      DW      0000, 1000
```

1000:0000 = A[0]

04

1000:0001 = A[1]

03

1000:0002 = A[2]

02

1000:0003 = A[3]

01

1000:0004 = A[4]

00

```
MOV     SI , WORD PTR A
MOV     AX , WORD PTR A+2
MOV     DS , AX

LDS     SI , A
```


LES (load pointer using ES)

- LES transfers 32-bit pointer variable from source operand to destination operand and ES.
- The source operand must be a memory operand.
- The destination operand may be any 16-bit general purpose register.
- The first word of the pointer variable is transferred into 16-bit general purpose register.
- The second word of the pointer variable transferred into ES.

Accessing array in extra segment

```
B      DW      0000, 2000
```

2000:0000 = B[0]

04

2000:0001 = B[1]

03

2000:0002 = B[2]

02

2000:0003 = B[3]

01

2000:0004 = B[4]

00

```
MOV  DI , WORD PTR B  
MOV  AX , WORD PTR B + 2  
MOV  ES , AX  
  
LES  DI , B
```

Thank you