# 8086 Memory Addressing Modes

***Course Teacher:***

**Md. Obaidur Rahman, Ph.D.**
Professor
Department of Computer Science and Engineering (CSE)
Dhaka University of Engineering & Technology (DUET), Gazipur.

**Course ID:** CSE - 4503
**Course Title:** Microprocessors and Assembly Language
Department of Computer Science and Engineering (CSE),
Islamic University of Technology (IUT), Gazipur.

# Lecture References:

- **Book:**
  - *Microprocessors and Interfacing: Programming and Hardware, Chapter # 2,* **Author:** Douglas V. Hall
  - *Assembly Language Programming And Organization of the IBM PC, Chapter # 4,* **Author:** Ytha Yu, Charles Marut.

- **Lecture Materials:**
  - *IBM PC Organization,* CAP/IT221
  - M. A. Sattar, Microprocessor and Microcontroller

# Memory Model Segmentation

- **.model small**
  - Most widely used memory model.
  - The code must fit in 64k.
  - The data must fit in 64k.
- **.model medium**
  - The code can exceed 64k.
  - The data must fit in 64k.
- **.model compact**
  - The code must fit in 64k.
  - The data can exceed 64k.
- **.medium and .compact are opposites.**

# Basic Structure of Assembly Language Program

```
.MODEL SMALL/COMPACT/MEDIUM
.STACK 100H
.DATA
----
----
.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX
….
---
MAIN ENDP
    END MAIN
RET
```

# How to Define Data Segment

- **db:** define byte
- **dw:** define word (2 bytes)
- **equ:** define a numeric constant to a name

```
.data
data_byte DB 255 ; Define a data with 1-byte
data_word DW 512 ; Define a data with 2-bytes
data_string DB "Hello" ; define 5 consecutive bytes with ASCII valuees
max_int equ 65535  ; define maximum value of integer =65535
```

# How to Define Stack Segment

**.STACK size**

Where, size is an optional number that specifies the stack area size in bytes.

**.STACK 100h**

Here, it sets 00100h bytes for the stack. If size is omitted then 1 KB size is set aside for the stack.

# Addressing Mode and It's Categories

- The different ways in which a microprocessor can access data are referred to as its addressing modes.

- Addressing modes of 8086 Microprocessor are categorized as:
  - *Addressing Data*
  - *Addressing Program codes in memory*
  - *Addressing Stack in memory*
  - *Addressing I/O*
  - *Implied addressing*

# 1. Addressing Data

I. Immediate addressing
II. Direct addressing
III. Register [direct] addressing
IV. Register indirect addressing
V. Base-plus-index addressing
VI. Register relative addressing
VII. Base-relative-plus-index addressing

# 1. Addressing Data

## i. Immediate addressing

- Data is immediately given in the instruction

**MOV BL, 44**

**MOV BX, 256**

## ii. Register [Direct] addressing

- Data is in a register (here BX/BL register contains the data)

**MOV AX, BX**

**MOV AL, BL**

# 1. Addressing Data

## iii. Register [Indirect] addressing

- Register supplies the address of the required data

**MOV AX, [BX]** ;

Store the value of (DSx10h+BX) in AX

## iv. Direct addressing

- Data address is directly given in the instruction

**MOV AX, [1234H]** ; AX ← Value in (DSx10h+1234)

**MOV AX, DATA_VAR** ; DATA_VAR is a 16-bit variable

# 1. Addressing Data

## v. Base-plus-index addressing

- Base register is either BX or BP
- Index register is either DI or SI

**MOV DX, [BX+DI]** ;

Store the value of (DSx10h+BX+DI) in DX

## vi. Register relative addressing

- Register can be a base (BX, BP) or an index register (DI,SI)
- Mainly suitable to address array data

**MOV DX, [BX+2]** ;

Store the value of (DSx10h+BX+2) in DX

# 1. Addressing Data

## VII. Base-relative-plus-index addressing

- Suitable for array addressing

**MOV DX, [BX+DI+10]** ;

Store the value of (DSx10h+BX+DI+10) in DX

# 2. Addressing Program Codes in Memory

- Used with JMP and CALL instructions
- 3 distinct forms:
  - Direct
  - Indirect
  - Relative

# 2. Addressing Program Codes in Memory

- Address is directly given in the instruction

  **JMP 1000: 0000**

  or **JMP doagain** ; doagain is a **label** in code

  **CALL 1000:0000**

  or **CALL doagain** ; doagain is a **procedure** in code

- Often known as *far* jump or *far* call

# 2. Addressing Program Codes in Memory

- Address can be obtained from
  - **a)** any GP registers (AX,BX,CX,DX)
  - **b)** any relative registers ([BP],[BX],[DI],[SI])
  - **c)** any relative register with displacement

**JMP [DI]** ; Jump to memory location (CSx10h + DI)

**CALL [BX]** ; Call the content in memory location (CSx10h + BX)

# 3. Addressing Stack in Memory

- **PUSH** and **POP** instructions are used to move data to and from stack (in particular from stack segment).

  ***PUSH AX***

  ***PUSH BX***

  ***POP AX***

  ***POP BX***

- (Alternative for SWAP Operation: ***XCHG AX, BX***)
- **CALL** also uses the stack to hold the return address for procedure.

# 4. Addressing Input and Output Port

- IN and OUT instructions are used to address I/O ports
- Could be ***direct addressing***

  **IN AL, 05h** ; Here 05h is a input port number

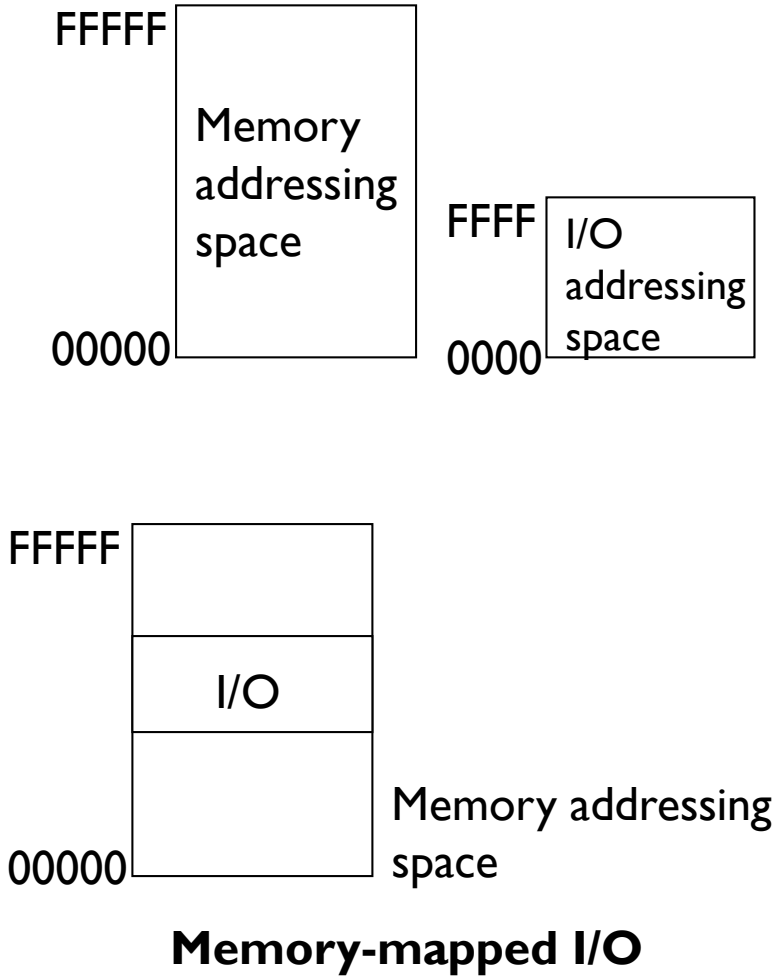
- or ***indirect addressing***

  **OUT DL, AL** ; DL contains the address of I/O port

- Only DL register can be used to point a I/O port

# 4. Addressing Input and Output Port

- **Mapped I/O**
  - A device is mapped to a memory location. Sending data to the particular location causes interaction with the device.
  - 16-bit I/O port addresses
  - Uses same memory read and write signals
  - Most processors use this I/O mapping




**Memory-mapped I/O**

# 5. Implied Addressing

- No explicit address is given with the instruction
- implied within the instruction itself
- Examples:

```
CLC  ; clear carry flag
HLT  ; halts the program
RET  ; return to DOS is equivalent of
       MOV AH, 4Ch
       INT 21h
```

# Thank You !!