Ref: 'Online course on EE-390', KFUPM

Extra Handout No 1

By Dr Sheikh Sharif Iqbal

80x86 Physical Addresses and Assembly Language Syntax

Objective:

- To discuss the process used by 80x86 microprocessor to generate the physical addresses (PA's) of main memory system

- To introduce the syntax of Assembly language statements

-To present a basic Assembly language data-transfer (or MOV) instruction of the for 80X86 system.

Slide 1: **Generating Physical Addresses of 80x86 main-memory:**

- The internal registers of 8086/8088 processors are 16-bit (4 hex digit) wide, whereas the 1-MByte main memory locations require 20 bit (5 hex digit) wide physical address (PA).

- Thus, Physical address are divided into two 16-bit parts, called offset and segment addresses, before storing it into CPU registers.

- The Segment address are made of the leftmost 4-hex digits of the Base address (as the rightmost digit are fixed to "0_H") related to for Code, Data, Extra and Stack segment memory locations and stored into 16-bit CS, DS, ES and SS registers, respectively.

Slide 2: **Generating Physical Addresses PA's (cont'd)**

- Thus, PA address is expressed either as a 5 hex-digit number or as combination of its segment-address (SA) & offset-address (OA) parts.

- The method used to relate these two expressions of PA can be written as: PA= 10 * SG + OA. Thus, if SG=2000_H and OA=$45AB_H$, then related PA can be expressed as, PA=2000:$45AB_H$=$245AB_H$

OA =		4	5	A	B
SG =	2	0	0	0	
PA =	2	4	5	A	B

- Note the rightmost hex digit of the base address is ignored as it should be set to 0H and only the leftmost hex digits are loaded into segment address (SG)

- Thus, the 20-bit physical address of the next executable instruction stored in the code segment memory location is given by;
$(PA)_{\text{within Code-Segment memory}} = (CS)_{\text{register value}} X10 + (IP)_{\text{register value}} = CS:IP$

- Note that the segment part of the PA is stored in CS register and offset part of the PA is stored in IP register

- Thus, the physical address of the data to be read/written in the stack segment memory location is given by;
$(PA)_{\text{within Stack-Segment memory}} = (SS)_{\text{register value}} X\ 10 + (SP)_{\text{register value}} = SS:SP$

- Note that the segment part of the PA is stored in SS register and offset part of the PA is stored in SP register

Slide 3: **Generating Physical Addresses PA's (cont'd)**

- The physical address of source data to be read (into the CPU registers) from the Data segment memory location are given by;
$(PA)_{\text{withih Data-Segment memory}} = (DS)_{\text{register value}} \text{ X } 10 + (SI)_{\text{register value}} = DS:SI$

-The physical address of destination data to be written (from the CPU registers) into the Data segment memory location are given by;
$(PA)_{\text{within Data-Segment memory}} = (DS)_{\text{register value}} \text{ X } 10 + (DI)_{\text{register value}} = DS:DI$

- Note that the segment part of the PA is stored in DS register and offset part of the PA is stored in SI or DI registers depending on accessing source or destination memory locations, respectively.

- Similarly, PA's for accessed extra segment memory locations are;
$(PA)_{\text{within Extra-Segment memory}} = (ES)_{\text{register value}} \text{ X10+ } (SI)_{\text{register value}} = ES:SI$
$(PA)_{\text{within Extra-Segment memory}} = (ES)_{\text{register value}} \text{ X10+} (DI)_{\text{register value}} = ES:DI$

- Note that the segment part of the PA is stored in ES register and offset part of the PA is stored in SI or DI registers depending on accessing source or destination memory locations, respectively.

Slide 4: **Generating Physical Addresses PA's (cont'd):**

-Thus, 20 Bit Physical Address pointing to a Memory storage **=** Related Segment Register value * **10 +** Offset Register value

- Typical segment and offset register combinations are:

$(PA)_{\text{in Code-Segment memory}}$ = **CS***10 + **IP = CS:IP**;
*Remember: **CS:IP** expression is also called the Logical Address*

$(PA)_{\text{in Data-Segment}}$ = **DS***10 + **SI** = **DS:SI** and **PA**= **DS***10 **+ DI** = **DS:DI**

$(PA)_{\text{in Extra-Seg. memory}}$ = **ES***10 + **SI** = **ES:SI** and **PA** = **ES***10 + **DI** = **ES:DI**

$(PA)_{\text{in Stack-Segment}}$ = **SS***10 + **S**P = **SS:SP**

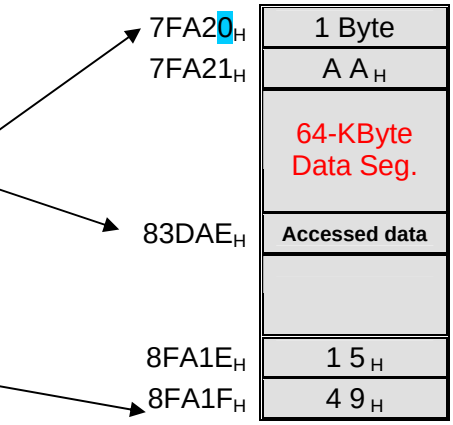
- Remember that **lowest nibble** (*or lowest hex digit*) of **base address** (*lowest physical address of a segment*) should be "$\mathbf{0}_H$"➔ $(PA)_{\text{Seg Base}} = 1234\mathbf{0}_H$-

- A demonstrative example of this process is given in the next slide.

Slide 5: **Generating Physical Addresses PA's (cont'd):**

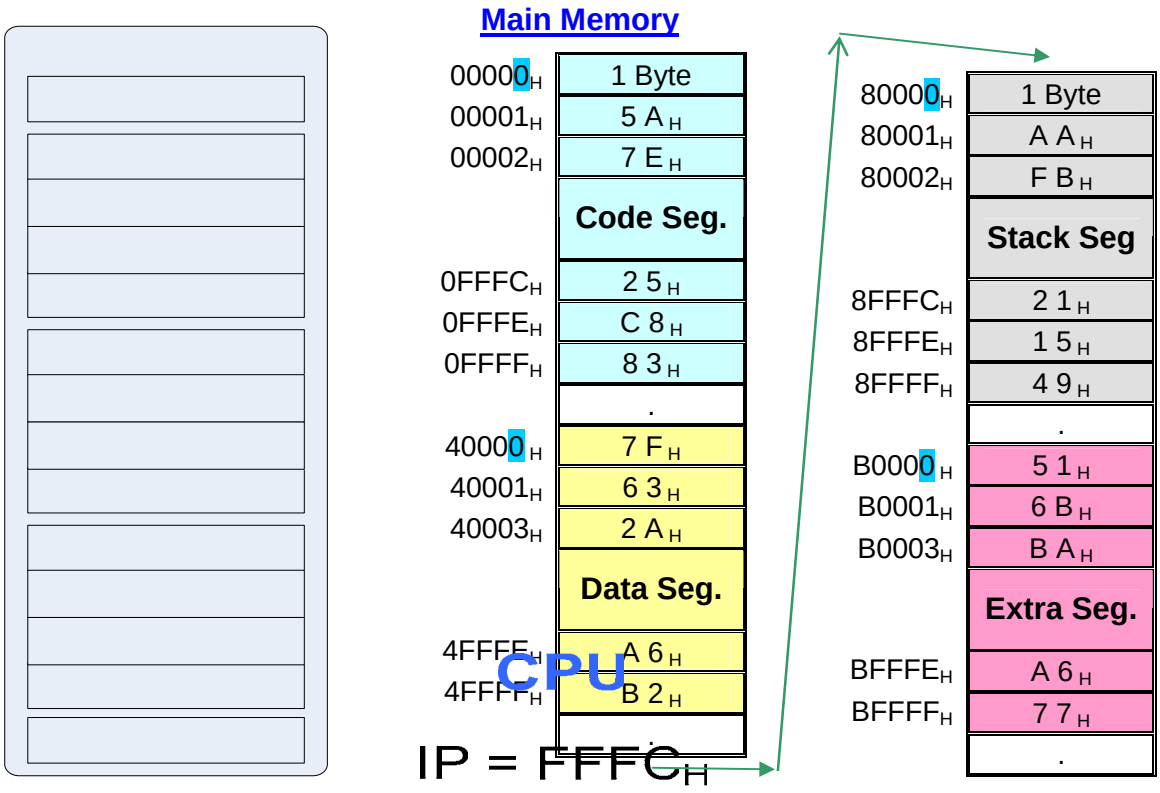
Thus: for DS=7FA2$_H$ and the offset is 438E$_H$;

- The calculated physical address (PA) of memory location is: 7FA2x10$_H$ + 438E$_H$ = 83DAE$_H$
- The calculate the lower range of the data segment is: 7FA2 x10$_H$ + 0000$_H$ = 7FA20$_H$
- The calculate the upper range of the data segment is: 7FA2 x10$_H$ + FFFF$_H$ = 8FA1F$_H$
- The Logical address of the memory location is : **7FA2 : 438E** $_H$



Slide 6: **Example-1 on Physical Addresses Generation:**

Q. For the given values of the CPU registers, determine the PA's of the (a) next code to be executed, (b) source data stored in Data-segment, (c) Stack-segment memory location to be accessed, (d) Extra-segment memory location where destination data can be stored



IP = FFFC$_H$

CS = 0000$_H$

DS = 4000$_H$

SS = 8000$_H$

ES = B000$_H$

Accumulator Reg., (AX)

Base Register, (BX)

Counter Register, (CX)

Data Register, (DX)

SP = 0002$_H$

Slide 7: **Example-1 on Physical Addresses Generation (cont'd):**

Animate

- The **solution** of the previous example problem is as follows:

(a) Instruction codes reside in the Code-segment memory location, so for the given CPU register values, the PA that points to the Code-segment is; **PA** = CS:IP = CSX10+IP ➔ 0000:FFFC$_H$ = 0FFFC$_H$.

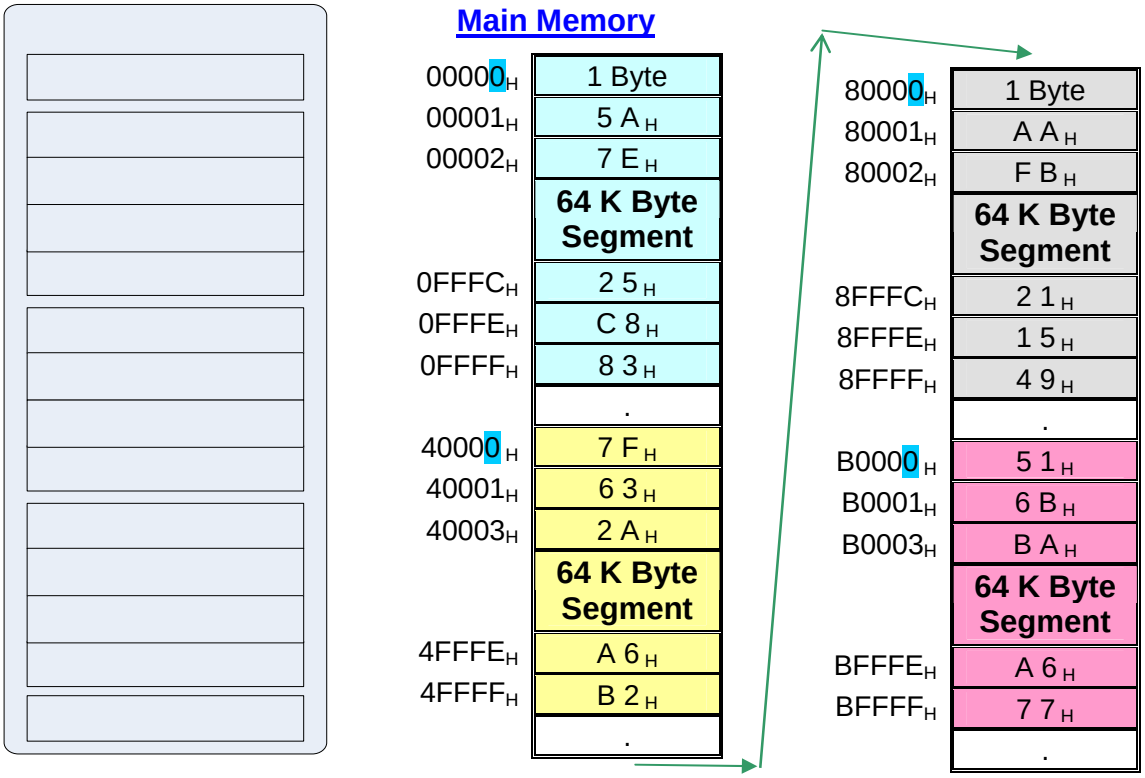
(b) Similarly, the PA that points to the source data stored in Data-segment is; **PA** = DS:SI = DSX10+SI ➔ 4000:FFFE$_H$ = 4FFFE$_H$.

(c) The word memory location to be accessed in the Stacked-segment is; **PA** = SS:SP = SSX10+SP ➔ $8000:0002_H = 80002_H$.

(d) And the PA that points to the destination data stored in Extra Data-segment is; **PA** = ES:DI = ESX10+DI ➔ $8000:0001_H = 80001_H$.

Slide 8: **Example-2 on Physical Addresses Generation:**

Q. If Memory addresses $B0000_H$ to $BFFFF_H$ is selected to be the new Code-segment, then to point to point to a location with PA=$B1234_H$, what values should be loaded into the related CPU registers.



Solution: This instruction involves the Code-segment and will only affect the CS and IP registers of the CPU. Since the Base address of the Code-segment is specified to be '$B0000_H$', the segment-address (SA) part of the PA that will loaded into the CS register will be '$B000_H$' and the offset-address (OA) part of the PA that will be loaded into the IP register will be '1234_H'.

CPU

Instruction Pointer, (IP)

Code Segment, (CS)

Data Segment, (DS)

Stack Segment, (SS)

Slide 10: **Introduction to Assembly Language Programming:**

-The native language of 80x86 processors is machine language and programs written in this language are called machine-codes (in binary).

- In early computers, instructions were manually converted into binary machine codes and entered into the computer using panel switches.

- But the idea of writing a "program" that will efficiently translate the instructions into binary machine-codes (later called Assemblers and Compilers) were a breakthrough in the field of programming.

- Let us now learn how to program 80x86 processors using a low-level language, called Assembly language, where "Assemblers" (TASM or MASM) are used to convert its instructions to machine codes.

- Assembly language program is often used for real time application due to its efficient machine-codes, which occupies less memory and executes faster compared to that of higher level languages.

***Hyperlink data on Programming**: Program is a Sequence of Commands or Instructions that defines the operation to be carried out by the microcomputer. Whereas a Software consist of a Wide verity of programs that can be run by the micro-computer, such as, Languages, Operating systems, Application programs, diagnostics etc.*

***Hyperlink data on Low level language**: When a program written using a low-level language is converted or translated into machine language, it generates an efficient machine-code that occupies less memory and executes faster.*

Hyperlink data on TASM or MASM: "Microsoft Assembler (MASM) and Turbo Assembler (TASM) are two popular Assemblers or translators used to convert Assembly language programs into machine-language.

***Hyperlink data* on** real time *: Real time means that the task required by the application must be completed before any other input to the program that will alter its operation can occur. Such as, program that controls the operation of a floppy disk drive.*

***Hyperlink data on High level language**: When a program is written using a user friendly high-level language (C, Pascal etc) and is converted or translated into machine code by a complier, it generates a relatively less efficient machine-*

code that occupies more memory spaces and consequently slows the execution process.

Slide 11: **Assembly Language Syntax for 80x86 processors:**

- Assembly language programs, also called source-codes, are often described using alphanumeric symbols rather than binary codes.

- To be executed by 80x86 processors, the source code needs to be converted into machine-code using “Assemblers” (MASM or TASM)

- Syntax of an assembly language statement consists of four fields:

Label**:** **OpCode** **Oprand** ; Comment

where, “Opcode” and “Operand” fields often consist the “Instruction”.

- The “Label” is an optional field and used to identify the address of the instruction (*often used in jumping*). The maximum length of a label differs between assemblers (4 to 32 characters), whereas all labels begins with a valid character (A..Z) and are suffixed by a colon

---

Slide 12: **Assembly Language Syntax for 80x86 (cont'd):**

- "Opcode" or operation code is a compulsory part of the instruction that specifies the operations to be performed by the microprocessor. Opcodes consists of 3-5 letter abbreviations, called mnemonics and often require additional data on which operations will be performed.

Hyperlink data mnemonics: A mnemonic is an abbreviation which represents the actual operation code of the instruction. Mnemonics are used because they are more meaningful than hex or binary values, reduce the chances of making an error and are easier to remember than bit values.

- The "Operand" field consists of source/destination data that may be required/generated by the "opcode" to complete the operation. Often different addressing modes are used to specify these operands.

- The "Comment" is an optional field used by the programmer to explain how the coded program works. Remember, the "Assembler" doesn't generate the related machine-codes for comments.

- The 80x86 microprocessor supports 117 basic instructions, each of which can be expressed using one assembly language statement. Next slide discusses the most common data transfer instruction.

- The remaining 116 instructions, supported by 80x86 assembly language programs, will be explained in later modules, as needed.

Slide 13: **Assembly language Data Transfer Instruction:**

- In this slide, a basic assembly language instruction used to copy byte/word data from one storage location to another is introduced

Mnemonic	Meaning	Format	Operation	Flags affected
MOV	Move	**MOV D,S**	(S) ➔(D)	None

- Thus, the assembly language statement for this instruction is;

MOV *Destination-operand(*D*)* **,** *Source-operand(S)*

- Note optional "Label" and "Comment" fields are not included
- the "OpCode" or operation to be performed is specified by a 3 letter abbreviation or mnemonic, "MOV"
- the "Operand" fields consisted of a source data location and destination or resulted data location separated by a comma.

Such as; **MOV AL, BL** ; **MOV CH,2A$_H$** and **MOV AL, [DS:1122$_H$]**

- In example 1, BL is the source location and AL is the destination location. Thus, a byte-data stored in BL register of the CPU will be copied into the AL register.
- In example 2, CH register is the destination location and a Hexadecimal data of 2A will be copied into CH register.
- In example 3, the memory location with PA=DS:1122H is the source location and AL register is the destination location. Thus, this instruction copies byte data contents of the memory location DS:1122H into AL register.

Slide 14: **Assembly language Data Transfer Instruction (cont'd)**

- The list of important operand combinations are shown below:

Allowed or accepted operand combination of MOV instruction	
Destination operand	**Source operand**
Any Memory location	AX or AL (accumulator)
CPU internal Registers	**CPU internal Registers**
AX or AL (accumulator)	**Any Memory location**
CPU internal Registers	Any Memory location
Any Memory location	CPU internal Registers
CPU internal Registers	**Any Constant numbers**
Any Memory location	Any Constant numbers

➔ **As in example 1 of previous slide**

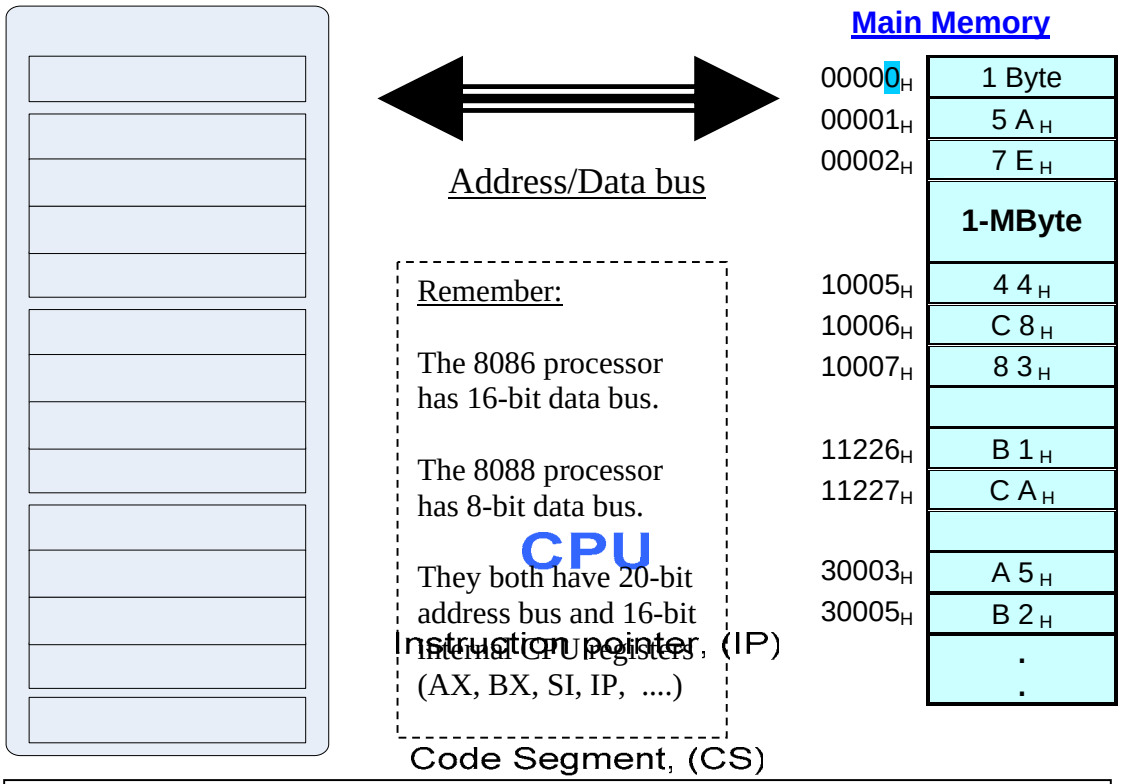
➔ **As in example 3 of previous slide**

➔ **As in example 2 of previous slide**

Not-allowed or unaccepted operand combination of MOV instruction (only a few are presented)	
Destination operand	**Source operand**
Any Memory location	Any Memory location
Instruction pointer (IP)	Contents of Registers, Memory or Constant
8-bit register content	16-bit register content
Segment register	Segment register
Constant data	Register/memory location

Slide 15: **Assembly language Data Transfer Instruction (cont'd):**

Example: For the given values, executing each instruction and find the content of the destination- operands.



(a) MOV CL,AH ➔ CX =_____ (b) MOV AL,[SI] ➔ AX =_____

(c) MOV [ES:DI] ➔ DH =_____ (d) MOV AX,SP ➔ AX =_____

(e) MOV DH,[SP] ➔ DX =_____

Solutions: (a)AA34$_H$, (b)3444$_H$, (c) content of PA=30003$_H$=90$_H$, (d)1006$_H$, (e)B18B$_H$

$DS=1000_H$

$SS=1122_H$

$ES=3000_H$

$AX=3456_H$

$BX=1239_H$

$CX=AA89_H$

$DX=908B_H$

$SP=1006_H$

Base pointer, (BP)

$SI=0005_H$