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

Extra Handout No 3

By Dr Sheikh Sharif Iqbal

Software model of 8086 and 8086 systems

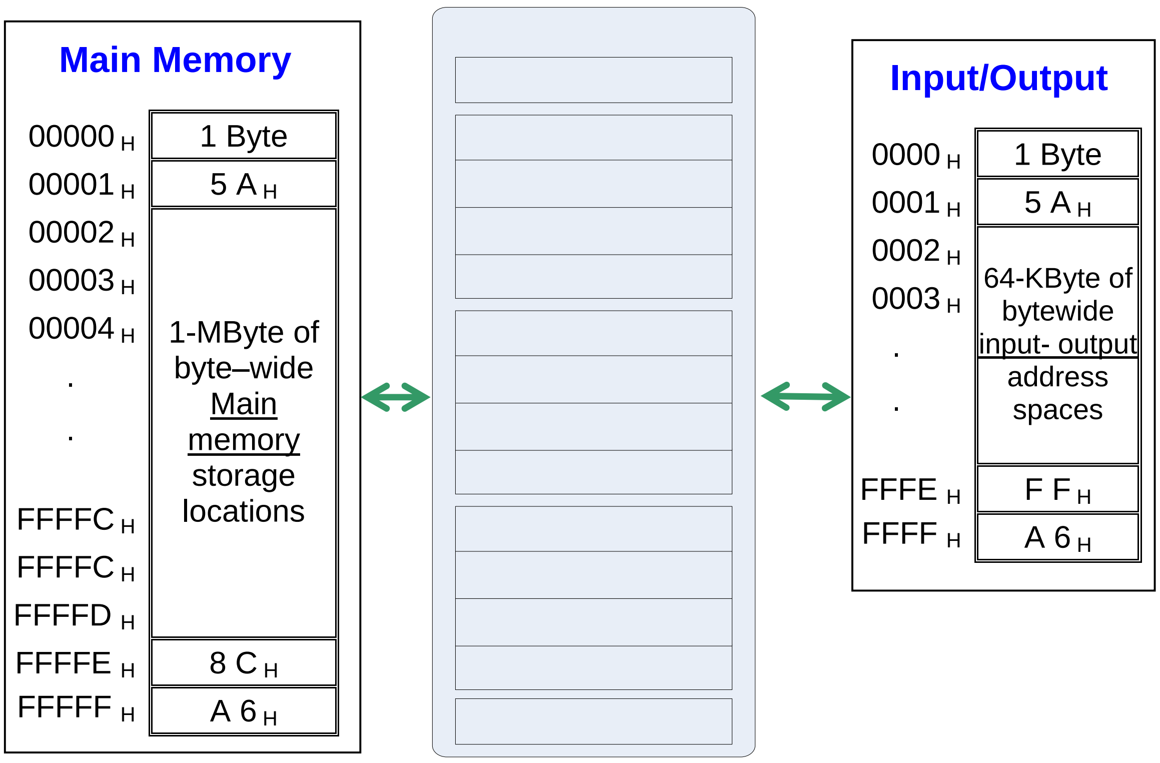
Objective:

- To present the software model of 80x86 microprocessor

- To introduce the Main-memory and input-output address spaces supported by 8086/8088 system

- To discuss the internal registers of 8086/8088 microprocessor

Slide 1: **Software Model of 80x86 Sysmtes**



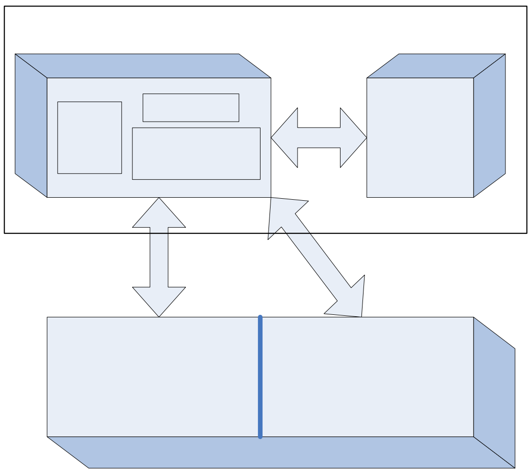
- Intel's recent microprocessors are fundamentally based on 80x86. So, the software model of 8086/8088 system is shown here to demonstrate its behavior from a software/programming point of view.

- It is clear from the figure that in this model, we have access to 14 16-bit CPU registers, 1 Mega or million of byte-wide main memory storages and 64 thousand byte wide input output address spaces for peripheral devices.

- Thus, this model focuses more on the control-signals and registers of the microprocessor that can be accessed by the programmer from software prospective. This allows the programmer to know; how external memory and input/output peripherals are organized, how information is arranged in registers, and how CPU registers interacts with memory and input/output devices.

Slide 2: **Main-memory of 80x86 systems (Software model)**

- Computer system has several forms of memory storage, each with individual purposes.
- Detail discussion on all types of storage media are presented in later modules. This section will only discuss the organization of Main memory.



- Main memory (RAM) stores instructions/data related to the program, which is being executed.

- 8086 and 8088 systems can support 1-MByte of main-memory spaces (for temporary storage) and 64 KByte of input-output spaces.

- Since, 1 Mega = 1048575_D = $FFFFF_H$, converting the hex number into binary reaffirms the need for a 20-bit address bus (in binary).

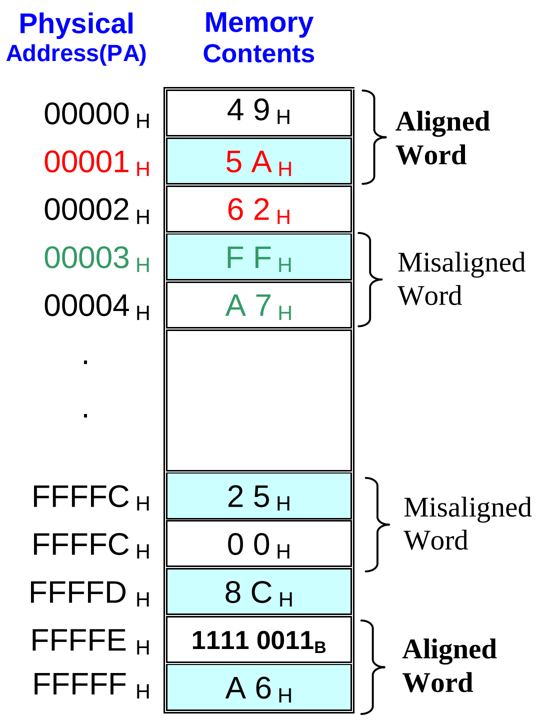
- Thus, the physical address, pointing to the byte-wide storage locations in the main memory, ranges from 00000_H to $FFFFF_H$.

Slide 3: **<u>Software Organization of the 80x86 Main-memory</u>**

- The Main memory of 1 Mega byte-wide storage locations of an 80x86 computer system are shown here.

- Note, PA=00001_H stores a data Byte of "$5A_H$" and Word of "$625A_H$", where $5A_H$➔L.S.Byte & 62_H➔M.S.Byte and the Double-word of "$A7FF625A_H$"

- Note for a stored word data at physical address of 00001, the lower addressed byte is the Lease Significant byte of the word and the higher addressed byte is the most significant byte of the word. Similar rule also applies for the stored double-word



Physical Address(PA)	Memory Contents
00000_H	49_H
00001_H	$5A_H$
00002_H	62_H
00003_H	FF_H
00004_H	$A7_H$
.	
.	
$FFFFC_H$	25_H
$FFFFC_H$	00_H
$FFFFD_H$	$8C_H$
$FFFFE_H$	$1111\ 0011_B$
$FFFFF_H$	$A6_H$

- To permit efficient use of memory, word data's are stored in 'even' and 'odd' physical address (PA) boundaries.

- Words stored at even PA's (0_H, 2_H....$FFFFE_H$) are called aligned words and odd PA's (1_H, 3_H .... $FFFFF_H$), are called misaligned words. Note that the aligned word of "$5A49_H$" is stored in PA=00000_H *or* 0_H

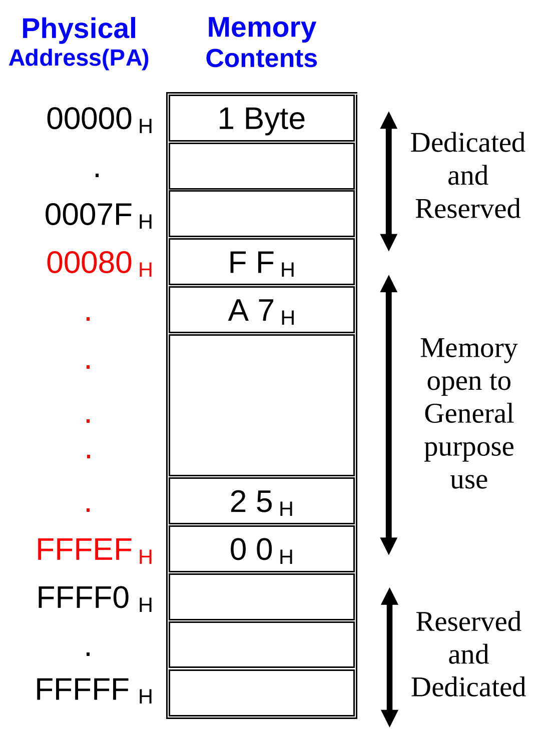
- Note that two aligned words are shown in figure with even physical address of 00000_H and $FFFFE_H$. Also two misaligned words shown in the figure are stored at odd physical address of 00003_H and $FFFFC_H$.

Slide 4: **Mapping of 80x86 systems Main-memory**

- The 1 MByte main-memory of an 80x86 systems is divided into three parts: (a) General-use, (b) Dedicated and (c) Reserved storage locations

- The general-use part ranges from P.A. of 80_H to $FFFEF_H$ and is open to the user for storing programs.

- Dedicated memories ranges from P.A. of “0_H to 13_H”and “$FFFF0_H$ to $FFFFB_H$” and are used to process system interrupts, reset-functions and exceptions.



- Reserved memories ranges from P.A. of “14_H to $7F_H$”and “$FFFFB_H$ to $FFFFF_H$” and are used to process user defined interrupts for future hardware and software products.

Slide 5: **Segmentation of 80x86 systems Main-memory**

- Although 80x86 supported a large general-use memory, the absence of addressing mechanism required it to be divided into 16_D, 64KB segments

- Only four of these 64KB segments remain active at a time and can be accessed by the 80x86 processor.

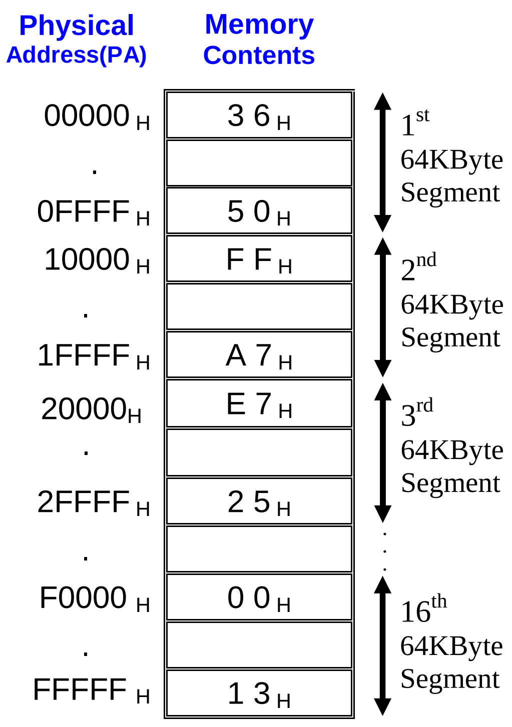
- These active segments are named:

(a)Code Segment: Stores instruction codes

(b)Stack Segment: Temporary information

(c) Data Segment: Stores program data

(4) Extra Segment: Also for data storage

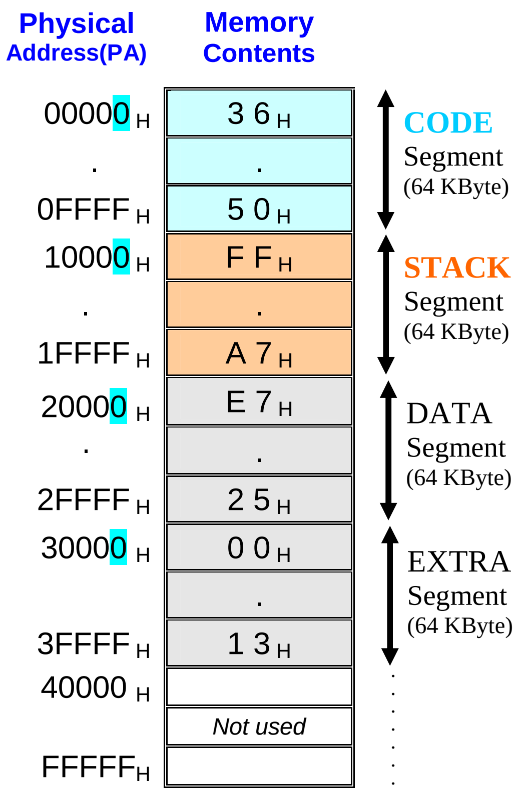


- Thus, Four segment give a maximum of 256 Kilo-Byte of active memory storage, of which 64 Kilo-Byte are for storing program instruction codes, 64 Kilo-Byte are for stack storage and 128 Kilo-Byte are for storing program data.

Slide 6: **Segmentation of 80x86 systems Main-memory (cont'd)**

- The 64 KByte (65535_H=$FFFF_H$) segments are allowed to be contiguous, adjacent, disjointed and partially or totally overlapping to each other.

- This point of overlapping segments will be clear in the lecture about DEBUG or Turbo Debugger.



- The only restriction in allocating segments are to have a '0_H' number as the least-significant hex-digit of the Base address (*the P.A pointing to the beginning of any segment*)

- Thus, Base address are the starting address of any 64 KByte segment.

- Physical address are also expressed interims of its segment base part (that is left most 4 hex digits of base address) and offset part (*that is displacement between the base address and the pointed memory location*)

Thus, a data segment memory location with PA=22356_H can also be written as 2000_H:2356_H, where segment base=2000_H & offset is 2356_H

Slide 7: Input-output Ports of 80x86' system:

- The 8086/8088 processors supports 64 Kbyte (65535_H=$FFFF_H$) byte-wide input/output (I/O) ports, accessed by dedicated instructions.

- In the software model of 80x86 systems, input-output ports are considered to be byte address spaces, either as part of the main memory or as an isolated device.

- The storage/retrieval techniques of byte and word data's are similar to that of main-memory of 80x86 processor

Physical Address(PA)	**I/O Contents**
0000_H	1 Byte
0001_H	$A\,7_H$
0002_H	$5\,0_H$
0003_H	$F\,F_H$
. . . .	64 K Byte address spaces
$FFFE_H$	$2\,5_H$
$FFFF_H$	$0\,0_H$

- More information on the operation and classes of input output address spaces are given in later modules of this course.

Slide 9: **<u>Software Model of the CPU in an 80x86 system:</u>**

- The software model of CPU consists of a number of 16-bit registers for dedicated operation.
- These internal registers are shown in the figure. Note that they are grouped according to their functions as will be explained later.

- The 2nd group of four registers is called Data-registers (AX … DX) and stores intermediate results, to be acted upon by next instruction.

- The 1st group of registers (DS ..) is called segment registers, which combines with 'IP' or the 3rd group of registers (SP… DI) to generate physical addresses (PA's) of main memory storage locations.

- The programmer should know the purpose, functions, operations and limitation of these registers.

- The detail definitions of these registers are given in the next section.

Slide 10: **Software Model: Definition of CPU registers:**

- Code Segment Register (CS) stores the leftmost sixteen-bits (4-hex digits) of the base address related to the 64-KByte Code segment memory locations. Remember that the base addresses of any segments are restricted to have a “0_H” as the rightmost hex-digit.

- In Slide 5, it is already shown that the base address consist of the four left-most hex digits with a “0_H” as the Least significant (or rightmost)hex digit

- Instruction Pointer register (IP) is a 16-bit register that stores the offset part of the physical address, which when combined with the CS register values, generates the physical address that points to memory locations within the 64-KByte Code Segment area.
- The method used to combine the values of IP and CS to generate the PA will be illustrated in next lecture of this module..

- Data Segment Register (DS) stores the leftmost sixteen-bits of the base address related to the 64KByte Data segment memory locations

---

Slide 11: **Software Model: Definition of CPU registers (cont’d)**

- Extra Segment register (ES) stores the leftmost sixteen-bits (4-hex digits) of the base address related to the 64-KByte Extra segment memory locations. (as the rightmost hex-digit of this PA is “$\mathbf{0}_H$”)

- Source Index register (SI) stores the offset address part of the physical address, which when combined with either DS or ES register contents, points to the <u>source data</u> stored within the 64-KByte Data or Extra Segment memory locations, respectively.

- Destination Index register (DI) stores the offset address part of the physical address, which when combined with either DS or ES register contents, points to the <u>destination data</u> stored within the 64-KByte Data or Extra Segment memory locations, respectively.

Slide 12: **Software Model: Definition of CPU registers (cont'd)**

- Stack Segment register (SS) stores the leftmost sixteen-bits (or 4-hex digits) of the base address related to the 64-KByte Stack segment memory locations. (as rightmost hex-digit of this PA is "$\mathbf{0}_H$")

- Stack Pointer Register (SP) and Base Pointer Register (BP) stores the offset part of the physical address, which when combined with the values of SS-register, generates the physical address that points to memory locations within the 64-KByte Stack Segment area.
- The difference between SP and BP will become clear in the lecture discussing Addressing modes.

- Data register (AX, BX, CX, DX) are 16-bit general purpose registers used for arithmetic calculations, temporary data storage, data transfer and special instructions.
- Further detail of data registers are presented in next slide.

Slide 13: **Software Model: Definition of CPU registers (cont'd)**

- The names of these data registers come from their functions:

  - AX is called Accumulator register and is the most commonly used register. This is also use for special instructions like MUL, DIV, CBW, LAHF, IN, OUT etc.
  - BX is called Base register and used by several addressing modes and special instruction like XLAT.
  - CX is called the counter register and used by special instructions like LOOP, SHIFT etc.
  - DX is called the Data register are used by special instructions like MUL, DIV, IN etc.

- The purpose and functions of these assembly language instructions will be introduced in module 2.

- All of these four 16-bit registers can be accessed as a whole for word-data operation or as two 8-bit registers for byte-data operation.

Such as; $(AX)_{word} = (AH)_{byte}\,(AL)_{byte}$; $(BX)_{word} = (BH)_{byte}\,(BL)_{byte}$;
$(CX)_{word} = (CH)_{byte}\,(CL)_{byte}$; $(DX)_{word} = (DH)_{byte}\,(DL)_{byte}$;

- Thus, if **AX** = $23F5_H$, this means AH = 23_H and AL = $F5_H$

*The Least-Significant-Byte of AX is stores in **AL** register AND Most-Significant-Byte of AX is stores in **AH** register*

Slide 14: **Software Model: Definition of CPU registers (cont'd)**

- Status Register (SR), also called flags register, reports the status of the flags after the execution of every instruction.

- Often SR register values determine the result of the executed instruction. Such as carry/borrow information for arithmetic instruction

- In 80x86 model, the 16-bit status or flag register is defined as:

Bit 15	....	*TF*	*DF*	*IF*	OF	SF	ZF	AF	PF	CF

(a) Carry Flag (CF): If Carry or Borrow occurred from the MSB of resulted DATA, then CF=CY=1 (set) otherwise CF=NC=0 (reset)
Example: $10001111_B + 11110000_B = \underline{0}1111111_B$ <u>and</u> CF ➔ CY

(b) Parity FLAG (PF): If the number of binary '1's' in the resulted DATA is even then PF=PE=1 (set) otherwise PF=PO=0 (reset).
Example: $10001111_B + 11110000_B = 01111111_B$ <u>and</u> PF ➔ PO

Slide 15: **Software Model: Definition of CPU registers (cont'd)**

(c) Auxiliary FLAG (AF): If Carry or Borrow occurred from the M.S.Nibble of resulted DATA then AF=AC=1(set) else AF=NA=0
Example: $10001111_B + 01001000_B = 1101\underline{0}111_B$ <u>and</u> AF ➔ AC

(d) Zero FLAG (ZF): If the execution of previous instruction results in a DATA=0, then ZF=ZR=1 (set), otherwise ZF=NZ=0 (reset).
Example: $00000001_B - 00000001_B = 0_B$ <u>and</u> ZF ➔ ZR

(e) Sign FLAG (SF): If the MSB of resulted signed DATA is "1_B", then SF=NG=1 (negative data) otherwise SF=PL=0 (≈positive).
Example: $10000001_B + 00000001_B = \underline{1}0000010_B$ <u>and</u> SF ➔ NG

(f) Overflow FLAG (OF): Indicates that Signed data is out of range.

(g) Direction FLAG (DF): Auto-decrement or Auto Increment in address after execution of string operation; DN=1 (set) or UP=0

(h) Trap FLAG (TF) decides operating mode (single-step or continues)

- The advantages of these flags will become clear during executing arithmetic and shift instructions, covered in later modules.

Slide 17: **Reminder:**

$(PA)_{\text{in Code-Segment memory}} = \underline{\mathbf{CS}}*10 + \mathbf{IP}$; $(PA)_{\text{in Data-Segment}} = \mathbf{DS}*10 + \mathbf{SI} = DS*10 + \mathbf{DI}$

$(PA)_{\text{in Extra-Segment memory}} = \mathbf{ES}*10 + \mathbf{SI} = ES*10 + \mathbf{DI}$; $(PA)_{\text{in Stack-Segment}} = \mathbf{SS}*10 + \mathbf{SP}$;

20 Bit Physical Address pointing to Memory = Related Segment Register value * **10** + Offset Register value

20-bit **Physical address** = 10*16-bit segment **base address** + 16-bit **offset or effective address**.

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