8086/8088 Microprocessor: Internal Architecture

Course Teacher:

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

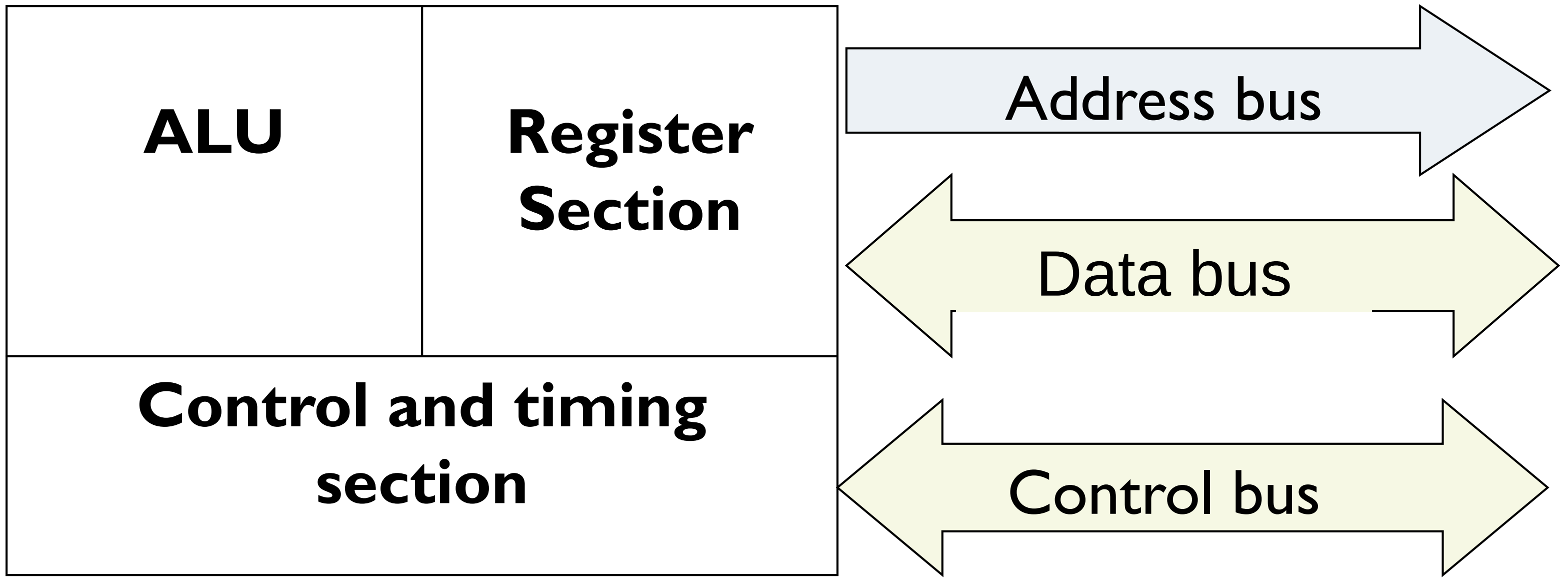
Course ID: CSE - 4503
Course Title: Microprocessors and Assembly Language
Department of Computer Science and Engineering (CSE)
Islamic University of Technology (IUT), Gazipur.

Lecture References:

- **Book:**
  - *Microprocessors and Interfacing: Programming and Hardware, Chapter # 2,* **Author:** Douglas V. Hall

- **Lecture Materials:**
  - *IBM PC Organization,* CAP/IT221

Internal Structure of a Microprocessor

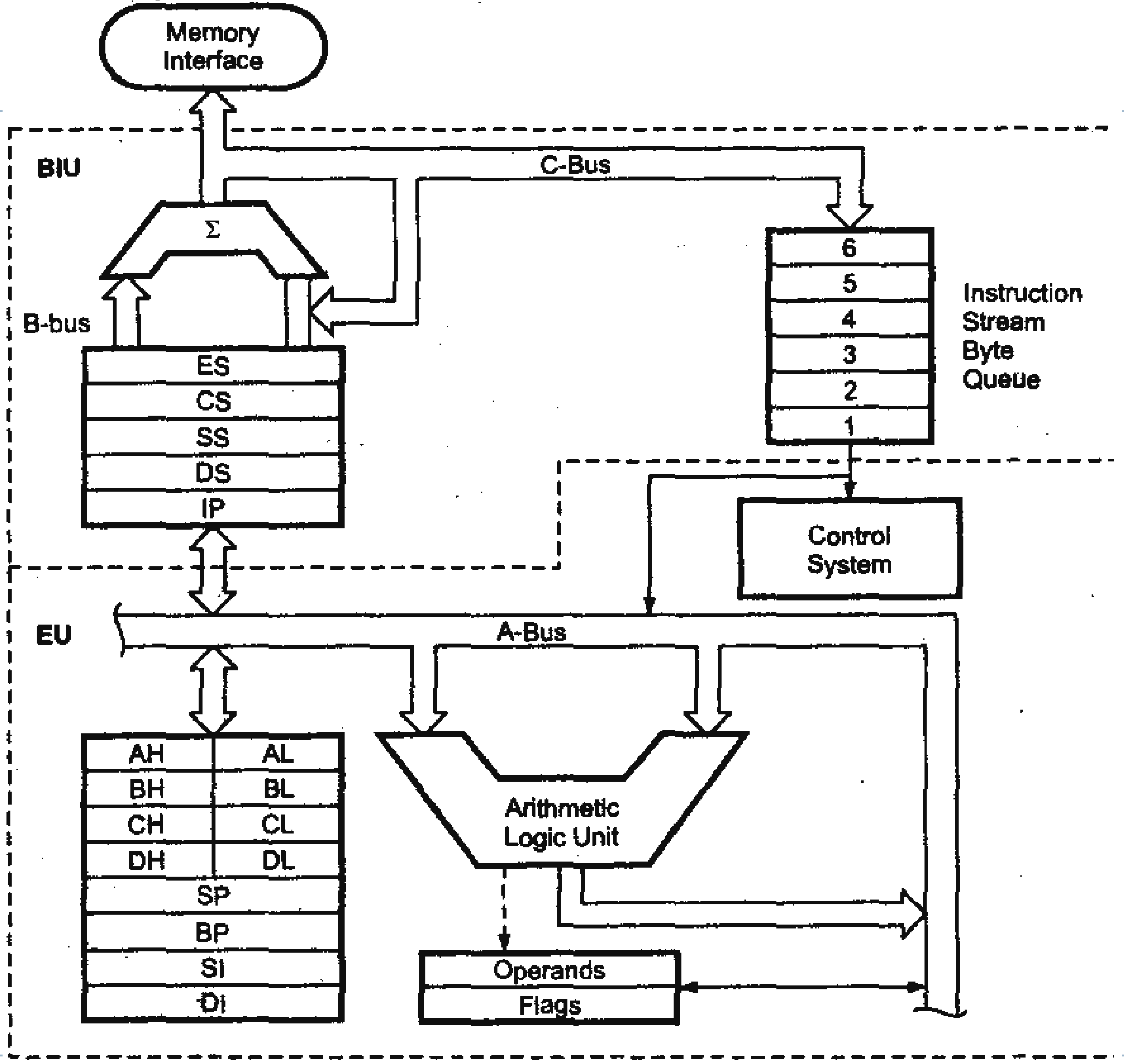


Block Diagram of a Microprocessor

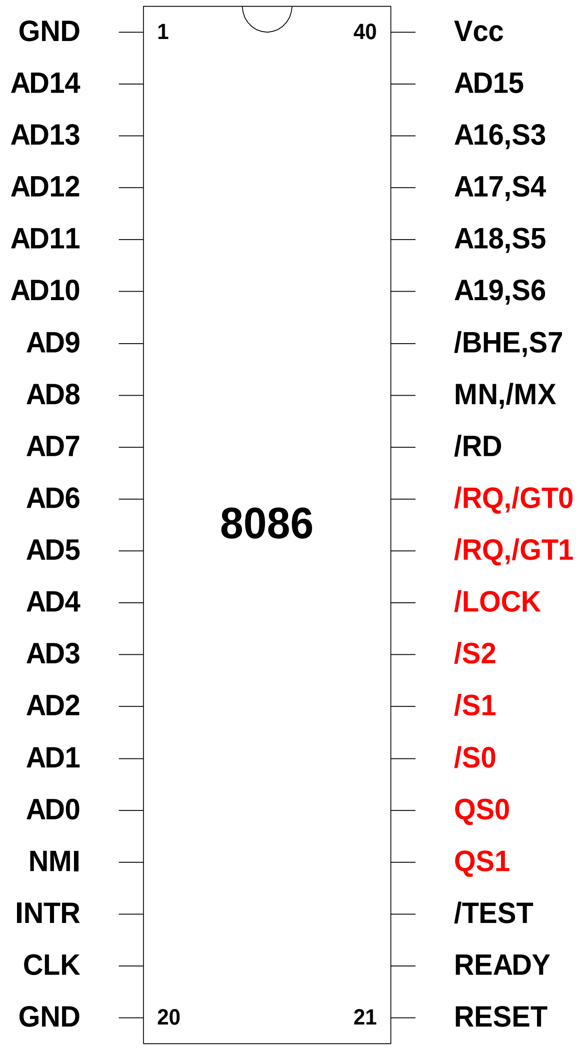
8086/8088 Microprocessor

- **Intel 8086**
  - The microprocessor 8086 can be considered to be the basic processor for the Intel X86 family from 1978.
  - It has a 20-bit address bus along with 16-bit data bus.
  - With the knowledge of 8086 16-bit processor, one can study the further versions of this processor 80286, 80406 and Pentium.
- **Intel 8088**
  - The Intel 8088 have 20-bit address bus with 8-bit data bus (allowing the use of cheaper and fewer supporting logic chips).
  - 8086/8088 have the same instruction set, it forms the basic set of instructions for other Intel families.

16-Bit 8088/8086 Intel Processor Architecture



16-Bit 8086 Intel Processor (Pin Diagram)



Organization of the 8088/8086

- 2 main components:
  1. **Execution Unit (EU)**
  2. **Bus Interface Unit (BIU)**

- **EU:** ALU + Registers (AX, BX, CX, DX, SI, DI, BP, and SP) + FLAGS register.
  - **ALU:** performs arithmetic & logic operations.
  - **Registers:** store data
  - **FLAGS Register:** Individual bits reflect the result of a computation.

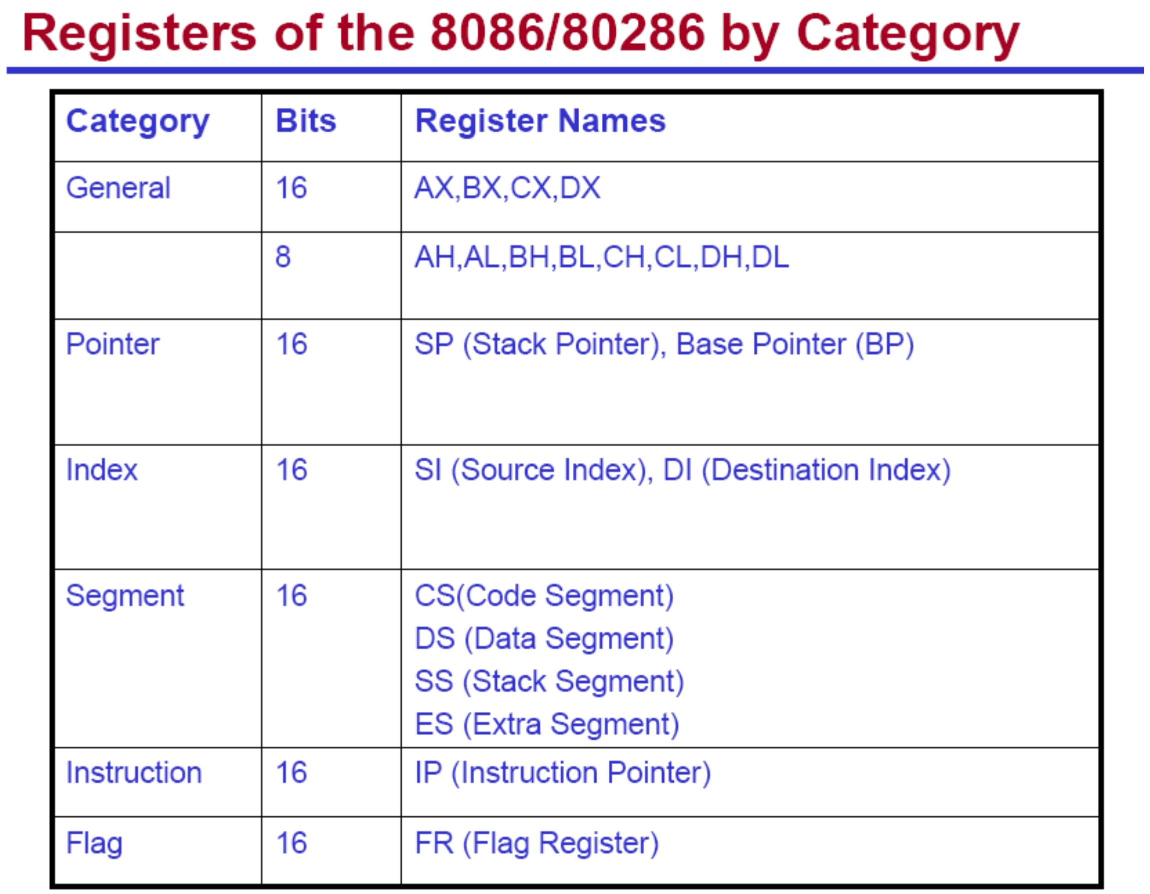
Organization of the 8088/8086

- **BIU:** facilitates communication between the EU & the memory or I/O circuits.
  - Responsible for transmitting addresses, data, and control signals on the buses.
  - **Registers** (CS, DS, ES, SS, and IP) hold addresses of memory locations.
  - **IP** (instruction pointer) contain the address of the next instruction to be executed by the EU.

8086 Registers and Memory

Number of Registers: 14, each of that16-bit registers

Memory Size: 1M Bytes

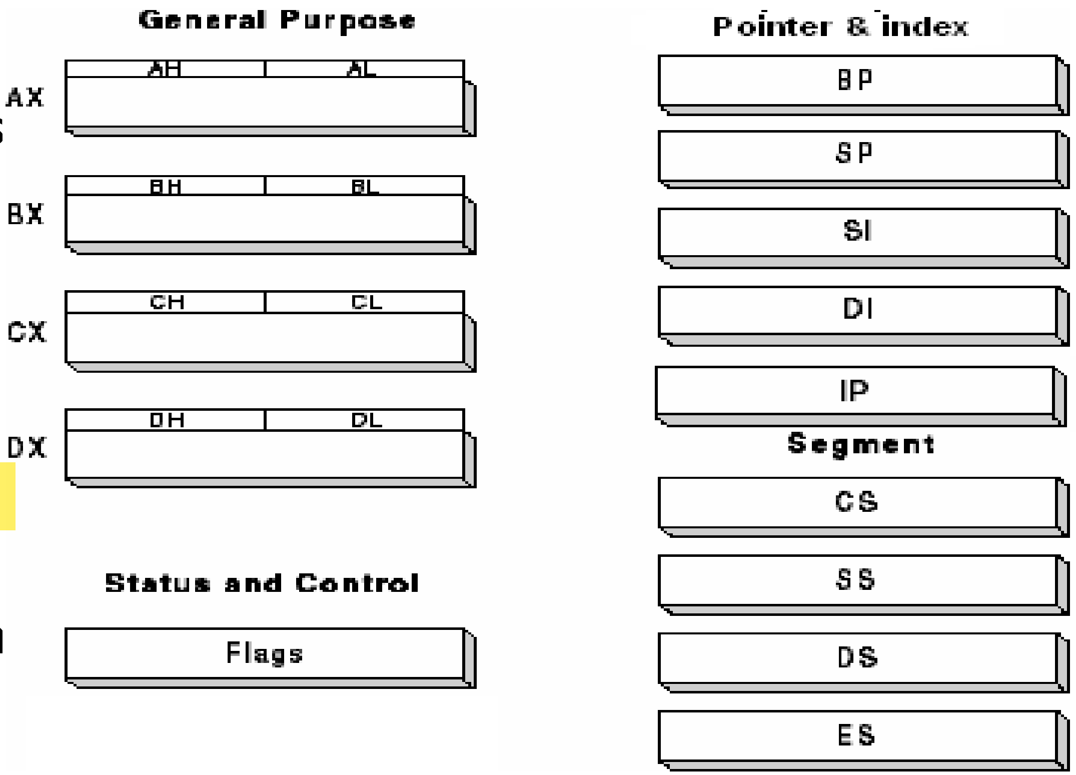


Registers of the 8086/80286 by Category

Category	Bits	Register Names
General	16	AX,BX,CX,DX
	8	AH,AL,BH,BL,CH,CL,DH,DL
Pointer	16	SP (Stack Pointer), Base Pointer (BP)
Index	16	SI (Source Index), DI (Destination Index)
Segment	16	CS(Code Segment) DS (Data Segment) SS (Stack Segment) ES (Extra Segment)
Instruction	16	IP (Instruction Pointer)
Flag	16	FR (Flag Register)

8086 Register Categories

- **Data registers (4)**: General data registers hold data for an operation (AX, BX, CX, DX).
- **Address registers (9)**: (Segment, Pointer and Index registers) hold the address of an instruction or data.
- **Status register (1)**: FLAG register keeps the current states of the processor.



General Data Registers

- These are 16-bit registers and can also be used as two 8 bit registers: ***low and high bytes*** can be accessed separately
- **AX (Accumulator)**
  - Most efficient register for arithmetic, logic operations and data transfer: the use of AX generates the shortest machine code.
  - In multiplication and division operations, one of the numbers involved must be in AL or AX
- **BX (Base)**
  - The base address register (offset)
- **CX (Counter)**
  - Counter for looping operations: loop counter, and in the shift and rotate bits
- **DX (Data)**
  - Used in multiply and divide, also used in I/O operations

Memory Segment and Offset Concept

- The 8086 processor assign a 20-bit physical address to its memory locations.

  $2^{20} \rightarrow$ **1 Mbyte**

  20 bits $\rightarrow$ 5 hex digits

  First addresses: 00000, 00001,…,0000A,…FFFFF.

  But Registers are 16-bits and can address only 2^{16} = **64 KBytes.**

- **Partition the memory into segments**
  - One way four (4) 64 Kbytes segments might be positioned within the 1 Mbyte address space of an 8086 processor.

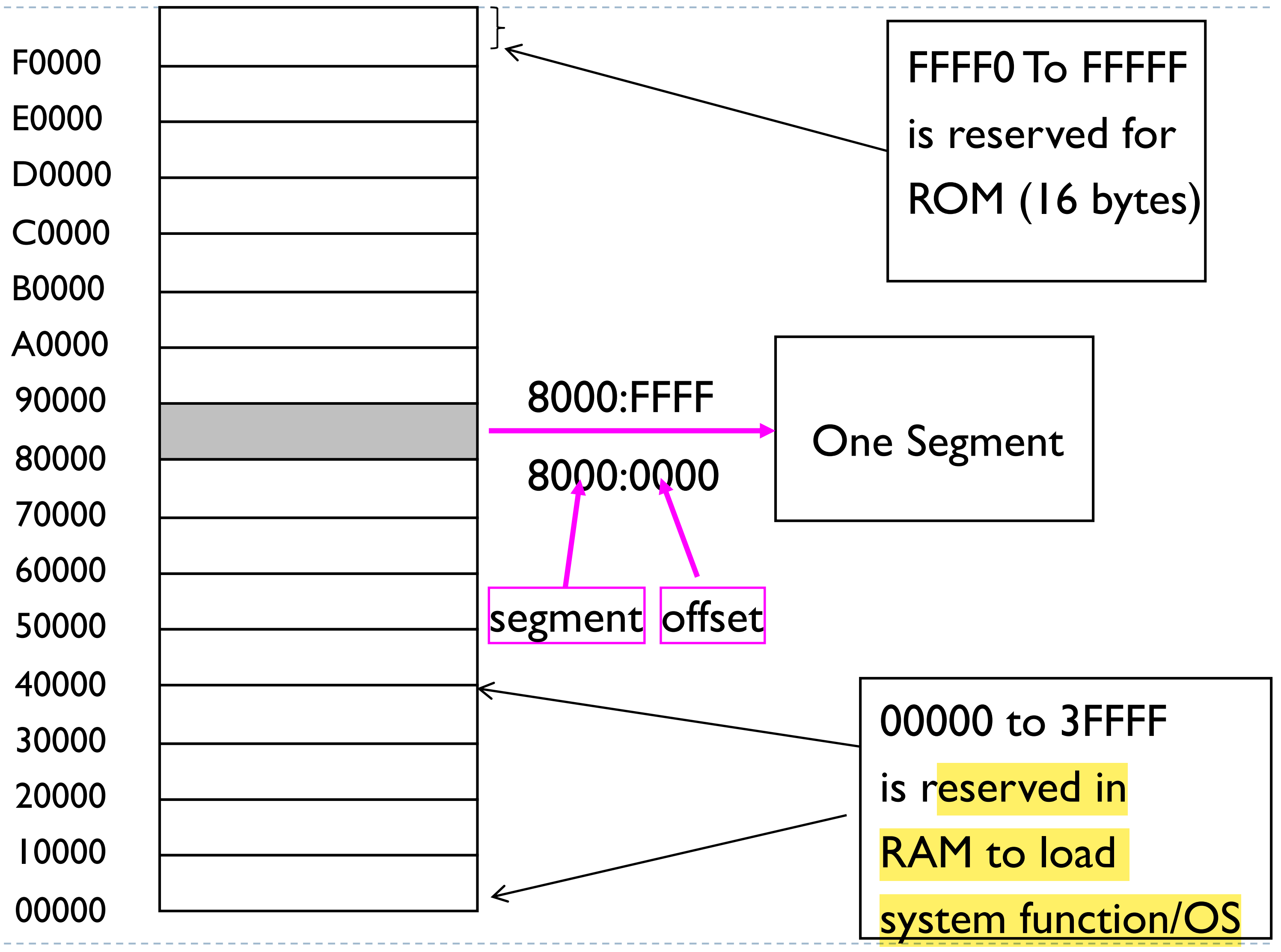
Memory Segment and Offset Concept

- ***Memory segment*** is a block of 2^{16} (64) KBytes consecutive memory bytes.

- Each segment is identified by a 16-bit number called **segment number**, starting with 0000 up to FFFFh. Segment registers hold segment number.

- Within a segment, a memory location is specified by giving an **offset** (16-bit) = It is the number of bytes from the beginning of the segment (0$\rightarrow$ FFFFh).

Memory Segment and Offset Concept



Memory Segment and Offset Concept

- A memory location may be specified by a ***segment number*** and ***offset*** (logical address).

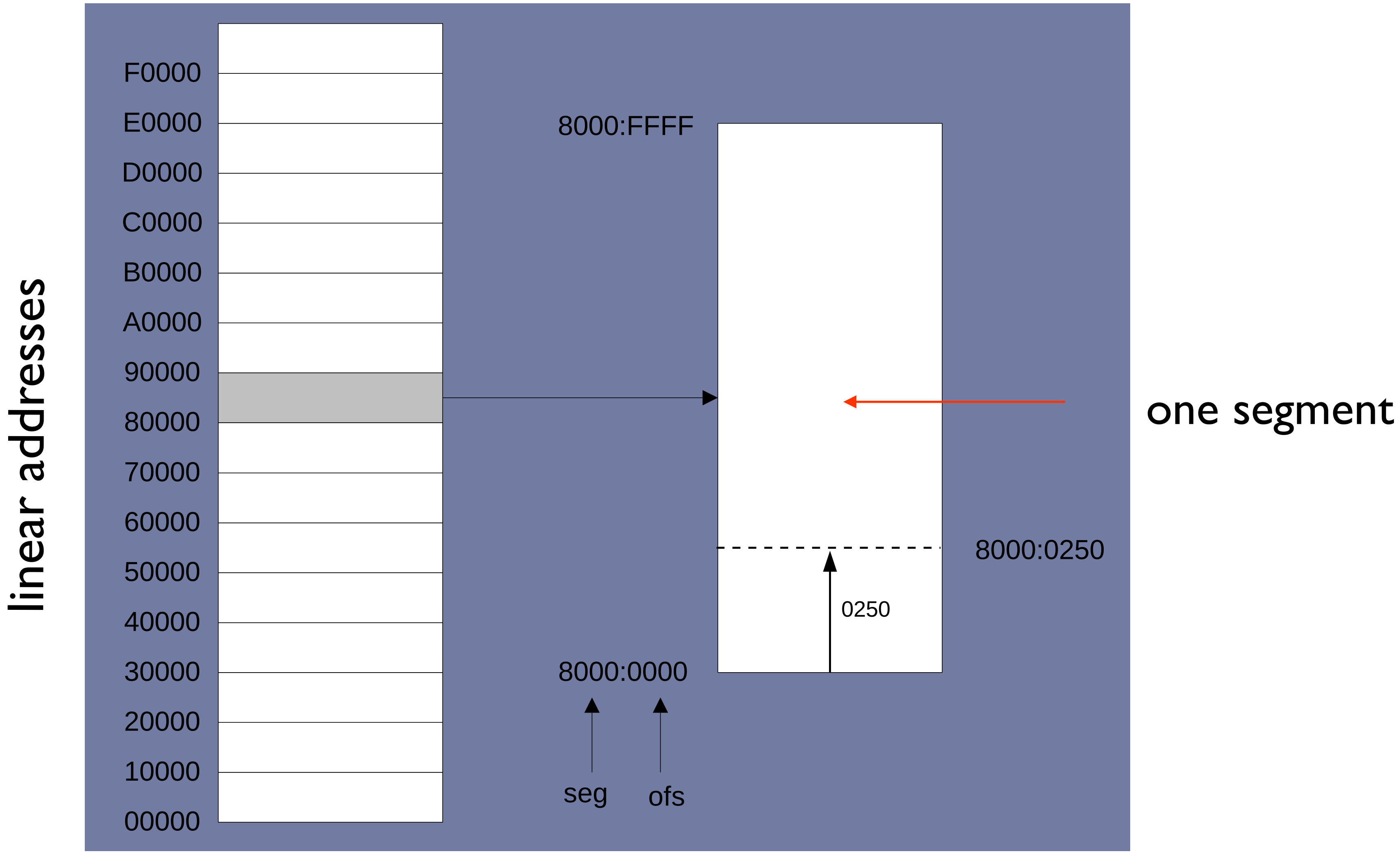
  Example :

$$\begin{tikzcd} & \text{A4FB : 4872}_{h} \arrow[dl] \arrow[dr] & \\ \text{Segment} & & \text{Offset} \end{tikzcd}$$

- **Offset:** is the distance from the beginning to a particular location in the **segment.**

- **Segment Number:** defines the starting of the segment within the memory space.

Memory Segment and Offset Concept



Memory Segment and Offset Concept

- **Segmented Memory Address:**
- Start location of the segment must be 20 bits $\rightarrow$ the absolute address is obtained by appending a hexadecimal zero to the segment number, i.e. , **multiplying by 16(10_h).**
  - Adds 4 Nibble bits at the lower portion of each 16-bit address.
- So, the **Physical Memory Address** is equal to:

Physical Address = Segment number X 10_h + Offset

- Physical **Address** for **A4FB : 4872**

$$\begin{array}{r} \textbf{A4FB0 h} \\ \textbf{+ 4872 h} \\ \hline \textbf{A9822 h} \text{ (20 Bits)} \end{array}$$

Physical Location of Segments

- **Segment 0**
  - **starts** at address 0000:0000 → 00000 h
  - **ends** at address 0000:FFFF → 0FFFF h
- **Segment 1**
  - **starts** at address 0001:0000 → 00010 h
  - **ends** at address 0001:FFFF → 1000F h
- Overlap occurs between the Segment 0 and 1 having varying size.
  - **Advantage:** Utilization of memory would be higher.

Physical Location of Segments

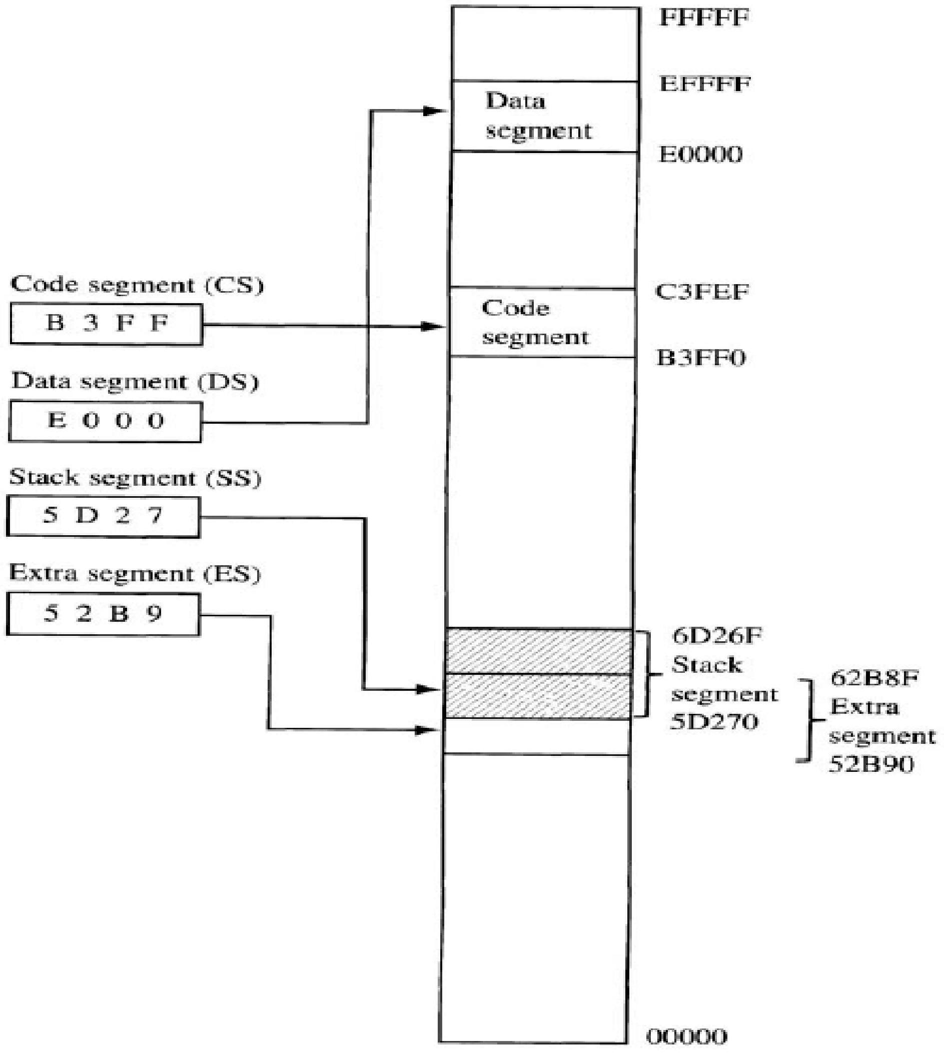
Segment	Physical Address (hex)
	…
	10021
	10020
End of Segment 2	1001F
	1001E
	…
	10010
End of Segment 1	1000F
	1000E
	…
	10000
End of Segment 0	0FFFF
	0FFFE
	…
	00021
Start of Segment 2	00020
	0001F
	…
	00011
Start of Segment 1	00010
	0000F
	…
	00003
	00002
	00001
Start of Segment 0	00000

Segment Registers

- Four Segment Registers in the BIU are used to hold the upper 16-bits of the starting addresses of four memory segments, namely
  - **Code segment CS**: holds segment address of the code segment.
  - **Data Segment DS:** holds segment address of the data segment.
  - **Extra Segment ES:** extra segment : holds alternate segment address of the data segment.
  - **Stack Segment SS:** holds segment address of the stack segment and used when sub-program executes.
- Codes , data , and stack are loaded into different memory segments (registers).

Memory Segment and Segment Registers

Segment Registers



Instruction Pointer (IP) and Code Segment Register

- **IP (Instruction pointer):**
  - Points to the next instruction.
  - Offset address relative to CS
- **Code segment CS**: holds segment address of the code segment.

- Suppose, CS = B3FFh and IP = 12ABh

- Physical **Address** of the next instruction:

$$\begin{array}{r} \textbf{B3FF0 h} \\ \underline{\textbf{+ 12AB h}} \\ \textbf{B529B h} \text{ (20 Bits)} \end{array}$$

Stack Pointer and Index Registers

- Used for offset of data, often used as pointers. Unlike segment registers, they can be used in arithmetic and other operations.
- **SP (Stack Pointer)**:
  - Used with SS for accessing the stack segment.
  - Holds **Offset** address relative to SS
  - Always points to word (byte at even address)
  - An empty stack will had SP = FFFEh
- **BP (Base Pointer)**:
  - Used with SS to access data on the stack. However, unlike SP, BP can be used to access data in other segments.
  - Primarily used to access parameters passed via the stack
  - Holds **Offset** address relative to SS

Stack Pointer and Stack Segment Register

- Suppose, SS = 5D27h and SP = FFFEh

- Physical **Address** of the Top of Stack (ToS) information/data:

```
   5D270 h
 + FFFE h
 ---------
   6D26E h (20 Bits)
```

Data Pointer and Index Registers

- **SI (Source Index):**
  - Source of string operations. Used with DS (or ES).
  - Can be used for pointer addressing of data with effective address (EA)
  - Used as source in some string processing instructions
  - Offset address relative to DS
- **DI (Destination Index):**
  - Destination of string operation. Used with ES (or DS).
  - Can be used for pointer addressing of data
  - Used as destination in some string processing instructions
  - Offset address relative to ES

Source Index and Data Segment Register

- Suppose, DS = E000h and SI (EA) = 437Ah

- Physical **Address** of the data:

$$\begin{array}{r} \textbf{E0000 h} \\ \textbf{+ 437A h} \\ \hline \textbf{E437A h} \end{array} \text{ (20 Bits)}$$

Flag Register

- **Flags Register:** A 16-Bits register specify status of CPU and information about the results of the arithmetic operations.
- Flags Register determines the current state of the processor.
- It is modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program.
- Generally you cannot access these registers directly.

Bit Position

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	O	D	I	T	S	Z	x	A	x	P	x	C

O = Overflow	S = Sign
D = Direction	Z = Zero
I = Interrupt	A = Auxiliary Carry
T = Trap	P = Parity
x = undefined	C = Carry

Flag Register

- **Carry Flag (CF)** - this flag is set to '1' when there is an unsigned overflow. For example when you add bytes 255 + 1 (result is not in range 0...255). When there is no overflow this flag is reset to 0.
- **Parity Flag (PF)** - this flag is set to '1' when there is even number of one bits in result, and reset to '0' when there is odd number of one bits.
- **Auxiliary Flag (AF)** - set to '1' when there is an unsigned overflow for low nibble (4 bits).
- **Zero Flag (ZF)** - set to '1' when result is zero. For non-zero result this flag is reset to '0'.

Flag Register

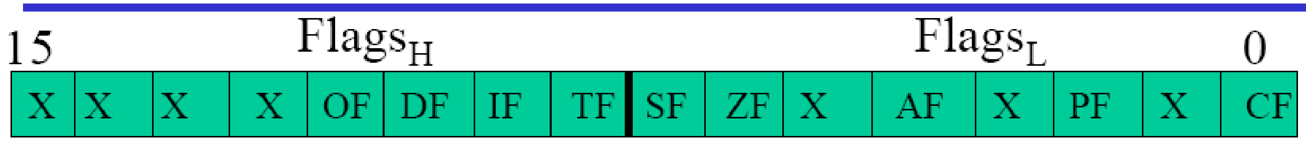
- **Sign Flag (SF)** - set to '1' when result is negative. When result is positive it is reset to '0'. (This flag takes the value of the most significant bit).
- **Trap Flag (TF)** - Used for on-chip single-step debugging.
- **Interrupt enable Flag (IF)** - when this flag is set to '1' CPU reacts to interrupts from external devices.
- **Direction Flag (DF)** - this flag is used by some instructions to process data chains, when this flag is set to '0' - the processing is done forward, when this flag is set to '1' the processing is done backward.
- **Overflow Flag (OF)** - set to '1' when there is a signed overflow. For example, when you add bytes 100 + 50.

Rule for Overflow Flag

```
Overflow Flag
-------------

The rules for turning on the overflow flag in binary/integer math are two:

1. If the sum of two numbers with the sign bits off yields a result number
   with the sign bit on, the "overflow" flag is turned on.

   0100 + 0100 = 1000 (overflow flag is turned on)

2. If the sum of two numbers with the sign bits on yields a result number
   with the sign bit off, the "overflow" flag is turned on.

   1000 + 1000 = 0000 (overflow flag is turned on)

Otherwise, the overflow flag is turned off.
 * 0100 + 0001 = 0101 (overflow flag is turned off)
 * 0110 + 1001 = 1111 (overflow flag is turned off)
 * 1000 + 0001 = 1001 (overflow flag is turned off)
 * 1100 + 1100 = 1000 (overflow flag is turned off)

Note that you only need to look at the sign bits (leftmost) of the three
numbers to decide if the overflow flag is turned on or off.
```

Flag Register (Example)

Flag (Status) Register



- Six of the flags are status indicators reflecting properties of the last arithmetic or logical instruction.
- For example, if register AL = 7Fh and the instruction ADD AL,1 is executed then the following happen
  - AL = 80h
  - CF = 0; there is no carry out of bit 7
  - PF = 0; 80h has an odd number of ones
  - AF = 1; there is a carry out of bit 3 into bit 4
  - ZF = 0; the result is not zero
  - SF = 1; bit seven is one
  - OF = 1; the sign bit has changed
- Can be used to transfer program control to a new memory location

```
ADD AL,1
JNZ 0100h
```

Thank You !!