Microprocessors, Micro-controllers and Assembly Language

(CSE – 305)

Week -1

Reference Book:

Assembly Language Programming and Organization of the IBM PC

Ytha Yu and Charles Marut

Introduction to Microprocessors

- **What is a Microprocessor?**
- **What is a Microcontroller?**
- **What is Embedded system?**
- **What is Assembly Language?**

Introduction to Microprocessors

What is a Microprocessor?

A microprocessor is an electronic chip that functions as a central processing unit or the brain of a computer. It is made up of millions of transistors, diodes and resistors and it is responsible for any arithmetic or logical operation. The microprocessors operate on binary data in the form of “0” & “1”.

The microprocessors do not have RAM or ROM inside it. But they do have registers to stores the results from the ALU (Arithmetic Logic Unit). It includes interfaces to connect the external ROM and RAM to the processors.

Introduction to Microprocessors

What is a microcontroller?

A microcontroller is a chip optimized to control electronic devices. It is stored in a single integrated circuit which is dedicated to performing a particular task and execute one specific application.

A **microcontroller** contains one or more CPUs (processor cores) along with memory and programmable input/output peripherals.

Introduction to Microprocessors

As its name suggests, Embedded means something that is attached to another thing. An embedded system can be thought of as a computer hardware system having software embedded in it. An embedded system can be an independent system or it can be a part of a large system. An embedded system is a microcontroller or microprocessor based system which is designed to perform a specific task. For example, a fire alarm is an embedded system; it will sense only smoke.

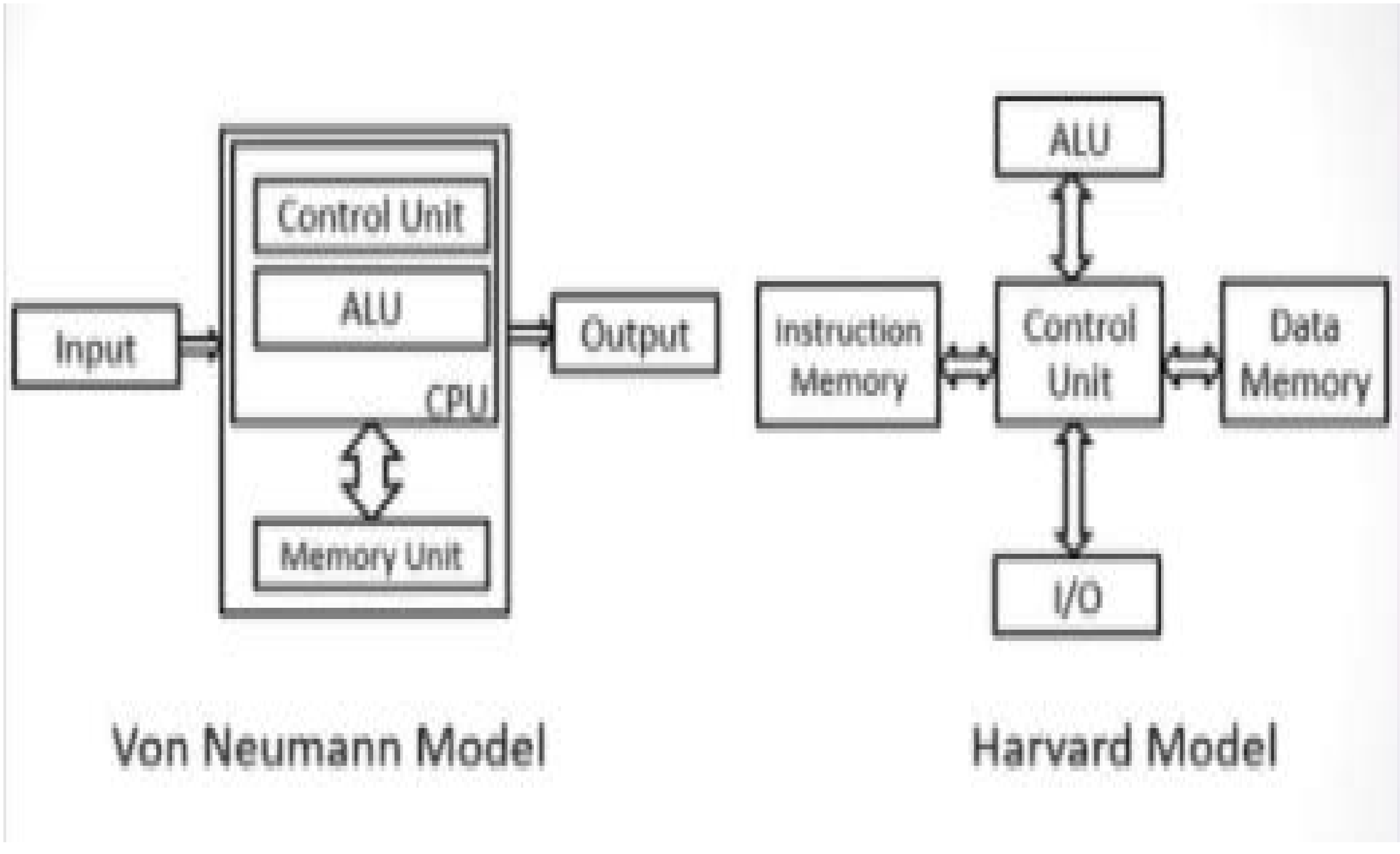
An **embedded system** is a computer system—a combination of a computer processor, computer memory, and input/output peripheral devices—that has a dedicated function within a larger mechanical or electrical system

Introduction to Microprocessors

KEY DIFFERENCES

- Microprocessor consists of only a Central Processing Unit, whereas Micro Controller contains a CPU, Memory, I/O all integrated into one chip.
- Microprocessor uses an external bus to interface to RAM, ROM, and other peripherals, on the other hand, Microcontroller uses an internal controlling bus.
- Microprocessors are based on Von Neumann model, Micro controllers are based on Harvard architecture
- Microprocessor is complicated and expensive, with a large number of instructions to process but Microcontroller is inexpensive and straightforward with fewer instructions to process.

Introduction to Microprocessors



Introduction to Microprocessors

The **diode** is the most used semiconductor device in electronics circuits. It is a two-terminal electrical check valve that allows the flow of current in one direction. They are mostly made up of silicon but germanium is also used. Usually, they are used for rectification.

A **resistor** is a component or device designed to have a known value of resistance. **OR,**
Those components and devices which are specially designed to have a certain amount of resistance and used to oppose or limit the electric current flowing through it are called resistors.

A **transistor** is a semiconductor device used to amplify or switch electronic signals and electrical power.

Introduction to Microprocessors

- The microprocessor is the heart of the computer and it is a hardware component.

Introduction to Microprocessors

- Intel introduced it’s 4 bit microprocessor 4004 in 1971 (In the 4-bit microprocessor or computer architecture will have a data path width or a highest operand width of 4 bits or a nibble) and it’s 8 bit microprocessor 8008 in 1972 (In computer architecture, 8-bit integers, memory addresses, or other data units are those that are 8 bits (1 octet or 1 Byte) wide.).

- These microprocessors could not survive as general purpose microprocessors because of their design and performance limitations.

- Then the launch of a first general purpose 8 bit microprocessor 8080 in 1974 by Intel is considered to be the first major stepping stone towards the development of advanced microprocessors.

Introduction to Microprocessors

- The microprocessor 8085 (introduced in 1976) followed by 8080, with a few more added features to it's architecture, which resulted in a functionally complete microprocessor.

- The main limitations of 8-bit microprocessor were
  – Low speed,
  – Low memory addressing capability
  – Limited number of general purpose registers
  – Less powerful instruction set

- All these limitations lead to the launching of 8086 microprocessor.

- In the family of 16 bit microprocessors, Intel's 8086 was the first one to be launched in 1978.

Introduction to Microprocessors

Main Differences between 8085 and 8086 Microprocessor

8085 Microprocessor	8086 Microprocessor
It is an 8-bit microprocessor.	It is a 16-bit microprocessor.
It has an 8-bit wide data bus.	It has 16-bit wide data bus.
It can address 2^8 = 256 I/O ports.	It can address 2^{16} = 65,536 I/O ports.
It has 16-bit wide address bus.	It has 20-bit wide address bus.
It has an 8-bit ALU (Arithmetic Logic Unit).	It has a 16-bit ALU.
It can process an 8-bit of data in one machine cycle.	It can process 16-bit of data in one machine cycle
The maximum accessible memory (RAM) it can access is 2^{16} = 64KB.	The maximum accessible memory of 8086 is 2^{20} = 1MB.

Introduction to Microprocessors

Main Differences between 8085 and 8086 Microprocessor

8085 Microprocessor	8086 Microprocessor
It has an on-chip oscillator of 3 MHz.	It is available in 3 versions with a clock frequency of 5 MHz, 8 MHz and 10 MHz.
It is an Accumulator based Microprocessor.	It is a General Purpose Register based Microprocessor.
It does not have Multiplication and division instruction.	Its ALU can perform multiplication and division.
It has 5 flags (carry, auxiliary carry, parity, zero and sign flag).	It has 9 flags (carry, auxiliary carry, parity, zero, sign, trap, interrupt, direction and overflow flag).
It does not have an instruction queue.	It has an instruction queue of 6 bytes that is stored in FIFO (First In First Out) register.

Introduction to Microprocessors

Main Differences between 8085 and 8086 Microprocessor

8085 Microprocessor	8086 Microprocessor
It supports only the single mode of operation. i.e. a single processor mode. It does not support external processors.	It has two modes of operation; minimum (single Processor mode) and maximum mode (Multiprocessor mode) which supports external processors.
It is cheaper than the 8086 microprocessor.	It is expensive than the 8085 microprocessor.

Introduction to Microprocessors

- The 8086 microprocessor has a much more powerful instruction set along with the architectural developments which imparts substantial programming flexibility and improvement in speed over the 8-bit microprocessor.

- The peripheral chips designed earlier for 8085 were compatible with microprocessor 8086 with slight or no modifications.

Architecture of 8086

- The architecture of 8086 supports a 16 bit ALU, a set of 16 bit registers and provides the segmented memory addressing capability, a rich instruction set, powerful interrupt structure, fetched instruction queue for overlapped fetching and execution etc.

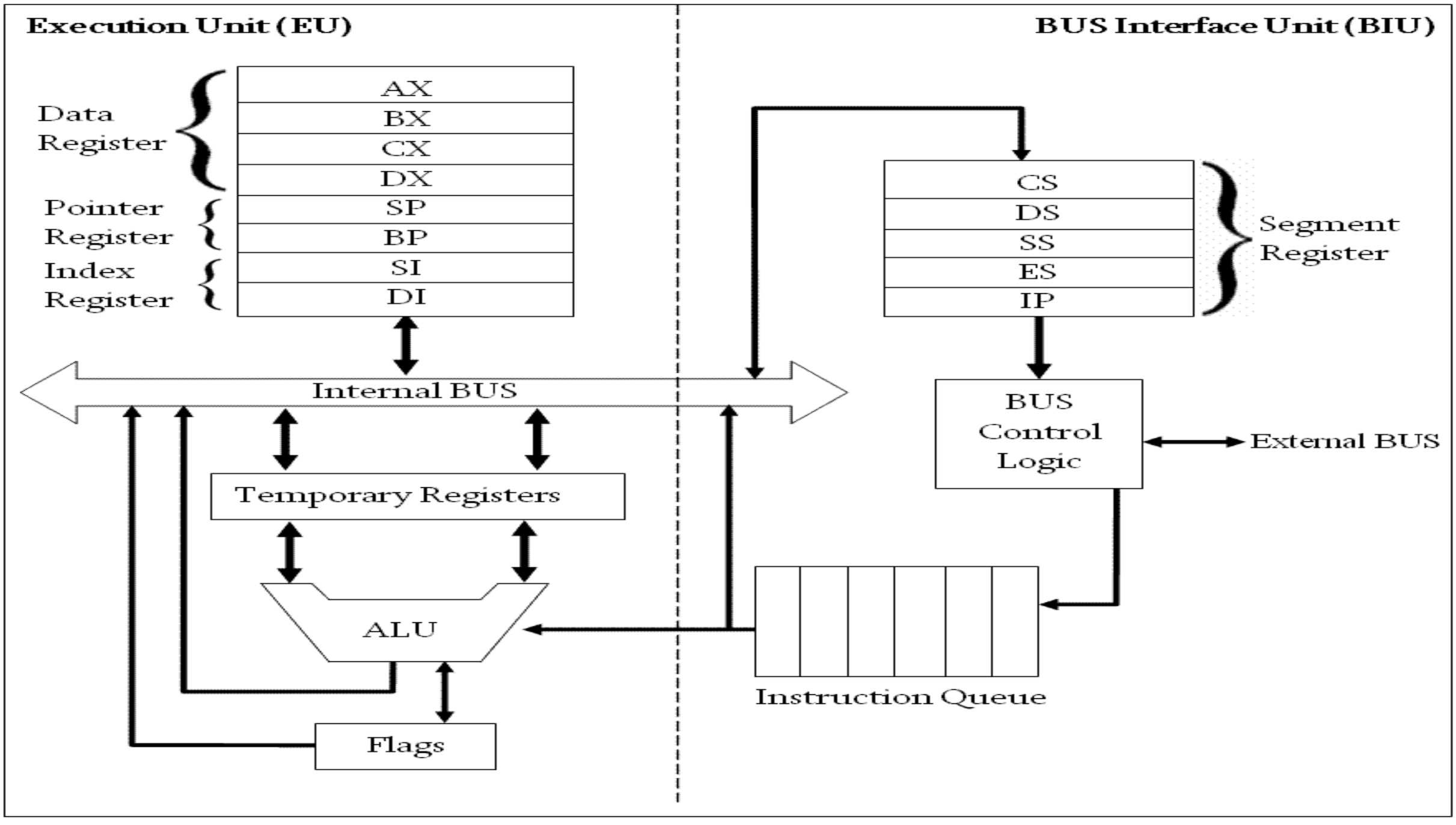
- The internal block diagram, shown in Fig 1.2, describes the overall organization of different units inside the chip.

- The complete architecture of 8086 can be divided into two parts.

  – Bus interface unit

  – Execution Unit

Intel 8086 Microprocessor Organization



Registers

- Information inside the microprocessor is stored in registers. To speed up the processor operations, the processor includes some internal memory storage locations, called registers.

- Registers are classified according to their functions.
  - Data register holds the data for an operation
  - Address register holds the address for an instruction or data.
  - Status register keeps the current status of the processor.

Registers

- 8086 has four general registers.
  - Address registers: 1) segment 2) pointer and 3) index register
  - Status Register: 4) FLAGS Register
- There are total fourteen 16-bit registers.

8086 Registers

General Register

Segment Register

Pointer and Index Register

FLAGS Register

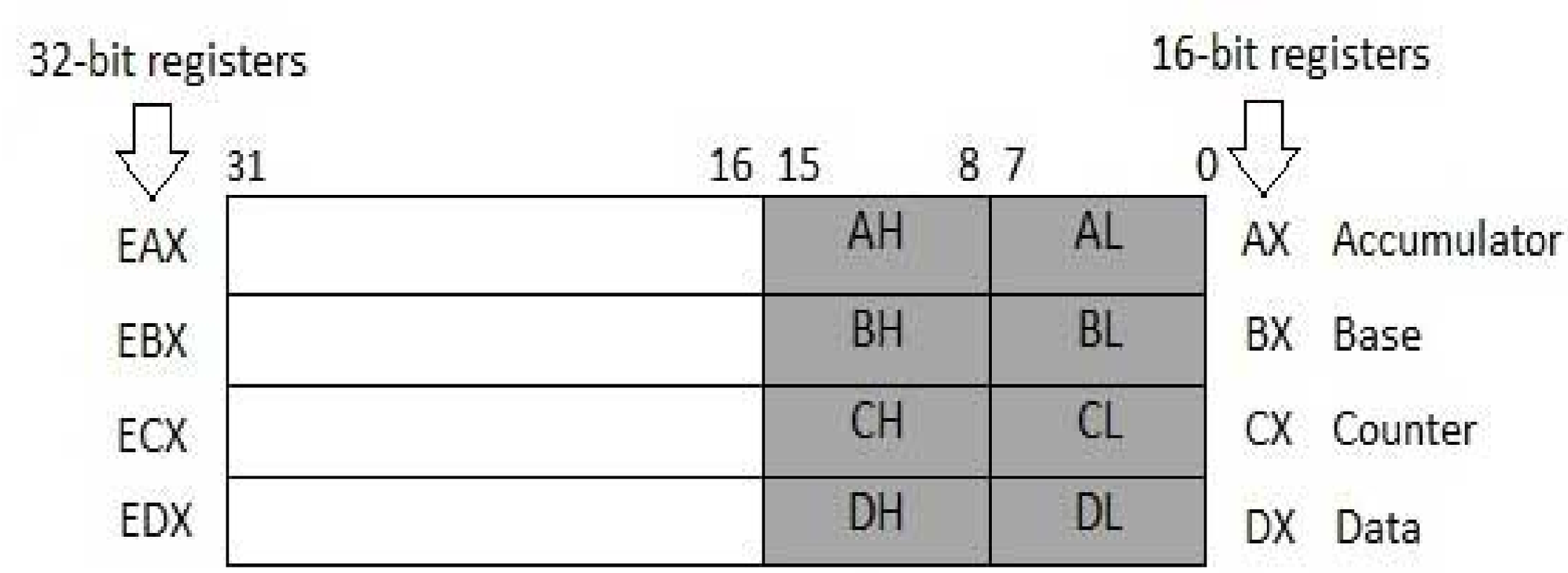
80x86 Registers

AX	Accumulator	AH	AL
BX	Base	BH	BL
CX	Counter	CH	CL
DX	Data	DH	DL
	CS (code segment)		
	DS (data segment)		
	SS (stack segment)		
	ES (extra segment)		
	IP (instruction pointer)		
	SP (stack pointer)		
	BP (base pointer)		
	SI (source index)		
	DI (destination index)		
	Flag Register		

8086 Registers

Four 32-bit data registers are used for arithmetic, logical, and other operations. These 32-bit registers can be used in three ways −

- As complete 32-bit data registers: EAX, EBX, ECX, EDX.
- Lower halves of the 32-bit registers can be used as four 16-bit data registers: AX, BX, CX and DX.
- Lower and higher halves of the above-mentioned four 16-bit registers can be used as eight 8-bit data registers: AH, AL, BH, BL, CH, CL, DH, and DL.



AX: Accumulator Register

- **AX is the primary accumulator (**intermediate storage of arithmetic and logic data); it is used in input/output and most arithmetic instructions. For example, in multiplication operation, one operand is stored in EAX or AX or AL register according to the size of the operand.

- Thus, AX is preferred register to use in **arithmetic, logic** and **data transfer** instructions.

- In **multiplication and division**, one of the numbers involved must be in AX or AL.

- Input and Output also require the use of AL or AX

BX: Base Register

- **BX is known as the base register**, as it could be used in indexed addressing (A method of generating an effective **address** that modifies the specified **address** given in the instruction by the contents of a specified **index** register)

- BX also serves as an address register

CX: Count Register

- CX used as Program **Loop counter.** CX registers store the loop count in iterative operations.

- CX also used as a counter (REP-repeat) to control **string operations.**

- CL is used as count in **bit rotation and shifting instructions**

DX: Data Register

- DX registers are used in input/output operations. It is also used with AX register for multiply and divide operations involving large values.

Memory Recall

- **List One special function of AX, BX, CX, and DX**

Segment Registers

- Segments are specific areas defined in a program for containing data, code and stack. There are three main segments −

Code Segment − It contains all the instructions to be executed. A 16-bit Code Segment register or CS register stores the starting address of the code segment.

Data Segment − It contains data, constants and work areas. A 16-bit Data Segment register or DS register stores the starting address of the data segment.

Stack Segment − It contains data and return addresses of procedures or subroutines. It is implemented as a 'stack' data structure. The Stack Segment register or SS register stores the starting address of the stack.

Architecture of 8086

- For generating a physical address from contents of these two registers, the content of a segment register also called as segment address is shifted left bit-wise four times and to this result, content of an offset register also called as offset address is added, to produce a 20 bit physical address.

Architecture of 8086

- Thus, the segment register indicates the base address of a particular segment, while the offset indicates the distance of the required memory location in the segment from the base address.

- Since the offset is a 16-bit number, each segment can have a maximum of 64k locations.

- The bus interface unit has a separate adder to perform this procedure for obtaining a physical address while addressing a memory

Segment Registers: CS,DS,SS,ES

- Memory is a collection of bytes.
- Each memory bytes has an address starting with 0.
- The 8086 assigns 20-bit physical address to its memory location.[i.e. we can address 2^20(1MB) of memory]. Thus the first byte of the memory addresses:

Binary representation	Hex Representation
0000 0000 0000 0000 0000	**00000h**
0000 0000 0000 0000 0001	**00001h**
0000 0000 0000 0000 0010	**00010h**
0000 0000 0000 0000 0011	**00011h**
0000 0000 0000 0000 0100	**00100h**

***So what will be the highest address of 20-bit memory address?

Using 20-bit Address in 16-bit Processor

- To explain segment register's function, lets have a look on the idea of memory segments.
- How can we fit 20 bit address into 16 bit register?

  Memory Partitioning into segments

- A memory segment is a block of 2^16(64KB) consecutive memory bytes.
- Each segment is identified by segment number.[starts with 0]
- A **segment** number is 16-bit [thus, highest value FFFFh].
- Within a segment, a memory location is specified by giving an offset.
- **Offset:** Number of bytes from the beginning of segment.

  i.e. for a 64KB segment, the offset can be given as16-bit number.

  - **The first byte in a segment has offset 0000h.**
  - **The Last byte in a segment has offset FFFFh**

Segment : Offset address

- There are often many different **Segment:Offset** pairs which can be used to address the same location in your computer's memory.
- A memory segment may be specified by providing a segment number and an offset.
- The scheme works like this: The value in any register considered to be a Segment register is multiplied by 16 (or shifted one hexadecimal byte to the left; add an extra 0 to the end of the hex number) and then the value in an Offset register is added to it. So, the Absolute address for any combination of Segment and Offset pairs is found by using the formula:

Segment : Offset address

- Memory segment is written in the form of segment : offset.

- The representation of segment : offset is known as logical address (**Logical address** is generated by CPU in perspective of a program. On the other hand, the **physical address** is a location that exists in the memory unit)

  e.g. A4FB:4872h means offset 4872h within segment A4FBh

How to obtain 20-bit physical address in a 16-bit microprocessor?

1. The 8086 shifts the segment address 4-bits to the left [i.e. multiply by 10h].

2. Add the offset address to the segment address.

i.e. to get the 20-bit physical address from A4FB:4872h,

A4FB0h [multiplied segment with 10]

4872h

======

A9822h [20-bit physical address]

How to obtain 20-bit physical address in a 16-bit microprocessor?

Absolute memory Location = (Segment value * 16) + Offset value.

The Absolute or Linear address for the **Segment:Offset** pair, **F000:FFFD** can be computed quite easily in your mind by simply inserting a zero at the end of the Segment value (which is the same as multiplying by 16) and then adding the Offset value:

F 0 00 0
+ FFFD

FFFFD or 1,048,573(decimal)

Task

- **Find the 20-bit address of**
- **ABC4:12BAh ---- ACEFA**
- **923F:E2FF -> ------ A06EF or 657,135**(decimal)

Example-1

- **Physical address of a memory location is 1256Ah, find the address in segment : offset form for segment 1256h?**

Physical address = Segment X 10h + offset

Offset = Physical address - Segment

Lets consider, X = offset in 1256h. Thus,

- 1256Ah=12560h [segment 1256 multiplied by 10] + X
- X = 1256Ah-12560h
- X = Ah
- X = 000Ah

Segment : offset = 1256:000Ah

Example-2

- **Physical address of a memory location is 1256Ah, find the address in segment : offset form for segment 1240h?**

Lets consider, X = offset in 1240h. Thus,

- 1256Ah=12400h [segment 1256 multiplied by 10] + X
- X = 1256Ah-12400h
- X = 16Ah
- X = 016Ah

Segment : offset = 1240:016Ah

Calculate the Segment Number

- **A physical address 80FD2h and offset BFD2h is given. Calculate the segment .**

Physical address = Segment X 10h + offset

Thus,

- Segment X 10h = Physical address - offset
- Segment = (Physical address - offset) / 10h
- Segment = (80FD2h - BFD2h)/10h
- Segment = (75000h)/10h

Segment = 7500h

Task

- Find the physical address of memory location **0A51:CD90h ?**

- A memory location has physical address **4A37Bh.**Compute the offset if segment number is **40FFh.**

- Compute the Segment if offset number is **123Bh.**

Program Segments

- Machine language program consists instruction and data.
- Processor uses stack to implement procedure calls.
- The program code are loaded into **Code Segment (CS)** of memory.
- Data are loaded into **Data segment (DS)** of memory
- Stack are loaded into **Stack Segment (SS)** of memory
- The 8086 uses four segment registers (CS,DS,SS, ES) to **hold segment numbers**.
- Any program needs access for second data segment may use Extra segment (ES).

Pointer and Index Registers

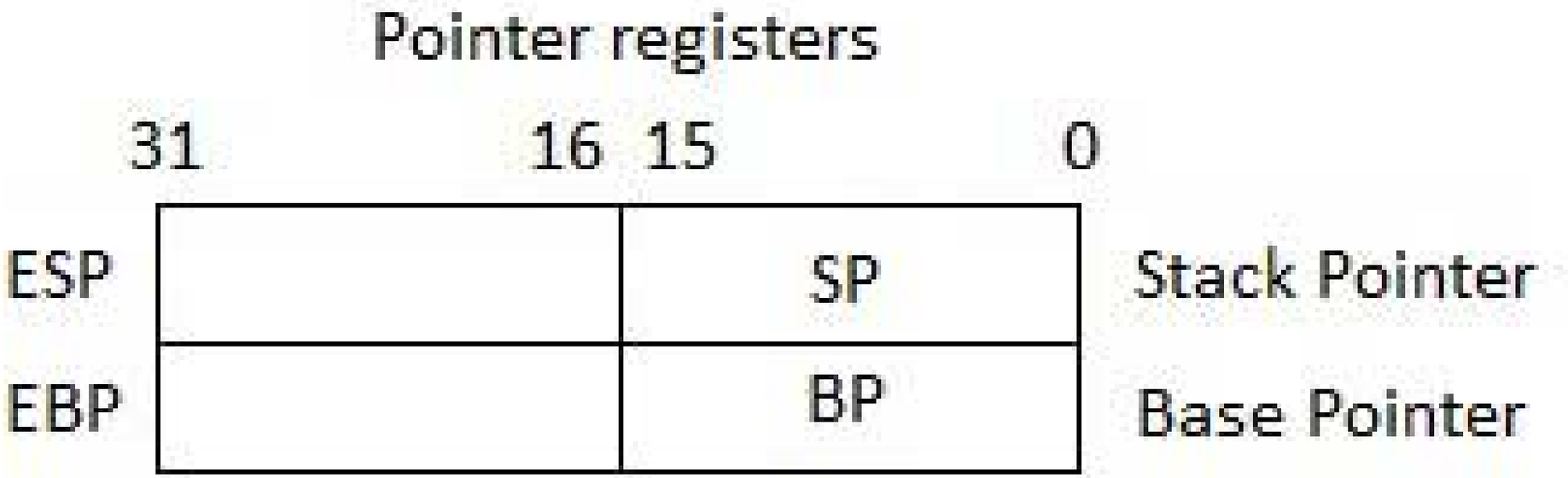
- The pointer registers are 32-bit EIP, ESP, and EBP registers and corresponding 16-bit right portions IP, SP, and BP. There are three categories of pointer registers −

Instruction Pointer (IP) − The 16-bit IP register stores the offset address of the next instruction to be executed. IP in association with the CS register (as CS:IP) gives the complete address of the current instruction in the code segment.

Stack Pointer (SP) − The 16-bit SP register provides the offset value within the program stack. SP in association with the SS register (SS:SP) refers to be current position of data or address within the program stack.

Base Pointer (BP) − The 16-bit BP register mainly helps in referencing the parameter variables passed to a subroutine. The address in SS register is combined with the offset in BP to get the location of the parameter. BP can also be combined with DI and SI as base register for special addressing.

Pointer and Index Registers



- The registers **SP,BP,SI and DI** usually **point** to the memory locations.

- Registers **contain the offset address** of Memory location

- Unlike segment registers **Index** registers can be used in **arithmetic** and other operations.

Stack Pointer (SP)

- The stack pointer (**SP**) is used **together with SS** to **access** the stack segment.

Base Pointer (BP)

- BP is used to access data on the stack.
- Unlike **SP, BP** can be used to access data in the other segments

Source Index (SI)

The 32-bit index registers, ESI and EDI, and their 16-bit rightmost portions. SI and DI, are used for indexed addressing and sometimes used in addition and subtraction. There are two sets of index pointers −

Source Index (SI) − It is used as source index for string operations.

Destination Index (DI) − It is used as destination index for string operations.

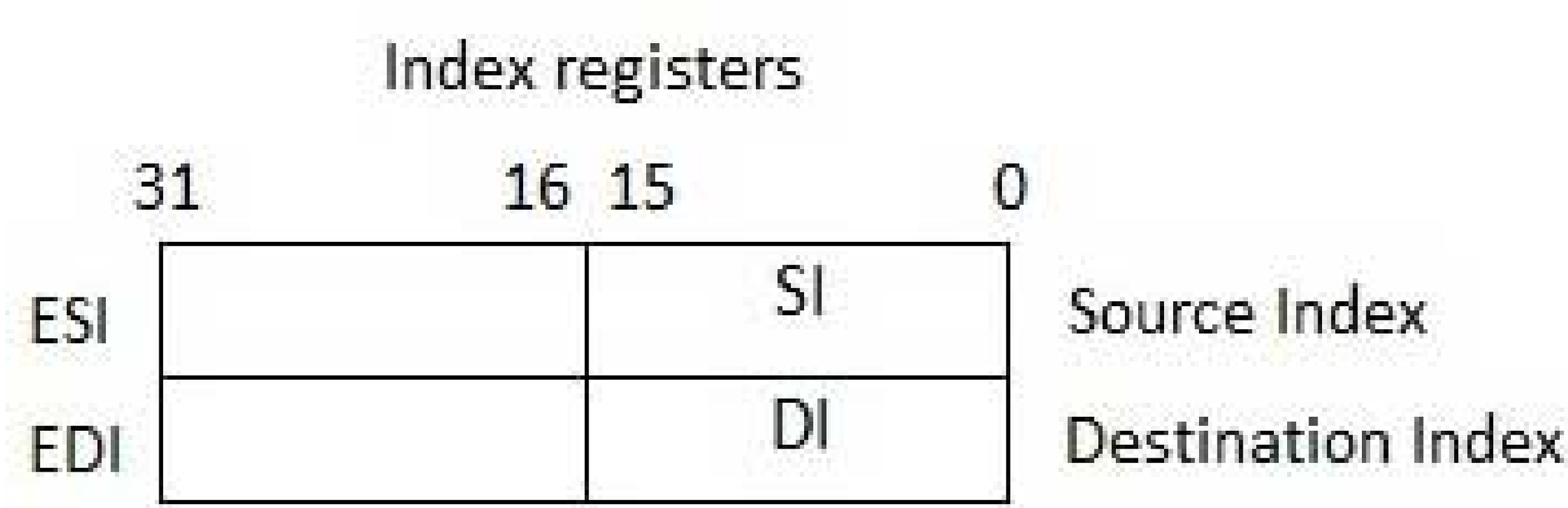
- **SI** is used to **point to memory locations** in the data segment addressed by **DS**. Consecutive memory locations can be accessed by **incrementing** the content of SI.

Destination Index (DI)

- **DI** is also used to point memory location.

- **String operations** use **DI** to access memory locations addressed by ES.

Index registers



Control Registers

The 32-bit instruction pointer register and the 32-bit flags register combined are considered as the control registers.

Many instructions involve comparisons and mathematical calculations and change the status of the flags and some other conditional instructions test the value of these status flags to take the control flow to other location.

The common flag bits are:

Flag:					O	D	I	T	S	Z		A		P		C
Bit no:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Control Registers

Overflow Flag (OF) − It indicates the overflow of a high-order bit (leftmost bit) of data after a signed arithmetic operation.

Direction Flag (DF) − It determines left or right direction for moving or comparing string data. When the DF value is 0, the string operation takes left-to-right direction and when the value is set to 1, the string operation takes right-to-left direction.

Interrupt Flag (IF) − It determines whether the external interrupts like keyboard entry, etc., are to be ignored or processed. It disables the external interrupt when the value is 0 and enables interrupts when set to 1.

Trap Flag (TF) − It allows setting the operation of the processor in single-step mode. The DEBUG program we used sets the trap flag, so we could step through the execution one instruction at a time.

Sign Flag (SF) − It shows the sign of the result of an arithmetic operation. This flag is set according to the sign of a data item following the arithmetic operation. The sign is indicated by the high-order of leftmost bit. A positive result clears the value of SF to 0 and negative result sets it to 1.

Control Registers

Zero Flag (ZF) − It indicates the result of an arithmetic or comparison operation. A nonzero result clears the zero flag to 0, and a zero result sets it to 1.
Auxiliary Carry Flag (AF) − It contains the carry from bit 3 to bit 4 following an arithmetic operation; used for specialized arithmetic. The AF is set when a 1-byte arithmetic operation causes a carry from bit 3 into bit 4.
Parity Flag (PF) − It indicates the total number of 1-bits in the result obtained from an arithmetic operation. An even number of 1-bits clears the parity flag to 0 and an odd number of 1-bits sets the parity flag to 1.
Carry Flag (CF) − It contains the carry of 0 or 1 from a high-order bit (leftmost) after an arithmetic operation. It also stores the contents of last bit of a *shift* or *rotate* operation.

Memory Recall (Bonus-1)

- **What is the primary difference between Index registers and segment registers?**

Instruction Pointer (IP)

- The memory registers are for **data access**.
- The 8086 uses **CS** and **IP** registers to **access instructions.**
- **CS** contains the **segment number** and IP contains the **offset** of next register.
- IP is updated each time after an instruction execution to **point to the next pointer**.
- Unlike other registers, IP can **not be directly** manipulated by an instruction. (i.e. an instruction may **not** contain IP as its **operand**.)

FLAGS Register

- **FLAGS** register is used to **indicate the status** of the microprocessor.
- Indication is done by setting of **individual bits** [flags].
- There are two kinds of FLAGS

  - **Status flags: Reflect the result** of an instruction executed by the processor. [More: chapter-5]

    e.g. If AX-BX results to **0**, the **ZF** (Zero Flag) is set to **1** (True)**.**

  - **Control flags: Enable** or **Disable** certain operations of the processor

    e.g. if **IF (Interrupt Flag)** is cleared (set to 0), inputs from keyboard are ignored by the processor. [More: chapter-11]

Execution Unit (EU)

- EU contains ALU circuits.
- ALU performs **arithmetic** and **logical** operations.
- **Data operations** are stored in **registers**.
- A **register** is like **memory location**, however, we refer to it by name not number.
  - **i.e. AX, BX, CX, DX, SI, DI, SP, BP**
- Also, EU Contains **temporary registers** for **holding operands** for the ALU and FLAGS registers.
- FLAG register's **individual bits** reflect the **result of computation**

Bus Interface Unit (BIU)

- BIU **enables communication** between the EU and memory or I/O circuits.

- Primarily responsible for transmitting address, data and control signals on the buses.

- BIU registers are: **CS,DS, ES and IP**
    - BIU registers hold the addresses of the memory locations

EU and BIU

- EU and BIU are connected by **internal bus** and they work together.

- While EU executes an Instruction, BIU fetches up to six bytes of the next instruction and places instructions in instruction queue (IQ).

- The overall process is called ***instruction prefetch*** and it’s purpose is to speed up the processor.

- However, if EU needs to communicate with memory, BIU suspends instruction prefetch and performs required operations.

Address Vs Contents

- The stored data in a memory byte are called **contents/value.**

Address	Contents
The address of a memory byte is **FIXED** and different from other addresses(**unique**).	Contents are **NOT** unique as they deal with current data.
The number of bits in an address depend on the processor [i.e. Intel 8086 = 20-bit & Intel 80286=24-bit]	Contents of memory byte are always 8 bits

Memory byte addressing

- Suppose a processor uses 20 bits for an address. How many memory bytes can be addressed using this processor?
  - A bit can have two possible values (i.e. 0 or 1)
  - So, in a 20-bit address, we can have **2^{20}** or **10,48,576**
- In computer terminology 2^{20} = 1 Mega
- Therefore, 20-bit address can be used to address **1 MB**.

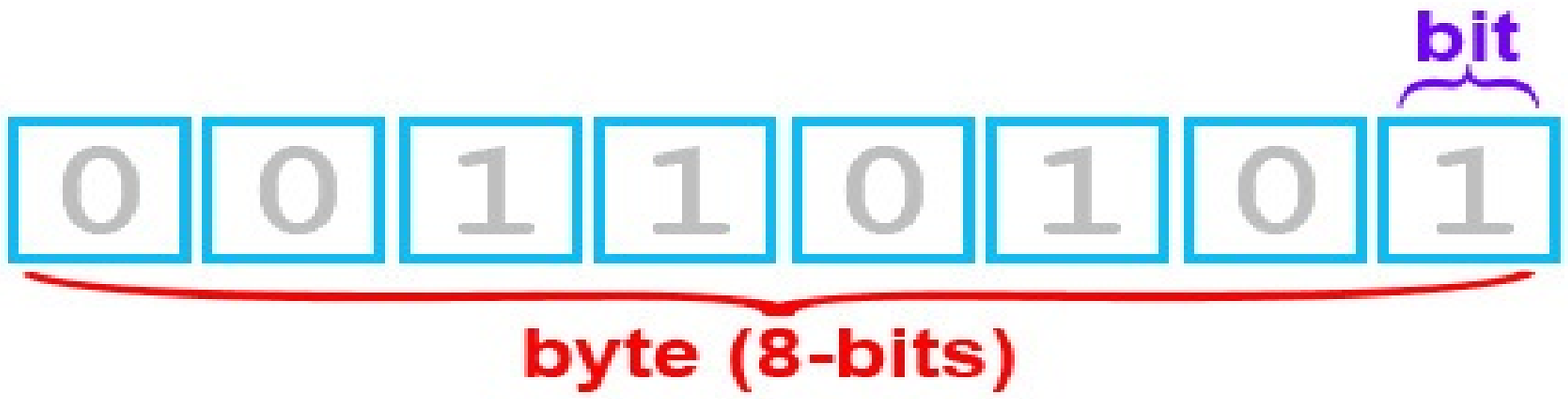
Memory Word

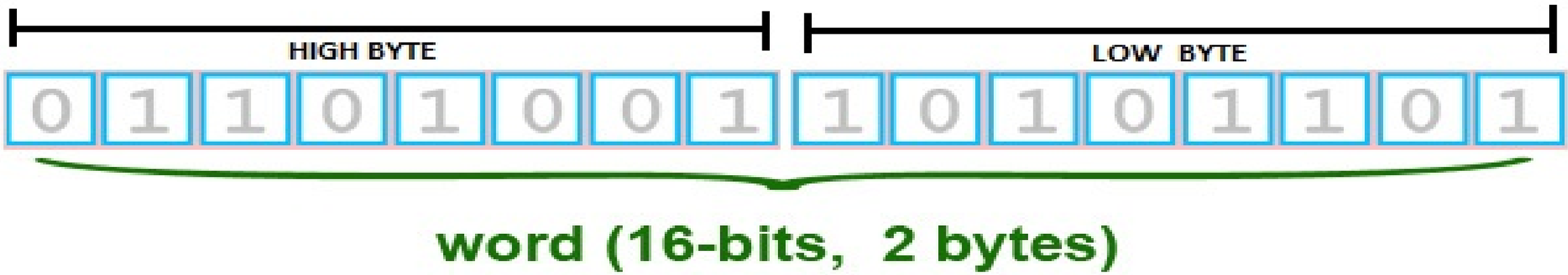
- In a Microcomputer, **Two bytes = a word**

- So to store a word data, IBM PC needs :
  - A pair of successive memory bytes
  - **A pair of memory bytes = Memory word**

- The **lower** address of the two memory bytes is the memory address.
  - i.e. a memory word with address 2 is made up of address 2 and 3
- A microprocessor can detect memory byte or memory word from memory **location/address.**

Bit Positions in byte and Word





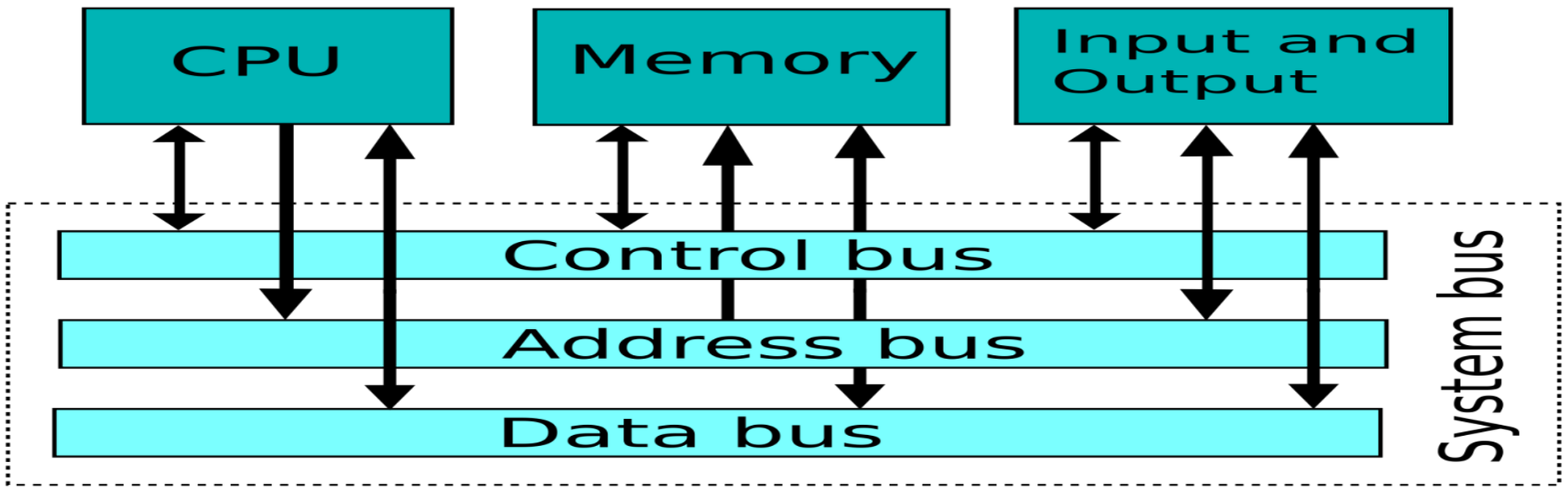
- Bit positions are numbered from **Right to left**
- **Bit 0-7 = low byte** [Lower address of word]
- **Bit 8-15 = high byte** [Higher address of word]

BUSES(cont'd…)

Address Bus: The CPU places the **address** of memory location on address bus to **read** the contents.

Data Bus: CPU receives the data, sent by memory circuits on the data bus.

Control Bus: CPU sends control signals on control bus perform read operation in memory.



Programming Languages

- **Machine Language:** Bit strings (i.e. 0 & 1)

- **Assembly language:**
  - Symbolic names are used to represent operations, registers and memory locations(i.e. MOV AX, A)
  - Assembly program must be converted into machine language using assembler.

- **High-Level language:**
  - Allows programmer to write program in more natural language text.
  - A Compiler is needed to translate high-level programs into machine language

Advantages

High-level

- Closer to natural language. So, algorithm conversion in easier.
- Less instruction and time required than assembly language.
- Programs can be executed in any machine

Assembly

- So close to the machine language. So programs are faster and shorter.
- Reading or writing to specific memory location, I/O ports is easy.
- It can be a sub program of a high-level language.

Simple Assembly Program

```
.MODEL SMALL
 .STACK 100H
 .DATA

 .CODE
 MAIN PROC
    MOV AH,1
    INT 21H

    MOV AH, 2
    MOV DL, "@“
    INT 21H

    MOV AH, 4CH
    INT 21H
 MAIN ENDP
END MAIN
```

```
DATA SEGMENT

DATA ENDS

CODE SEGMENT
   ASSUME CS:CODE, DS:DATA
START:
    MOV AH,1
     INT 21H
     MOV AH, 2
     MOV DL, "@“
     INT 21H

    MOV AH,4CH
    INT 21H
END START
CODE ENDS
```

Simple Assembly Program

```
.MODEL SMALL                          DATA SEGMENT
 .STACK 100H
 .DATA                                DATA ENDS

 .CODE                                CODE SEGMENT
 MAIN PROC                              ASSUME CS:CODE, DS:DATA
    MOV AH,1                          START:
    INT 21H                             MOV AH,1
                                         INT 21H
    MOV AH, 2                            MOV AH, 2
    MOV DL, AL    ; “@”                  MOV DL, AL     ; “@”
    INT 21H                              INT 21H

    MOV AH, 4CH                       MOV AH,4CH
    INT 21H                           INT 21H
 MAIN ENDP                          END START
END MAIN                            CODE ENDS
```

Simple Assembly Program

NEW LINE

```
MOV AH, 2                    ; carriage return
MOV DL, 0DH
INT 21H
MOV DL, 0AH                   ; line feed
INT 21H
```

Space in between

```
mov dl, ' '
mov ah, 2
int 21h
```

Simple Assembly Program

```
.MODEL SMALL
 .STACK 100H
 .DATA
     MSG DB "HELLO WORLD$"
 .CODE
 MAIN PROC
    MOV AX,DATA    ; @DATA
    MOV DS,AX
    MOV AH,09H
    MOV DX,OFFSET MSG
         ; LEA DX,MSG
    INT 21H
    MOV AH,4CH
    INT 21H
 MAIN ENDP
END MAIN
```

```
DATA SEGMENT
    MSG DB "HELLO WORLD$"
DATA ENDS

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START:
    MOV AX,DATA     ; @DATA
    MOV DS,AX
    MOV AH,09H
    MOV DX,OFFSET MSG
         ; LEA DX,MSG
    INT 21H
    MOV AH,4CH
    INT 21H

END START
CODE ENDS
```

Simple Assembly Program

```
.MODEL SMALL
.STACK 100H
.DATA
A DB 2
B DB 3
.CODE
MAIN PROC
    MOV AX,DATA
    MOV DS,AX

    MOV AH,2
    MOV DL,A
    INT 21H

    MOV AH,2
    MOV DL,B
    INT 21H

    MOV AH,4CH
    INT 21H

MAIN ENDP
END MAIN
```

```
DATA SEGMENT
    A DB 2
    B DB 3
DATA ENDS

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START:
    MOV AX,DATA
    MOV DS,AX


    MOV AH,2
    MOV DL,A
    INT 21H

    MOV AH,2
    MOV DL,B
    INT 21H

    MOV AH,4CH
    INT 21H

END START
CODE ENDS
```

Simple Assembly Program

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC
MOV AX,DATA
   MOV DS,AX

   MOV AH,1
   INT 21H

   MOV AH,2
   MOV DL,AL
   INT 21H

   MOV AH,1
   INT 21H

   MOV AH,2
   MOV DL,AL
   INT 21H

   MOV AH,4CH
   INT 21H

MAIN ENDP
END MAIN
```

```
DATA SEGMENT
DATA ENDS

CODE SEGMENT
   ASSUME CS:CODE, DS:DATA
START:
   MOV AX,DATA
   MOV DS,AX

   MOV AH,1
   INT 21H

   MOV AH,2
   MOV DL,AL
   INT 21H

   MOV AH,1
   INT 21H

   MOV AH,2
   MOV DL,AL
   INT 21H

   MOV AH,4CH
   INT 21H
END START
CODE ENDS
```

Simple Assembly Program

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC
    MOV AX,DATA
   MOV DS,AX

   MOV AH,1
   INT 21H

MOV BL,AL

mov dl, ' '
mov ah, 2
int 21h

MOV AH,2
MOV DL,BL
INT 21H
```

```
DATA SEGMENT
DATA ENDS
CODE SEGMENT
   ASSUME CS:CODE, DS:DATA
START:
   MOV AX,DATA
   MOV DS,AX

   MOV AH,1
   INT 21H

MOV BL,AL

mov dl, ' '
mov ah, 2
int 21h

MOV AH,2
MOV DL,BL
INT 21H
```

Simple Assembly Program

```
MOV AH, 2                    ; carriage return
MOV DL, 0DH
INT 21H
MOV DL, 0AH                  ; line feed
INT 21H

MOV AH,1
INT 21H
MOV BL,AL

mov dl, ' '
mov ah, 2
int 21h

MOV AH,2
MOV DL,BL
INT 21H

MOV AH,4CH
INT 21H

MAIN ENDP
END MAIN
```

```
MOV AH, 2                    ; carriage return
   MOV DL, 0DH
   INT 21H
   MOV DL, 0AH               ; line feed
   INT 21H

MOV AH,1
   INT 21H
   MOV BL,AL

   mov dl, ' '
   mov ah, 2
   int 21h

   MOV AH,2
   MOV DL,BL
   INT 21H

   MOV AH,4CH
   INT 21H

END START
CODE ENDS
```

Thanks !!!