MICROPROCESOR AND INTERFACING

(Internal Architecture)

Md. Sabbir Ejaz
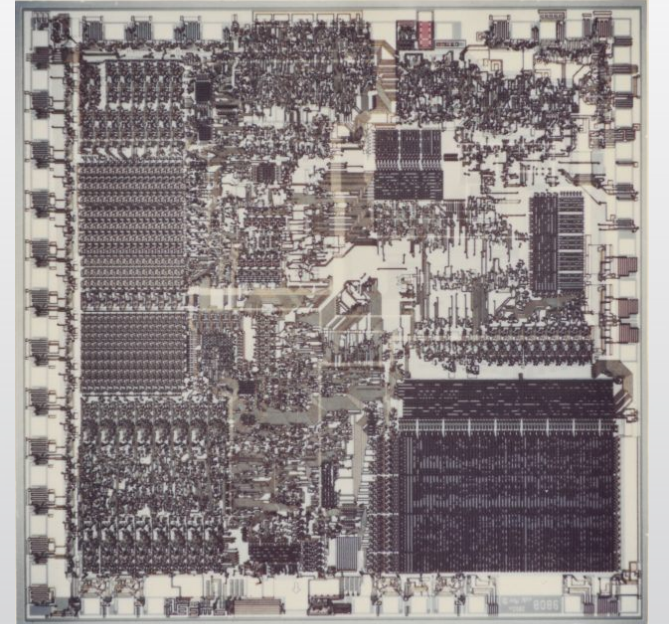
Basic Concepts of Microprocessors

What is Microprocessor?

The word comes from the combination **Micro and Processor**. Processor means a device that processes whatever.

The microprocessor is a programmable device that takes in numbers, performs on them arithmetic or logical operations according to the program stored in memory and then produces other numbers as a result.
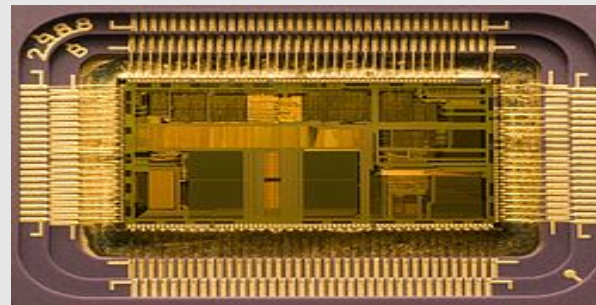
Basic Concepts of Microprocessors

Differences between:

- Microprocessor – silicon chip which includes ALU, register circuits & control circuits. It is a general purpose chip.

- Microcomputer – a computer with a microprocessor as its CPU. Includes memory, I/O etc.

- Microcontroller – silicon chip which includes microprocessor, memory & I/O in a single package, designed for a very specific task.

A Microprocessor-based System

- From the above description, we can draw the following block diagram to represent a microprocessor-based system:
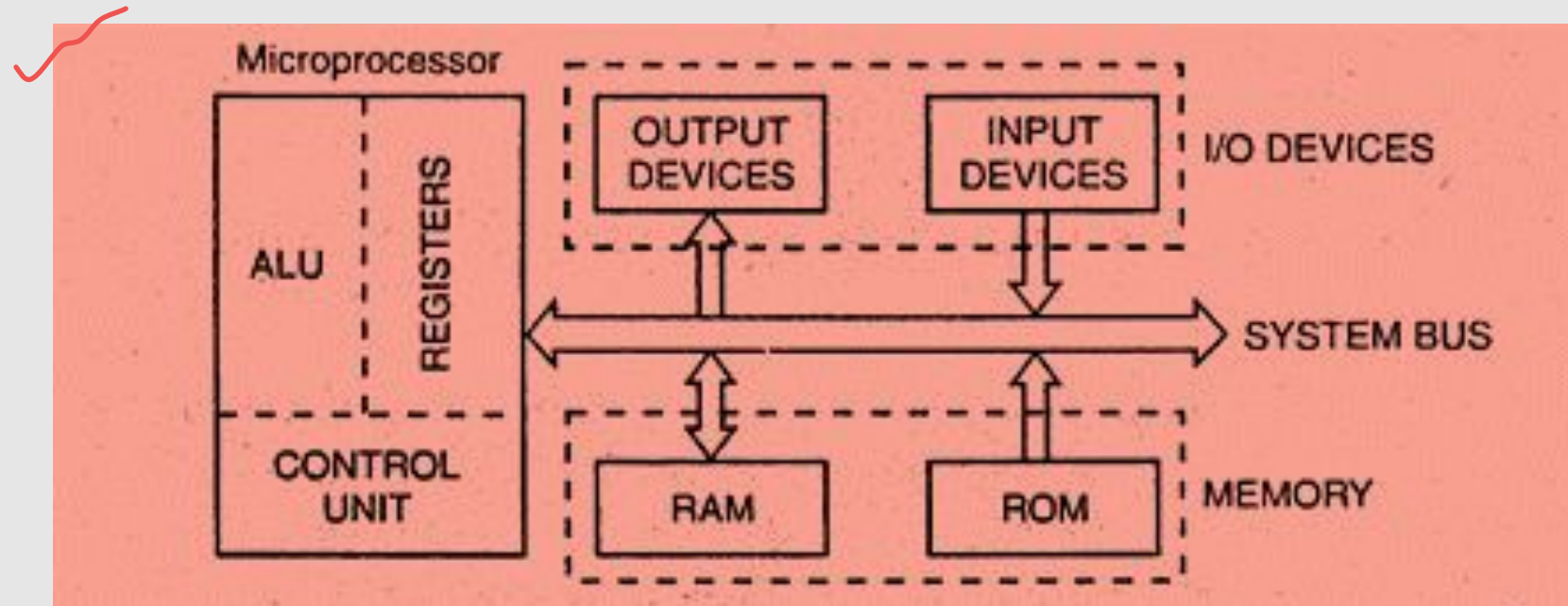

Fig: Block diagram of microprocessor based system

Architecture of Computers

- Von Neumann Architecture (stored program concept)
  - Only one main memory
  - fetch information through *program counter register* is machine instruction
  - fetch information with address specified through other register is data

- Harvard Architecture
  - Two memories (instruction & data)

Self Study:
  - Hall—chp1: fig: 1.8,1.9, 1.10, RISC vs CISC
  - Marut— chp 1,3
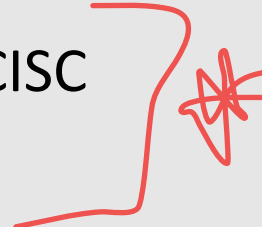
Data Bus

The CPU is connected to memory and I/O device through a strip of wires called a bus.

The data bus is '**bi-directional**'

- data or instruction codes from memory or input/output are transferred into the microprocessor

- the result of an operation or computation is sent out from the microprocessor to the memory or input/output.

- depending on the particular microprocessor, the data bus can handle 8 bit(8085) or 16 bit(8086) data.

Address Bus

The address bus is '**unidirectional**'

- Over which the microprocessor sends an address code to the memory or input/output.

- The size (width) of the address bus is specified by the number of bits it can handle.

- The more bits there are in the address bus, the more memory locations a microprocessor can access.

- A 20 bit address bus(8086) is capable of addressing 2^{20} (1MB) addresses.

Control Bus

The control bus is '**unidirectional**'

- Control bus carries commands from the CPU

- Define the address on the bus is memory address or I/O device address

- Microprocessor uses control bus to process data, that is what to do
  with the selected memory location

- Some control signals are Memory Read, Memory Write, I/O Read, I/O Write

- Sometimes returns status signals from the devices (bi-directional)

Bus System
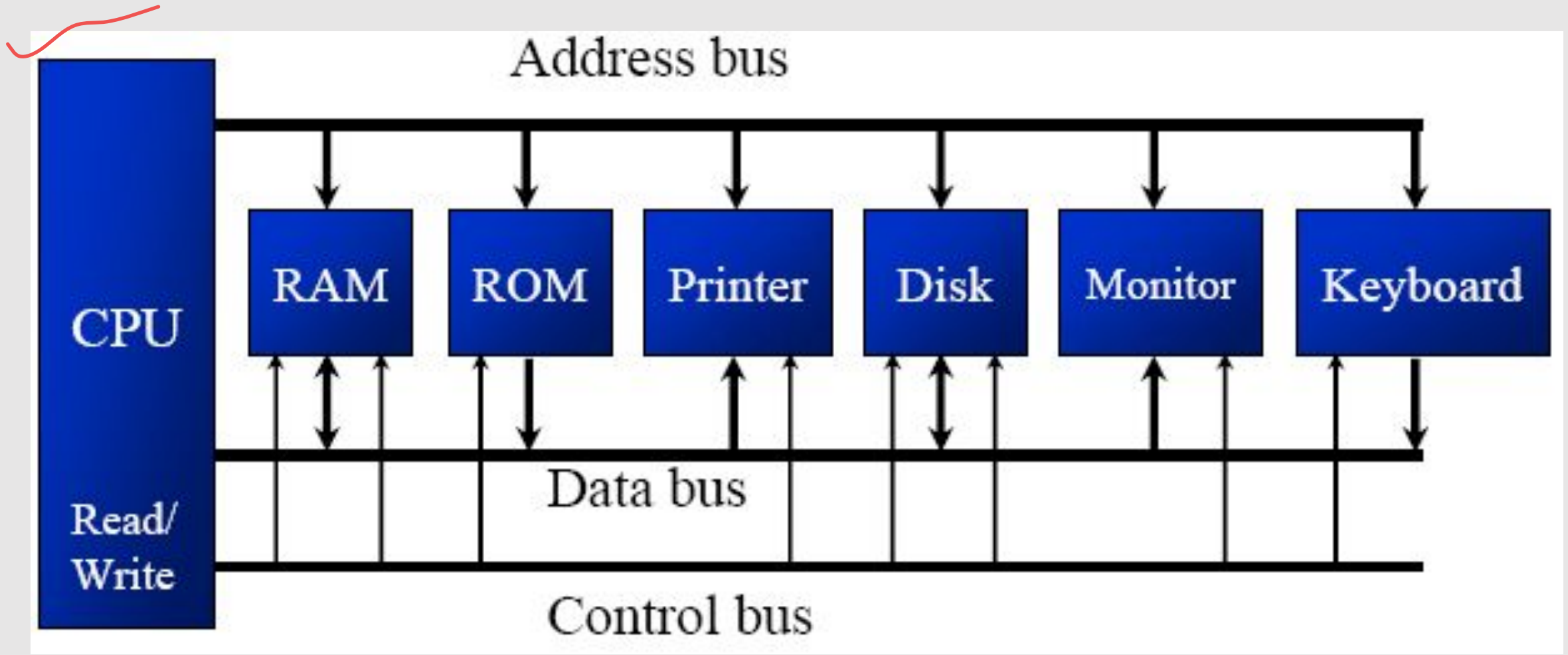
Intel 8086 Microprocessor
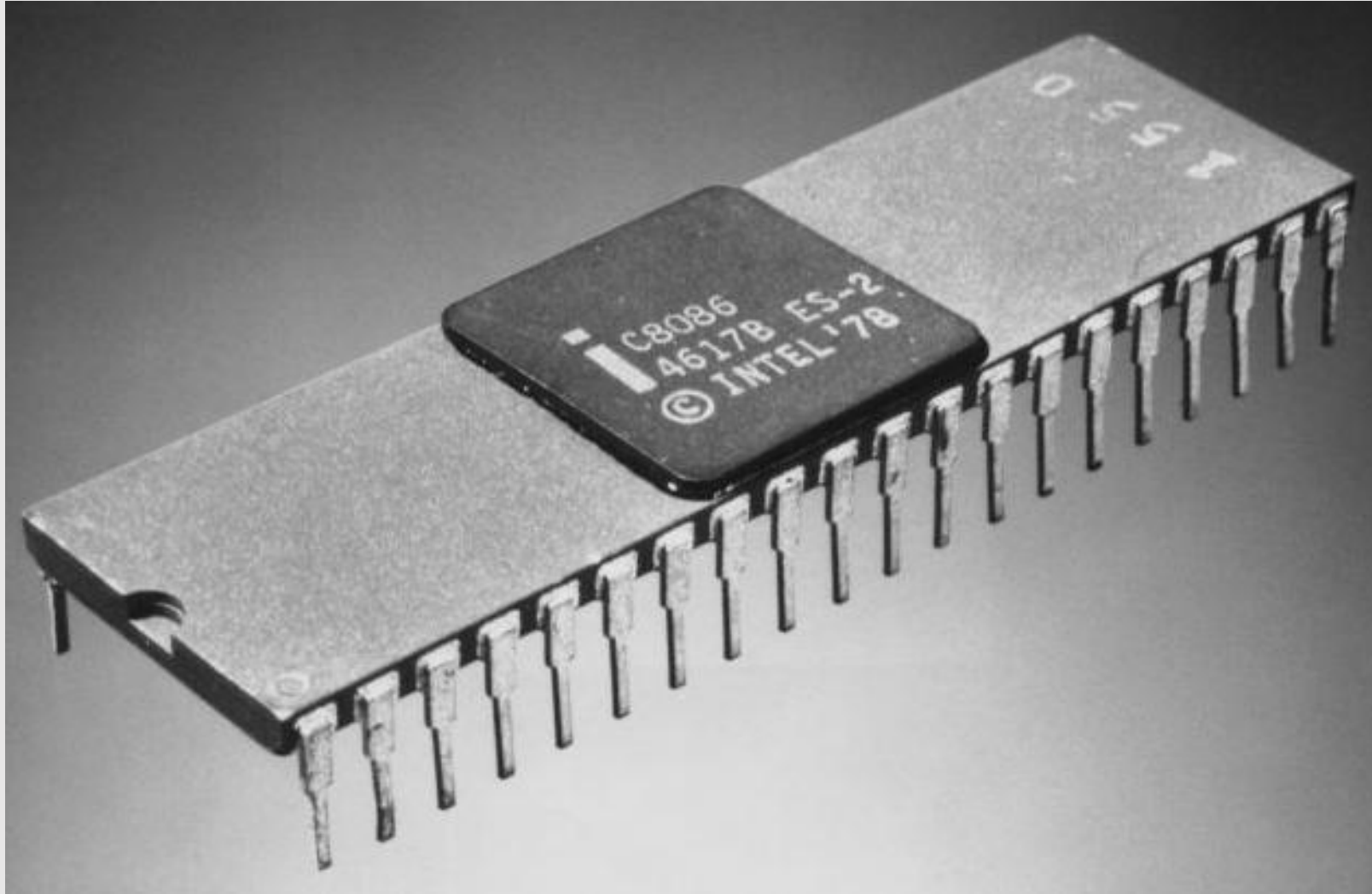
Features

- It is a 16-bit μp.

- It provides 16-bit registers.

- Word size is 16 bits.

- 8086 has a 20 bit address bus can access up to 2^{20} memory locations (1 MB).

- Address ranges from 00000H to FFFFFH

- Memory is byte addressable - Every byte has a separate address.

- It can support up to 64K I/O ports.

Features (Cont.)

- A 40 pin dual in line package.

- It has multiplexed address and data bus AD0-AD15 and A16-A19.

- 8086 is designed to operate in two modes, Minimum and Maximum.

- It can pre-fetches up to 6 instruction bytes from memory and queues them in order to speed up instruction execution.

- It requires +5V power supply.
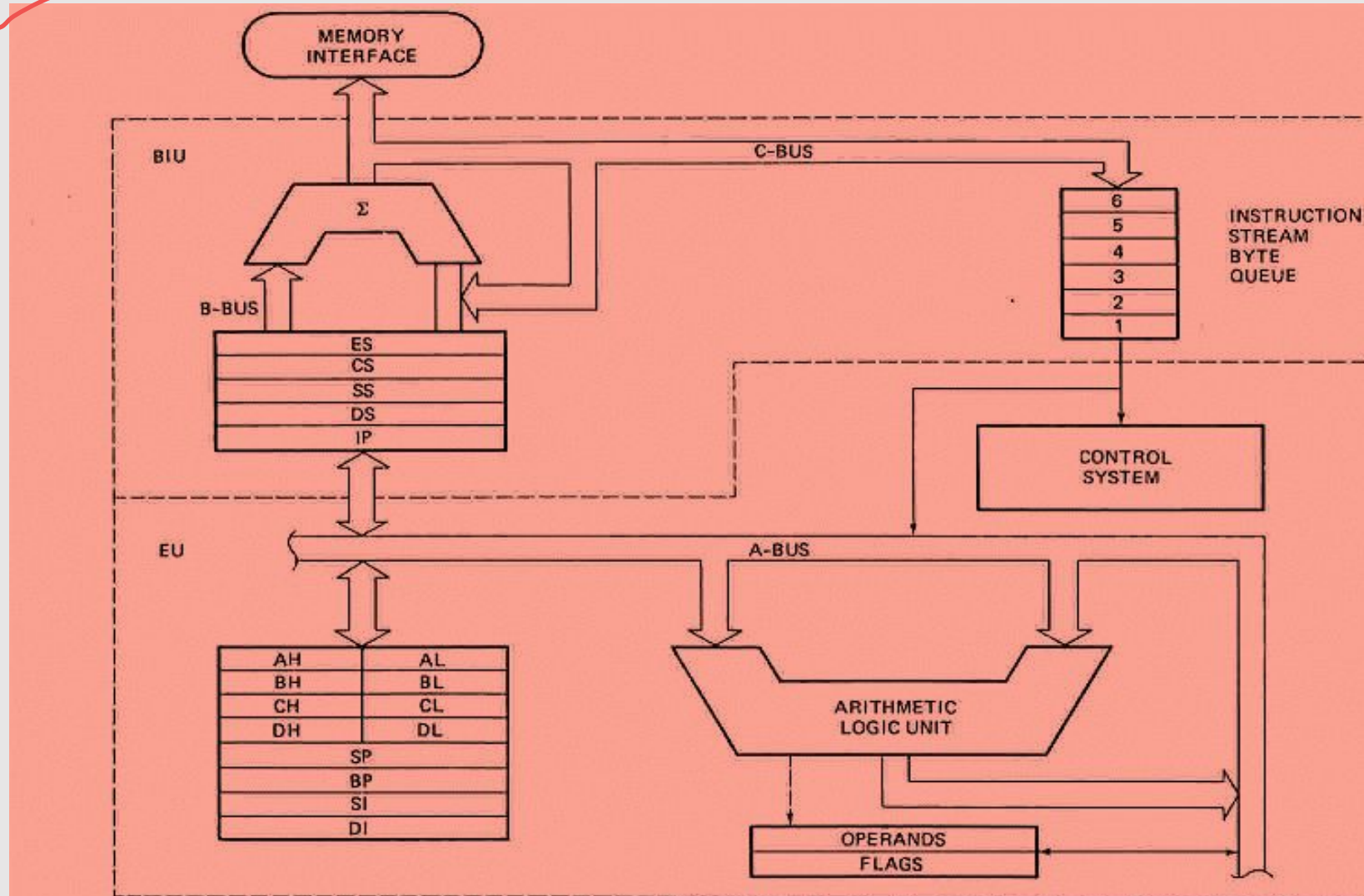
Intel 8086 Internal Architecture

FIGURE   8086 internal block diagram. (*Intel Corp.*)

Internal Architecture of 8086

Bus Interface Unit   Execution Unit

- 8086 has two blocks BIU and EU.

- The BIU handles all transactions of data and addresses on the buses for EU.

- The BIU performs all bus operations such as
  - Sends out address for memory locations
  - Fetches Instructions from memory
  - Read/Writes data to memory
  - Sends out addresses for I/O ports
  - Reads/Write data to I/O ports
  - The instruction bytes are transferred to the instruction queue.

Internal Architecture of 8086 (Cont.)

- EU performs the following functions--
  - Logic and arithmetic operation on memory or register
  - Receives the instruction from pre fetch queue and decodes it
  - Executes instructions from the instruction system byte queue.
  - Stores the information temporary in the register array

- Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining. This results in efficient use of the system bus and system performance.
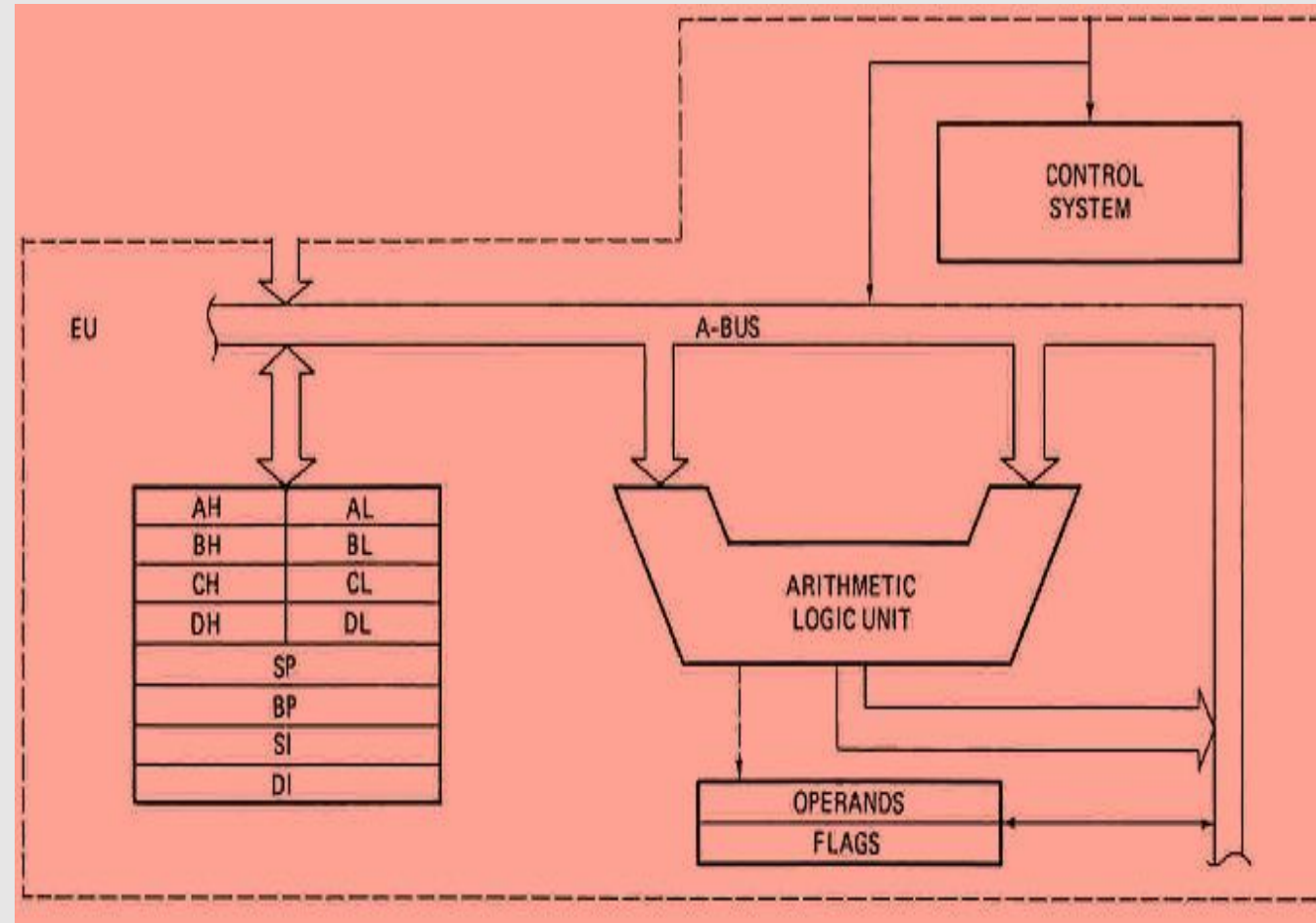
Internal Architecture of 8086 (Cont.)

- BIU contains
  - Instruction queue
  - Segment registers
  - Instruction pointer
  - Address adder.

- EU contains
  - Control circuitry
  - Instruction decoder
  - ALU
  - General purpose register
  - Pointer and Index register
  - Flag register.

Execution Unit

- Control System has two component
  - Instruction decoder-- works to translate or decode instructions which are fetched from the memory into a series of actions which EU caries out
  - Control unit-- generate timing and control signals to perform internal operation

- ALU is a 16 bit unit which performs the AND, OR, Exclusive, Addition, Subtraction, Increment, Decrement, Complement and Shift function.
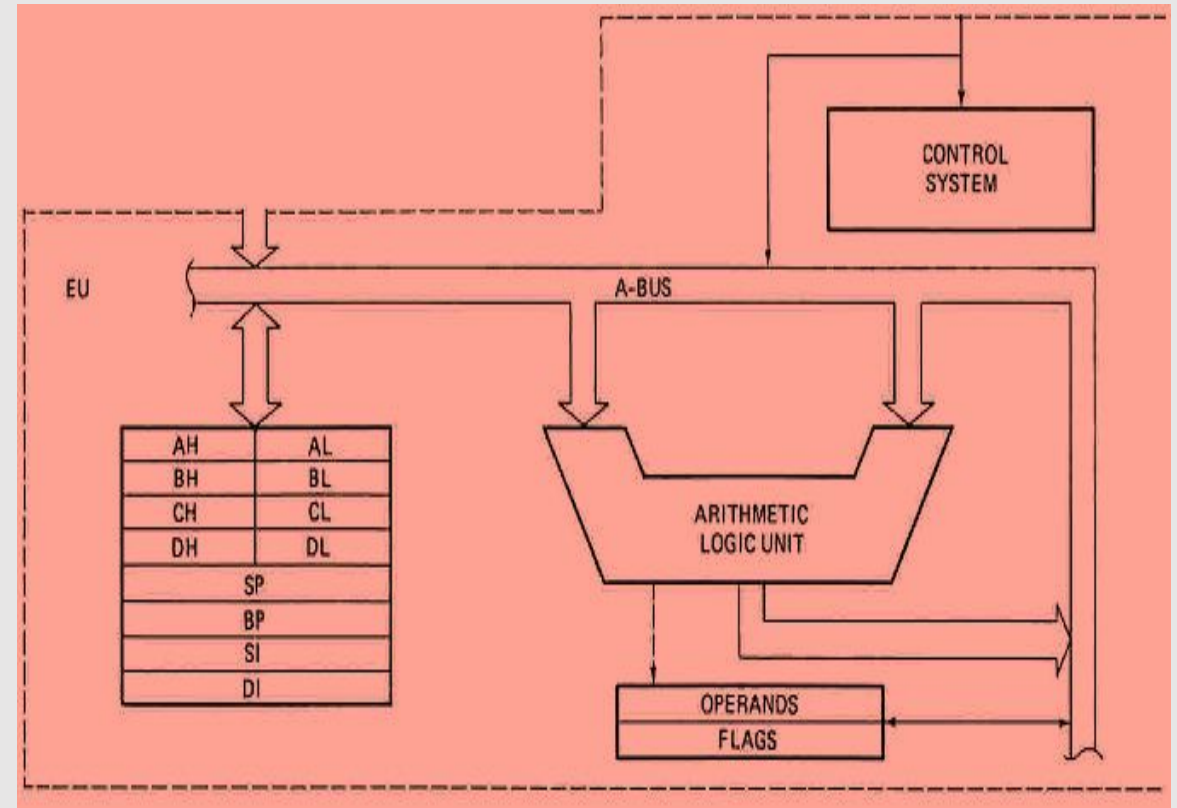
General Purpose Register

- EU has 8 general purpose register which are named AL, AH, BL, BH, CL, CH, DL, DH and 8 bits in size.

- Individually can store 8 bit data

- They can also be used as 16 bit registers when we take the different pairs of these registers. The possible pairs are--

    - AL, AH = AX
    - BL, BH = BX
    - CL, CH = CX
    - DL, DH = DX
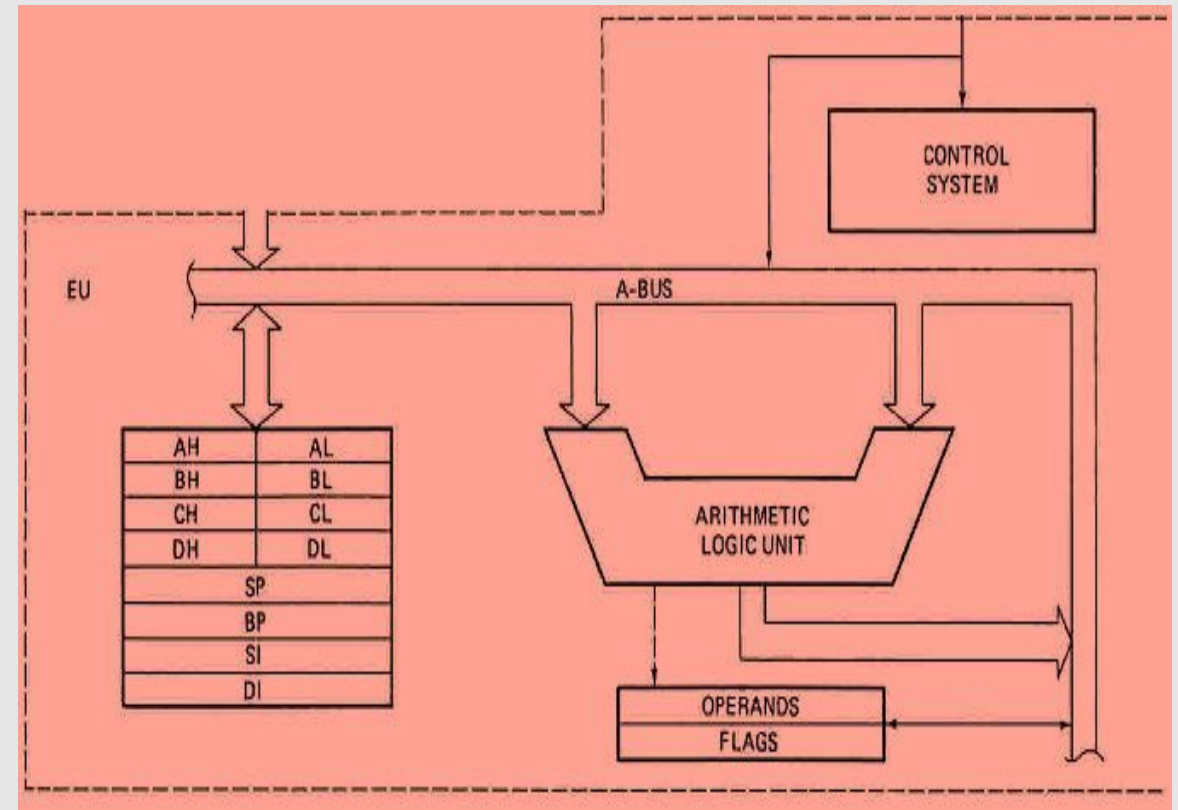
Execution Unit – Registers

16 bits

8 bits | 8 bits

AH	AL	Accumulator
BH	BL	Base
CH	CL	Count
DH	DL	Data
SP		Stack Pointer
BP		Base Pointer
SI		Source Index
DI		Destination Index

General Purpose register
AX
BX
CX
DX

Pointer

Index
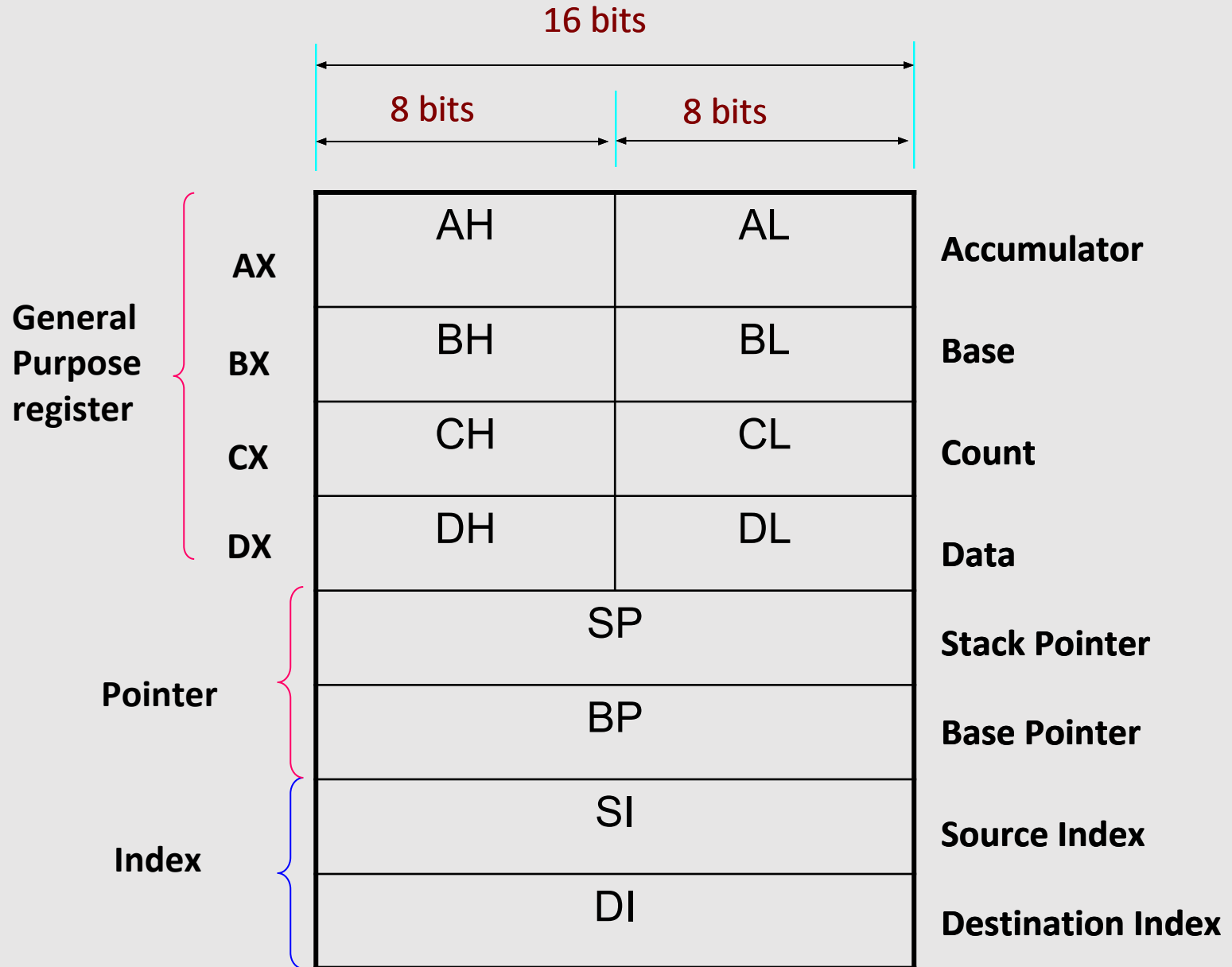
General Purpose Registers

Accumulator Register (AX)

- Consist of two 8-bit registers AL and AH, to form 16-bit register AX
- The I/O instructions use the AX or AL for inputting/outputting 16 or 8 bit data to or from an I/O port
- Multiplication and Division instructions also use the AX or AL

AH	AL	AX
BH	BL	BX
CH	CL	CX
DH	DL	DX

General Purpose Registers

Base Register (BX)

- Consist of two 8-bit registers BL and BH, to form 16-bit register BX
- This is the only general purpose register whose contents can be used for addressing the 8086 memory
- All memory references utilizing this register content for addressing use DS as the default segment register.

AH	AL	AX
BH	BL	BX
CH	CL	CX
DH	DL	DX

General Purpose Registers

Counter Register (CX)

- Consist of two 8-bit registers CL and CH, to form 16-bit register CX
- Instructions such as SHIFT, ROTATE and LOOP use the contents of CX as a counter

AH	AL	AX
BH	BL	BX
CH	CL	CX
DH	DL	DX

General Purpose Registers

Data Register (DX)

- Consist of two 8-bit registers DL and DH, to form 16-bit register DX
- Use to hold the high 16-bit result (data) in 16 X 16 multiplication or the high 16-bit dividend (data) before a 32/16 division and the 16-bit reminder after division

AH	AL	AX
BH	BL	BX
CH	CL	CX
DH	DL	DX

General Purpose Registers

Register	Purpose
AX	Arithmetic/logic/ I/O operation, Word multiply, word divide
AL	Byte multiply, byte divide, byte I/O, decimal arithmetic
AH	Byte multiply, byte divide
BX	Store address information
CX	String operation, loops
CL	Count in instruction that shift and rotate
DX	Word multiply, word divide, indirect I/O (Used to hold I/O address during I/O instructions. If the result is more than 16-bits, the lower order 16-bits are stored in accumulator and higher order 16-bits are stored in DX register)

Pointer and Index Register

Stack Pointer(SP) and Base Pointer(BP)
- SP and BP are used to access data in stack segment
- SP is used as an offset from the current Stack Segment Register
- SP contents are automatically updated(incremented/decremented) due to execution of a POP or PUSH instruction
- BP contains an offset address in the current Stack Segment Register

SP
BP
SI
DI

Pointer and Index Register

Source Index(SI) and Destination Index(DI)

- Used in indexed address
- SI register is used to point to memory locations in the data segment address by DS
- Same as SI register. There is a class of instructions called string operations that use DI to access memory locations address by ES
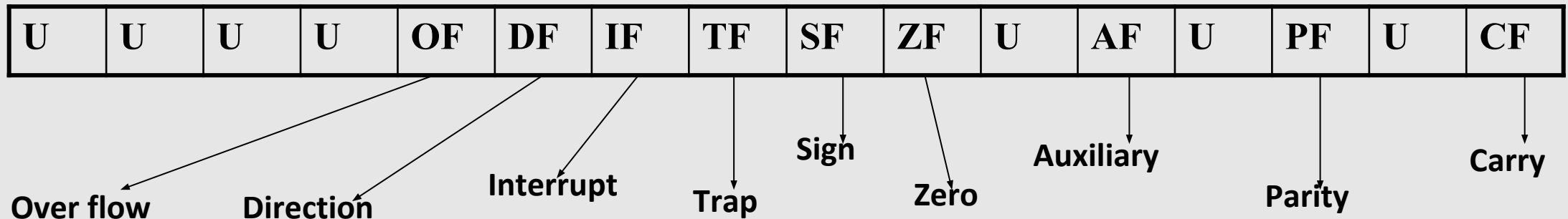
SP
BP
SI
DI

Flag Register

- A flag is a flip flop which indicates some conditions produced by the execution of an instruction or controls certain operations of the EU .

- Two type of flag register exist in 8086
    - Conditional flags : represent result of last arithmetic or logic instruction executed
    - Control flags : to control the certain operation of EU

Flag Register

- In 8086 The EU contains
  - ☐ a 16 bit flag register
  - ☐ 9 of the 16 are active flags and remaining 7 are undefined.
  - ☐ 6 flags indicates some conditions- status flags
  - ☐ 3 flags –control Flags

U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

Over flow   Direction   Interrupt   Trap   Sign   Zero   Auxiliary   Parity   Carry

U - Unused

Flag Register

Flag	Purpose
Carry (CF)	Holds the carry after addition or the borrow after subtraction. Also indicates some error conditions, as dictated by some programs and procedures .
Parity (PF)	PF=0;odd number of 1, PF=1;even number of 1.
Auxiliary (AF)	Holds the carry (half – carry) after addition or borrow after subtraction between bit positions 3 and 4 of the result (for example, in BCD addition or subtraction.)
Zero (ZF)	Shows the result of the arithmetic or logic operation. ZF=1; result is zero. ZF=0; The result is 6 Not 0
Sign (SF)	Holds the sign of the result after an arithmetic/logic instruction execution. SF=1; negative, SF=0; positive

Flag Register

Flag	Purpose
Trap (TF) A control flag	If set then processor enters the single step execution mode by generating internal interrupts after the execution of each instruction (debugging)
Interrupt (IF) A control flag	Controls the operation of the INTR (interrupt request) IF=0; INTR pin disabled. IF=1; INTR pin enabled. Maskable interrupts are recognized by the CPU(IF=1), otherwise they are ignored.
Direction (DF) A control flag	It selects either the increment(DF=0) or decrement(DF=1) mode for string manipulation. Lowest address to highest is incrementing.
Overflow (OF)	An overflow indicates the result has exceeded the capacity of the machine. Overflow occurs when signed numbers are added or subtracted.

Flag Register (OF)

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

- 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) OF=1

- 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) OF=1

Otherwise the "overflow" flag is turned off

- 0100 + 0001 = 0101 (overflow flag is turned off) OF=0
- 0110 + 1001 = 1111 (overflow flag turned off) OF=0
- 1000 + 0001 = 1001 (overflow flag turned off) OF=0
- 1100 + 1100 = 1000 (overflow flag is turned off) OF=0

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.

Execution Unit – Flag 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

```
7F = 0 1 1 1  1 1 1 1
                  +1
-----------------------
80 = 1 0 0 0  0 0 0 0
```

34F5 + 95EB = ?

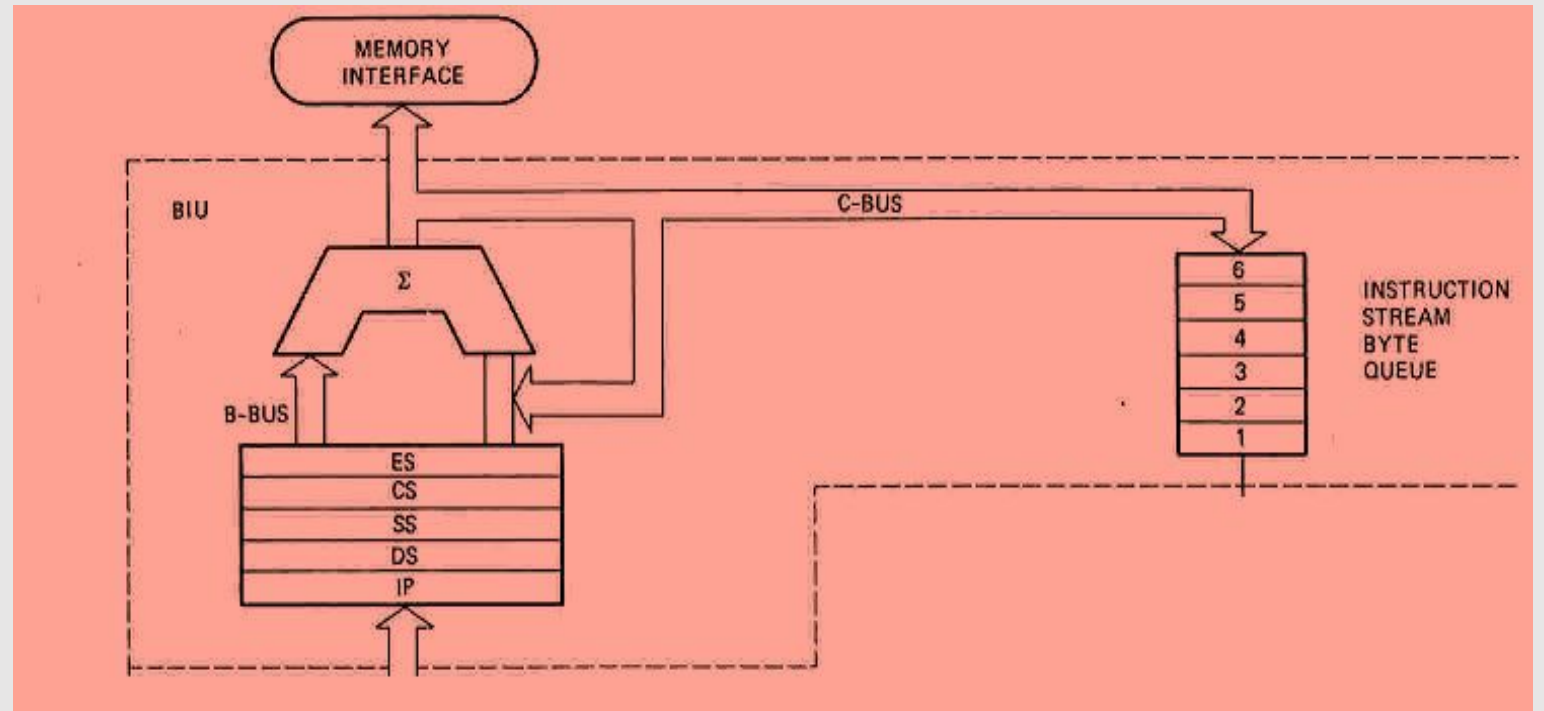**CF = 0 ; PF = 0 ; AF = 1**
ZF = 0 ; SF = 1 ; OF = 0

Example

- 35 + 19 = ?
- 35 + 5B = ?
- 35 + D3 = ?
- 9E + D3 = ?

```
mov  AL,43
add  AL,94
```

Bus Interface Unit (BIU)

Contains

- 6-byte Instruction Queue (Q)
- The Segment Registers (CS, DS, ES, SS)
- The Instruction Pointer (IP)
- The Address Summing block (Σ)

The Queue

- The execution unit (EU) is supposed to decode or execute an instruction.

- Decoding does not require the use of buses.

- When EU is busy in decoding and executing an instruction, the BIU fetches up to six instruction bytes for the next instructions.

- These bytes are called as the pre-fetched bytes and they are stored in a first in first out (FIFO) register set, which is called as a queue.

- With its **16** bit data bus, the BIU fetches two instruction bytes in a single memory cycle.

Significance of Queue

Significance of queue
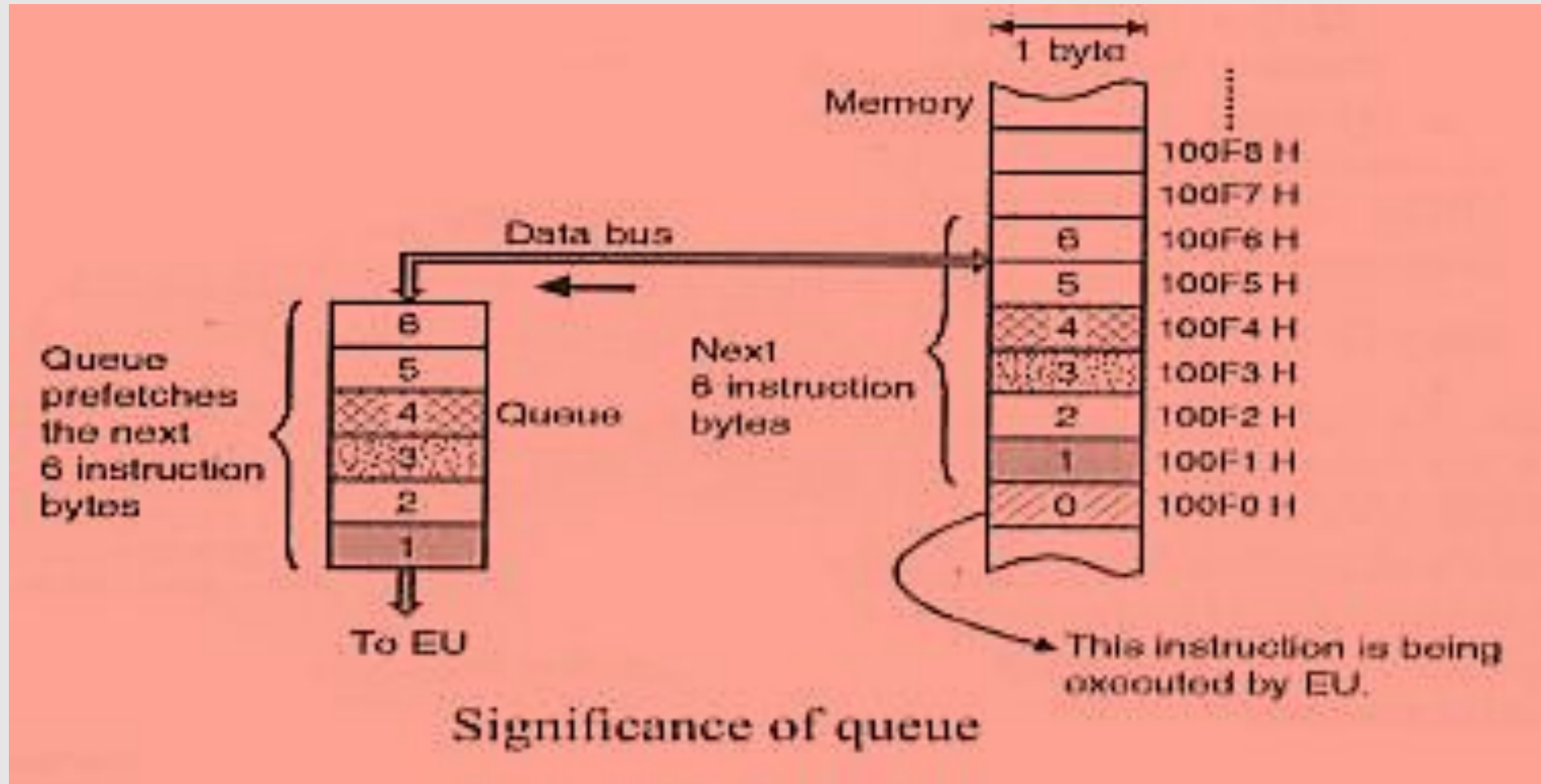
The Queue

- As shown in the above figure, while the EU is busy in decoding the instruction corresponding to memory location 100F0, the BIU fetches the next six instruction bytes from locations 100F1 to 100F6 numbered as 1 to 6.

- These instruction bytes are stored in the 6 byte queue on the first in first out (FIFO) basis.

- When EU completes the execution of the existing instruction and becomes ready for the next instruction, it simply reads the instruction bytes in the sequence 1, 2…. from the Queue.

- Thus the Queue will always hold the instruction bytes of the next instructions to be executed by the EU.

The Queue

Pipelining:

- The process of fetching the next instruction when the present instruction is being executed is called as pipelining.
- Pipelining has become possible due to the use of queue.
- BIU (Bus Interfacing Unit) fills in the queue until the entire queue is full.
- BIU restarts filling in the queue when at least two locations of queue are vacant.

The Queue

Advantages of pipelining:

- The execution unit always reads the next instruction byte from the queue in BIU. This is faster than sending out an address to the memory and waiting for the next instruction byte to come.

- In short pipelining eliminates the waiting time of EU and speeds up the processing.

- The 8086 BIU will not initiate a fetch unless and until there are two empty bytes in its queue. 8086 BIU normally obtains two instruction bytes per fetch.

The Queue

8086 Queue is only Six Byte long:

- This is because the longest instruction in the instruction set of 8086 is six byte long.

- Hence with a six byte long queue it is possible to pre-fetch even the longest instruction in the main program.
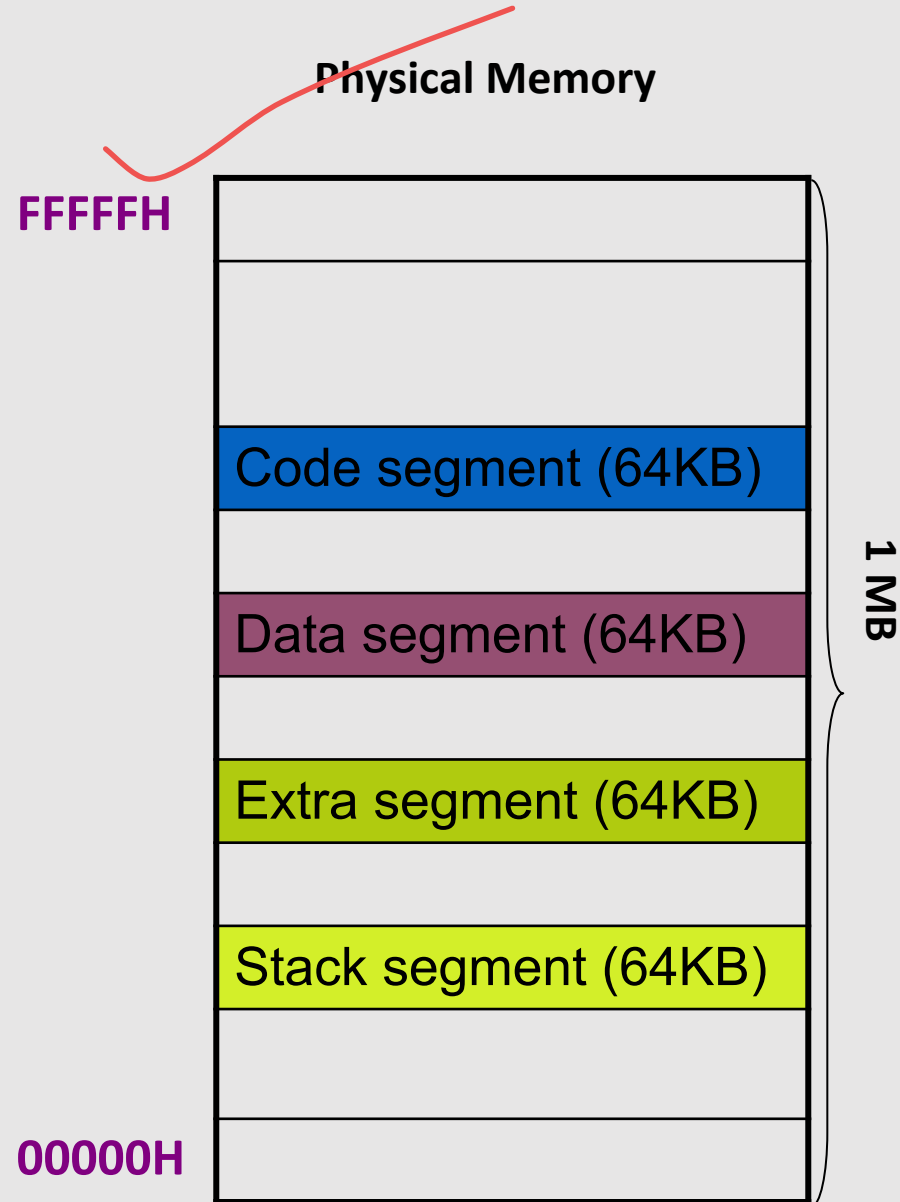
Memory Segmentation

- 8086 has a 20-bit address bus. So it can address a maximum of 1 MB of memory

- Physical available memory is divided into a number of logical segments

- 8086 can work with only 4 segments are Code, Data, Extra and Stack segments

- The size of each segment is 64 KB

- A segment may be located any where in the memory. Starting addresses will always be changing. They are not fixed

- Each segment is made up of memory contiguous memory locations. It is independent, separately addressable unit.

Memory Segmentation
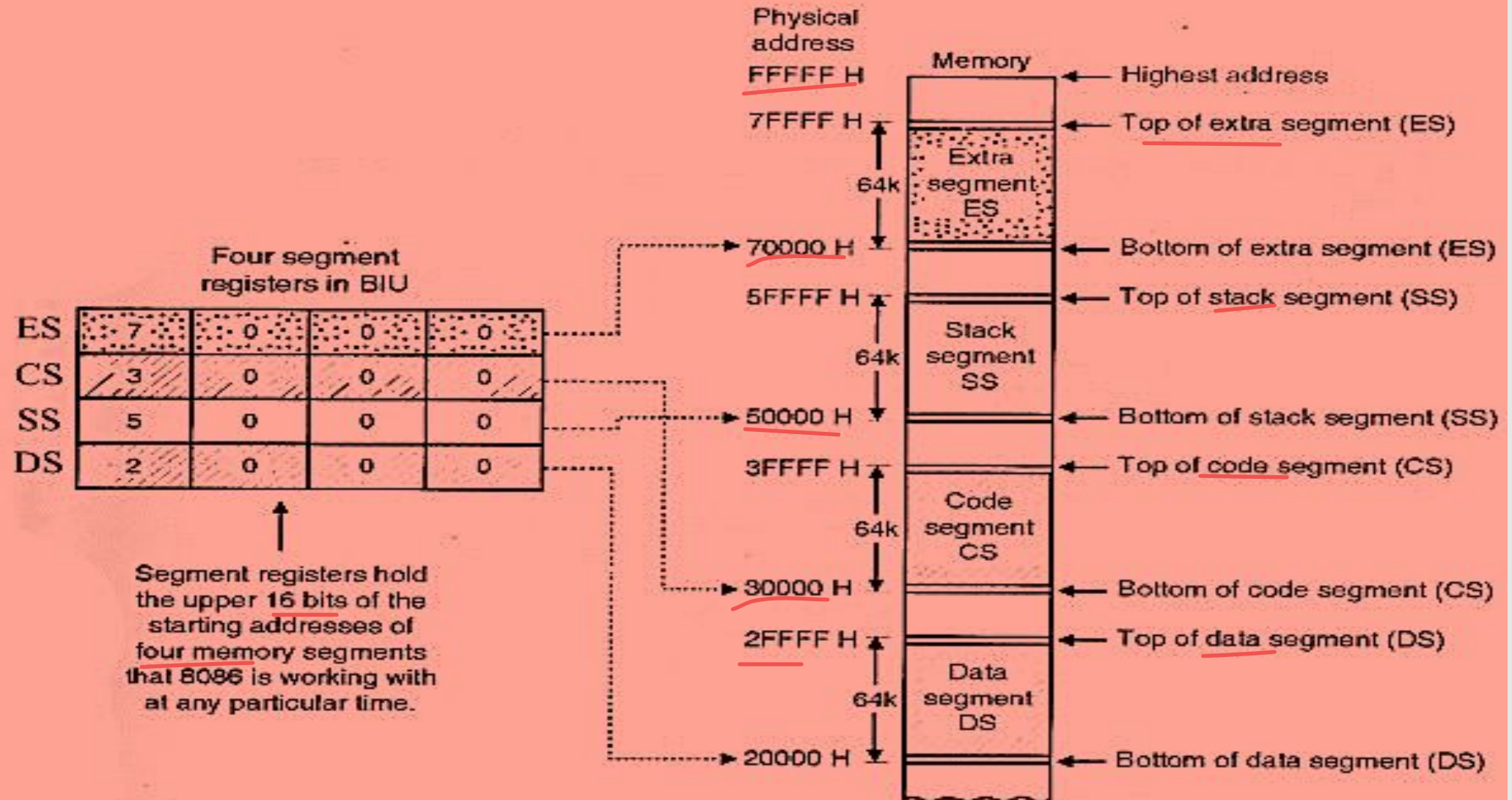
One way of positioning four 64k byte segments within the 1M byte memory space of an 8086

Memory Segmentation

Physical Memory

FFFFFH

Code segment (64KB)
Data segment (64KB)
Extra segment (64KB)
Stack segment (64KB)

1 MB

00000H

Segmented Memory

Advantages of Segmented Memory Scheme

- Allows the memory capacity to be 1Mb although the actual addresses to be handled are of 16 bit size.

- Allows the placing of code, data and stack portions of the same program in different parts (segments) of the memory, for data and code protection.

- Permits a program and/or its data to be put into different areas of memory each time program is executed, i.e. provision for relocation may be done .

- It allows to extend the address ability of a processor i.e. segmentation allows the use of 16 bit registers to give an addressing capability of 1 MB. Without segmentation, it would require 20 bit registers.

Segmented Memory

Code segment :

    The part of memory from where BIU is currently fetching instruction code bytes. This address plus the offset value contained in the instruction pointer (IP) indicates the address of an instruction to be fetched for execution. (CS : IP)

Stack segment :

    A section of memory set aside to store address and data while a sub program executes. This address plus the offset value contained in the stack pointer (SP) is used for stack operation. (SS : SP)

Data Segment and Extra Segment :

    Used for storing data values to be used in the program. String instruction always use ES and DI register for calculating physical address. (DS : DI , ES : DI)

Instruction Pointer & Summing Block

- The instruction pointer register contains a 16-bit offset address of instruction that is to be executed next.

- The IP always references the Code segment register (CS) to find the 20-bit physical address.

- The value of the instruction pointer is incremented after executing every instruction.

- Offset is the displacement of the memory location from the starting location of the segment.

- To form a 20bit address of the next instruction, the 16 bit address of the IP is added (by the address summing block) to the address contained in the CS, which has been shifted four bits to the left.
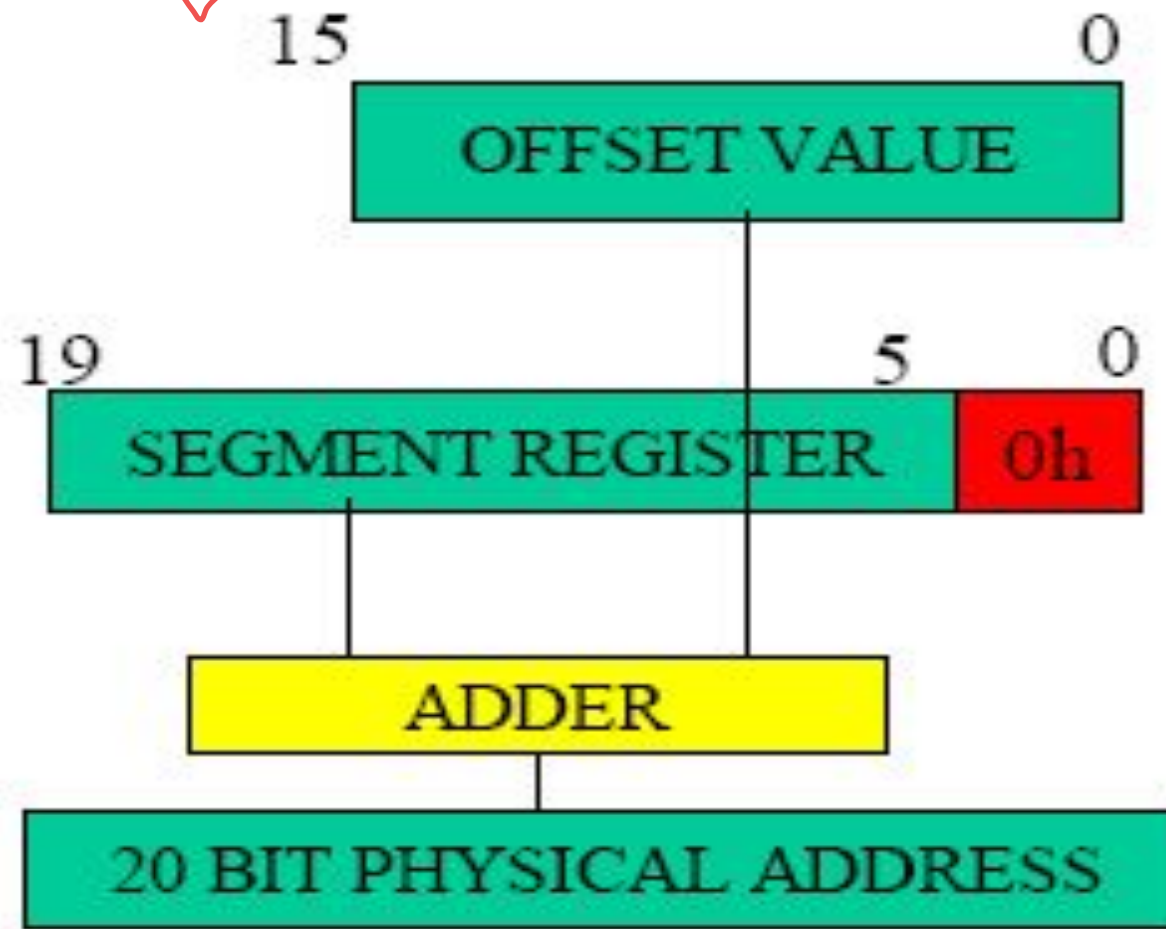
Instruction Pointer & Summing Block

- To form a 20bit address of the next instruction, the 16 bit address of the IP is added (by the address summing block) to the address contained in the CS, which has been shifted four bits to the left.

- To calculate the effective address of the memory, BIU uses the following formula:
    - Effective Address = Starting Address of Segment + Offset
    - To find the starting address of the segment, BIU appends the contents of Segment Register with 0H. Then, it adds offset to it.
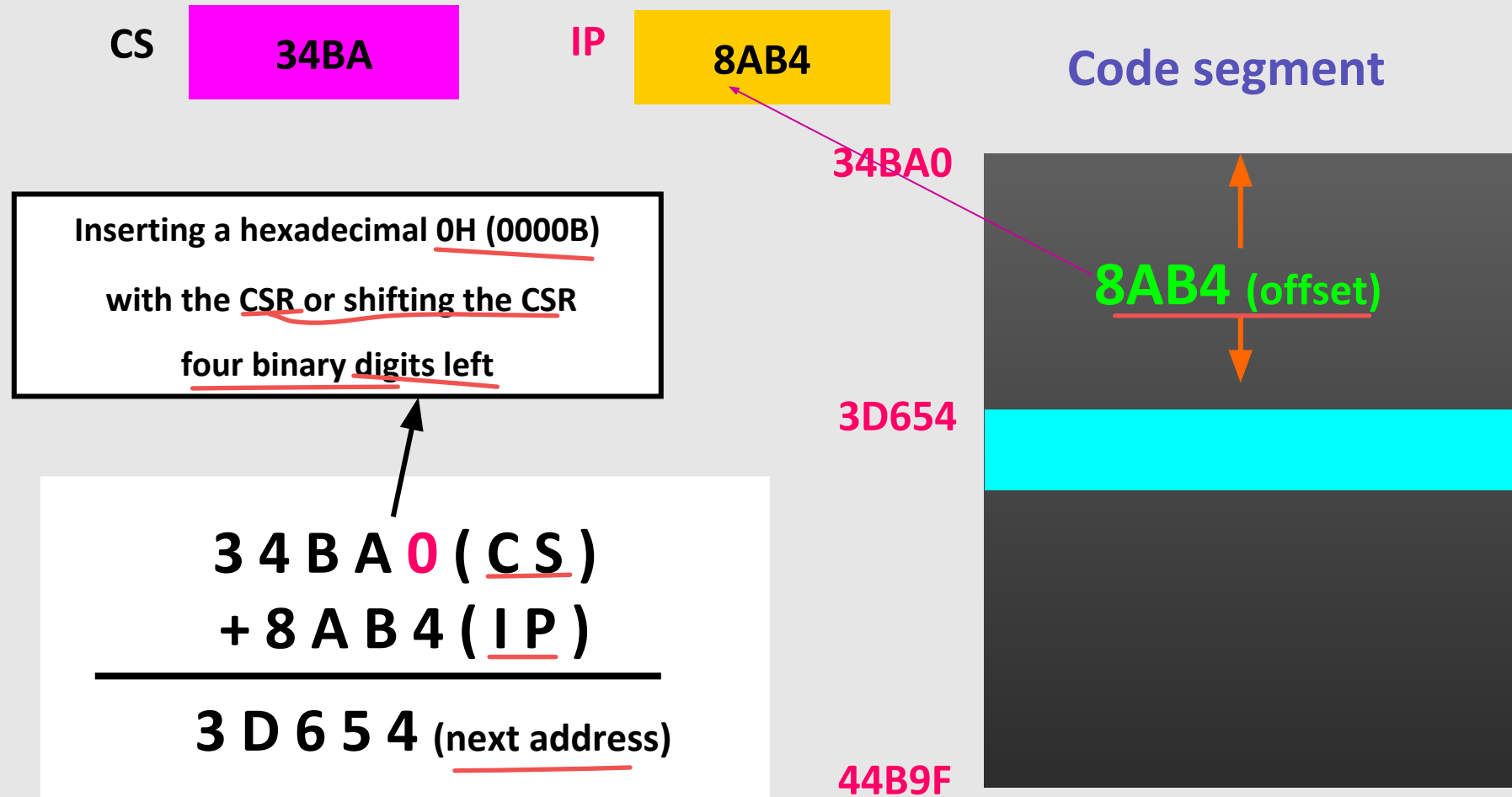
Therefore:

```
     EA = 2 2 2 2 0 H
        +   0 0 1 6 H
        ------------------
            2 2 2 3 6 H
```

Summing Block

- **The following examples shows the CS:IP scheme of address formation**

CS `34BA`   IP `8AB4`

Code segment

34BA0

8AB4 (offset)

Inserting a hexadecimal 0H (0000B)

with the CSR or shifting the CSR

four binary digits left

3D654

$$3\ 4\ B\ A\ 0\ (CS)$$
$$+\ 8\ A\ B\ 4\ (IP)$$
$$\overline{3\ D\ 6\ 5\ 4}\ \text{(next address)}$$

44B9F

Segment and Address Register Combination

Segment	Offset Registers	Function
CS	IP	Address of the next instruction
DS	BX, DI, SI	Address of data
SS	SP, BP	Address in the stack
ES	BX, DI, SI	Address of destination data (for string operations)