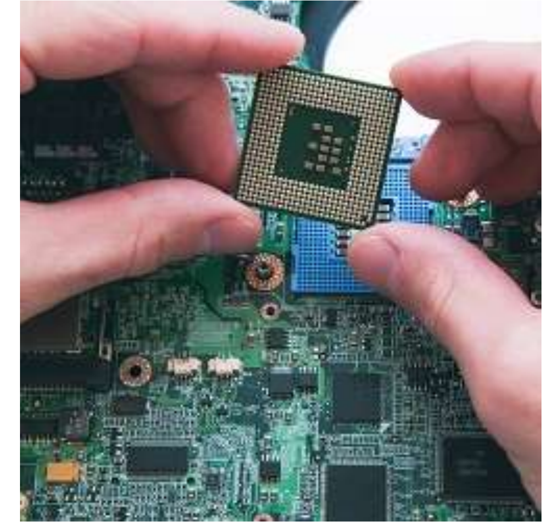
Ch-1
The 8086 microprocessor

Microprocessor

A **microprocessor** is an electronic component that is used by a computer to do its work.
It is a central processing unit on a single integrated circuit chip containing millions of very small components including transistors, resistors, and diodes that work together.



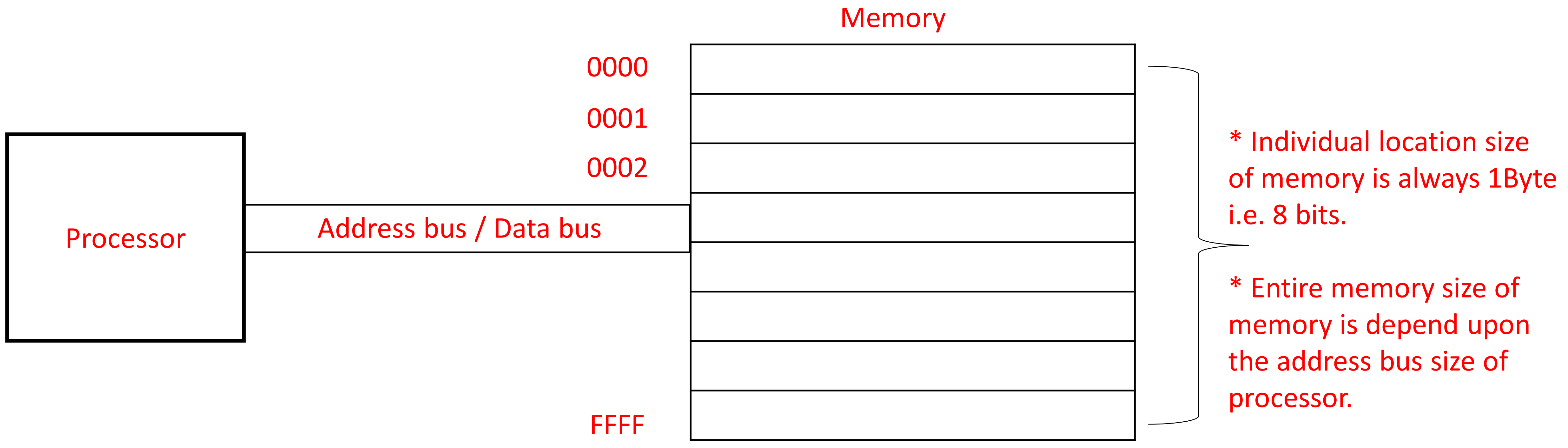
The microprocessor is the central unit of a computer system that performs arithmetic and logic operations, which generally include adding, subtracting, transferring numbers from one area to another, and comparing two numbers. It's often known simply as a processor, a central processing unit, or as a logic chip.

Evolution of Microprocessor

Intel Microprocess				
Name	**Year**	**Transistors**	**Clock speed**	**Data width**
8080	1974	6,000	2 MHz	8 bits
8085	1976	6,500	5 MHz	8 bits
8086	1978	29,000	5 MHz	16 bits
8088	1979	29,000	5 MHz	8 bits
80286	1982	134,000	6 MHz	16 bits
80386	1985	275,000	16 MHz	32 bits
80486	1989	1,200,000	25 MHz	32 bits
Pentium	1993	3,100,000	60 MHz	32/64 bits
Pentium II	1997	7,500,000	233 MHz	64 bits
Pentium III	1999	9,500,000	450 MHz	64 bits
Pentium IV	2000	42,000,000	1.5 GHz	64 bits
Pentium IV "Prescott"	2004	125,000,000	3.6 GHz	64 bits
Intel Core 2	2006	291 million	3 GHz	64 bits
Pentium Dual Core	2007	167 million	2.93 GHz	64 bits
Intel 64 Nchalem	2009	781 million	3.33 GHz	64 bits

Basic Functions of processor

1. Programmer write a program using HLL or ALP(instructions)
2. Assembler will convert HLL/ALP into machine code.(opcode of instructions (hex))
3. Loader is responsible to load program into memory (binary form)



1. Processor calculates the memory address and fetch the content of that memory location.
2. Processor decodes the fetched instruction and execute it.

Content of memory location can be Data/instructions

Features of 8086

Explain features of 8086 (each for 2 marks)

1. Basic Features
2. Special Features
3. Miscellaneous Features

1.Basic Features

- Processor Size
- Speed of processor
- Address bus size for memory
- Address bus size for I/O

(1) It is a 16-bit processor. This implies that

(a) It has a 16-bit ALU that can perform 16-bit operation simultaneously.

(b) It has 16-bit registers and internal data bus.

(c) It has 16-bit external data bus.

(2) It has three versions based on the basis of frequency of operation.

(i) 8086 → 5 MHz.

(ii) 8086-2 → 8 MHz.

(iii) 8086-1 → 10 MHz.

(3) 8086 has 20-bit address lines to access memory, hence it can access

$$2^{20} = 1 \text{ MB memory locations}$$

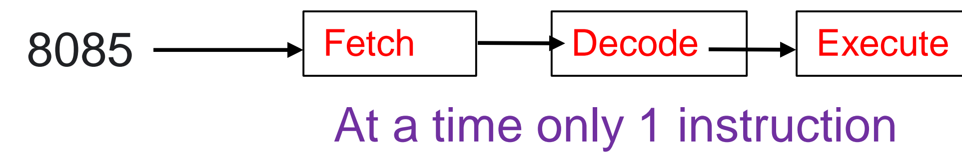
(4) It has 16-bit address lines to access I/O devices, hence it can access

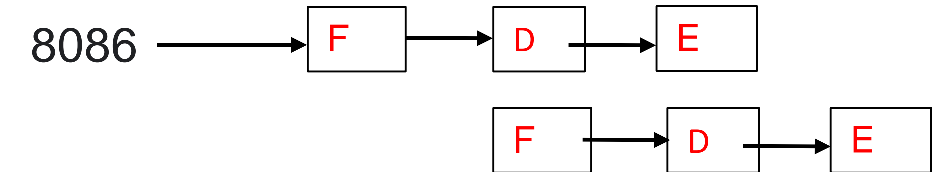
$$2^{16} = 2^{6} \times 2^{10} = 64 \times 1\text{ K}$$

$$= 64\text{ K I/O locations}$$

2. Special Features

- 8086 is a pipelined processor
- 8086 can operate in 2 modes
- 8086 uses memory banks
- 8086 uses memory segmentation





At a time 2 instruction

1) **8086 is a pipelined processor (i.e. it supports pipelined architecture)**

- It uses a two stage pipelining i.e. **Fetch stage** that pre-fetches up to 6 bytes of instructions stores them in the queue and **Execute stage** that executes these instructions.
- Pipelining improves the performance of the processor i.e. the operations are faster.

2) **8086 can operate in 2 modes**

a. **Minimum mode** → A system with only 1 processor i.e. 8086.
b. **Maximum mode** → A system with 8086 and other processors like 8087-(Math Co-processor), 8089-(IO processor) or multiple 8086 processors.

3) **8086 uses memory banks**

- The 8086 uses a memory banking system i.e. the entire data is not stored sequentially in a single memory of 1 MB but the memory is divided into two banks of 512KB each.

- The banks are called Lower bank (or even bank, because it stores the data bytes at even locations i.e. 0, 2, 4....) and Higher Bank (or odd bank, because it stores the data bytes at odd locations i.e. 1, 3, 5...).
- The benefit of this is that 16-bit data can be accessed in a single access even though the memory chip can store only 8-bit at a location.

4) **8086 uses memory segmentation**

- A 16-bit address in an instruction or a 16-bit address in a register can access a memory location, although 8086 has 20 address lines. This is made possible using the concept of Segmentation that divides the memory into logical components.
- Here the memory is divided into 16 segments of a capacity of 2^{16} (= 65536 B = 64 KB) each and is used as: Code, Stack, Data and Extra Segment.

3. Miscellaneous Features

- Interrupts
- Registers
- Instruction set
- Data size for ALU

(1) **It has 256 vectored interrupts :** There are also non-vectored interrupts in 8086, but they are routed to one of these interrupts.

(2) It has 14, 16-bit registers.

(3) It has a powerful instruction set, that supports multiply and divide operations also. (These operations were not possible in the processors earlier to 8086).

(4) 8086 can perform operations on bit, byte (8-bit), word (16-bit) or a string (block of data) types of data.

Architecture of 8086

Draw and explain architecture of 8086
- Diagram –
- Explanation –

Main task of Processor :

Fetch

Decode

Execute

Draw architecture of 8086 and explain any one unit

- Diagram –
- Explanation –

Draw architecture of 8086 (3 marks)

To understand any architecture we must know :

- How do you fetch the instruction?
- Where is getting decoded?
- Where is getting executed?

8086 is 16 bit processor

Why in two parts????

MEMORY INTERFACE

BIU

Σ

PA= seg x 10h + offset

B-BUS

Pipelining

INSTRUCTION STREAM BYTE QUEUE

6 5 4 3 2 1

6 Bytes

Register	Size
ES	(16)
CS	(16)
SS	(16)
DS	(16)
IP	(16)

BUS INTERFACE UNIT

CONTROL SYSTEM

EU

Register	High		Low	
AX (16)	(8)	AH	(8)	AL
BX (16)	(8)	BH	(8)	BL
CX (16)	(8)	CH	(8)	CL
DX (16)	(8)	DH	(8)	DL

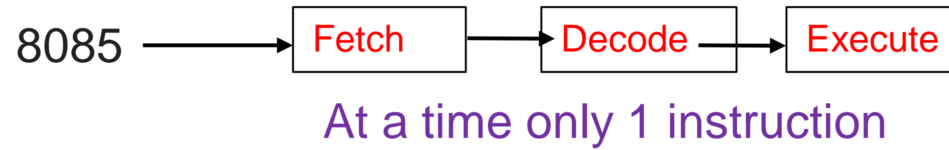
Register	Size
SP	(16)
BP	(16)
SI	(16)
DI	(16)

ALU ARITHMETIC LOGIC UNIT (16)

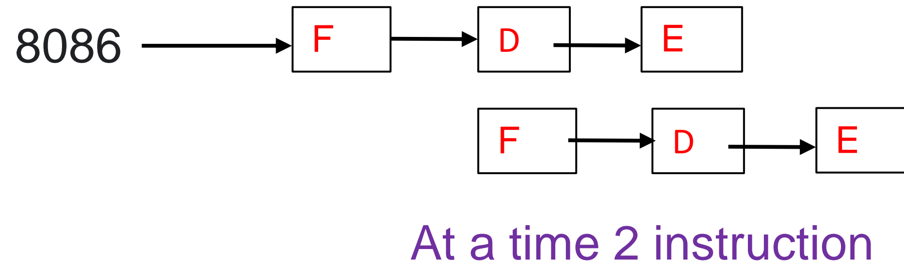
OPERANDS	(16)
FLAGS	(16)

EXECUTION UNIT

Pipelining :



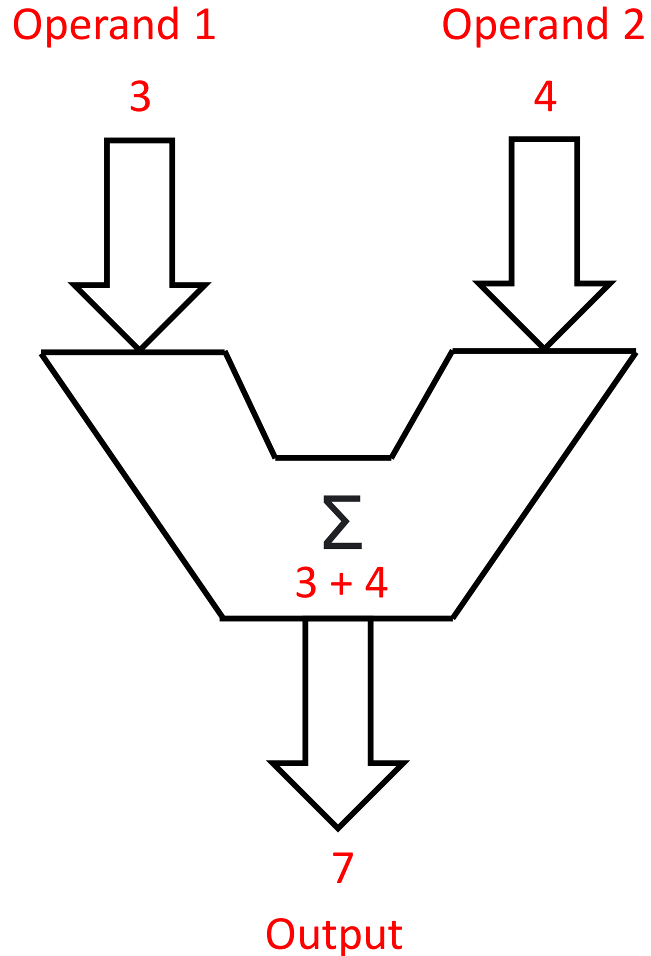
At a time only 1 instruction



At a time 2 instruction

- Pipelining is one of the special feature of 8086 processor.
- Due to pipelining 8086 processor architecture is divided into two parts.
- BIU will fetch the instruction from memory and it will pass that fetched instruction to the Execution unit.
- While execution unit executes the current instruction BIU will fetch the next instruction.
- Pipelining helps in order to increase the speed of processor.

Back



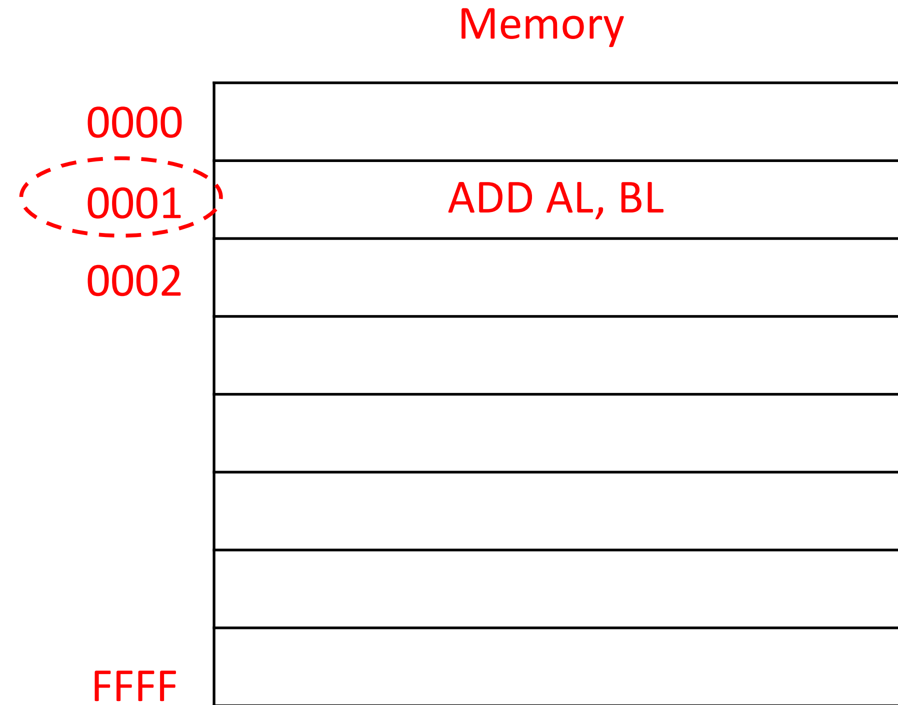
- In the architecture diagram of 8086 everything is in rectangle except two following shapes.
- This the diagram of arithmetic circuit.
- In the given circuit there are two inputs which are used to take operand values.
- operation will be performed on input operands and it will produce output.
- For example 3 + 4

***In the architecture of 8086 there are two such type of circuits are available from that one is **ALU** who is responsible to perform arithmetic operations and another one is to **calculate memory address.**

Back

There is an instruction ADD AL, BL (wants to add content of AL reg with content of BL register)

- This instruction will be loaded into the memory
- Processor will fetch this instruction from memory and to perform fetching processor has to calculate the physical address of that memory location.
- Hence arithmetic circuit in upper section is responsible to calculate the physical address of memory location where instruction is stored.
- ALU which is present at lower section i.e. in execution unit is responsible to perform actual addition of reg AL and BL data.



How to calculate physical address ????????

Formula to calculate physical address of memory location is : **PA= Seg x 10h + offset**

Overview of memory segmentation:

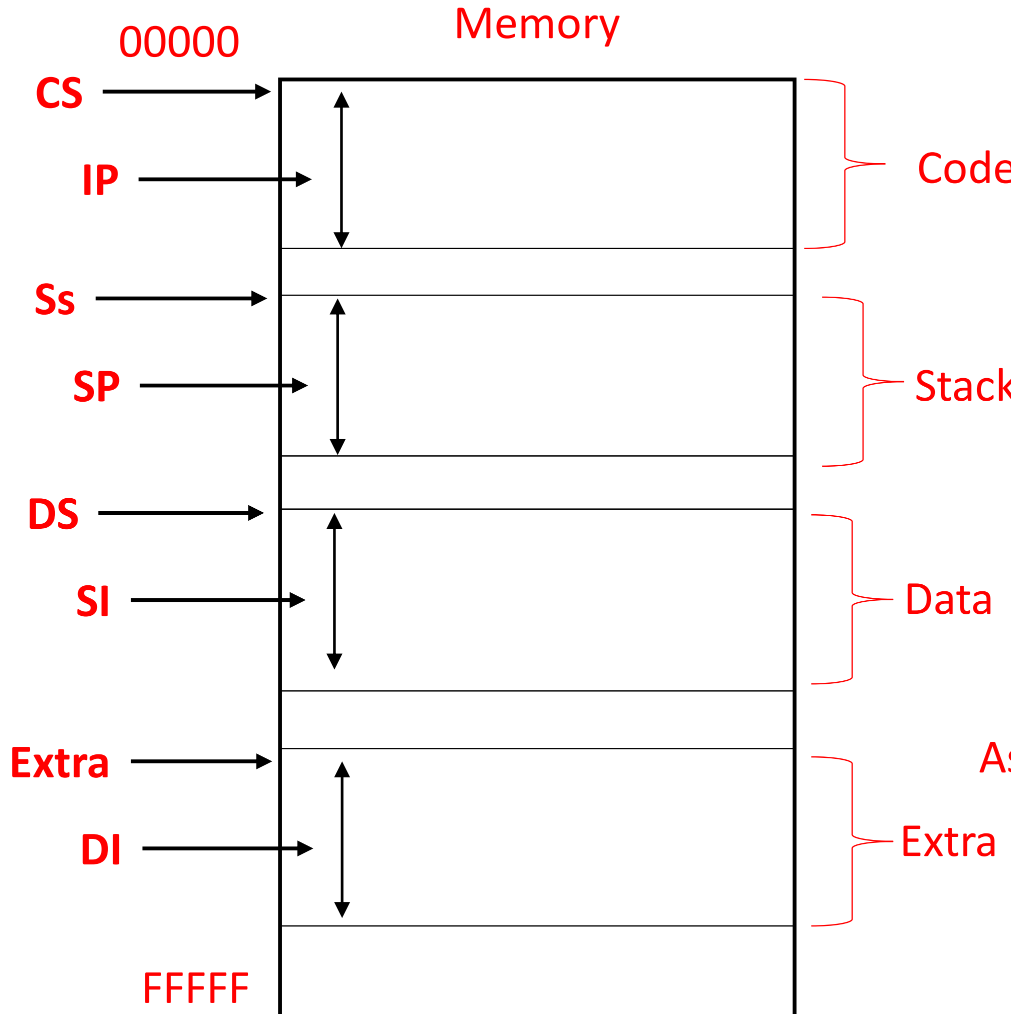
- This is entire memory supported by 8086.
- The amount of memory that you can access is depend upon the address size of processor. 8086 has 20 bit address bus hence entire memory size is 1 MB

Example :

- If you want to access any file from your hard disk you will access that file from file name.
- We don't know the physical address of that file.
- We know the logical address i.e. name

In 8086 memory is divided into 4 part i.e. segments

If you want to access any location of memory which will have its unique physical address.
We give segment address (starting addr of segment) and
offset addr (location within particular segment)



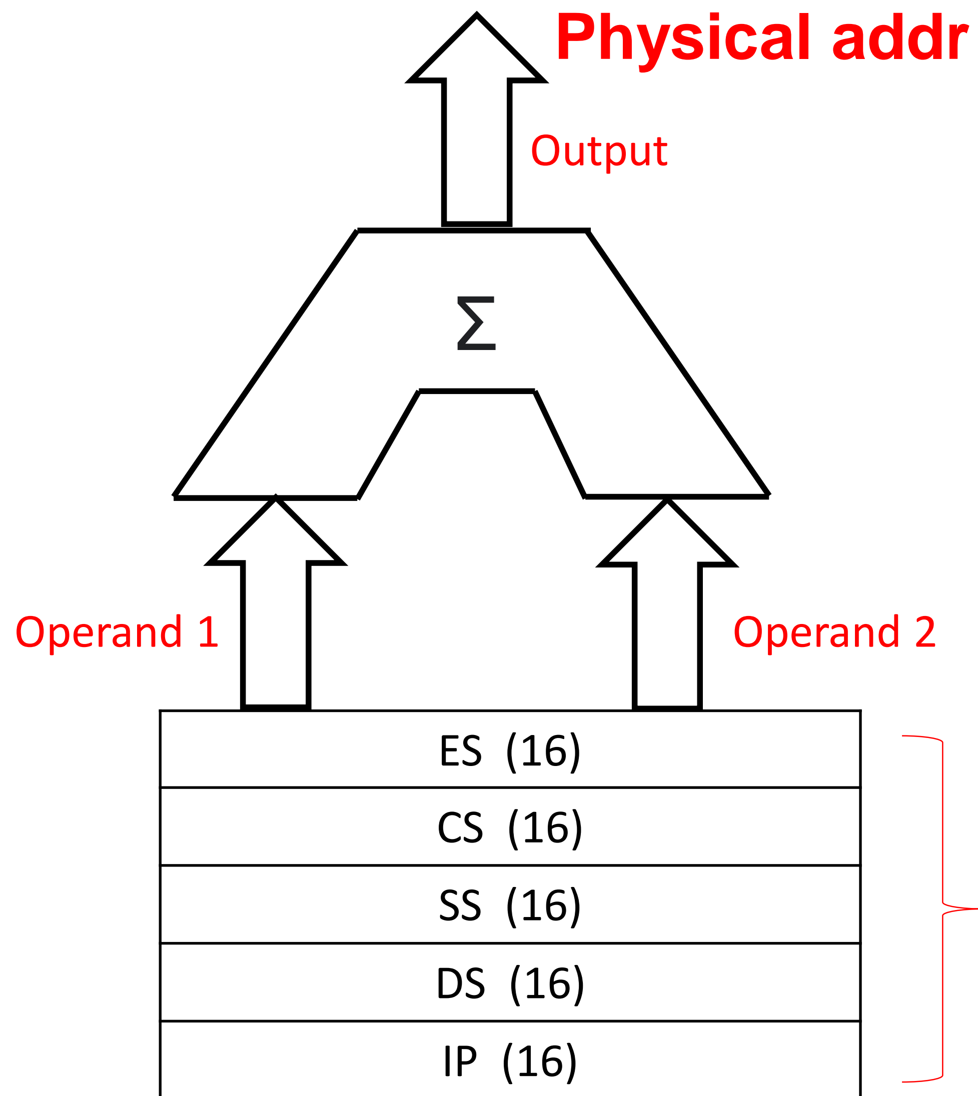
Example :

- There is a book which has 1000 pages.
- Assume there are 10 chapters and each chapter has 100 pages.
- We want to go on page no. 564.
- There are two methods :

1. Directly call 564
2. 5 th chapter **CS**
   64 page **IP**

Assume: CS= 1000 & IP= 2345

PA = Seg x 10H + offset
1000 x 10H + 2345
10000 + 2345
PA = 12345 H

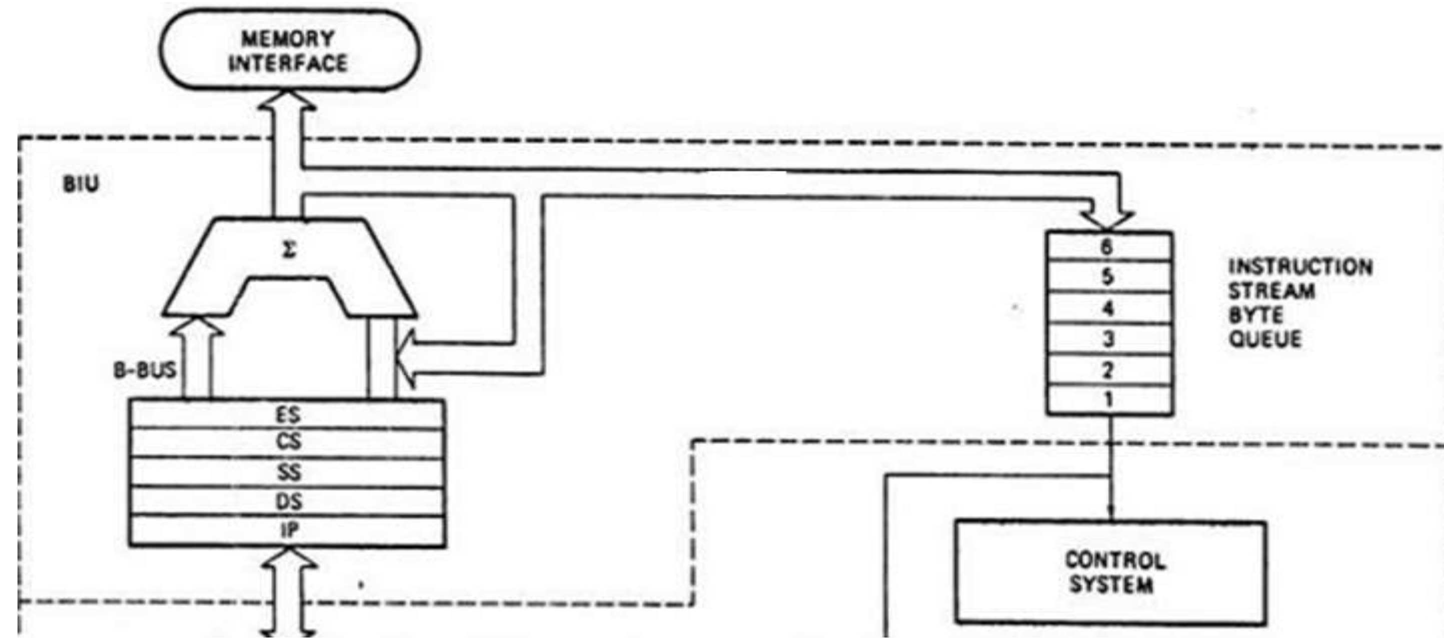


These are the registers which are used to stored starting address of segments

Back

Functions of BIU :

- Fetch the next Instruction from memory
- Calculate the Physical Addr
- Manage the instruction queue.



Managing of queue :

- Processor fetch the next instruction and calculate the physical addr of that instruction.
- After PA calculation processor will transfer that instruction into instruction queue through data bus (16 bit). This phase is called as prefetching phase.
- Maximum size of instruction queue is 6 bytes.

1 ← EI
2
3
4
5 ← 6 bytes of program
6
7
8 when EU executing the next
9 instruction BIU fetch the next
. 6 bytes of program

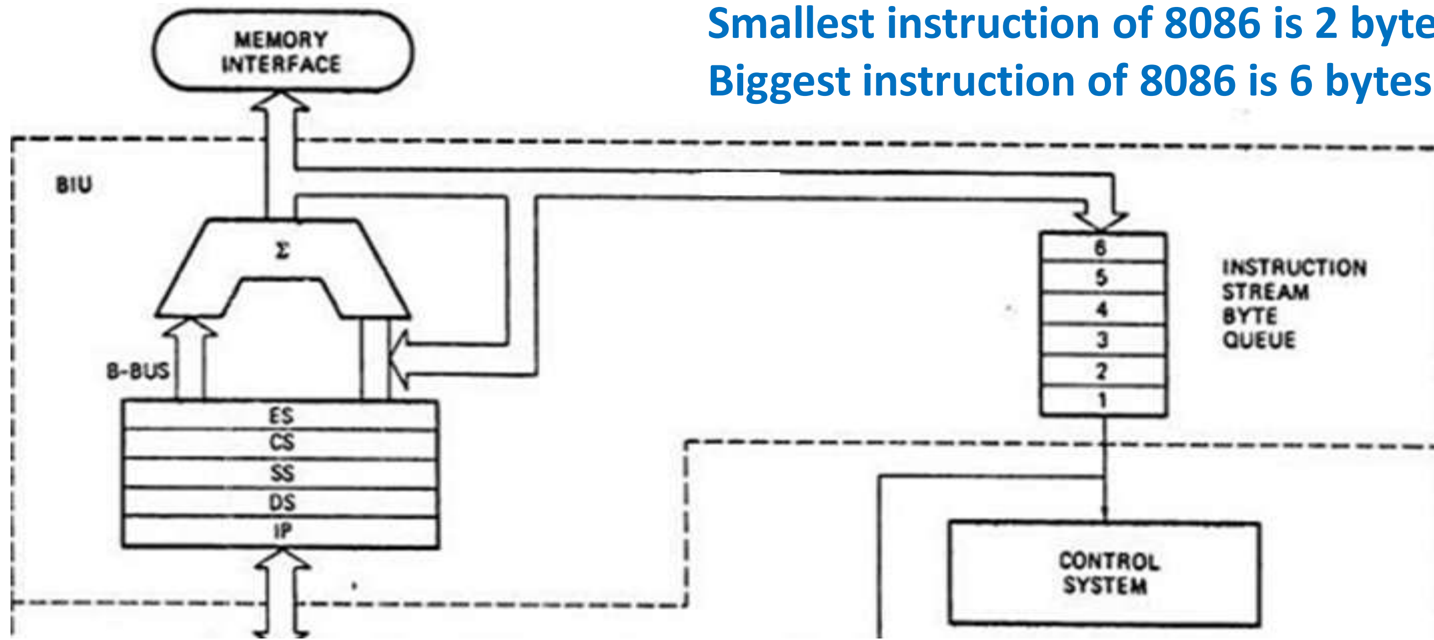
How many Instructions are in the queue?????
(nobody knows)

How many bytes are in the queue?????
(6 bytes)

All instructions are in different size :

1) ADD BL,CL : 1 byte
2) ADD CL, 02H : 2 bytes
3) ADD AX, 2000H : 3 bytes

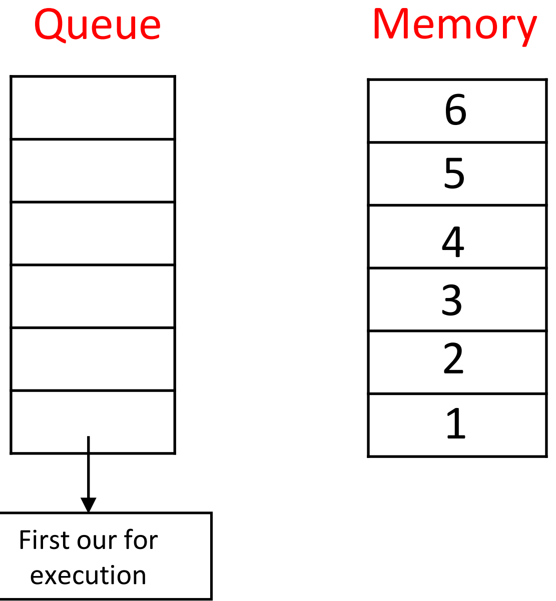
Smallest instruction of 8086 is 2 bytes
Biggest instruction of 8086 is 6 bytes



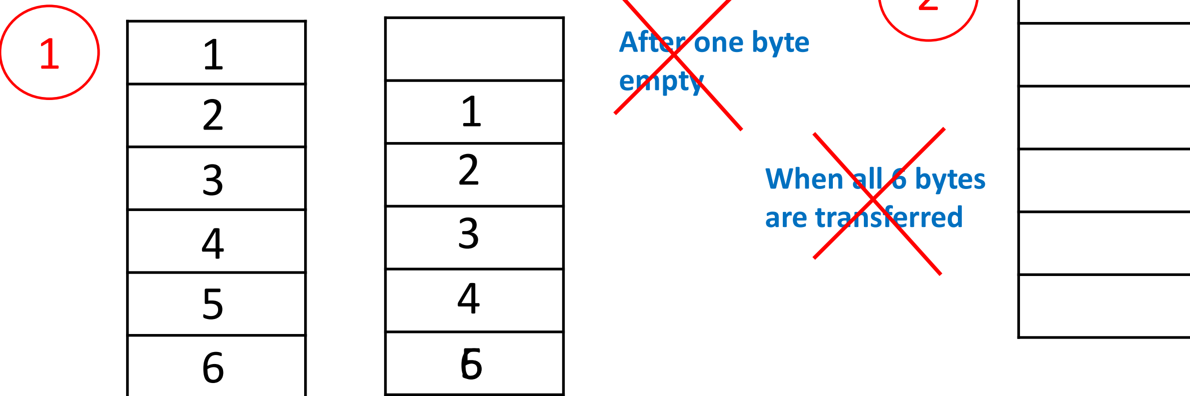
Smallest instruction of 8086 is 2 bytes

Biggest instruction of 8086 is 6 bytes

- 8086 has 6 bytes instruction queue
- It is FIFO type of queue i.e. First In First Out
- Processor will fetch the next **6 bytes** and store into the queue. EU removes one by one each byte for execution



When BIU will refill the queue???

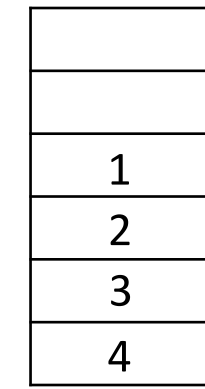


When first entered element removed for execution

First our for execution

3

When 2 bytes are transferred for execution.

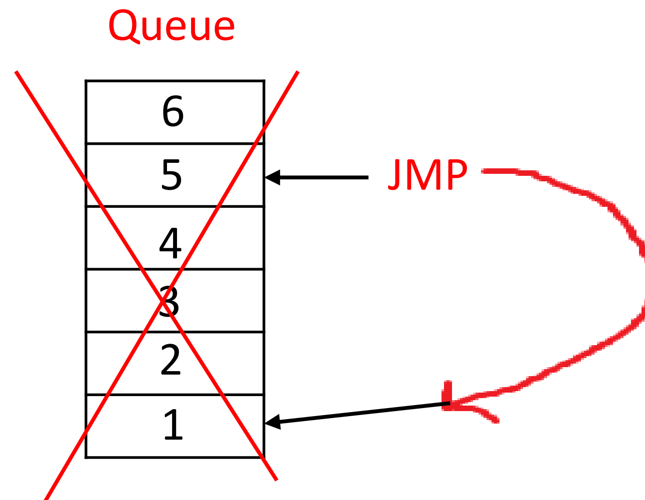


When BIU will refill the queue???

When 2 bytes are transferred for execution. Because the smallest instruction of 8086 is 2 bytes

When pipeline fails ???

- If a Jump (JMP) or CALL instruction appears in the main program, then all the existing instruction bytes in the Queue are flushed out and Queue is made empty.
- Then it is reloaded with the new instruction bytes which correspond to the new locations mentioned in the JMP or CALL instructions.
- The refilling of the queue then continues from the new locations corresponding to the main program.



MEMORY INTERFACE

1.BIU calculates the physical addr of this instruction **(Add BL,CL)** and fetch it from memory

2. Instruction will transfer into instruction Queue through data bus

3. There is a intermediate phase between fetching and executing is called decoding which is done by control system

8086 is 16 bit processor

Why in two parts????

BIU

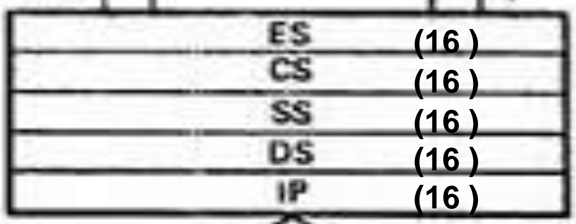
Add BL , CL

PA= seg x 10h + offset

B-BUS

INSTRUCTION STREAM BYTE QUEUE **6 Bytes**

Add BL , BL



9. Control system send notification to BL reg about the output because result is always stored into 1st operand register

5. Control System produces signal for reg. BL and CL to inform about the operation.

Add BL , CL

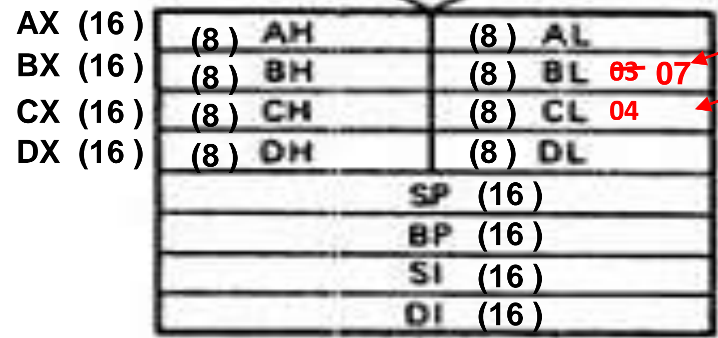
CONTROL SYSTEM

4.Decoding means identify the operation of an instruction and inform to execution unit about it

EU

07

7. Control system again inform to ALU about floating data



03

04

07

ARITHMETIC LOGIC UNIT (16)

Mov BL, 03
Mov CL, 04
Add BL , CL

10. BL replace output with its previous value

07

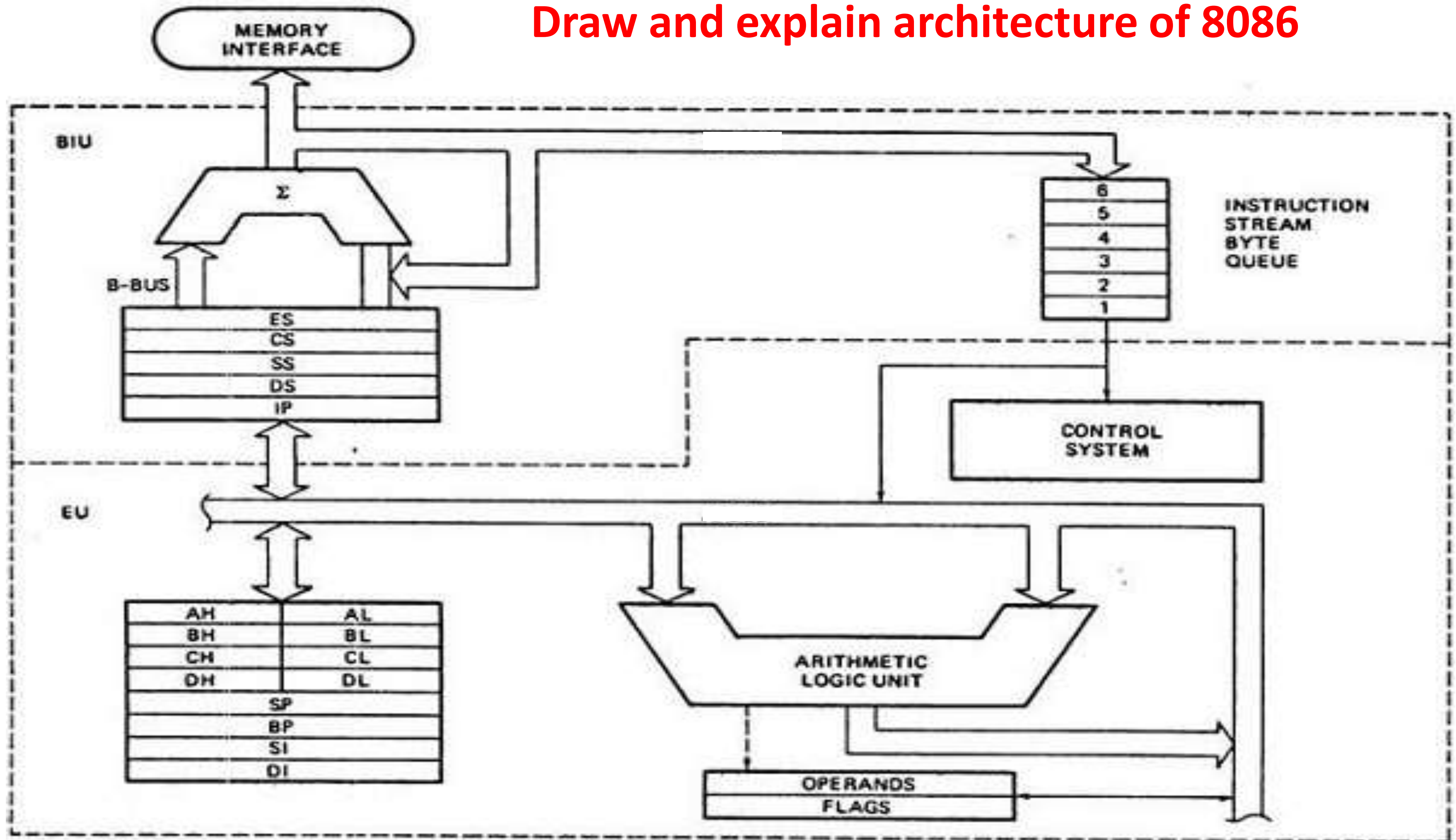
8. ALU take both operands and performs addition float the output on database

6. After receiving signal about the operation Reg. BL and CL float there content on data bus and also keep original value.

OPERANDS (16)

FLAGS (16)

Draw and explain architecture of 8086



The Execution Unit (EU)

Main function of EU is **decoding** and **execution** of the instructions. In order to carry out its tasks it has the following units :

- Arithmetic Logic Unit (ALU)
- General Purpose Registers. PPT
- Decoder
- Flag Register. → from PPT
- Control Unit
- Pointer and Index Register

Arithmetic Logic Unit (ALU)

- The ALU in the EU is a 16 bit unit i.e. it can perform 16-bit operation simultaneously
- It is capable of performing a variety of arithmetic and logic operations such as add, subtract, AND, OR, NOT, EX-OR, increment, decrement, shift etc.

Control Circuitry

The control circuit is a part of EU. It is used for directing the internal operations.

A Decoder

- "The process of translation from instructions into action is known as decoding"
- A decoder in the execution unit (EU) is used for translating the instructions fetched from the memory into a series of actions.
- The EU will actually carry out these actions.

Pointer and Index Register

The execution unit also contains the following 16 bit registers.

- Base Pointer (BP) register
- Stack Pointer (SP) register
- Source Index (SI) register
- Destination Index (DI) register

These registers can be used as general purpose 16-bit registers. But mainly they are used to hold the 16 bit offset of data word in one of the segments as given below :

Base Pointer (BP) Register

- This is a 16 bit register in the E.U. Its function is to hold the 16 bit offset relative to the stack segment (SS) register.
- But BP has a specific use. BP is used whenever we pass a parameter by way of stack.
- The BP register can also be used as an offset register in the addressing mode called **base addressing mode**.

Stack Pointer (SP) Register

- This is also a 16-bit register in the E.U. Its function is to hold the 16-bit offset address relative to stack segment (SS) register.
- SP is used for sequential access of stack segment.
- It always points to the top of stack.
- It is mainly used during instructions like PUSH, POP, CALL, RETURN etc.

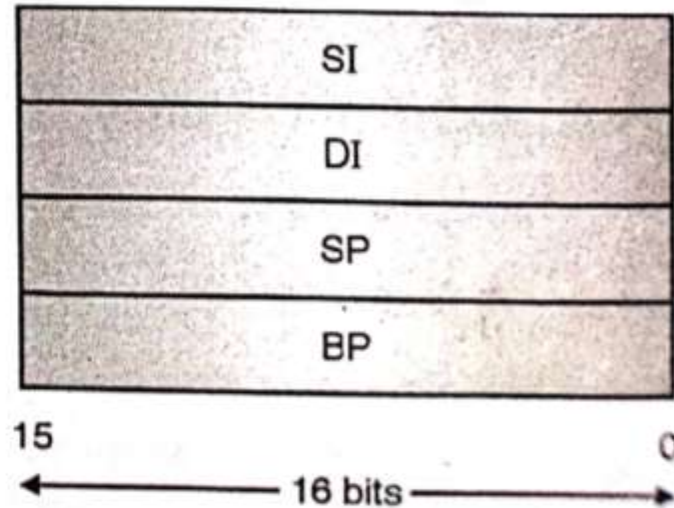
Source Index (SI) Register

- This register is used for holding the offset of a data word in the data segment (DS).
- The physical address of a location in the data segment can be generated by adding a hardwired zero to the segment base (16 bit) held by the data segment (DS) register and then by adding the offset in the SI register to it.

Data segment (DS) Register	→		2	0	0	0	0	← Hard wired 0
Source index (SI) register (offset)	→	+		1	F	2	3	
20 bit physical address in D.S.	→		2	1	F	2	3	

Destination Index (DI) Register

- This register is used for holding the 16 bit offset of a data word in the extra segment (ES).
- The source index (SI) register and destination index (DI) register are used for the string related instructions.
- For example if we want to move a block of data from memory to memory, then source index (SI) register can be used to point to the source memory address and destination index (DI) register is used to point to the destination memory address.



m(15.3)Fig. 2.4.2 : Pointer register of EU

The Bus Interfacing Unit (BIU)

The bus interface unit performs all the activities related to Bus. Specifically BIU has the following **five functions**

1. Instruction **fetching** (reading) from primary memory. (performed over the system bus)
2. R/W of data operand from/to primary memory. (performed over the system bus)
3. I/O of data from/to peripheral ports. (performed over the system bus)
4. Address generation for memory reference
5. Instruction queuing in an instruction queue.

In order to carry out its functions BIU has the following modules :

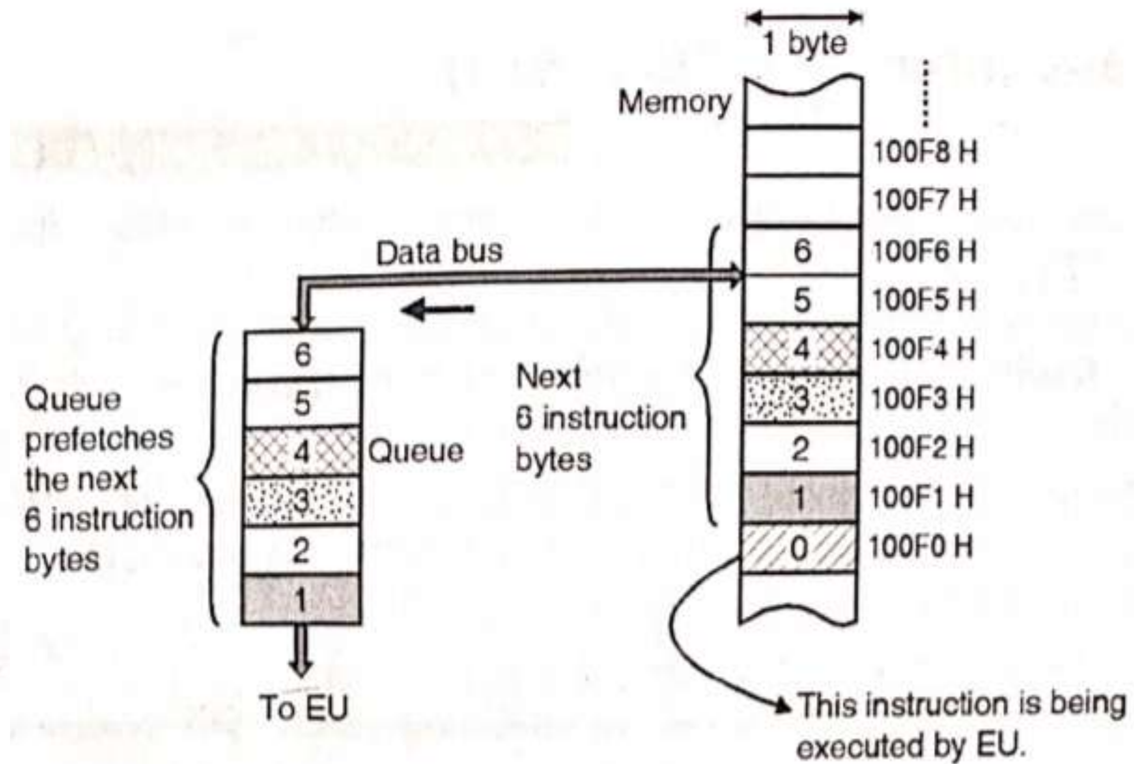
- Instruction queue.
- Segment registers
- An instruction pointer register (IP).
- Address generation and bus control.

The Instruction Queue

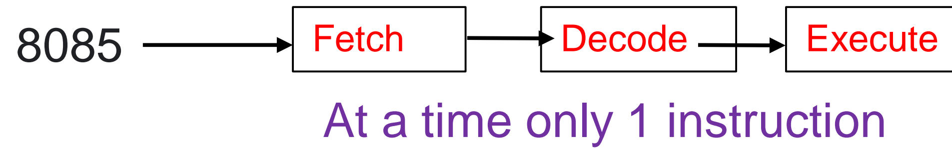
- The execution unit is supposed to decode or execute an instruction. Decoding does not require the use of buses.
- When EU is busy in decoding and executing an instruction, the BIU fetches upto six instruction bytes for the next instructions.
- These bytes are called as the prefetched bytes and they are stored in a first-in-first-out (FIFO) register set, which is called as a "**queue.**"

Significance of Queue

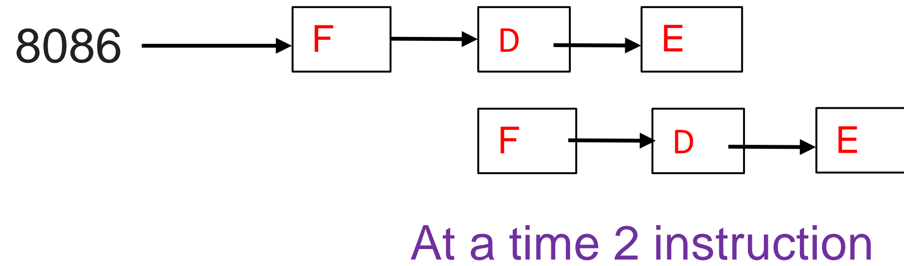
- To understand the significance of the queue, refer Fig. 2.5.1.
- As shown in Fig. 2.5.1, while the EU is busy in decoding the instruction corresponding to memory location 100F0, the BIU fetches the next six instruction bytes from locations 100F1 to 100F6 numbered as 1 to 6.
- These instruction bytes 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.



Pipelining :



At a time only 1 instruction



At a time 2 instruction

- Pipelining is one of the special feature of 8086 processor.
- Due to pipelining 8086 processor architecture is divided into two parts.
- BIU will fetch the instruction from memory and it will pass that fetched instruction to the Execution unit.
- While execution unit executes the current instruction BIU will fetch the next instruction.
- Pipelining helps in order to increase the speed of processor.

Advantages of Pipelining :

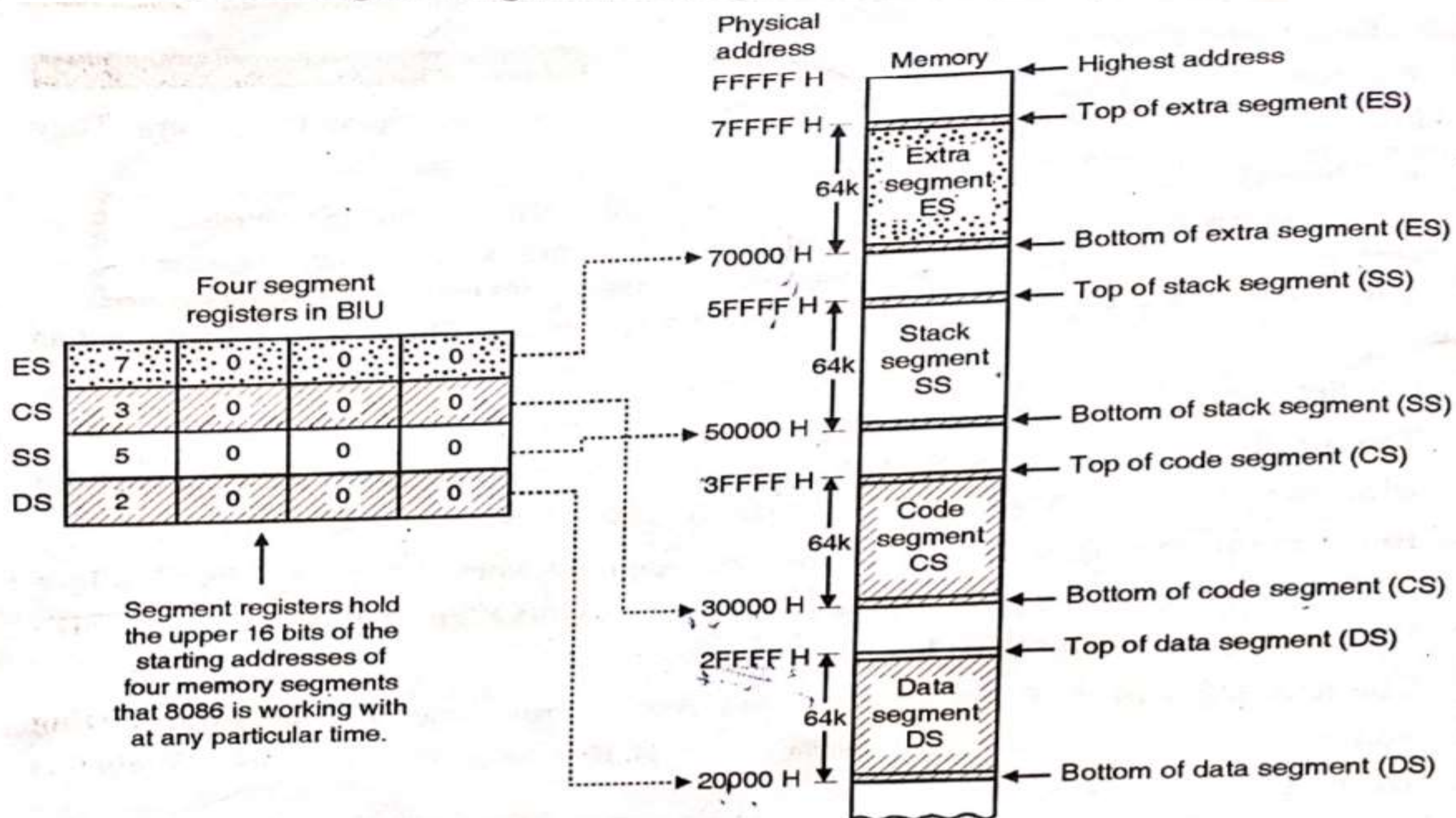
- The EU always reads the next instruction byte from the queue in BIU. This is much faster than sending out an address to the memory and waiting for the next instruction byte to come.
- In short pipelining eliminates the waiting time of EU and speeds up the processing.

Segment Registers

- The BIU contains four special purpose registers called as segment registers. They are :

• The code segment (CS) register	• The stack segment (SS) register
• The extra segment (ES) register and	• The data segment (DS) register

The purpose of using these segment registers and segmentation can be explained as follows :



- All these are 16 bit registers.
- The number of address lines in 8086 is 20. So the 8086 BIU will send out a 20 bit address in order to access one of the 1,048,576 or 1 Mb memory locations.
- But it is interesting to note that the 8086 does not work the whole 1,048,576 byte (1Mbyte) memory at any given time. However it works with only four 65,536 (64K-byte) segments within the whole 1 M-byte memory.
- The four segment registers actually hold (contain) the upper 16 bits of the starting addresses of the four memory segments of 64 K byte each with which the 8086 is working at that instant of time.
- A word is any two consecutive bytes in memory. Word is stored in memory with the most significant byte at the higher memory address. These bytes are stored sequentially from byte 0000H to byte FFFFH.
- Programs view memory space as a group of segments defined by the application.
- A segment is a logical unit of memory that may be upto 64 K bytes long.
- Each segment is made up of contiguous memory locations. It is independent, separately addressable unit.
- Note that these starting addresses will always be changing. They are not fix.
- This concept can be clearly understood by referring to Fig. 2.5.2.
- Fig. 2.5.2 shows one of the possible ways to position the four 64 k byte segments within the 1-M byte memory space of 8086. There is no restriction on the locations of these segments in the memory.
- Note that these segments can be separate from each other as shown in Fig. 2.5.2 or they can overlap.
- Note that the starting address or base address of the data segment is 20000H. The upper 16-bits of this i.e. 2000 are loaded into the data segment register (DS).

Code Segment (CS)

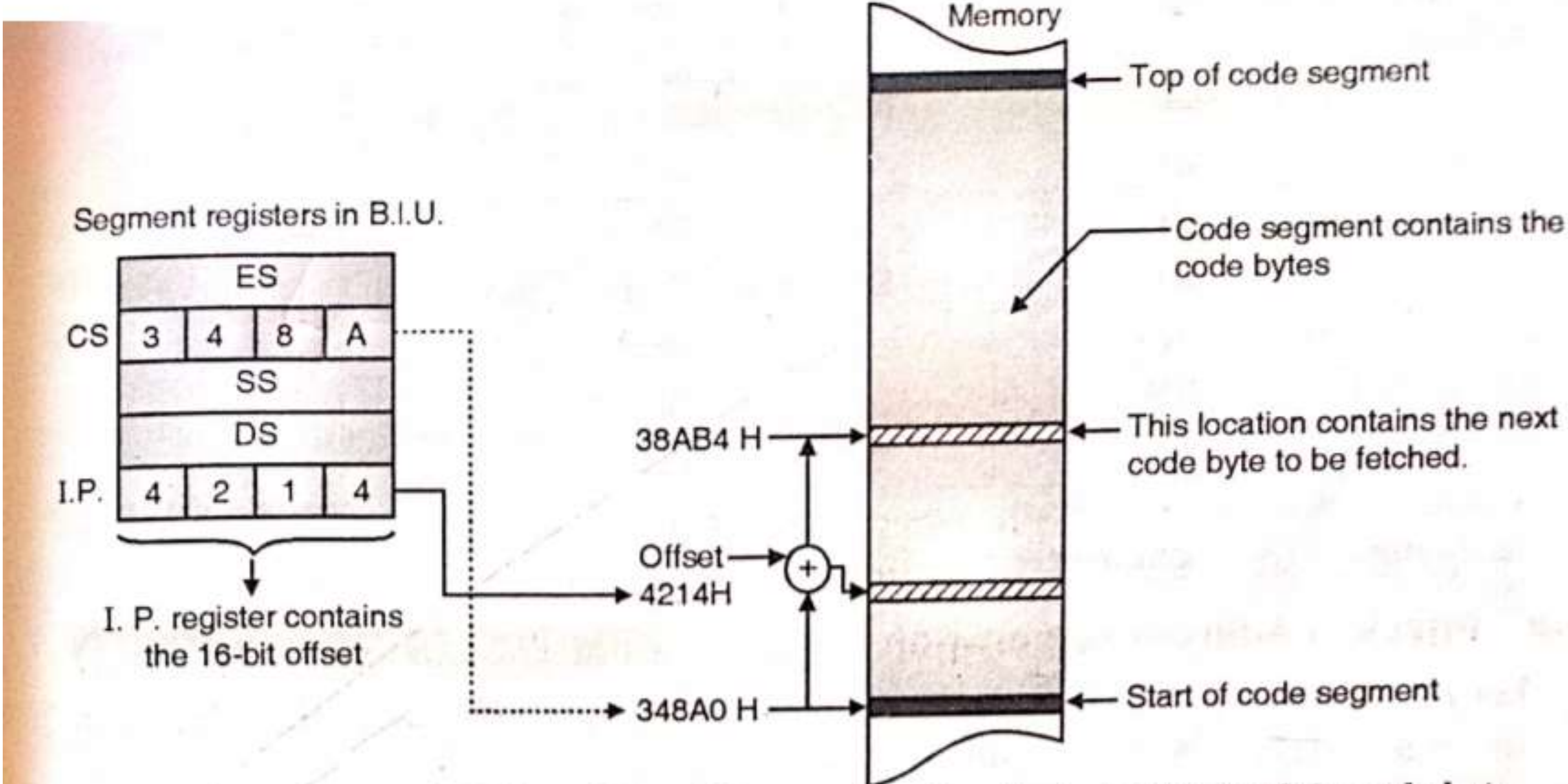
- Code segment (CS) is the part of memory from which the BIU fetches the instruction code bytes.
- The upper 16 bits of the starting address of code segment are held by the code segment (CS) register.
- But the memory address for the base of code segment is 20 bit long. So what about the lower 4-bits ? The answer is that the BIU always inserts zeros for the lowest 4-bits.
- So if the CS register contents are 3428 H then after adding zeros, the physical starting address of code segment becomes 34280 H.
- Therefore a segment will always start at an address with zeros in the lowest 4 bits.

Segment Base : The upper 16-bits of the starting address of a segment, stored in the segment register is called as the segment base.

Stack

- In Fig. 2.5.2 we have defined the stack segment from the memory location 50000 H to 5FFFF H.
- The stack is a section of memory which is reserved to store the addresses and data while executing a subroutine program.
- The stack segment (SS) register holds the upper 16 bits of the starting address of the stack area.

Instruction Pointer (IP) Register



Addition of IP to CS to produce the physical address of the code byte

Computation

- Another special purpose register in the BIU is the instruction pointer (IP) register.
- As discussed earlier the code segment (CS) register holds the upper 16 bits of the 20 bit starting address of the code segment. And code segment is the segment from which the BIU is currently fetching the instruction code bytes.
- The instruction pointer (IP) register holds the 16 bit address or **offset** of the next code byte within the code segment. This is illustrated in Fig. 2.5.3(a).

Rules of segmentation

1. If non-overlapping, there can be 16 segment in all, each of 64KB.
   (since, 1MB /64KB = 16)
2. Segments can overlap each other
3. A segment can begin at any location that is a multiple of 10H
4. At any given time a maximum of 4 segments and hence 256KB can be accessed
5. The memory of 8086 is a wrap around memory. This means that once the address generated crosses the 1MB limit, it accesses the location at the top 00000H. For e.g. if CS = FFFF and IP = 0010. Physical address = CS*10H + IP = 100000H. The address lines in 8086 are only 20, so the MSB '1' is discarded and the location being accessed is 00000H.

Physical Address Generation

- Let us now understand the generation of 20 bit physical address of the location in the code segment which contains the next code byte.
- The sequence of operation is as follows.

Physical Address Generation

Step 1 : The CS register contains the upper 16 bits of the starting address of the code segment.

∴ CS Register : | 3 | 4 | 8 | A | Segment base

Step 2 : The BIU will automatically insert zeros for the lowest four bits of the segment base address to get the 20 bit physical address for the starting of code segment.

∴ Starting address of code segment : | 3 | 4 | 8 | A | 0

↑ BIU adds this zero

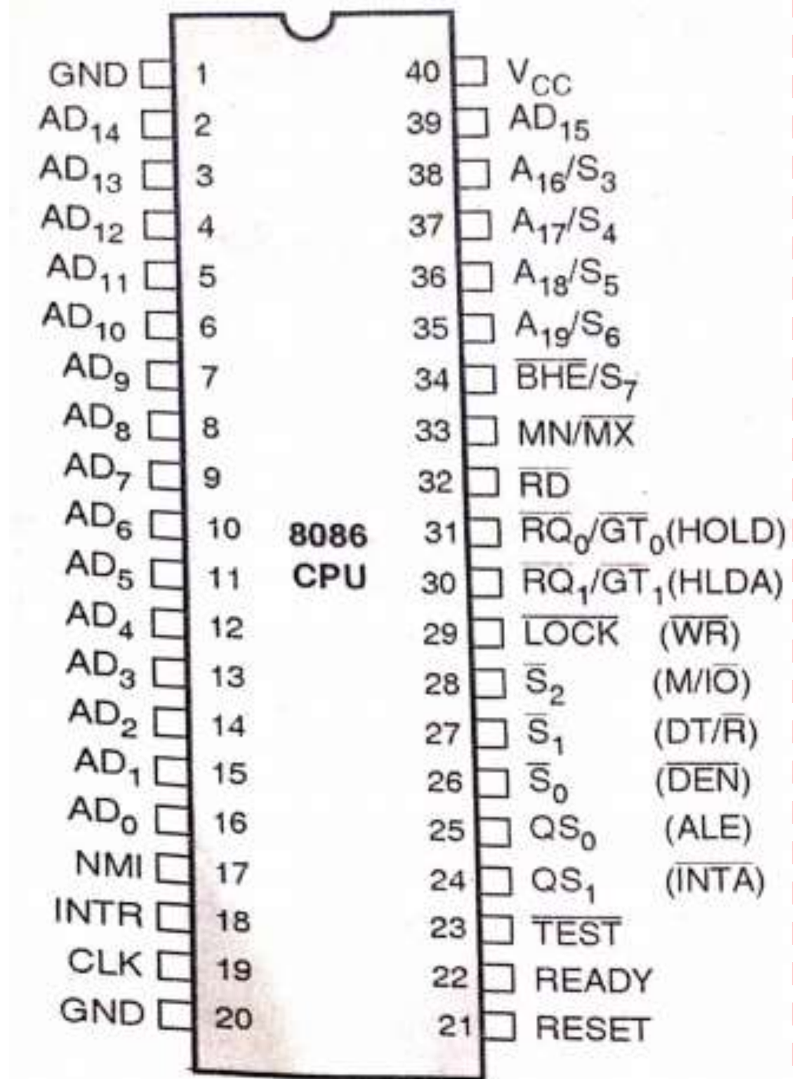
Step 3 : The I.P. register contains the offset or distance from this address. The offset here is 4214H.

∴ I.P. Register : | 4 | 2 | 1 | 4 |

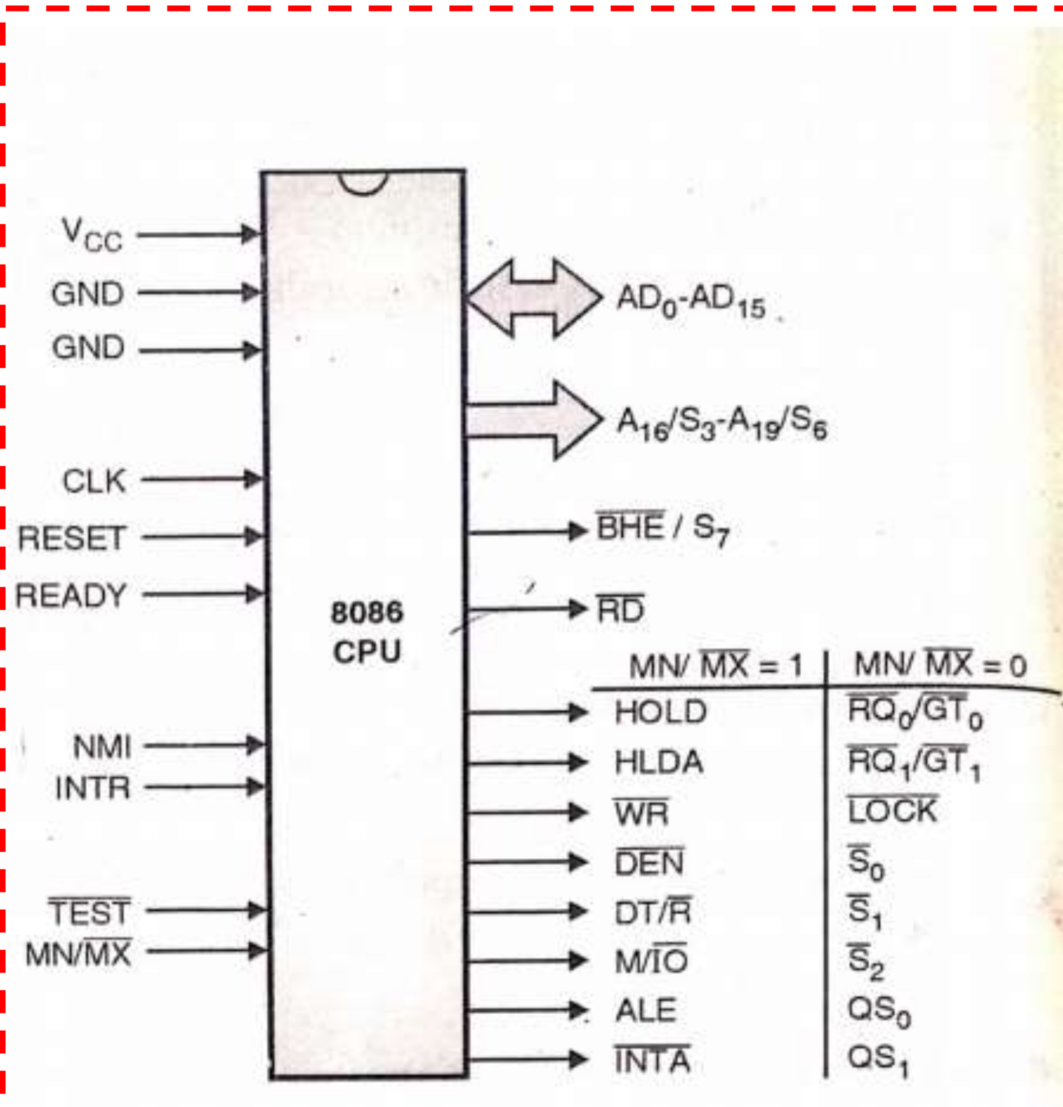
Step 4 : Add the starting address of code segment (20 bit) to the offset to get the physical address of the location containing the next code byte as follows :

Starting address of code segment →		3	4	8	A	0	← Hard wired 0
Offset in the I.P. Register →	+		4	2	1	4	
Physical address of the location containing the next code byte →		3	8	A	B	4	

Pin diagram of 8086



(a) Pin configuration



(b) Functional pin diagram

For Exam

Sr. No	Pins	Name of the pins
1.	Supply pins (3 pins)	V_{CC}, GND, GND
2.	Clock related pins (3 pins)	CLK, RESET, READY
3.	Address and Data pins (21 pins)	$AD_0 - AD_{15}$, $A_{16}/S_3 - A_{19}/S_6$, $\overline{BHE} / S_7$
4.	Interrupt pins (2 pins)	NMI, INTR
5.	Other control (3 pins)	$\overline{TEST}$, $MN / \overline{MX}$, $\overline{RD}$
6.	Mode multiplexed signals (8 pins) (MIN mode – MAX mode signals)	$HOLD - \overline{RQ_0} / \overline{GT_0}$, $HLDA - \overline{RQ_1} / \overline{GT_1}$, $\overline{WR} - \overline{LOCK}$ $\overline{DEN} - \overline{S_0}$, $DT / \overline{R} - \overline{S_1}$ $M / \overline{IO} - \overline{S_2}$, $ALE - QS_0$ $\overline{INTA} - QS_1$

1. Supply Pins (3) :

• V_{CC}	• GND	• GND

- Used for power supply i.e. + 5V on V_{CC} w.r.t. GND.
- Two separate GND pins for two layers of 8086 chip, improves the noise rejection.

2. Clock related Pins (3) :

• CLK	• RESET	• READY

CLK

- This pin provides the basic timing for the processor.
- 8086 does not has an on-chip clock generator hence an external clock generator like 8284 is used to provide the clock signal.
- It is asymmetric with 33% duty cycle, TTL clock signal.

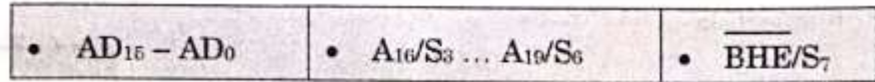
RESET

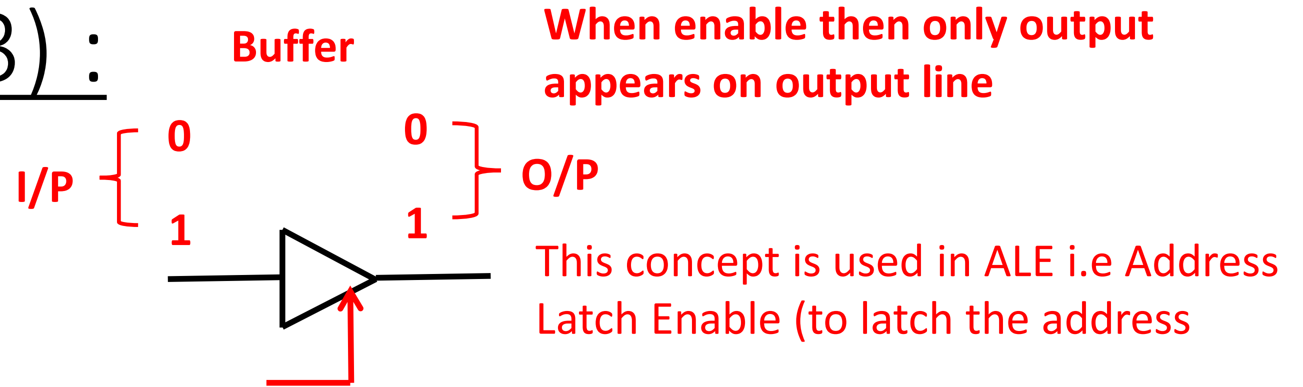
- It causes the processor to immediately terminate its present activity. The 8284 clock generator provides this signal.
- This signal must be active high for atleast 4 clock cycles
- It clears all the flag register, the Instruction Queue, the DS, SS, ES and IP registers and sets the bits of CS register.
- Hence the reset vector address of 8086 is FFFF0H (as CS = FFFFH and IP = 0000H).

READY

- It is an acknowledgement from the addressed memory or I/O that it will complete the data transfer specially meant for slow devices.
- μP samples the READY input between T2 and T3 of a M/C cycle.
- If READY pin is LOW, μP inserts wait-states between T2 and T3, until READY becomes HIGH.

3. Address and Data Pins (3) :

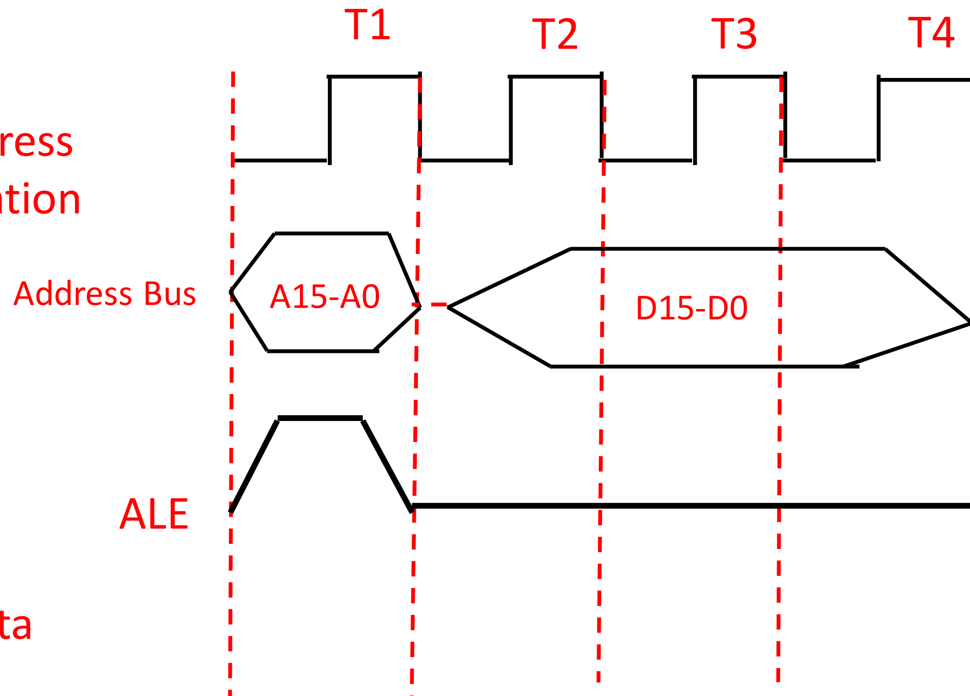




Tristate Buffer

$AD_{15} – AD_0$

- These are time multiplexed tristate data address lines i.e. for some time they have address and for some time data
- It gives the address $A_{15} – A_0$ during T1 of an Machine Cycle. (When ALE = 1)
- It gives the data $D_{15} – D_0$ after T1 of an M/C Cycle (Machine cycle).

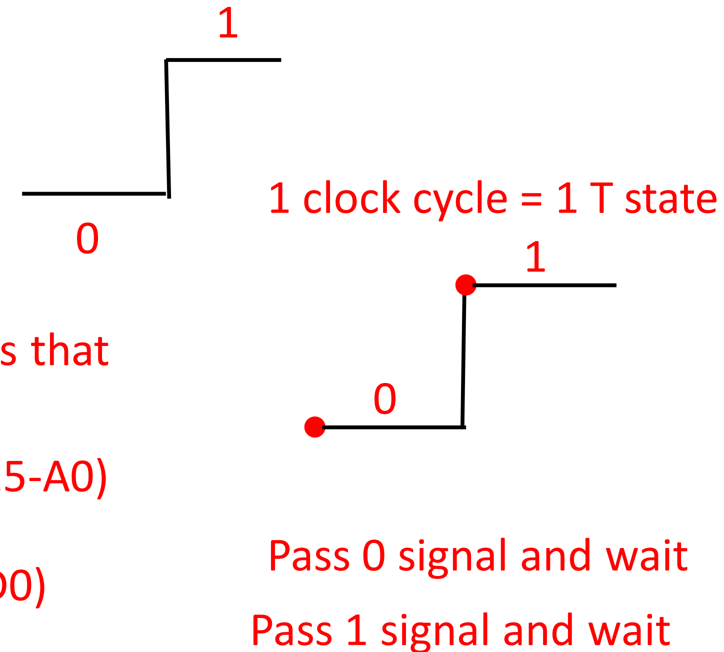


How to differentiate between data and address ???

Something is high which indicates that bus carries the address.

When ALE = 1 : Address bus (A15-A0)

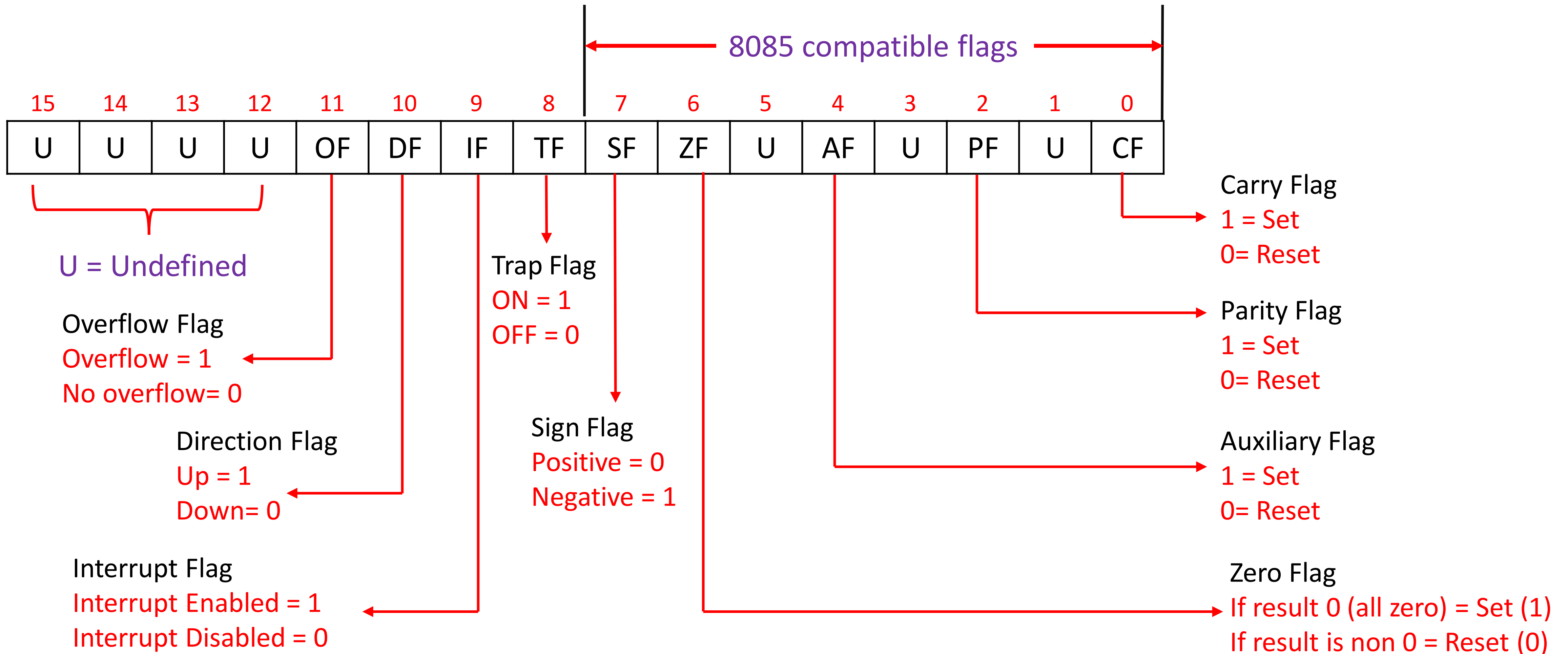
When ALE = 0 : Data bus (D15-D0)



Flag Register of 8086

Flag Register of 8086

- Flag register is a part of Execution Unit.
- It is a 16-bit register with each bit corresponding to a flip-flop.
- Flag register is used to give status of operation performed by processor.
- A flag is flip-flop.
- It indicates some condition produced by the execution of an instruction.



Carry Flag (CF)

- It can also be called as a final carry.
- This flag is set whenever there has been a carry out of, or borrow into, the MSB of the result (8 bit / 16 bit).
- The flag is used by the instruction that add and subtract multibyte numbers.
- CF = 1 , if there is a carry out from the most significant bit (MSB)
- CF = 0 , if no carry out from MSB

Example 1 (8 bit) :

ADD BL, CL where BL = 02 H and CL= 51 H

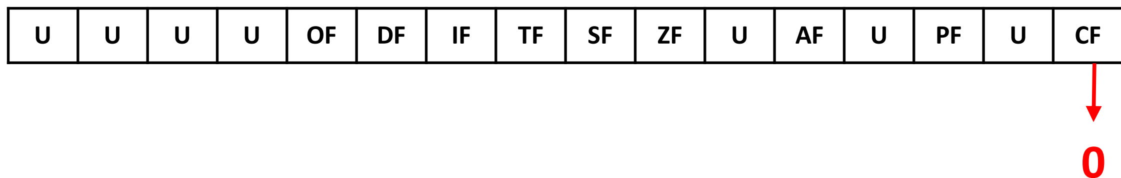
BL = 02 h = 0000 0010

+ CL = 51 h = 0101 0001

0101 0011

MSB LSB

Carry is not generated from MSB



Example 2 (8 bit) :

ADD BL, CL where BL = 83 H and CL= 81 H

BL = 83 h = 1000 0011

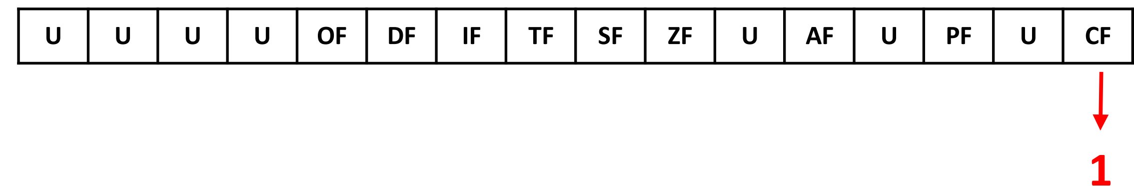
+ CL = 81 h = 1000 0001

1 1 1

0000 0100

MSB LSB

Carry is generated from MSB



Example 1 (16 bit) :

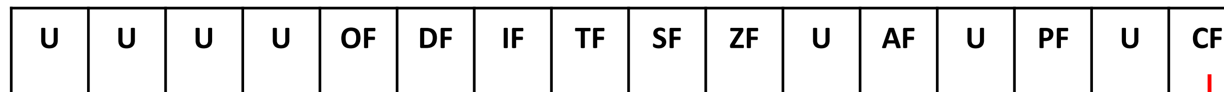
ADD BX, CX where BX = 0212 H and CX= 1251 H

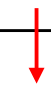
BX =	**0212 h**	**=**	**0 0 0 0**	**0 0 1 0**	**0 0 0 1**	**0 0 1 0**
CX =	**1251 h**	**=**	**0 0 0 1**	**0 0 1 0**	**0 1 0 1**	**0 0 0 1**

0 0 0 1 0 1 0 0 0 1 1 0 0 0 1 1

MSB LSB

Carry is not generated from MSB





0

Parity Flag (PF)

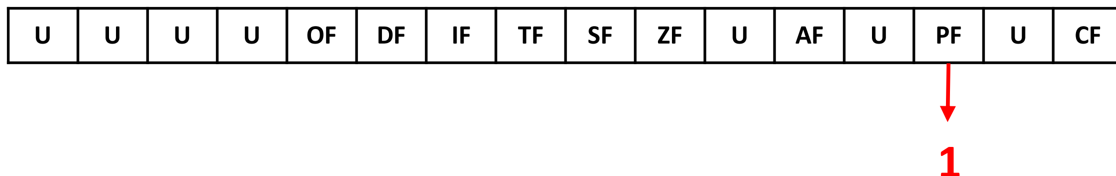
- This flag is normally used to check data transmission errors.
- PF = 1 , when the result has even parity, an even number of 1's
- PF = 0 , when the result has odd parity, an odd number of 1's

Example 1 :

ADD BL, CL where BL = 02 H and CL= 51 H

	BL =	02 h =	0000 0010
+	CL =	51 h =	0101 0001
			0000 0011

Result has even number of 1's

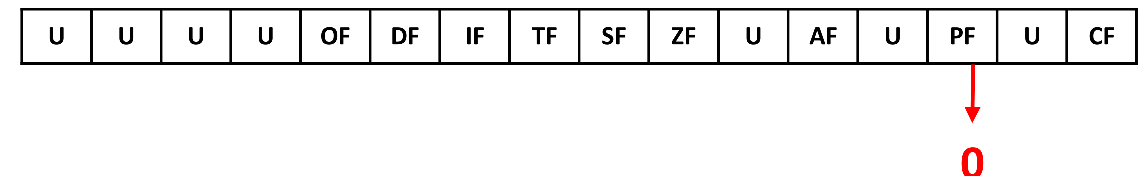


Example 2 :

ADD BL, CL where BL = 83 H and CL= 81 H

BL =	83 h =	1000 0011
CL =	81 h =	1000 0001
	carries: 1	1 1
		0000 0100

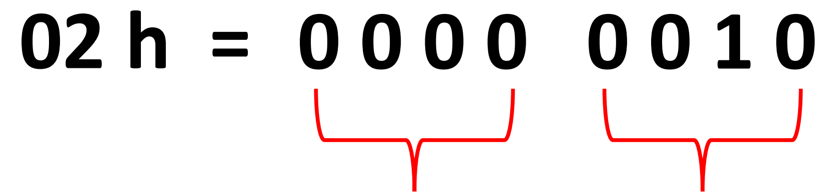
Result has odd number of 1's



Auxiliary Carry Flag (AF)

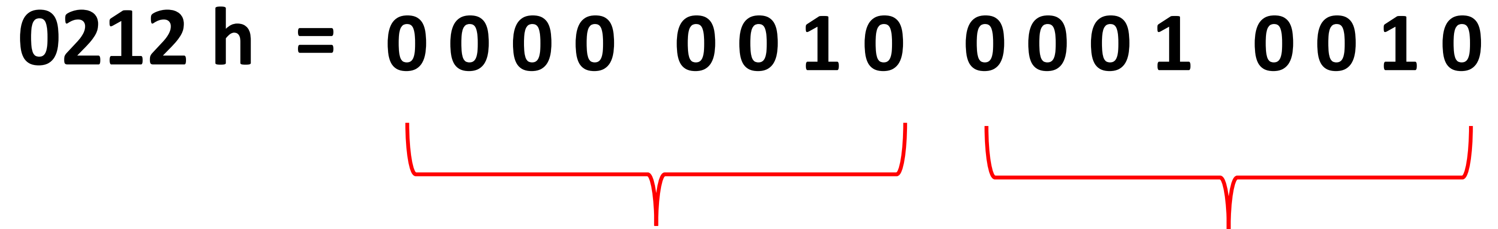
- It is a carry generated from lower nibble to upper nibble.
- AF = 1 , if carry or borrow generated from lower nibble to upper nibble.
- AF = 0 , if carry or borrow not generated from lower nibble to upper nibble.

8 bit data



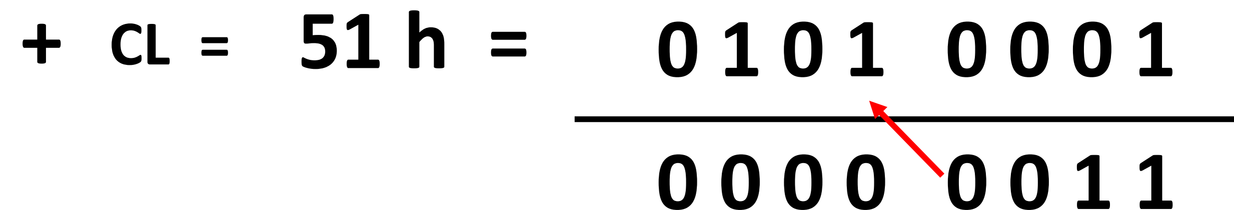
HB – Higher Nibble LB – Lower Nibble

16 bit data

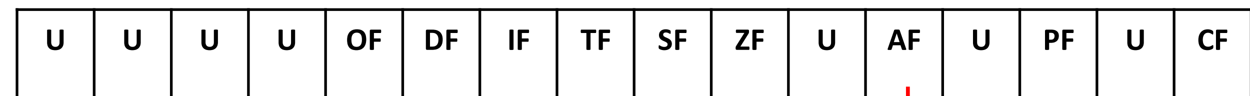


HB – Higher Nibble LB – Lower Nibble

ADD BL, CL where BL = 02 H and CL= 51 H



Carry is not generated from lower nibble to higher nibble.



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

0

ADD BL, CL where BL = 08 H and CL= 58 H

Carry is generated from lower nibble to higher nibble.

BL =	08 h =	0 0 0 0	1 0 0 0
+ CL =	58 h =	0 1 0 1	1 0 0 0
		1	
		0 0 0 0	0 0 0 0

U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF
											1				

Example 1 (16 bit) :

ADD BX, CX where BX = 0082 H and CX= 1281 H

BX =	0212 h =	0 0 0 0	0 0 0 0	1 0 0 0	0 0 1 0
CX =	1251 h =	0 0 0 1	0 0 1 0	1 0 0 0	0 0 0 1
			1		
		0 0 0 1	0 0 1 1	0 1 1 0	0 0 1 1

Carry is generated from lower nibble to higher nibble.

AF = 1

Zero Flag (ZF)

- This flag is normally used to check the result of operation is zero or non zero.
- This flag is monitor in Compare instruction.
- ZF = 1 , when the result consist all bits zero
- ZF = 0 , when the result is non zero which means at least one bit is 1.

Example 1 :

SUB BL, CL where BL = 02 H and CL= 02 H

```
    BL = 02 h =   0000 0010
 -  CL = 02 h =   0000 0010
                 ----------
                  0000 0000
```

Result has all zero

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

ZF ↓ 1

Example 2 :

ADD BL, CL where BL = 83 H and CL= 81 H

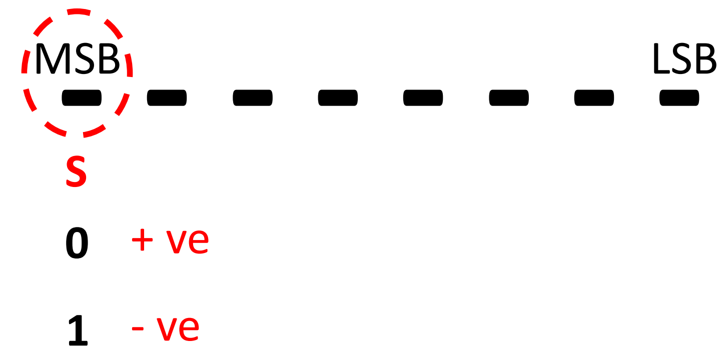
```
    BL = 83 h =    1000 0011
    CL = 81 h =    1000 0001
                 1        1 1
                 -----------
                   0000 0100
```

Result is non zero

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

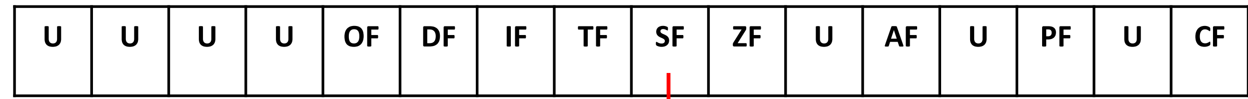
ZF ↓ 0

Sign Flag (SF)



- MSB of the result is used to indicate whether the result is positive or negative.
- SF = 0 , result is positive number
- SF = 1 , result is negative number

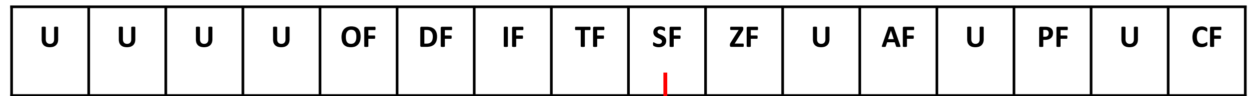
ADD BL, CL where BL = 02 H and CL= 51 H



0

	BL =	02 h =	0 0 0 0 0 0 1 0
+	CL =	51 h =	0 1 0 1 0 0 0 1
			0 0 0 0 0 0 1 1

	-23 h	1101 1101
+	-31 h	1100 1111
	- 54 h	1010 1100



1

Trap Flag (TF)

- Setting TF puts the processor into single step mode for debugging.
- In single stepping, microprocessor executes a instruction and enters into single step ISR.
- After that user can check registers or memory contents, if found ok, he/she will proceed the further, else necessary action will be taken.
- This utility is called as debug the program.
- if TF=1 , the CPU automatically generates an internal interrupt after each instruction, allowing a program to be inspected as it execute instruction by instruction.
- TF = 1 , Trap on (single instruction execution)
- TF = 0 , trap off (all instructions execution)

TF = 0 , trap off

Registers
AL =
BL = **04**
CL = **02**
DL =

Memory
MOV BL, 02h
MOV CL, 02h
ADD BL, CL

Processor execute all instructions and displays output

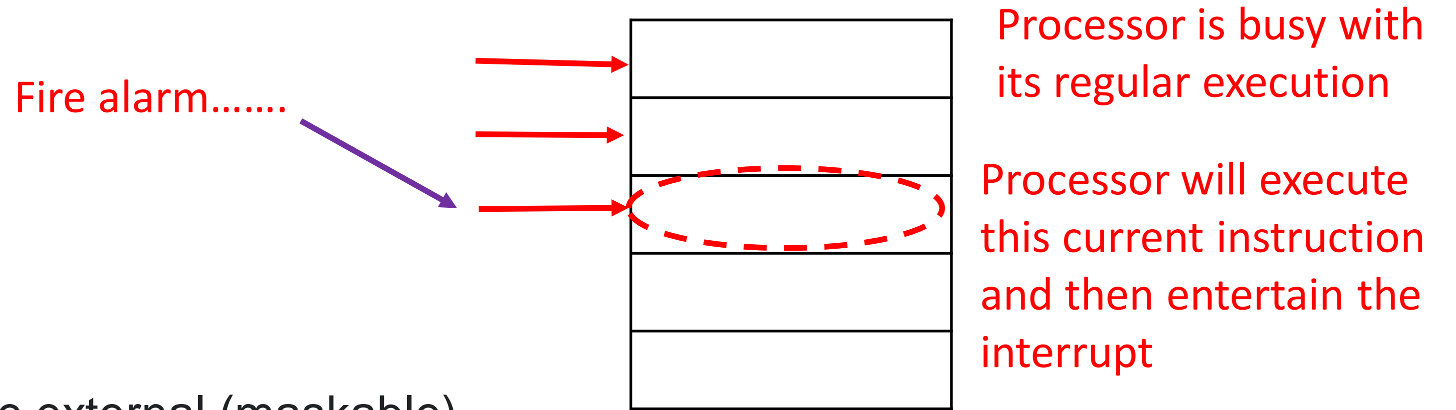
TF = 1 , trap ON

Registers
AL =
BL = ~~02~~ **04**
CL = **02**
DL =

Memory
MOV BL, 02h ←
MOV CL, 02h ←
ADD BL, CL ←

Processor instruction one by one and wait

What is an interrupt??? Ans : To disturb regular flow of processor execution

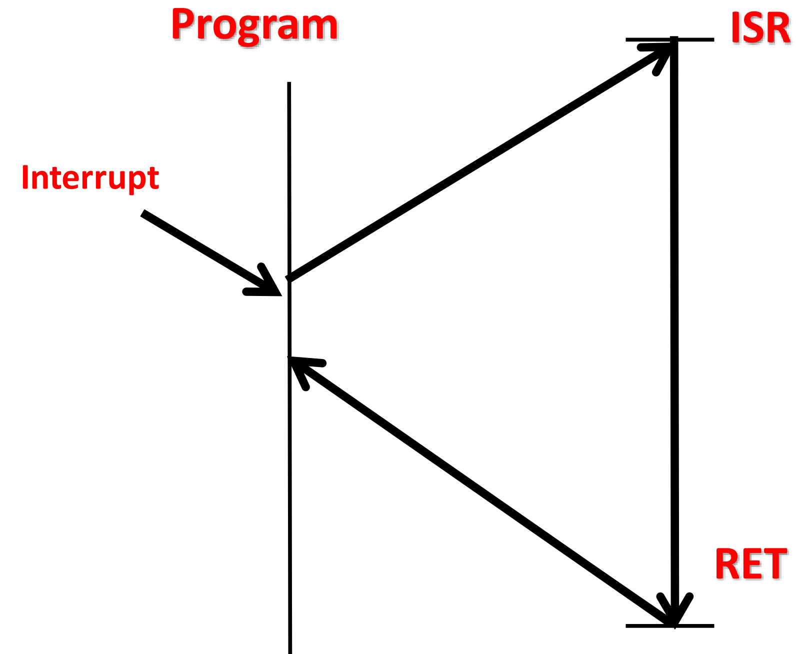


Interrupt Flag (IF)

- If user sets IF flag, the CPU will recognize external (maskable) interrupt requests.

- Clearing IF disables these interrupts.

- IF = 1 , interrupt enabled.
- IF = 0 , interrupt disabled.



Overflow Flag (OF)

- It indicates an overflow from the magnitude to the sign bit of result.
- If OF is set, an arithmetic overflow has occurred, that is a significant bit has been lost because the size of the result exceeded the capacity of its destination location.
- In 8086 interrupt on overflow instruction is available that will generate an interrupt in this situation.
- OF = 1 , signed overflow occur
- OF = 0 , no overflow

Number System

Signed Numbers

Positive & Negative
+ ve & - ve

Unsigned Numbers

they assume to be positive

No sign i.e. no + ve nor - ve

Unsigned Numbers

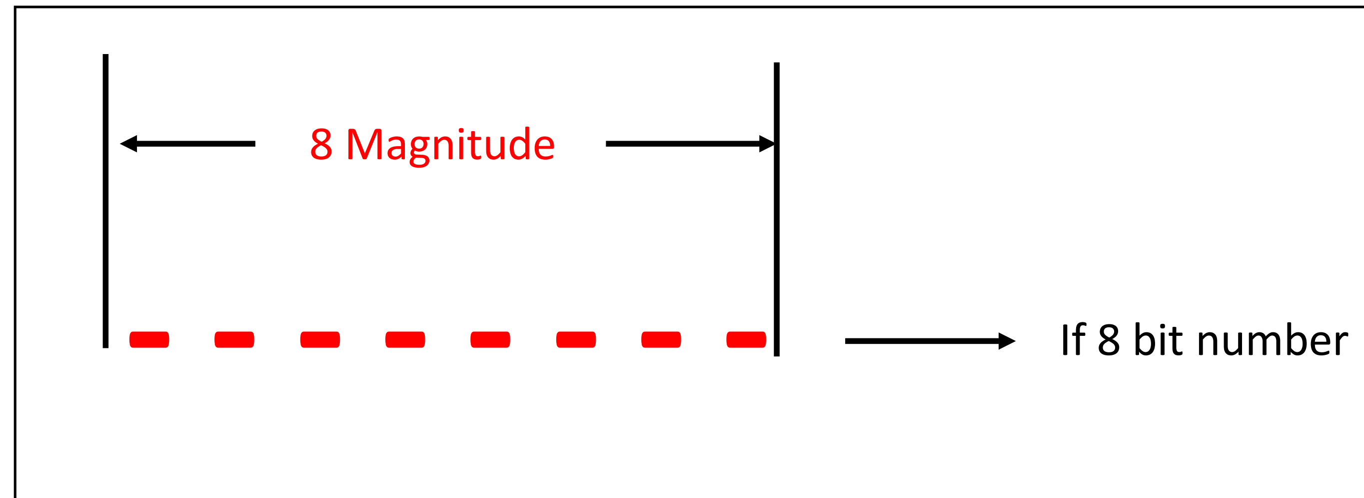
All Positive numbers

Example : Roll Number

For Unsigned numbers if magnitude is 8 then,

$2^8 = 256$

i.e. total 256 + ve numbers are used



Range for unsigned numbers

00	0000 0000
01	0000 0001
FF	1111 1111

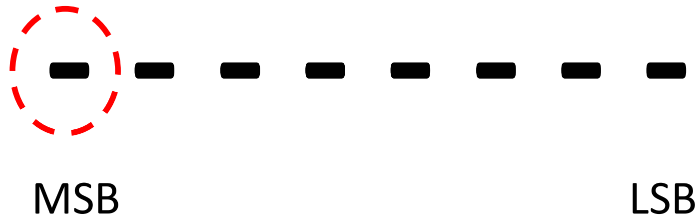
BACK

Signed Numbers

Positive & Negative
+ ve & - ve

How to find whether number is + ve or - ve?

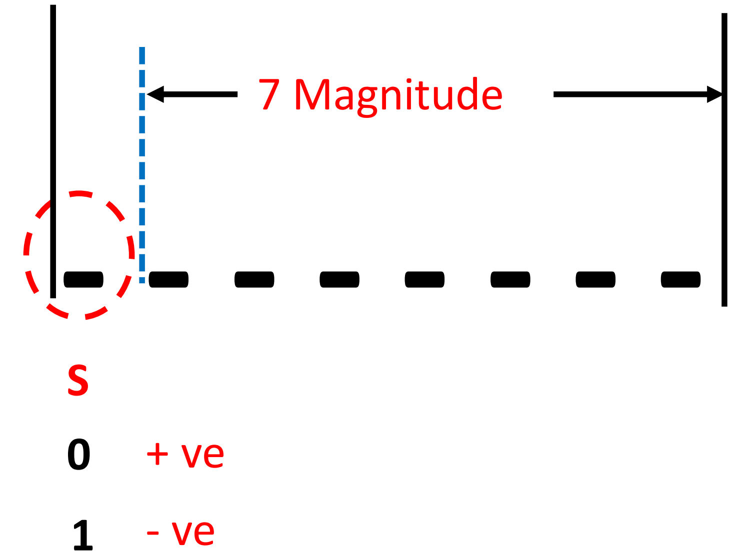
Answers :



If MSB of number is 0 then + ve

If MSB of number is 1 then - ve

If number is
8 bit



For signed numbers if magnitude is then 7 ,
$2^7 = 128$
i.e. 128 **- ve** numbers & 128 **+ ve** numbers

-128 to -1

0 to 127

Range for +ve and –ve numbers

Range for +ve and –ve numbers

Range for Positive numbers (0 to 127)

00	0000 0000
01	0000 0001
7F	0111 1111

Range for Negative numbers (- 80 to -01)

80	1000 0000
FF	1111 1111

What is unsigned and signed number following binary ?

1 0 0 0 0 0 1 1

Unsigned

83 H

Signed

~~03 H~~ 7D H

- ve number means 2's complement of given number

Shortcut for 2's complement : copy number as it is from right side till gets first 1 after 1 complement all numbers

example 1 :

+ 24 h	**0010 0100**
2's complement of 24 i.e. -24	**1101 1100**

example 1 :

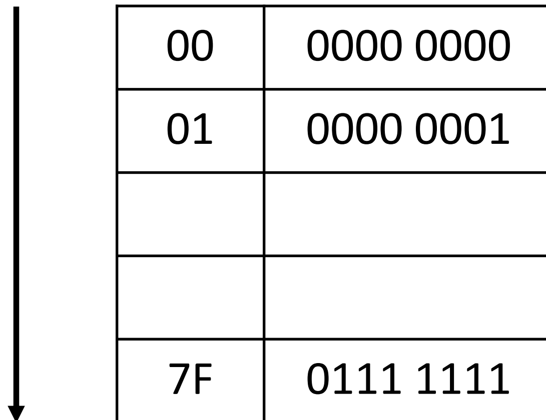
+ 5 h	**0101**
2's complement of 05 i.e. -5	**1011**

NEXT

Overflow Flag :

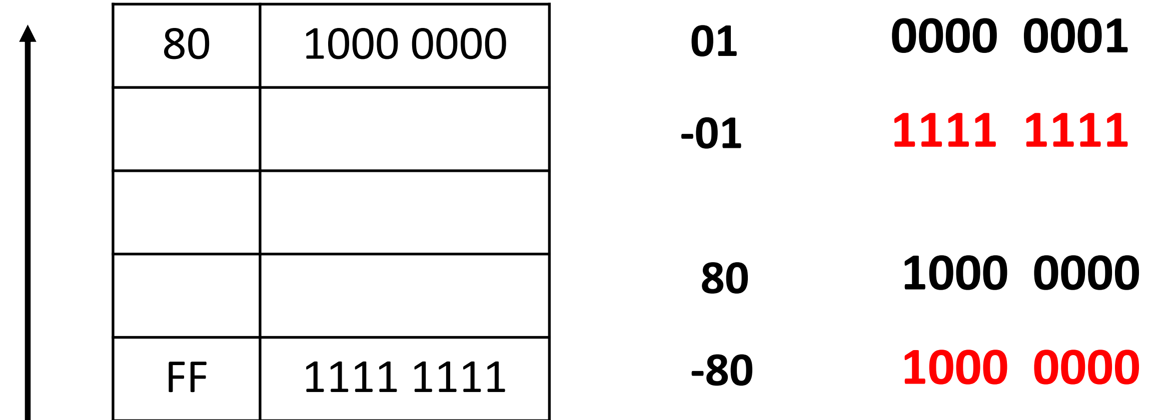
Overflow flag matters only for signed numbers

Range for Positive numbers (0 to 127)



00	0000 0000
01	0000 0001
7F	0111 1111

Range for Negative numbers (- 128 to -01)



80	1000 0000
FF	1111 1111

Range for Positive numbers : 00 to 7F

Range for Negative numbers : FF to 80

After addition if result is going beyond above ranges then overflow flag is set i.e. OF = 1

Example for overflow flag:

Overflow flag matters only for signed numbers

23 h	0010 0011		
+ 31 h	0011 0001		
54 h	0101 0100		
CY	AC	OF	P
0	0	0	0

-23 h	1101 1101		
+ -31 h	1100 1111		
- 54	1010 1100		
CY	AC	OF	P
1	1	0	1

27 h	0010 0111		
+ 39 h	0011 1001		
60 h	0110 0000		
CY	AC	OF	P
0	1	0	0

-27 h	1101 1001		
+ - 39 h	1100 0111		
- 60 h	1010 0000		
CY	AC	OF	P
1	1	0	1

42 h	0100 0010		
+ 43 h	0100 0011		
85 h	1000 0101		
CY	AC	OF	P
0	0	1	0

-42 h	1011 1110		
+ -43 h	1011 1101		
-85 h	0111 1011		
CY	AC	OF	P
1	1	1	1

1. Using MSB bit we can identify whether number is +ve or –ve
2. If MSB is 1 it means number is – ve
3. But sometimes it will give wrong sign bit
4. In such cases checking only MSB is not sufficient
5. We have to check range of both numbers
6. If number cross range it means there is overflow problem

Range for Positive numbers : 00 to 7F (0 to 127)

Range for Negative numbers : FF to 80 (-80 to -01)

Example :

7F h	0111 1111
+ 01 h	0000 0001
80 h	1000 0000

↑

Result is positive and answer gives sign bit negative.

General Purpose Registers of 8086

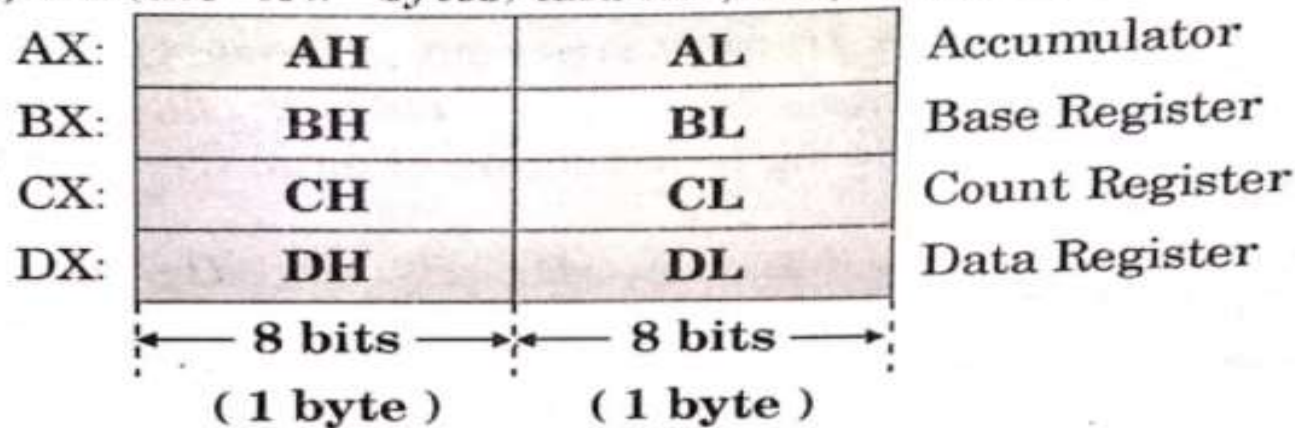
General Purpose Registers of 8086

- As shown in Fig. the execution unit (EU) has four general purpose 16-bit registers.
- Each one of them can be used for temporary storage of 8 bit data, 16-bit data or 32-bit data.
- 8-bit Registers - AH, AL, BH, BL, CH, CL, DH, DL. Any of these registers can be used as an 8-bit operand.
- 16-bit Registers - AX. BX, CX, DX, SI, DI, SP, BP. Any of these registers can be used as a 16-bit operand.
- 32-bit Registers – DX : AX together can be used for 32-bit operand.
- These registers can be used for general purpose computing when their other specialized functions do not interfere.

- The above mentioned register pairs are referred to as follows.

Pair of general purpose registers	Referred to as
1. Pair of AH and AL	AX register
2. Pair of BH and BL	BX register
3. Pair of CH and CL	CX register
4. Pair of DH and DL	DX register

- In short, the four general-purpose 16-bit data registers : AX, BX, CX, and DX, each of these is a combination of two 8-bit registers which are separately accessible as AL, BL, CL, DL (the "low" bytes) and AH, BH, CH, and DH (the "high" bytes).



- For example, if AX contains the 16-bit number 1234H, then AL contains 34H and AH contains 12H.
- The upper and lower halves of the data registers are separately addressable. This means that each data register can be used as a single 16-bit register or as two 8-bit registers.

Special functions of general purpose registers of 8086

1. Register AX : Accumulator

AX is the "16-bit accumulator" while AL is "8-bit accumulator"

Accumulator has the following special functions :

(i) Some of the operations, such as Multiplication and Division, require that one of the operands be in the accumulator and also the result is stored in accumulator. Some other operations, such as Addition and Subtraction, may be applied to any of the registers (that is, any of the eight general- and special-purpose registers) but are more efficient when working with the accumulator.

(ii) It works as a via register for I/O accesses i.e. a data is routed through accumulator for the communication of the processor and I/O devices.
For OUT instruction the data in accumulator (AL for 8-bit data and AX for 16-bit data) can only be given to the output device.
For IN instruction the data taken from the input device can be taken only in accumulator (AL for 8-bit data and AX for 16-bit data)

(iii) It also works as a via register for string instructions. Whenever a data is to be brought from memory or given to memory in case of string operations it is routed through accumulator only.

2. Register BX : Base

BX is the "base" register,

(i) It is the only general-purpose register which may be used for indirect addressing. (various addressing modes are discussed in chapter 4.)

(ii) For example, the instruction MOV [BX], AX causes the contents of AX to be stored in the memory location whose address is given in BX.

3. Register CX : Counter

CX is the "count" register. It works as a default counter register for three instructions viz :

(i) The looping instructions (LOOP, LOOPE, and LOOPNE), to indicate the number of iterations

(ii) The shift and rotate instructions (RCL, RCR, ROL, ROR, SHL, SHR, and SAR), to indicate number of shifts or rotations (Here only CL is used and not entire CX)

(iii) The string instructions (with the prefixes REP, REPE, and REPNE) to indicate the size of the string block.

4. Register DX : Data

DX is the "data" register

(i) It is used together with AX for the word-size MUL and DIV operations, when the operand size is greater than the register AX i.e. operand is 32-bit.

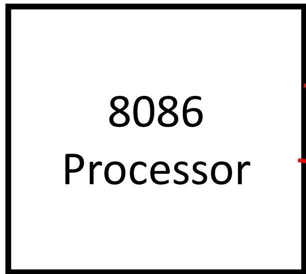
(ii) It also holds the port number for the IN and OUT instructions. For 16-bit address accesses of I/O ports only DX can be used as a pointer.

Summary of Implicit use of General Purpose Registers

Registers	Operations
AX	Word multiply, Word divide, Word I/O and Word string
AL	Byte multiply, byte divide, byte I/O, byte string and decimal / ASCII arithmetic.
BX	Store address information
CX	Counter for String operations and loops
CL	Counter for Variable shift and rotate
DX	Word multiply, word divide, Indirect I/O

Memory Bank of 8086

Why memory bank required??????



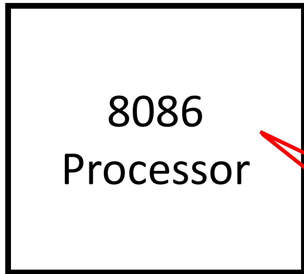
8086 is 16 bit processor hence can access 16 bits at a time.

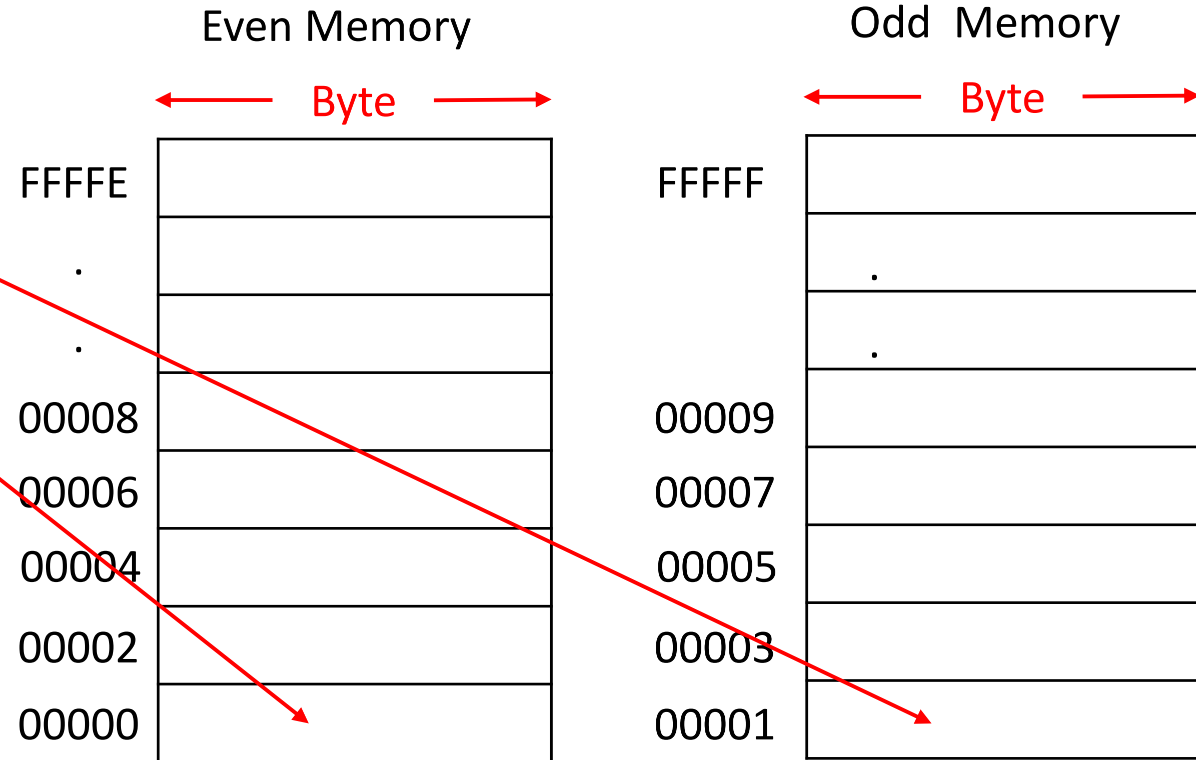
Hence processor will require 2 cycles to fetch 16 bits from memory.

To avoid above problem memory bank concept is used. Processor can access two consecutive memory locations in one cycle.

Memory bank:

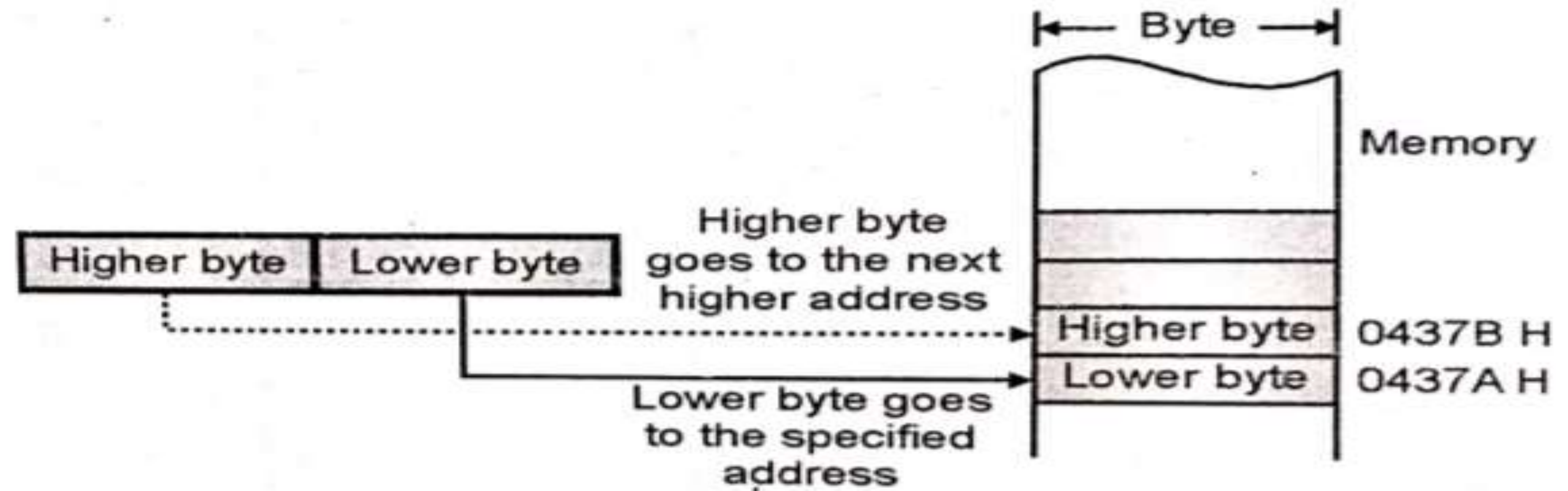
- Even Address Memory Bank
- Odd Address Memory Bank





Processor can read or write two memory locations of 16 bit in single cycle.

- We know that the 8086 has a 20-bit address bus, so it can address 2^{20} or 1,048, 576 addresses.
- At each address we can store an 8-bit data (1 byte). Hence the total memory capacity of 8086 is 1 M byte.
- However the data bus of 8086 is 16 bit and the processor is capable of processing 16 bit data i.e. words.
- The question is how to write a word (16 bit data) into a memory which is segmented to store the data in the byte form.

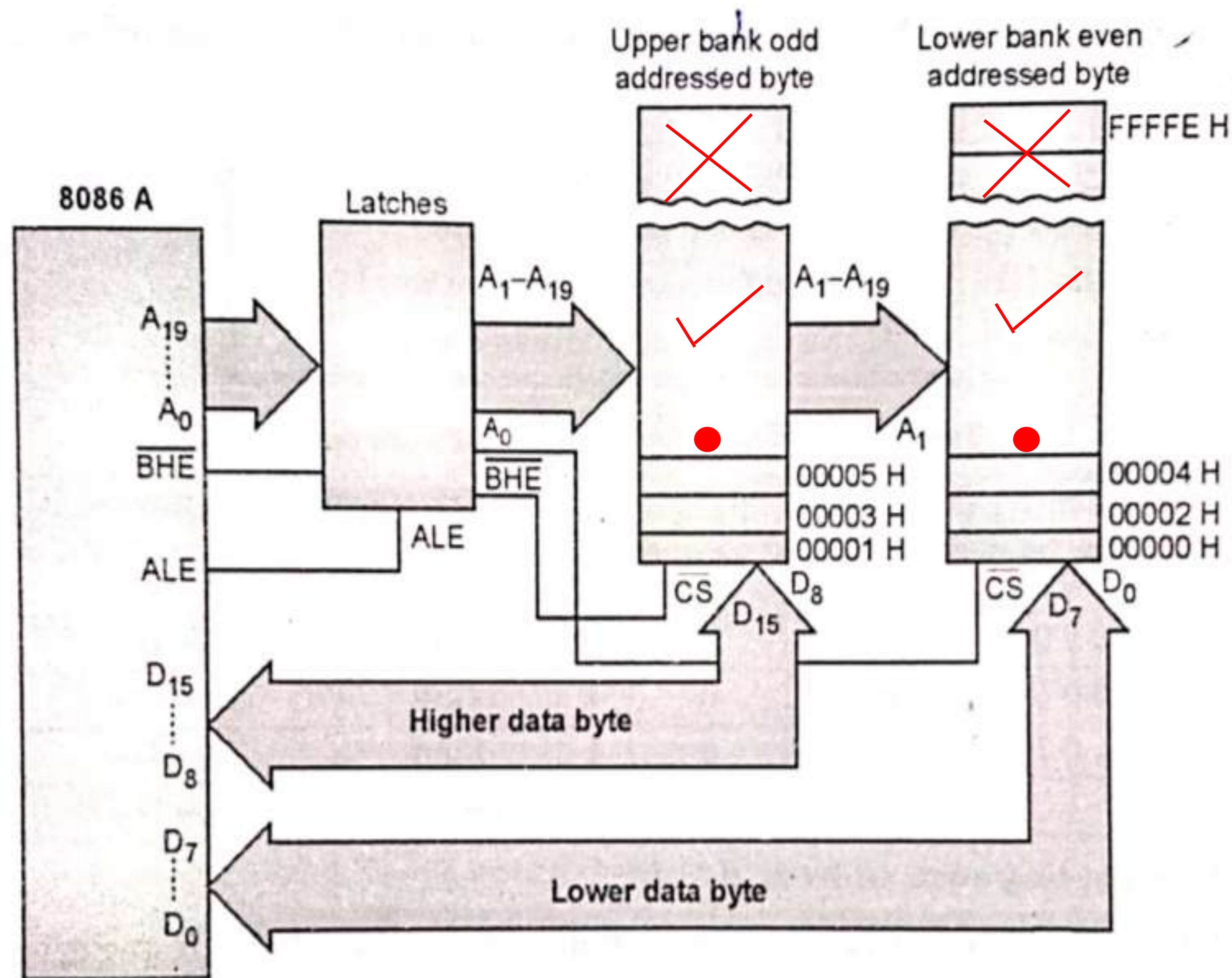


m(15.9)**Fig. 2.8.1 : How is a word stored in the 8086 memory ?**

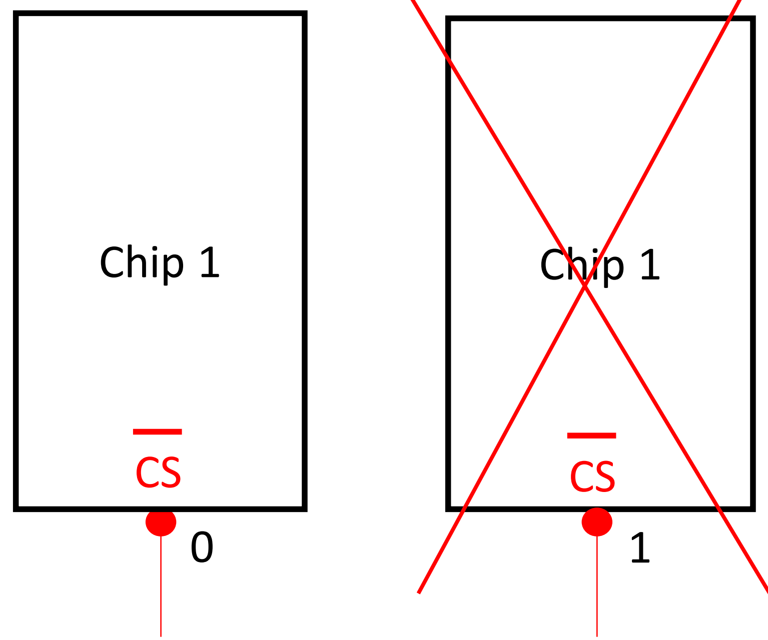
- The answer is that a word is written into two consecutive memory addresses.
- That means the lower byte is written into the specified memory address say 0437AH and the higher byte is written into the next higher address as illustrated in Fig. 2.8.1.
- In order to make it possible to read or write a word with one machine cycle, the memory of 8086 is divided into two "banks" of upto 524, 288 bytes i.e. 500 K bytes each, as shown in Fig. 2.8.2.

Block diagram of 8086 memory bank

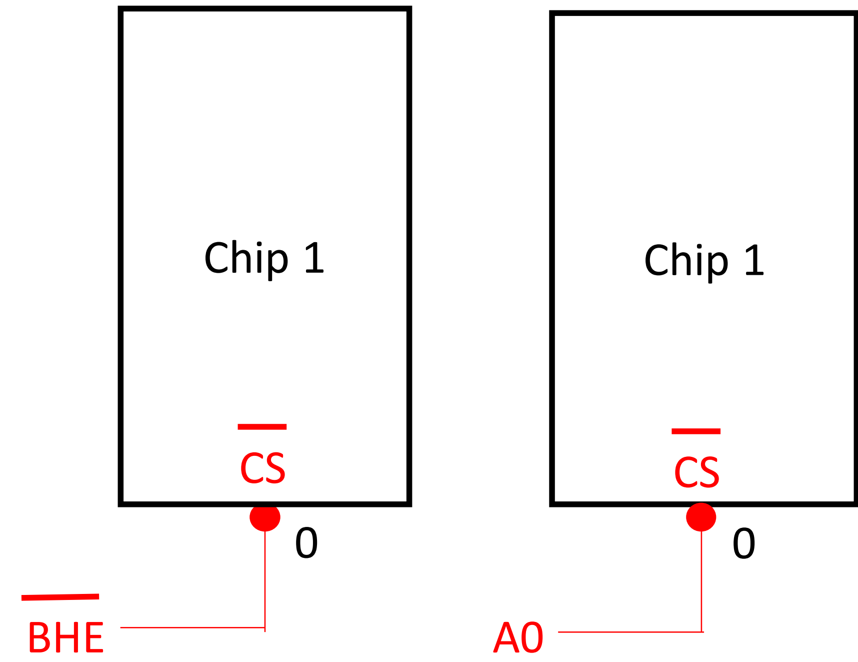
$\overline{BHE}$	A0	Bank
0	0	Both bank (16 bit)
0	1	Odd (8 bit)
1	0	even (8 bit)
1	1	No Bank



To start the working of Chip, every chip has to activate.
To activate or enable particular chip there is a pin which is called as chip select .
This pin is active low, hence CS=0 will enable the chip.



For 8086 memory chip :



- The even addressed memory bank contains all the bytes which have even addresses such as 00000, 00002, 00004, ... etc.
- The odd addressed memory bank contains all the bytes which have odd addresses such as 00001, 00003, 00005 etc.
- The even addressed bank is also called as lower bank and the data lines of this bank are connected to the lower eight data lines i.e. D_0 to D_7 of 8086 as shown in Fig.
- The odd addressed bank is also called as the upper bank and the data lines of this bank are connected to the upper eight data lines i.e. D_8 to D_{15} of 8086 as shown in Fig.
- The address line A_0 is used to enable the lower memory bank because A_0 is connected to the $\overline{CS}$ input of the lower bank.
- Address lines A_1 through A_{19} are used for selecting the desired memory device in the bank and to address the desired byte in that device.
- The bus high enable ($\overline{BHE}$) is used for enabling the memory in the upper bank.
- $\overline{BHE}$ is multiplexed out on a single line from 8086 at the same time when an address is sent out.
- The ALE signal is used to strobe in the address into the external latch. The same latch stores the $\overline{BHE}$ signal as well and holds it stable for the remaining machine cycle.

Different Types of memory access :

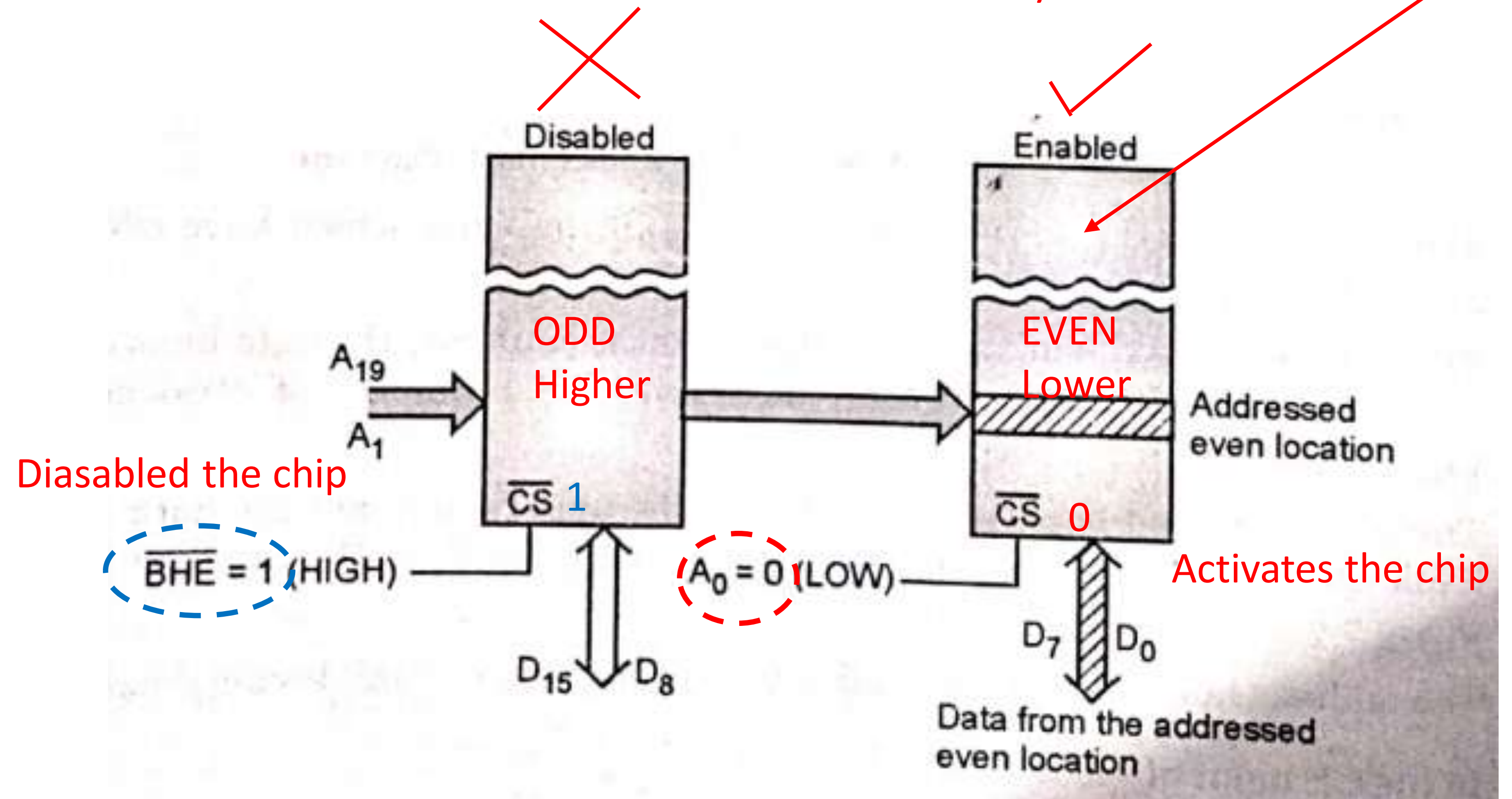
i. Accessing Even Addressed Byte — Lower byte i.e. Lower 8 bits
ii. Accessing Odd Addressed Byte — Higher byte i.e. Higher 8 bits
iii. Accessing Even Addressed word — All 16 bits , first lower then higher
iv. Accessing Odd Addressed Word — All 16 bits , first higher then Lower

1. Accessing Even Addressed Byte

Lower byte i.e. Lower 8 bits

How many cycles are required to perform this operation ??????

1 Cycle



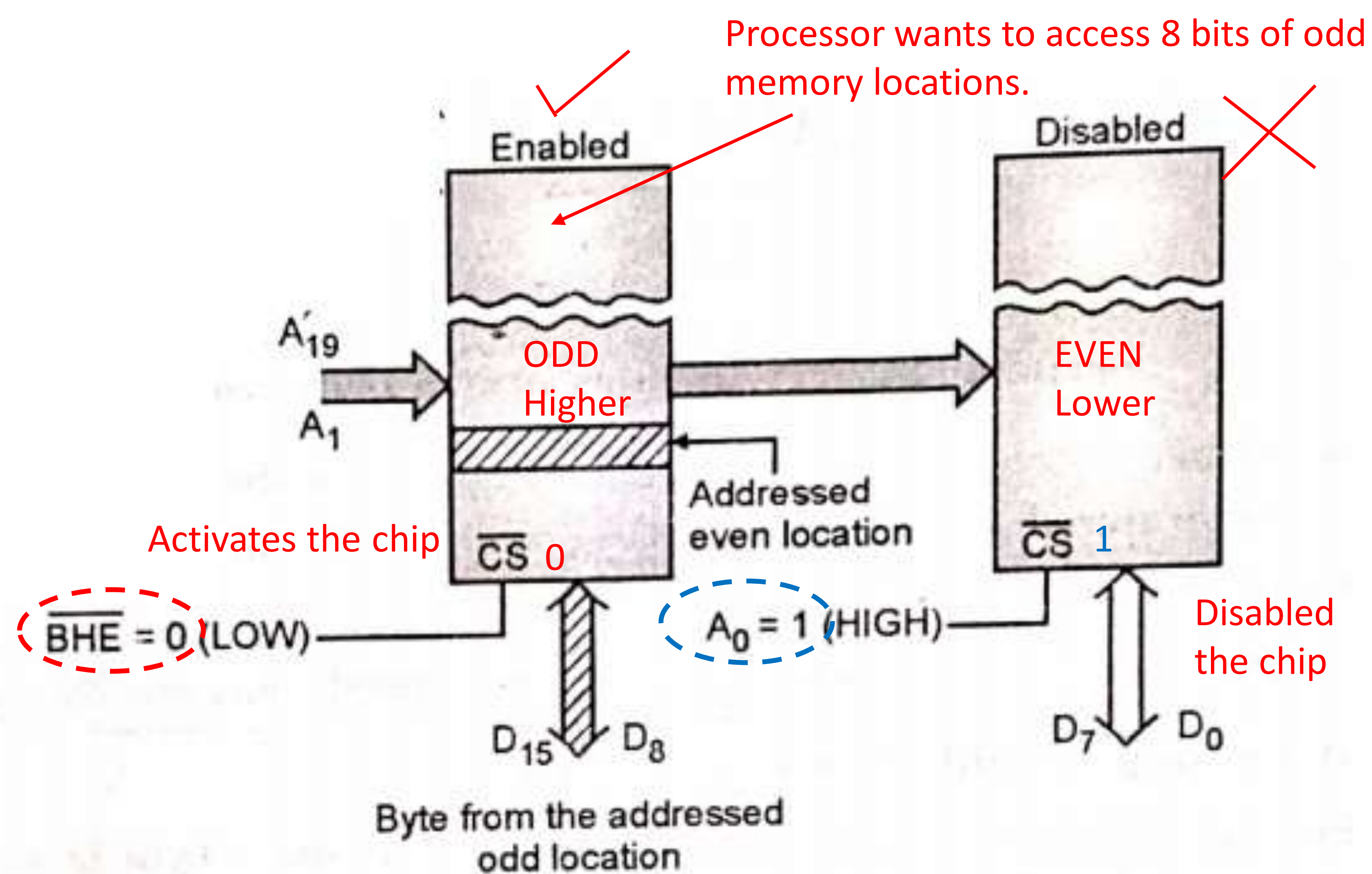
- The 8086 forces A_0 line LOW and $\overline{BHE}$ HIGH.
- This will enable the lower memory bank and disable the higher memory bank.
- The 8086 outputs the address of the desired even memory location on the address lines A_1 to A_{19}.
- The data stored (byte) at the addressed memory location appears on the data lines $D_0 - D_7$ as shown by the hatched lines in Fig.

2. Accessing Odd Addressed Byte

Higher byte i.e. Higher 8 bits

How many cycles are required to perform this operation ??????

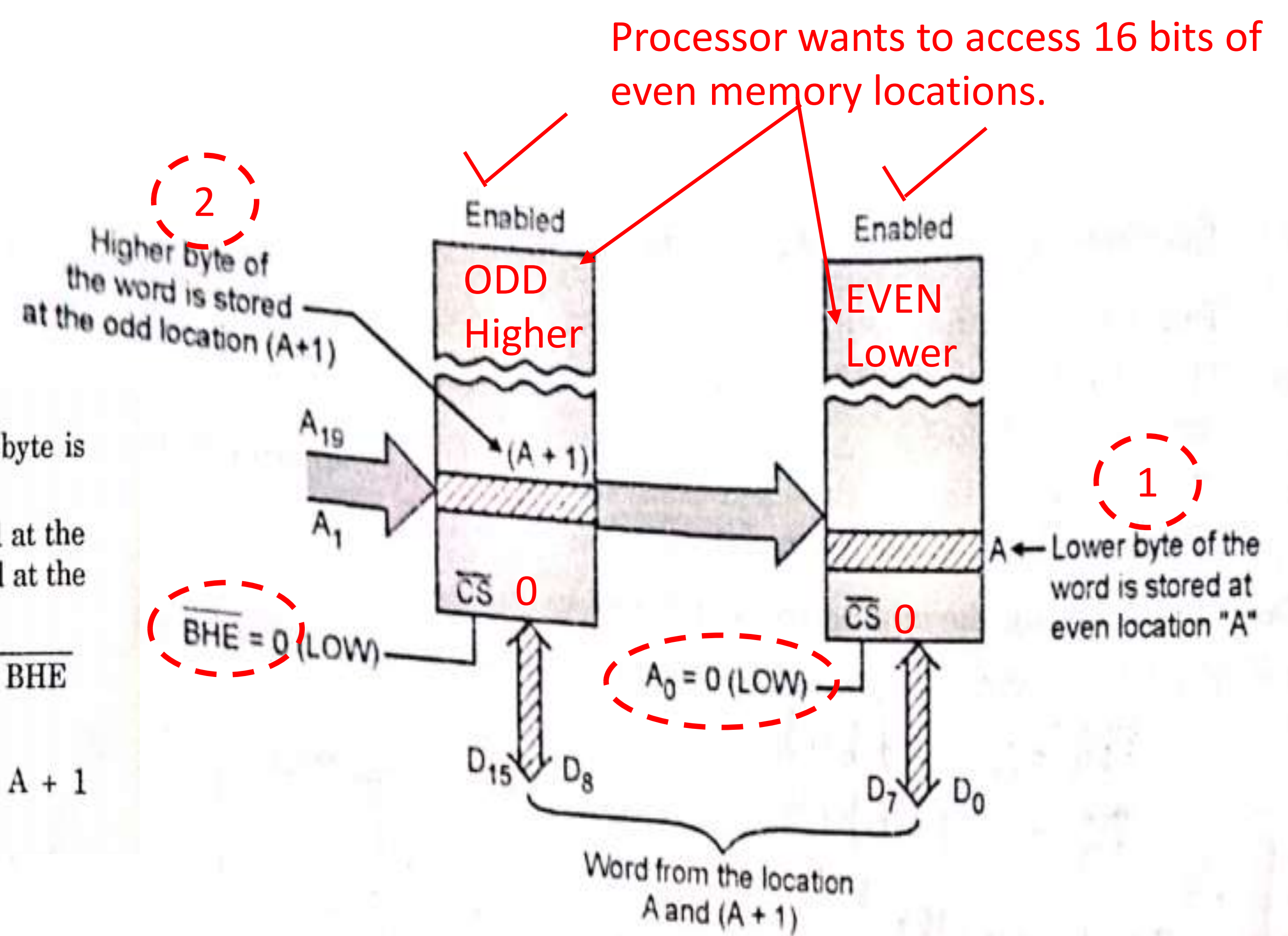
1 Cycle



- The 8086 forces $A_0 = 1$ (HIGH) and BHE = 0 (LOW).
- This will disable the lower (even) memory bank and enable the upper memory bank.
- The 8086 will output the address of desired odd memory location on A_1 to A_{19} address lines.
- The data byte on the addressed odd location appears on the data lines $D_8 - D_{15}$. as shown in Fig.

3. Accessing Even Addressed Word

All 16 bits , first lower 8 bits then higher 8 bits



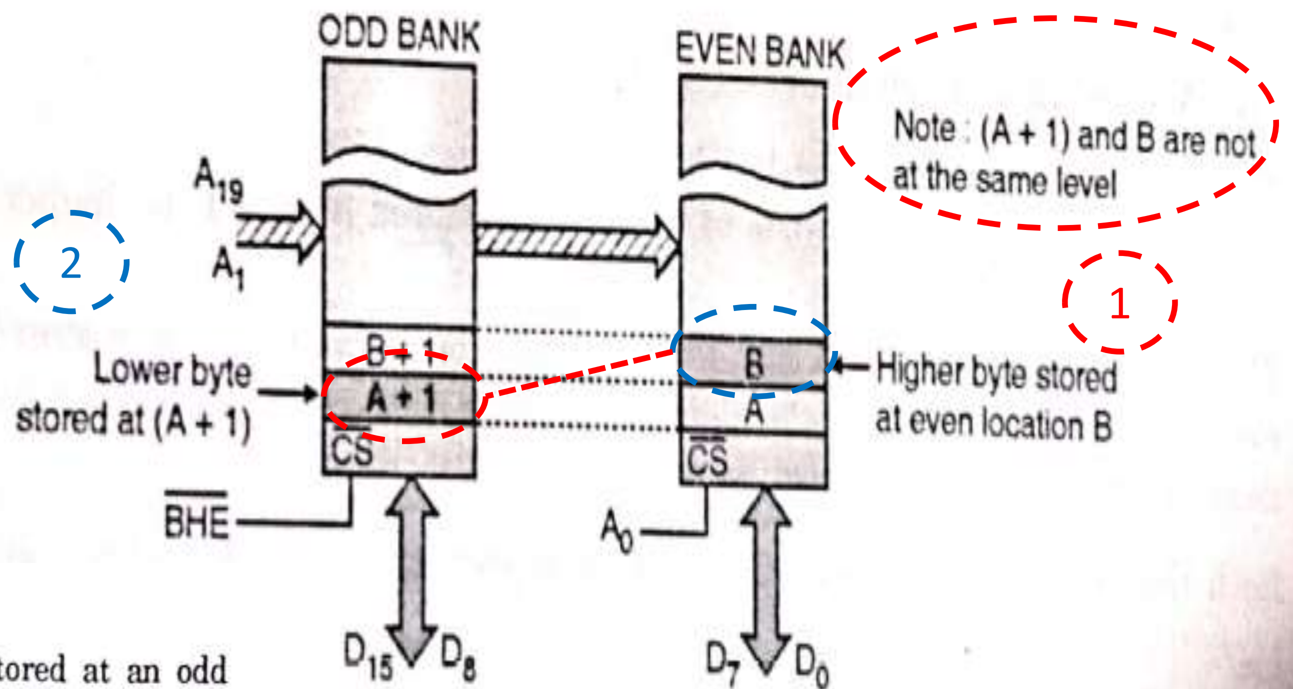
- Here the 8086 wants to read a 16 bit word.
- As stated earlier, the lower byte of this word is stored first and the higher byte is stored at the next higher.
- This is illustrated in Fig. 2.8.3(c). The lower byte of the word has been stored at the even memory location A and the higher byte of the same word has been stored at the next location (odd memory location A + 1).
- Both the memory banks are enabled because 8086 will force A_0 as well as $\overline{BHE}$ low.
- The address on the lines $A_{19} - A_1$ will point towards the locations A and A + 1 simultaneously.
- The lower byte stored at location A will now appear on the data lines $D_0 - D_7$ and the higher byte stored at location (A + 1) will appear on the data lines $D_8 - D_{15}$ simultaneously.

How many cycles are required to perform this operation ??????

1 Cycle

4. Accessing Odd Addressed Word

All 16 bits , first Higher 8 bits then Lower 8 bits



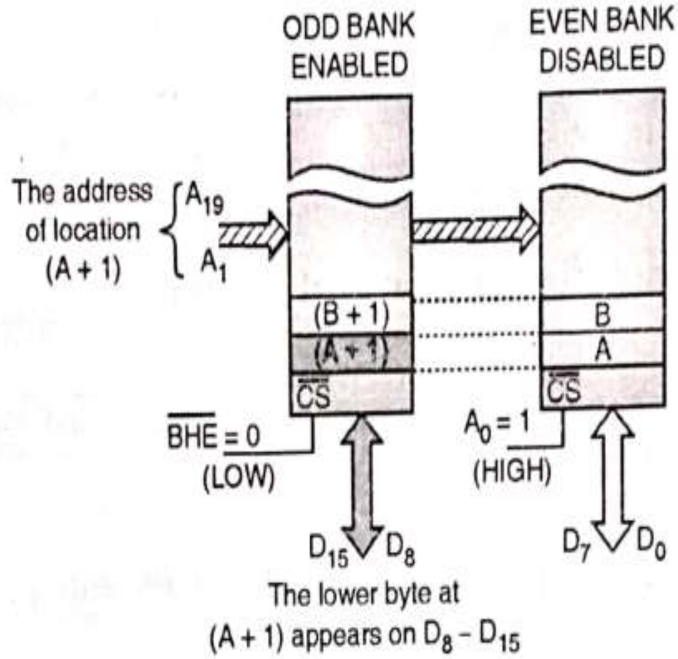
- In this case the 8086 wants to read a word, the lower byte is stored at an odd address. (say A + 1) and the upper byte of the word is at even address (i.e. at B immediately next to A + 1).
- Now refer Fig. 2.8.3(d) and notice that the locations (A + 1) and B are not at the same level.
- Therefore the 8086 cannot use the same address for both these locations, (A + 1) and B.
- Therefore it is not possible for 8086 to output a single address and read all the 16 bits of the word simultaneously.
- Hence the 8086 needs two bus cycles, to read the 16 bit word. One bus cycle is required to read the contents of location (A + 1) and other to read the contents of location B. The sequence of operations in these two bus cycles is as explained below.

How many cycles are required to perform this operation ??????

2 Cycle

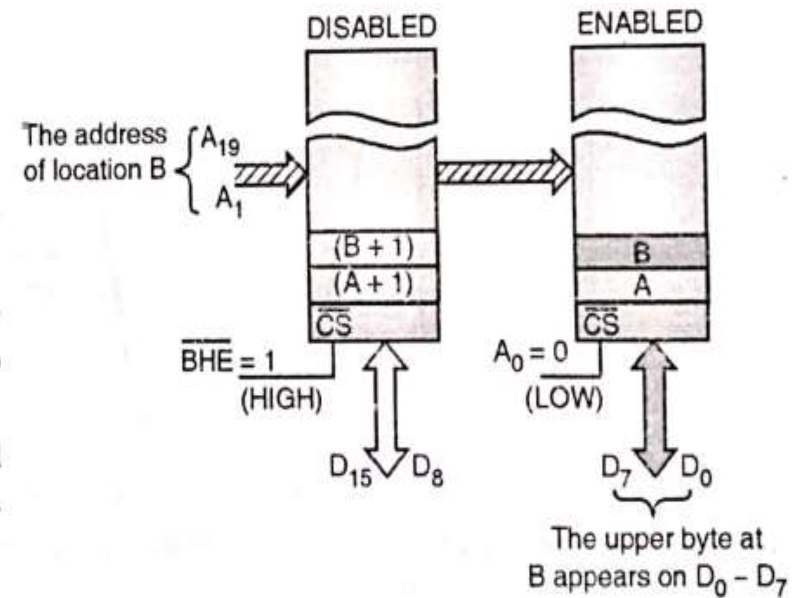
First bus cycle :

- In the first bus cycle, the 8086 forces $A_0 = 1$ (HIGH) and $\overline{BHE} = 0$ (LOW) so as to enable the odd bank.
- It then outputs the address on $A_1 - A_{19}$ lines to point at the location (A + 1).
- The contents of location (A + 1) will appear on the data lines D_8 to D_{15}, as shown in Fig. 2.8.3(e).



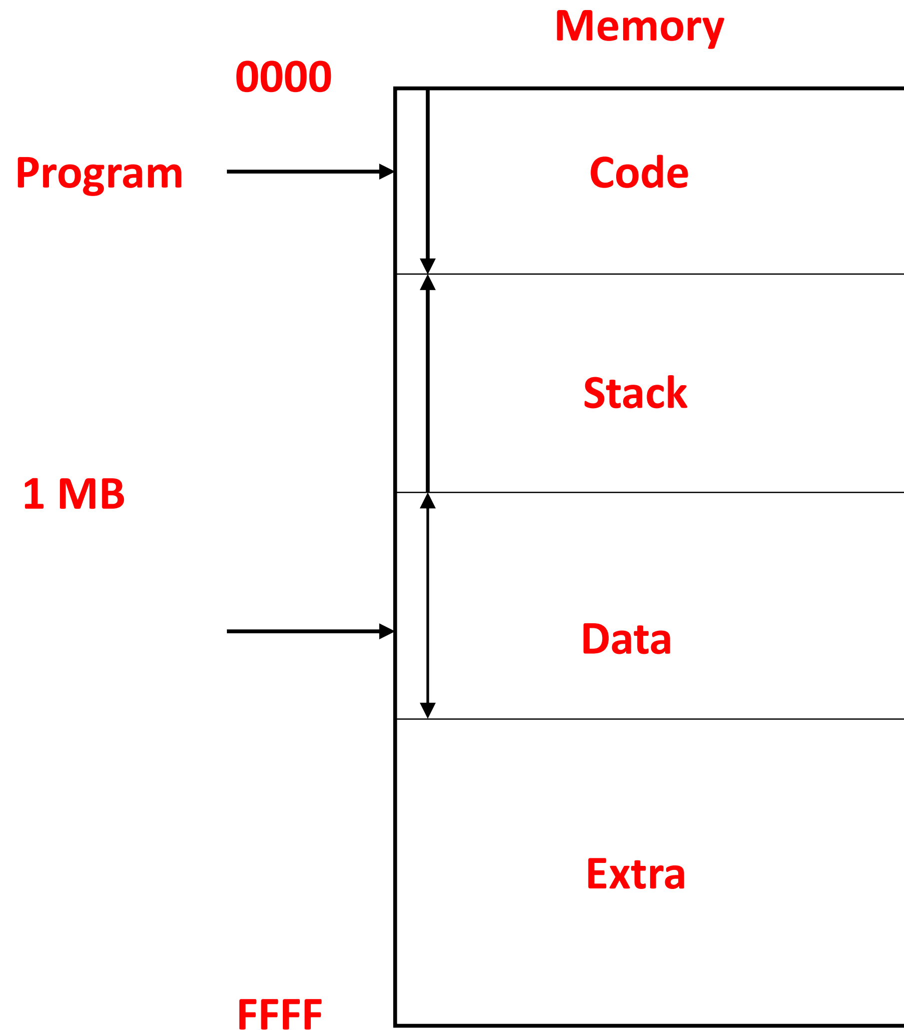
Second bus cycle :

- In the second bus cycle the 8086 forces $A_0 = 0$ (LOW) and $\overline{BHE} = 1$ (HIGH) so as to enable the lower (even) bank.
- It then outputs the new address on the $A_1 - A_{19}$ lines to point at location B.
- The contents of location B will appear on $D_0 - D_7$ as shown in Fig. 2.8.3(f).



Memory Segmentation of 8086

Problem : Memory accessed by 8085



- Memory and Processor are the different chips.
- Memory is used to stored code, stack and data.
- Code is referred as a program which is always stored in sequence.
- Data can stored in any direction.
- Stack is used to stored the data in last in first out manner.

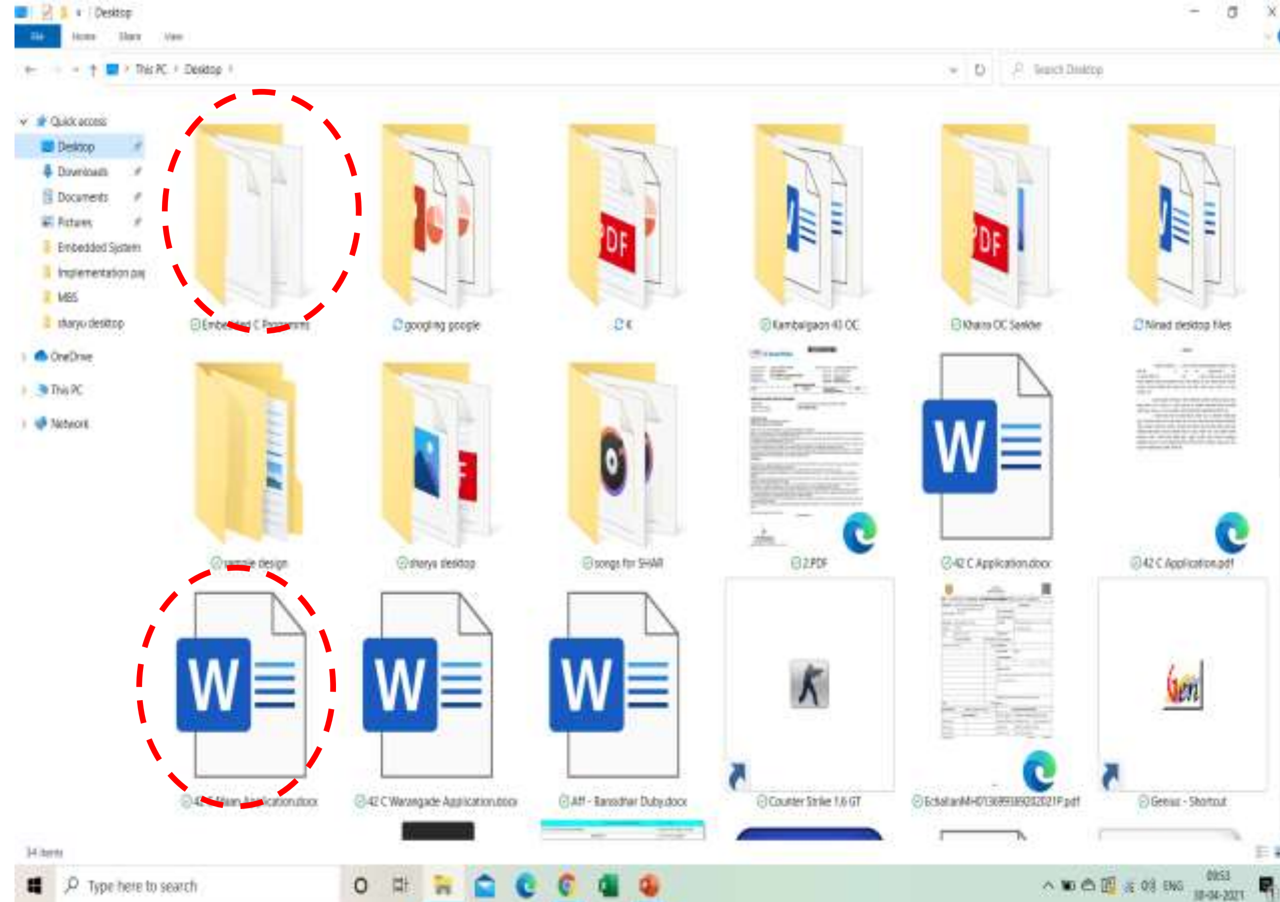
At some point they all are going to overwrite each other.

Solution :

Memory is divided into different parts which called as segments to automatically prevent the overwriting .

In 8086 memory segmentation concept was invented to avoid overwriting.

- **Programmers are responsible to create the segments at the time of initialization when start the program.**
- **Processor is responsible to manage the segments (avoid overwriting)**



Files and fodders are the example of segmentation.

There are two types of addresses :

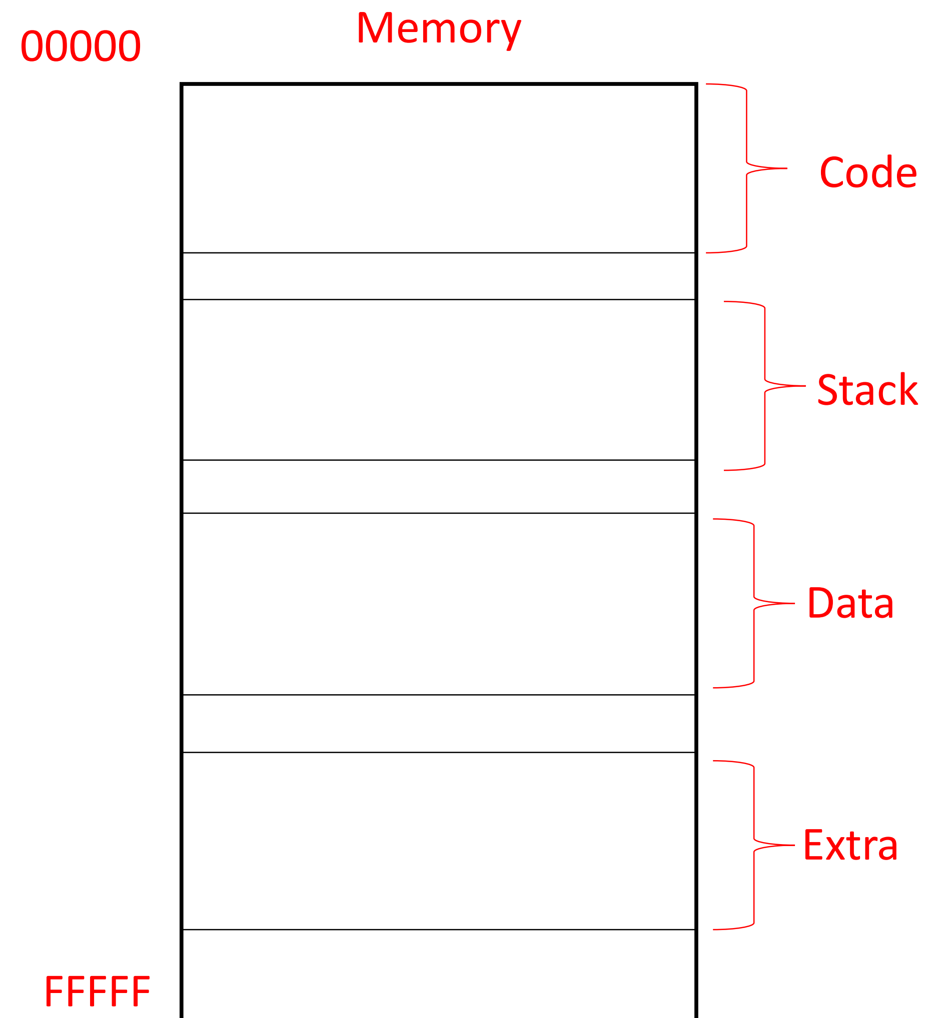
1. Virtual address
2. Physical address

- File/folder name is virtual address-known to programmer
- Actual location in memory is physical address – unknown to programmer.

- If address bus size is 8 bit then we have 2^8 = 256 unique address of memory locations.
- If address bus size is 16 bit then we have 2^{16} = 65535 = 64k memory locations.
- If we want 1MB memory hence 8086 having 2^{20} = 1MB i.e. 20 bit address bus.

PA = 12345

0001 0010 0011 0100 0101 = 20 bit



- Individual memory location size is 8 bit.
- Hence the address bus size is 20 bit then 2 and ½ memory locations are required to store one instruction in case of 8086.
- Which means wastage of ½ byte.
- If 16 bit address bus then no wastage of memory locations.

We want **20 bit address bus** to increase the memory size
We want **16 bit address** to provide equal memory locations

To achieve this we are going to use **16 bit virtual** address which will converted into **20 bit physical** address.

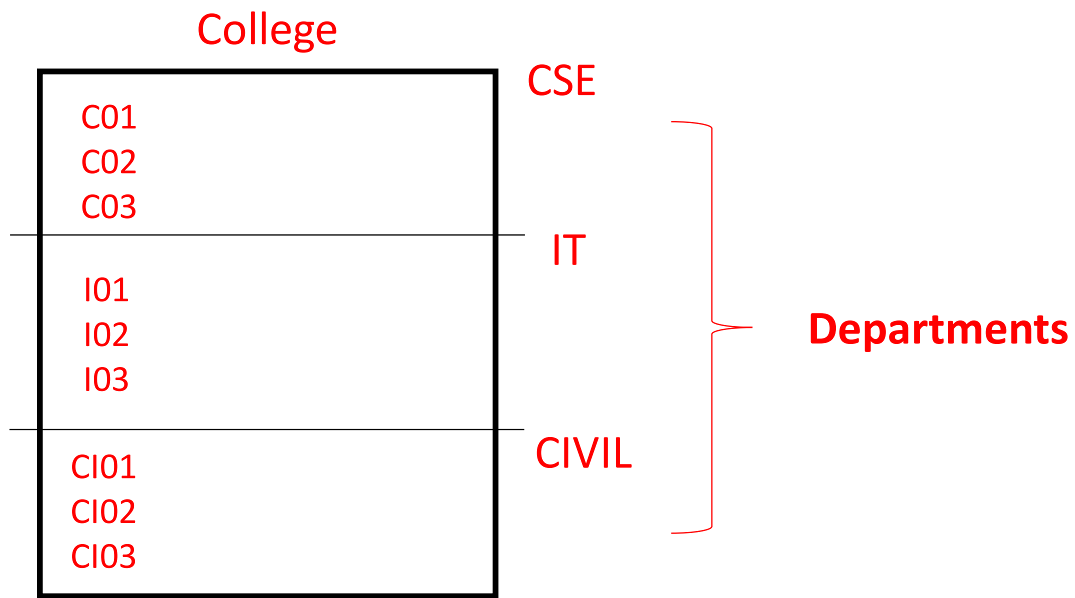
Virtual address = segment address + offset address

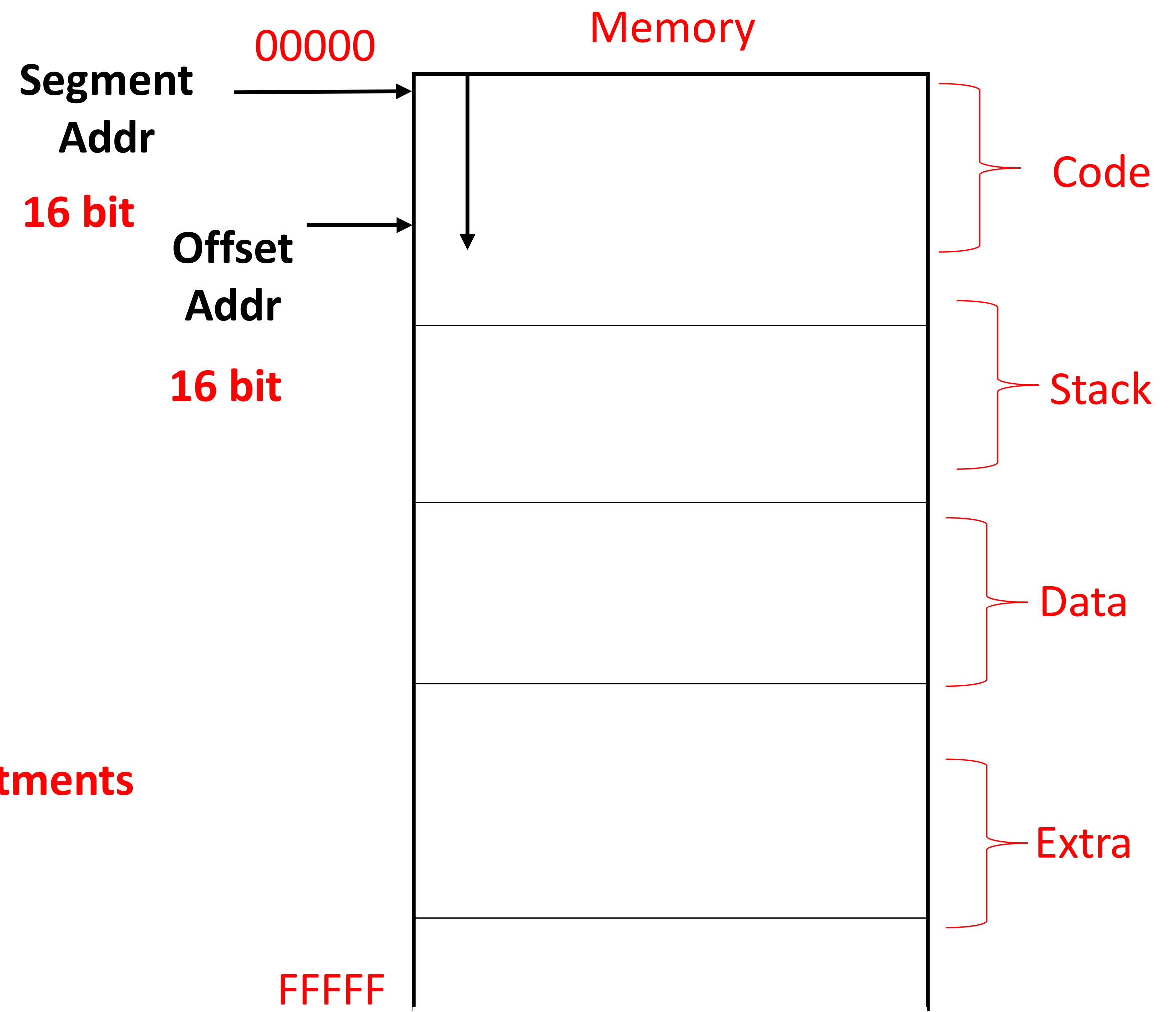
Both are 16 bit which means compatible.

Segment Addr gives the starting address of segment

Offset Addr gives the location which is present at that segment.

Example :





To call particular student from college

Segment Addr = Department Name
offset Addr = roll number

To call C01

Segment Addr = C
offset Addr = 01

To call C03

Segment Addr = C
offset Addr = 03

No need to change segment address when location is in same segment

There are three addresses :

1. Physical address — Not given by programmer

2. Segment address — Part of virtual address which is given only once

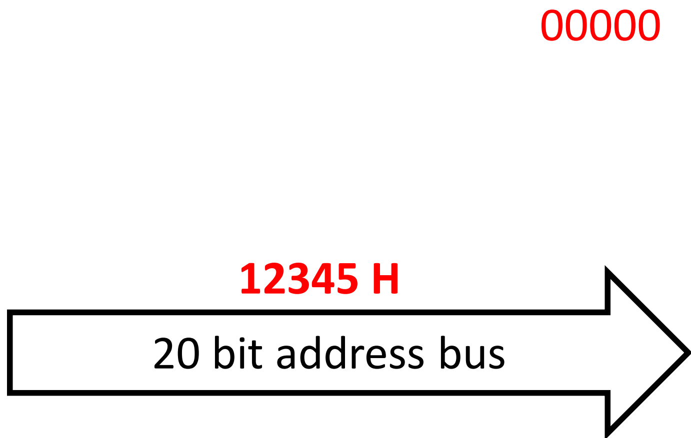
3. Offset address — Part of virtual address which is changed.

- Offset addresses are limited they are starting with 0000 to FFFF because they are 16 bits.

Maximum size of segment
= 2^{16} = 2^{6} X 2^{10} = 64 X 1KB = 64 KB

8086

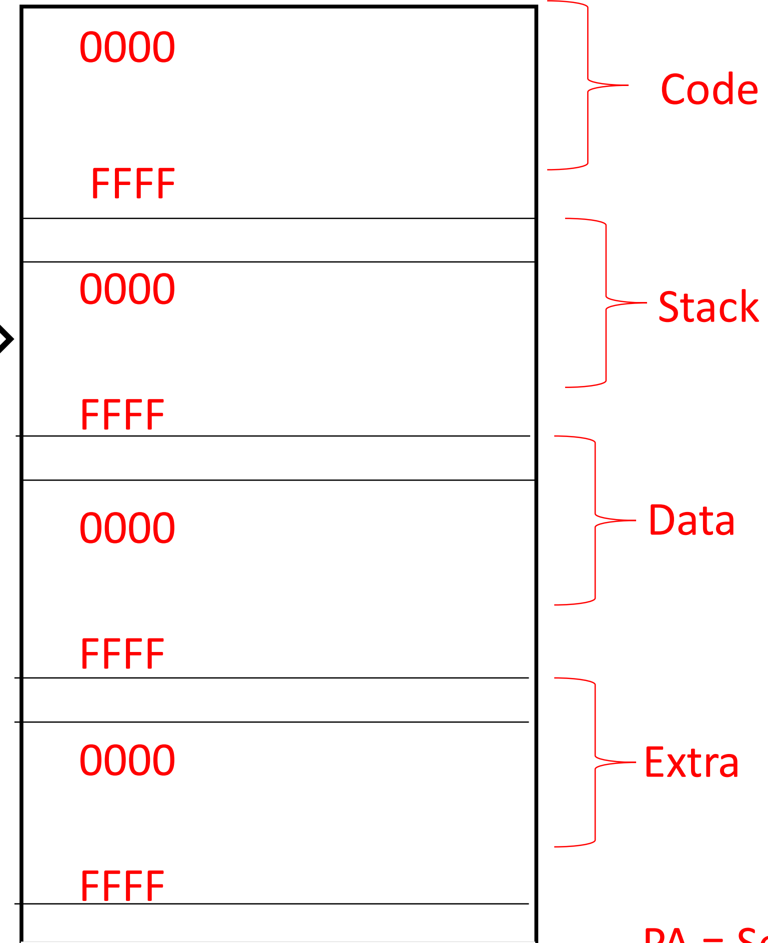
Seg reg	Off reg
CS 1000	IP 2345
SS	SP & BP
DS	SI
ES	DI



Processor gives the physical addr

Which is 20 bit

PA = Seg x 10H + offset



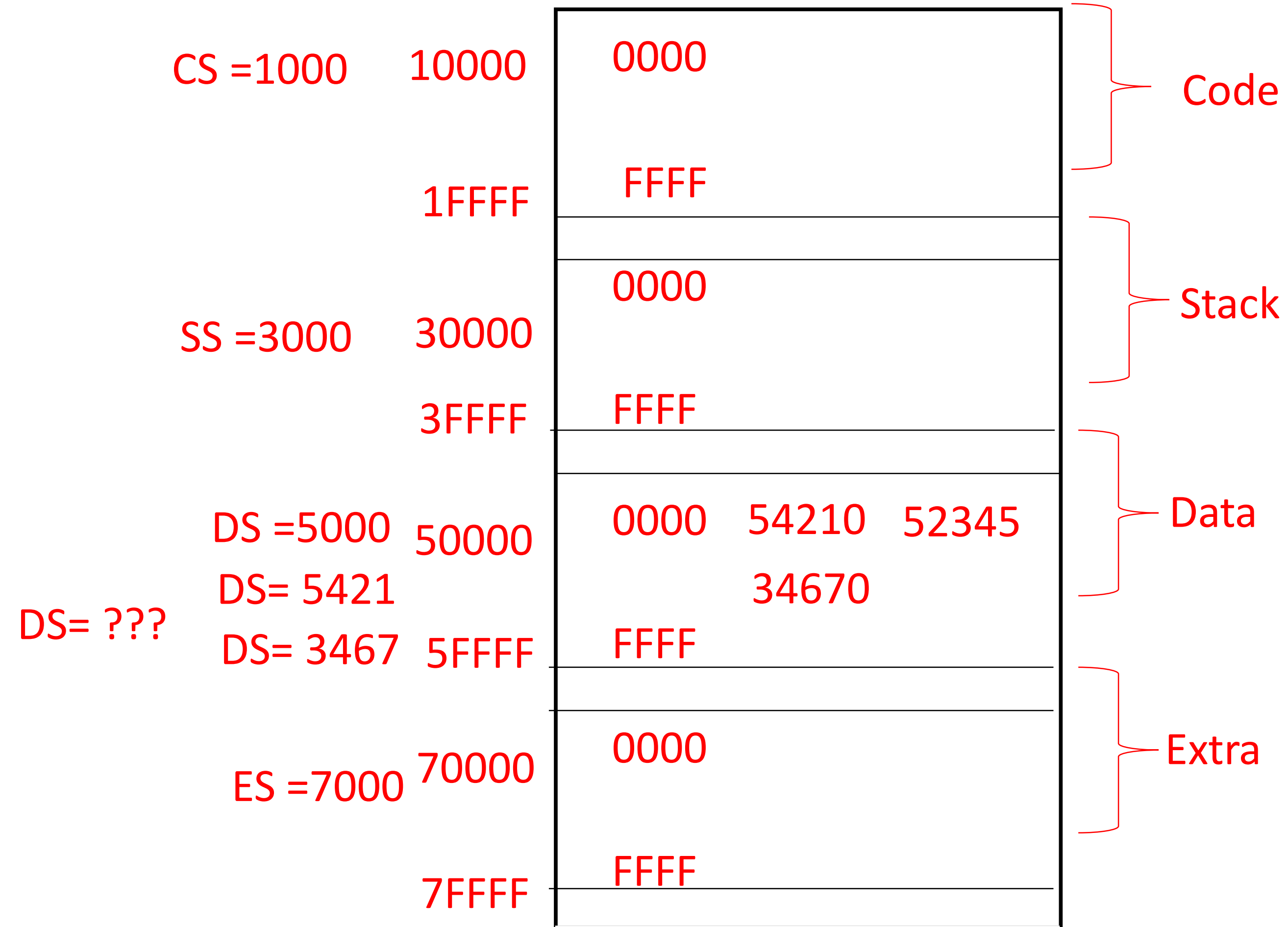
- 8086 having segment registers which are used to store the segment address and offset address of memory
- When processor wants to execute instruction from memory CS and IP is given
- Processor having 20 bit address bus

PA = Seg x 10H + offset
1000 x 10H + 2345
10000 + 2345
PA = 12345 H

If CS is 1000 then Code segment will actually start with 10000

8086

Seg reg	Off reg
CS 1000	IP 2345
SS 3000	SP & BP
DS	SI
ES	DI



If SS is 3000 then Code segment will actually start with 30000

If DS is 5000 then Code segment will actually start with 50000

If ES is 7000 then Code segment will actually start with 70000

Suppose we want to start DS from any random number

*****A segment can not begin at any location you want, it has to begin at location which is multiple of 10**

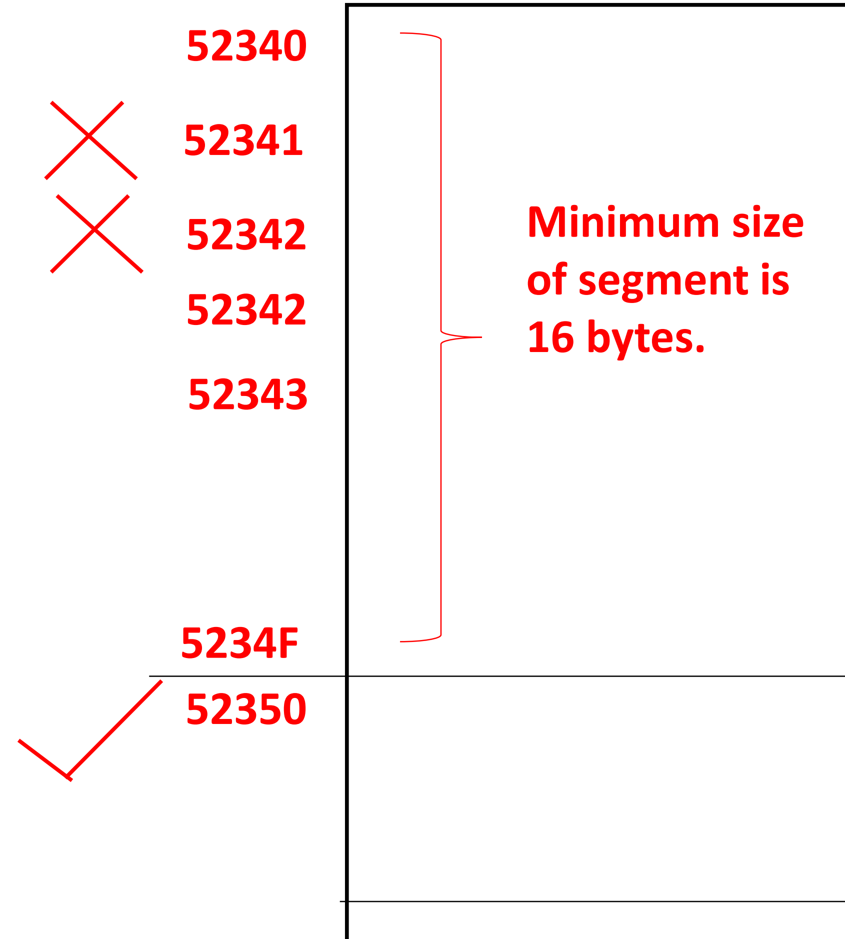
What is the minimum size of segment ????

8086

Seg reg	Off reg
CS	IP
SS	SP & BP
DS 5234	SI
ES	DI

- Because it is not multiple by 10

- Because it is multiple by 10



- Data segment starts with **52340** in memory
- We want only this locations after this we want next segment i.e. extra segment.

What is the next segment address????

16 = 10 H

Memory Segmentation of 8086 for exam

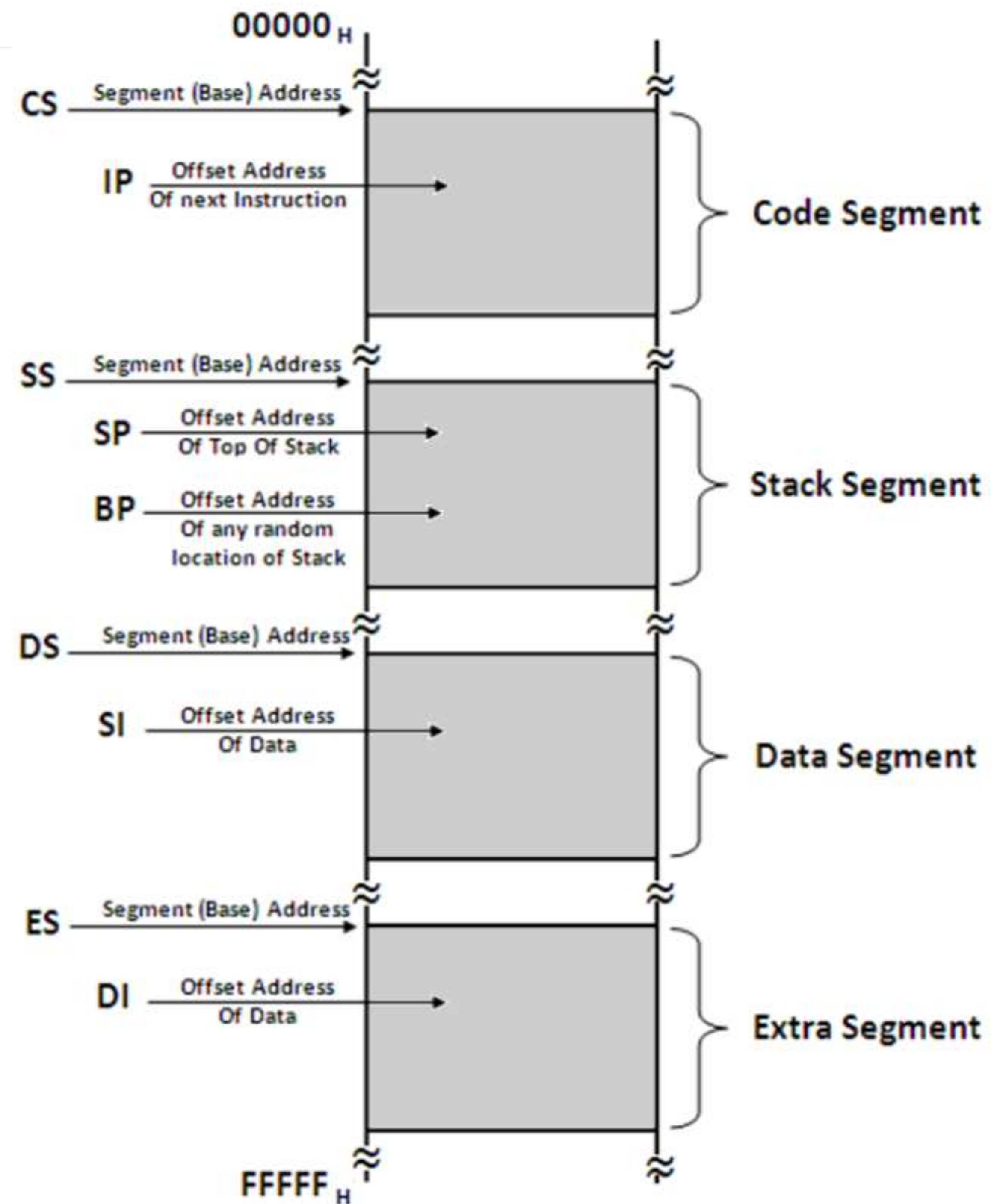
Memory Segmentation in 8086

Need For Segmentation / Concept of Segmentation

1) Segmentation means **dividing** the memory into **logically different parts called segments**.
2) 8086 has a **20-bit address bus**, hence it can access 2^{20} Bytes i.e. **1MB** memory.
3) But this also means that **Physical address** will now be **20 bit**.
4) It is **not possible** to work with a **20 bit address** as it is **not** a **byte compatible** number. (20 bits is two and a half bytes).
5) To avoid working with this incompatible number, we **create a virtual model** of the memory.
6) Here the memory is **divided into 4 segments**: Code, Stack Data and Extra.
7) The **max size** of a segment is **64KB** and the **minimum size** is **16 bytes**.
8) Now programmer can access each location with a **VIRTUAL ADDRESS**.
9) The Virtual Address is a **combination** of **Segment Address and Offset Address**.
10) **Segment Address indicates where the segment is located in the memory (base address)**
11) **Offset Address gives the offset of the target location within the segment.**
12) Since both, Segment Address and Offset Address are **16 bits each**, they both are **compatible numbers** and can be easily used by the programmer.
13) Moreover, **Segment Address is given only in the beginning** of the program, to initialize the segment. Thereafter, we **only give offset address**.
14) **Hence we can access 1 MB memory using only a 16 bit offset address for most part of the program. This is the advantage of segmentation.**
15) Moreover, dividing Code, stack and Data into different segments, makes the memory **more organized and prevents accidental overwrites** between them.
16) The **Maximum Size** of a segment is **64KB because offset addresses are of 16 bits.** 2^{16} = 64KB.
17) As max size of a segment is 64KB, programmer can create **multiple Code/Stack/Data segments** till the entire 1 MB is utilized, but **only one of each** type will be **currently active**.
18) The physical address is calculated by the microprocessor, using the formula:

    Physical Address = Segment Address x 10H + Offset Address

19) Ex: if Segment Address = 1234H and Offset Address is 0005H then
    Physical Address = 1234H x 10H + 0005H = 12345H
20) This formula automatically ensures that the **minimum size of a segment is 10H bytes** (10H = 16 Bytes).



Code Segment

This segment is used to hold the **program** to be executed.
Instruction are fetched from the Code Segment.
CS register holds the 16-bit **base** address for this segment.
IP register (Instruction Pointer) holds the 16-bit **offset** address.

Data Segment

This segment is used to hold **general data**.
This segment also holds the **source** operands during **string** operations.
DS register holds the 16-bit **base** address for this segment.
BX register is used to hold the 16-bit **offset** for this segment.
SI register (Source Index) holds the 16-bit **offset** address during String Operations.

Stack Segment

This segment holds the **Stack** memory, which operates in LIFO manner.
SS holds its **Base** address.
SP (Stack Pointer) holds the 16-bit **offset** address of the **Top** of the Stack.
BP (Base Pointer) holds the 16-bit **offset** address during **Random Access**.

Extra Segment

This segment is used to hold **general data**
Additionally, this segment is used as the **destination** during **String Operations**.
ES holds the **Base** Address.
DI holds the **offset** address during string operations.

Advantages of Segmentation:

1) It permits the programmer to access 1MB **using only 16-bit address**.
2) Its **divides** the **memory logically** to store Instructions, Data and Stack separately.

Disadvantage of Segmentation:

1) Although the total memory is 16*64 KB, **at a time only 4*64 KB memory can be accessed**.