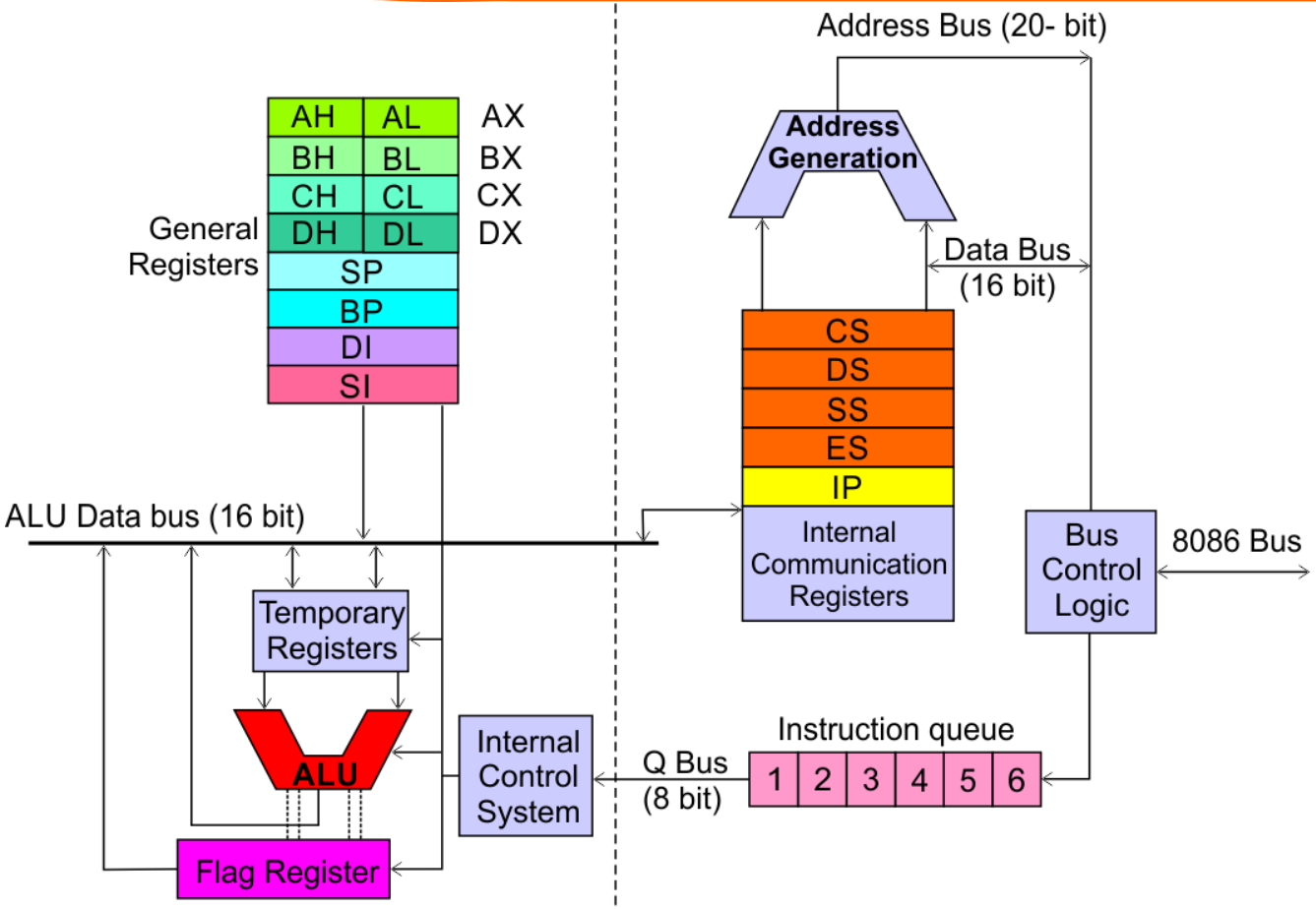
Course Coordinator : Ms.Simra Najm
Designation : Assistant Professor
Email: simranajm@neduet.edu.pk

Lecture:2
Computer Organization Microprocessor

Features

- ***It is a 16-bit µp.***
- ***8086 has a 20 bit address bus can access up to 2^{20} memory locations (1 MB).***
- ***It can support up to 64K I/O ports.***
- ***It provides 14, 16 -bit registers.***
- ***Word size is 16 bits.***

Architecture



Execution Unit (EU)

EU executes instructions that have already been fetched by the BIU.

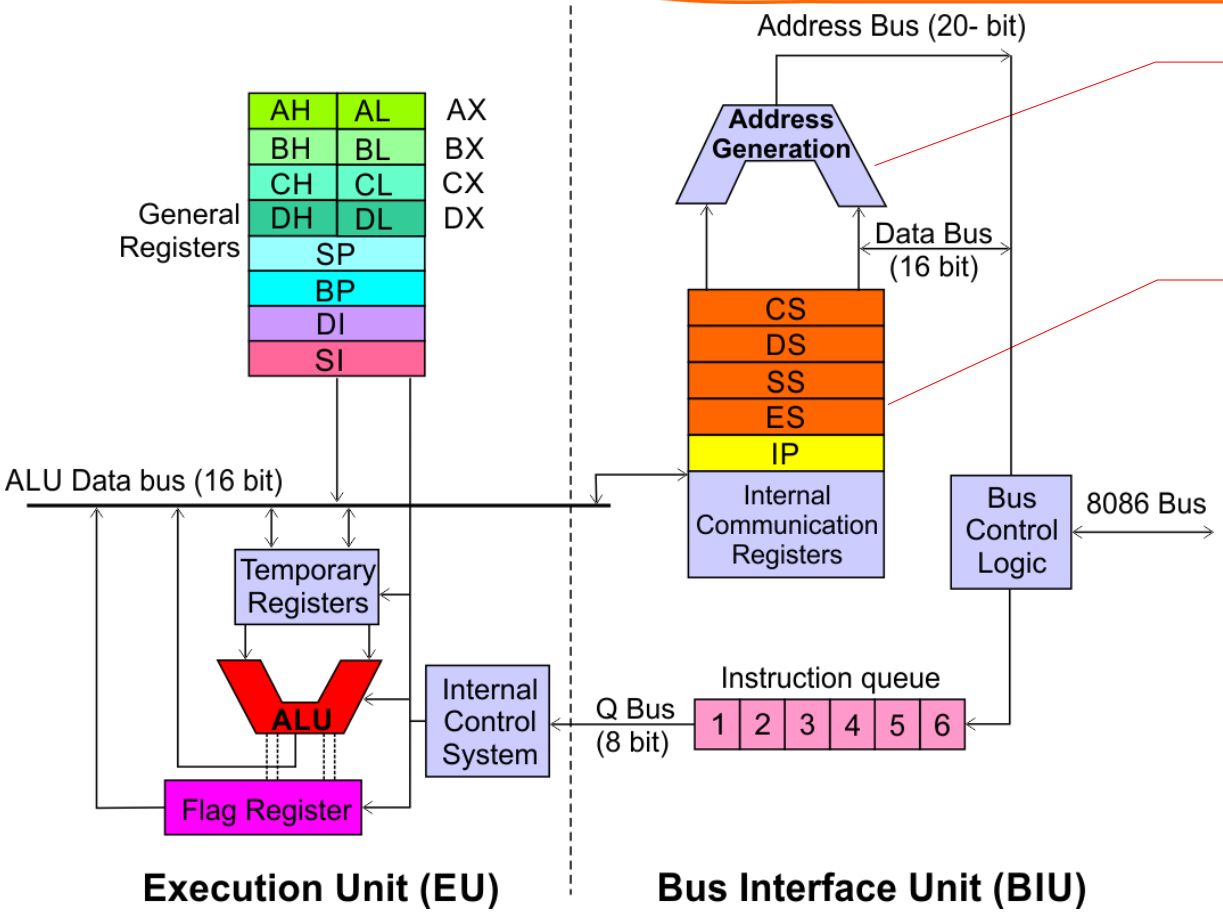
BIU and EU functions separately.

Bus Interface Unit (BIU)

BIU fetches instructions, reads data from memory and I/O ports, writes data to memory and I/ O ports.

Architecture

Bus Interface Unit (BIU)



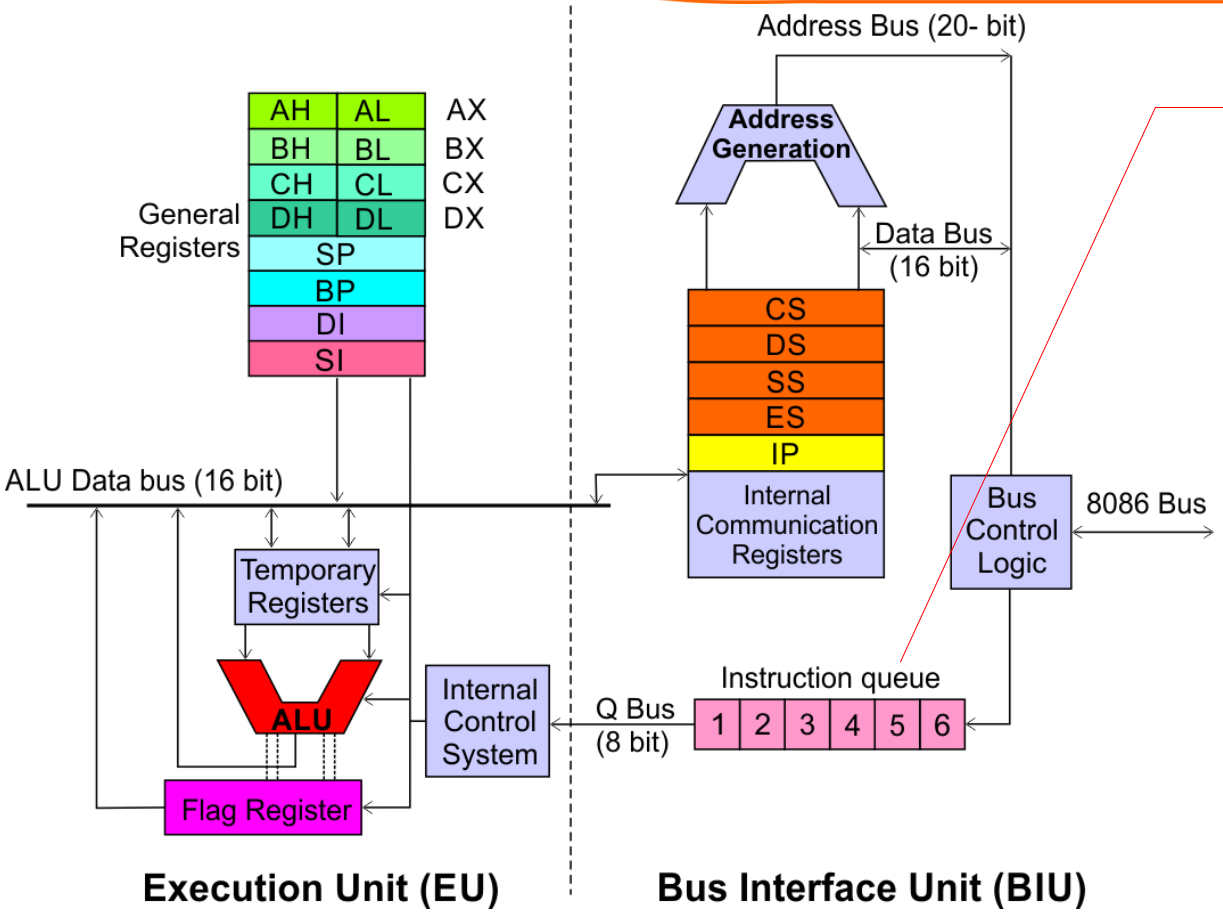
Dedicated Adder to generate 20 bit address

Four 16-bit segment registers

Code Segment (CS)
Data Segment (DS)
Stack Segment (SS)
Extra Segment (ES)

Architecture

Bus Interface Unit (BIU)



Instruction queue

- **A group of First-In-First-Out (FIFO) in which up to 6 bytes of instruction code are pre fetched from the memory ahead of time.**
- **This is done in order to speed up the execution by overlapping instruction fetch with execution.**
- **This mechanism is known as pipelining.**

Architecture

Execution Unit (EU)

EU decodes and executes instructions.

A decoder in the EU control system translates instructions.

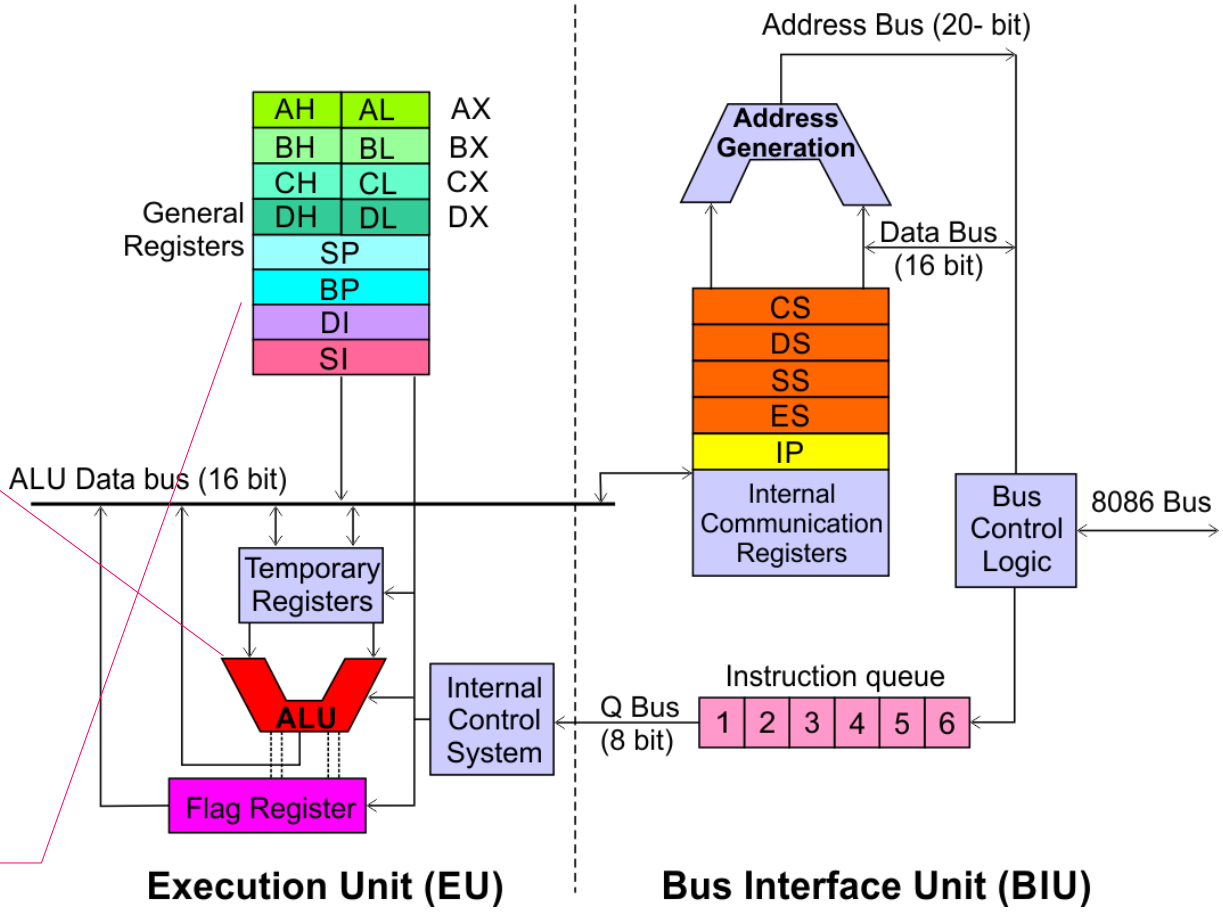
16-bit ALU for performing arithmetic and logic operation

Four general purpose registers(AX, BX, CX, DX);

Pointer registers (Stack Pointer, Base Pointer);

and

Index registers (Source Index, Destination Index) each of 16-bits



Some of the 16 bit registers can be used as two 8 bit registers as :

AX can be used as AH and AL
BX can be used as BH and BL
CX can be used as CH and CL
DX can be used as DH and DL

Internal architecture of 8086

- **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 instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands. The instruction bytes are transferred to the instruction queue.**
- **EU executes instructions from the instruction system byte queue.**

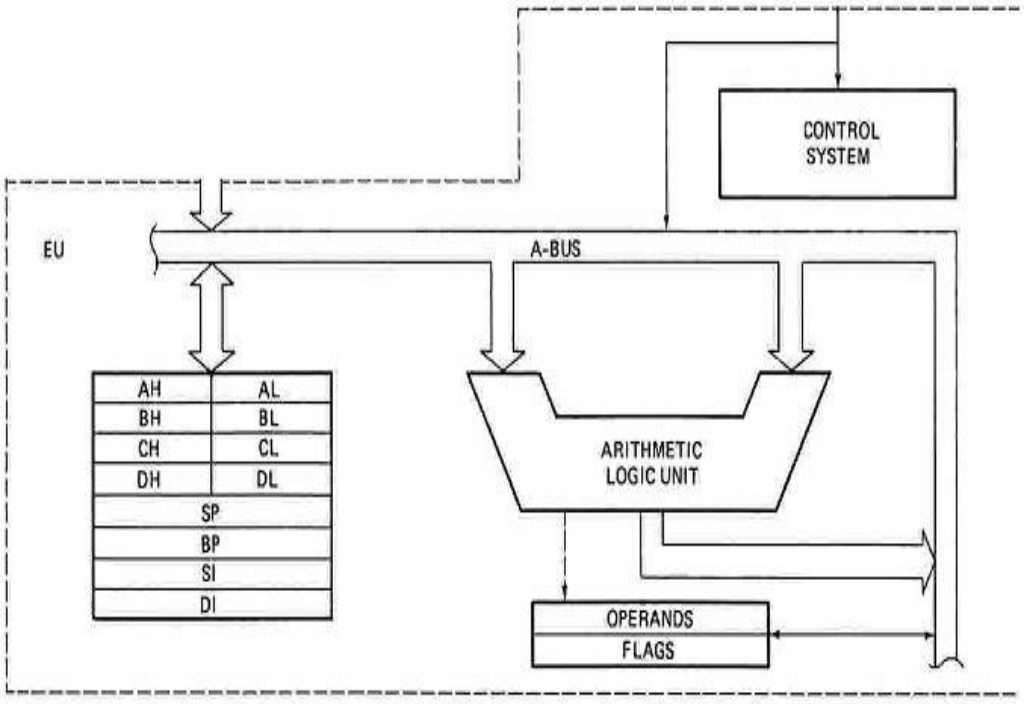
- **Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining. This results in efficient use of the system bus and system performance.**
- **BIU contains Instruction queue, Segment registers, Instruction pointer, Address adder.**
- **EU contains Control circuitry, Instruction decoder, ALU, Pointer and Index register, Flag register.**

EXECUTION UNIT

- **Decodes instructions fetched by the BIU**
- **Generate control signals,**
- **Executes instructions.**

The main parts are:

- **Control Circuitry**
- **Instruction decoder**
- **ALU**



THE QUEUE (Q)

- The BIU uses a mechanism known as an **instruction stream queue** to implement a ***pipeline architecture.***
- This queue permits pre-fetch of up to **6 bytes** of instruction code. Whenever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by pre-fetching the next sequential instruction.

- These pre-fetching instructions are held in its **FIFO queue**. With its **16** bit data bus, the BIU fetches two instruction bytes in a single memory cycle.
- After a byte is loaded at the input end of the queue, it automatically shifts up through the **FIFO** to the empty location nearest the output.

- The **EU accesses the queue from the output end**. It reads one instruction byte after the other from the output of the queue.

- The intervals of no bus activity, which may occur between bus cycles are known as ***Idle state*.**

Segment: Offset Notation

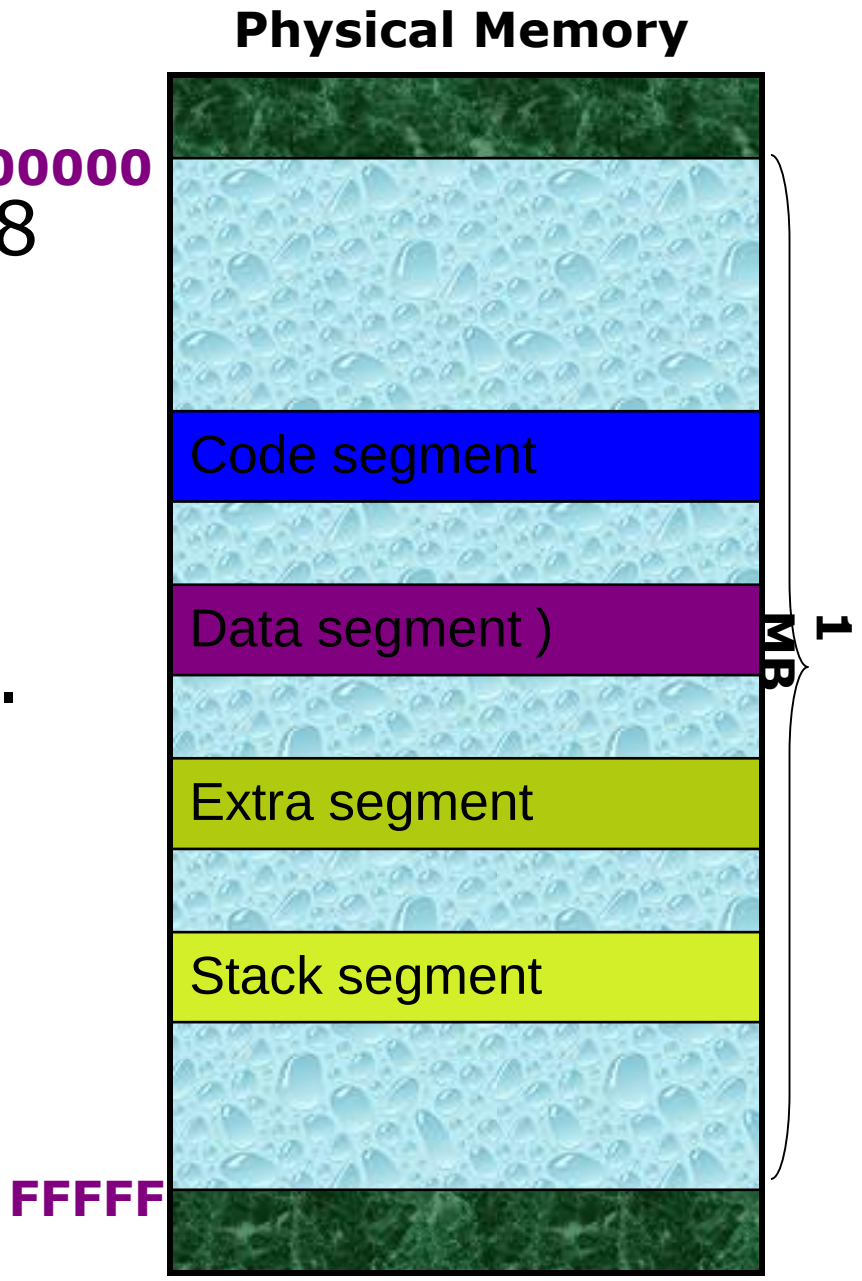
- A simple scheme would be to order the bytes in a serial fashion and number them from 0 (or 1) to the end of memory
- The scheme used in the 8086 is called segmentation
- Every address has two parts, a SEGMENT and an OFFSET (Segmnet:Offset)
- The segment indicates the starting of a 64 kilobyte portion of memory, in multiples of 16
- The offset indicates the position within the 64k portion
- Physical address = (segment * 10h) + offset

Segment registers

- In 8086/88 the processors have 4 segments registers
- **Code Segment register (CS),**
- **Data Segment register (DS),**
- **Extra Segment register (ES) and**
- **Stack Segment (SS) register.**
- All are 16 bit registers.
- Each of the Segment registers store the upper 16 bit address of the starting address of the corresponding segments.

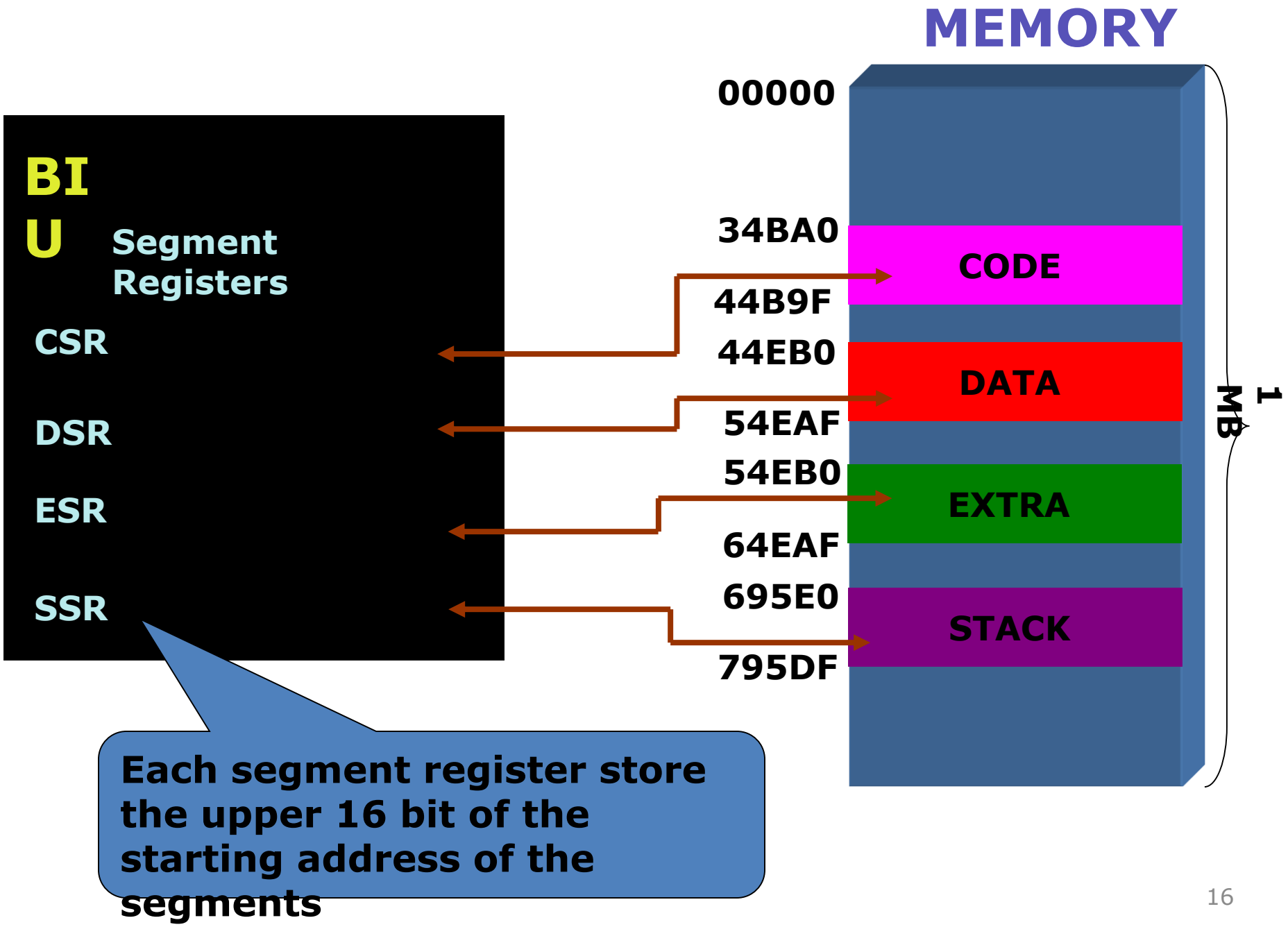
Segmented Memory

- The memory in an 8086/88 based system is organized as segmented memory.
- The CPU 8086 is able to address 1Mbyte of memory.
- The Complete physically available memory may be divided into a number of logical segments.



- A segment is an area that begins at any location which is divisible by 16.
- A segment may be located any where in the memory
- Each of these segments can be used for a specific function.
  - Code segment is used for storing the instructions.
  - The stack segment is used as a stack and it is used to store the return addresses.
  - The data and extra segments are used for storing data byte address.

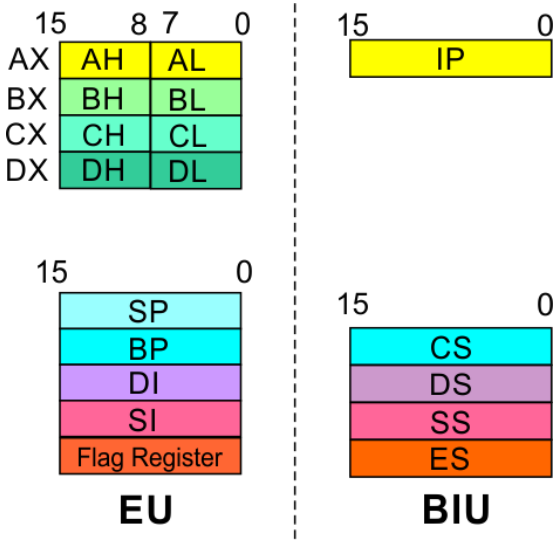
*** In the assembly language programming, more than one data/ code/ stack segments can be defined. But only one segment of each type can be accessed at any time.**


MEMORY
BIU
Segment Registers
CSR
DSR
ESR
SSR
00000
34BA0
CODE
44B9F
44EB0
DATA
54EAF
54EB0
EXTRA
64EAF
695E0
STACK
795DF
1 MB
Each segment register store the upper 16 bit of the starting address of the segments

Segment Registers

Code Segment Register

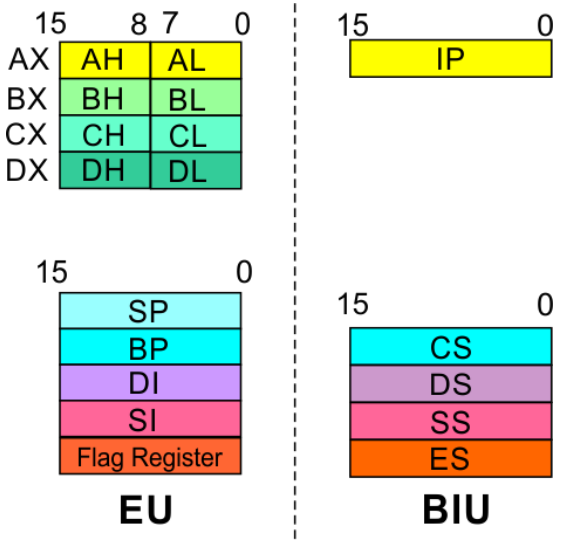
- 16-bit
- CS contains the base or start of the current code segment; IP contains the distance or offset from this address to the next instruction byte to be fetched.
- BIU computes the 20-bit physical address by logically shifting the contents of CS 4-bits to the left and then adding the 16-bit contents of IP.
- That is, all instructions of a program are relative to the contents of the CS register multiplied by 16 and then offset is added provided by the IP.



Segment Registers

Data Segment Register

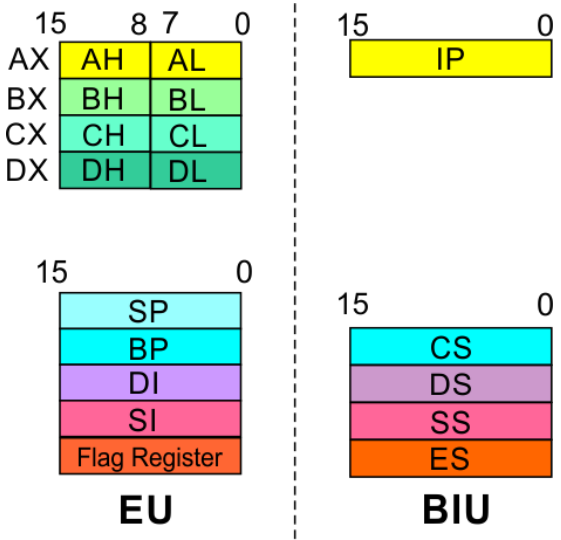
- 16-bit
- Points to the current data segment; operands for most instructions are fetched from this segment.
- The 16-bit contents of the Source Index (SI) or Destination Index (DI) or a 16-bit displacement are used as offset for computing the 20-bit physical address.



Segment Registers

Stack Segment Register

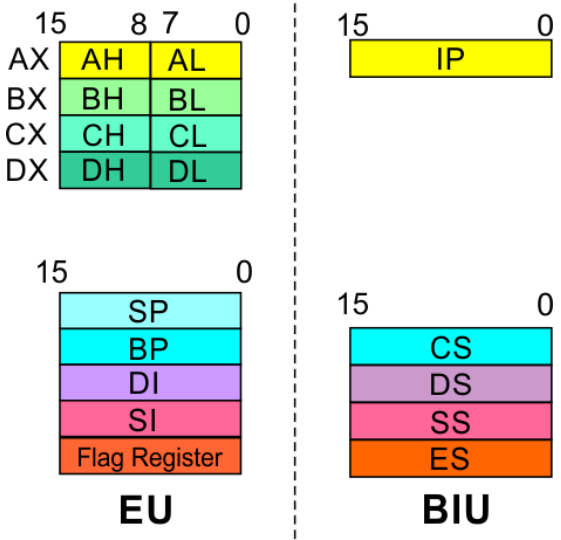
- 16-bit
- Points to the current stack.
- The 20-bit physical stack address is calculated from the Stack Segment (SS) and the Stack Pointer (SP) for stack instructions such as PUSH and POP.
- In based addressing mode, the 20-bit physical stack address is calculated from the Stack segment (SS) and the Base Pointer (BP).



Segment Registers

Extra Segment Register

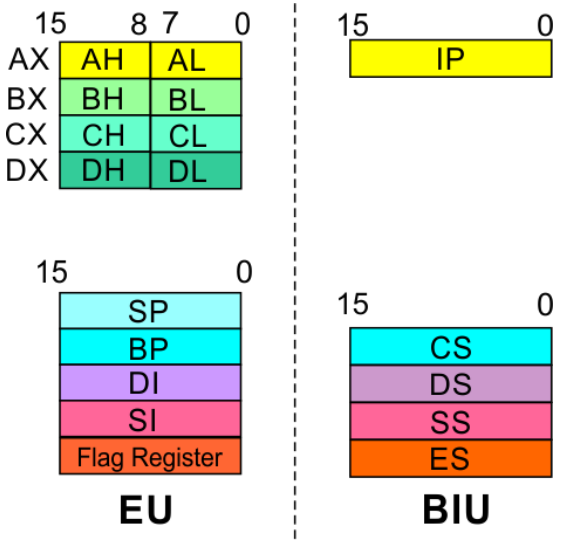
- 16-bit
- Points to the extra segment in which data (in excess of 64K pointed to by the DS) is stored.
- String instructions use the ES and DI to determine the 20-bit physical address for the destination.



Segment Registers

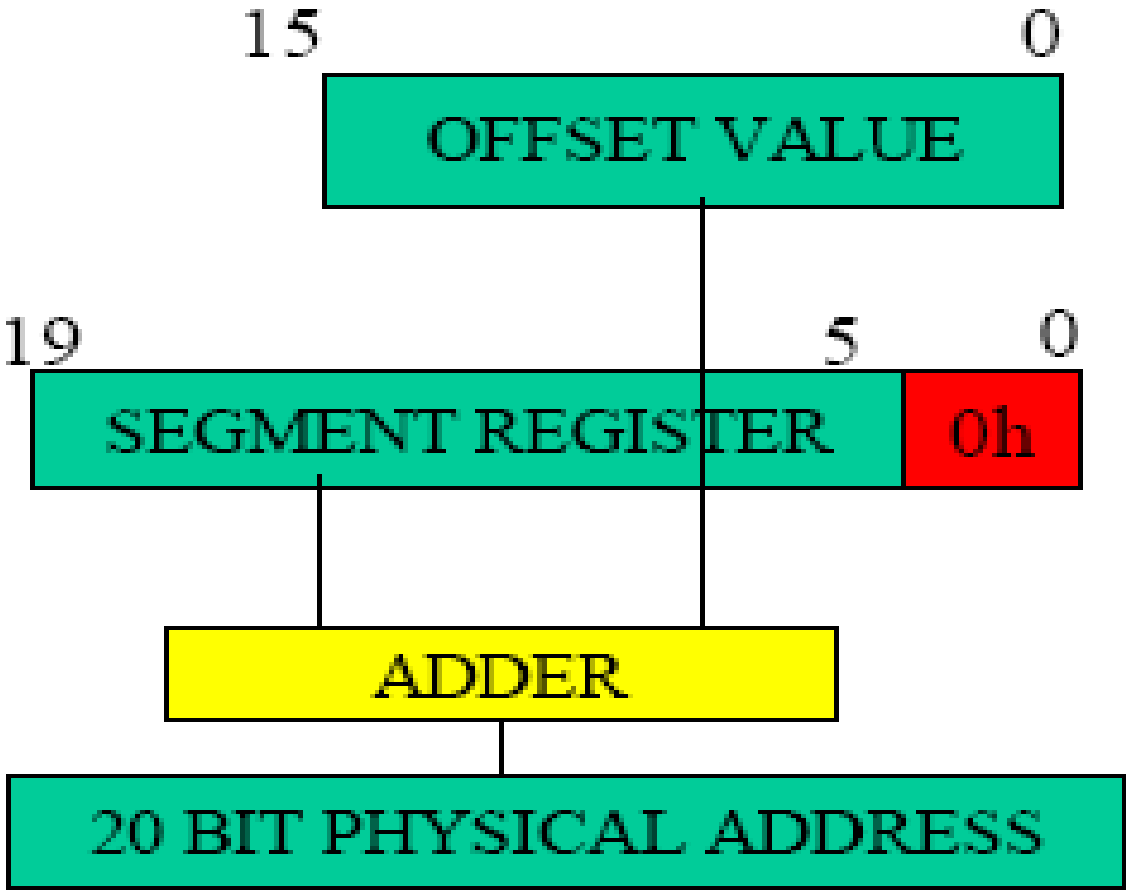
Instruction Pointer

- 16-bit
- Always points to the next instruction to be executed within the currently executing code segment.
- So, this register contains the 16-bit offset address pointing to the next instruction code within the 64Kb of the code segment area.
- Its content is automatically incremented as the execution of the next instruction takes place.

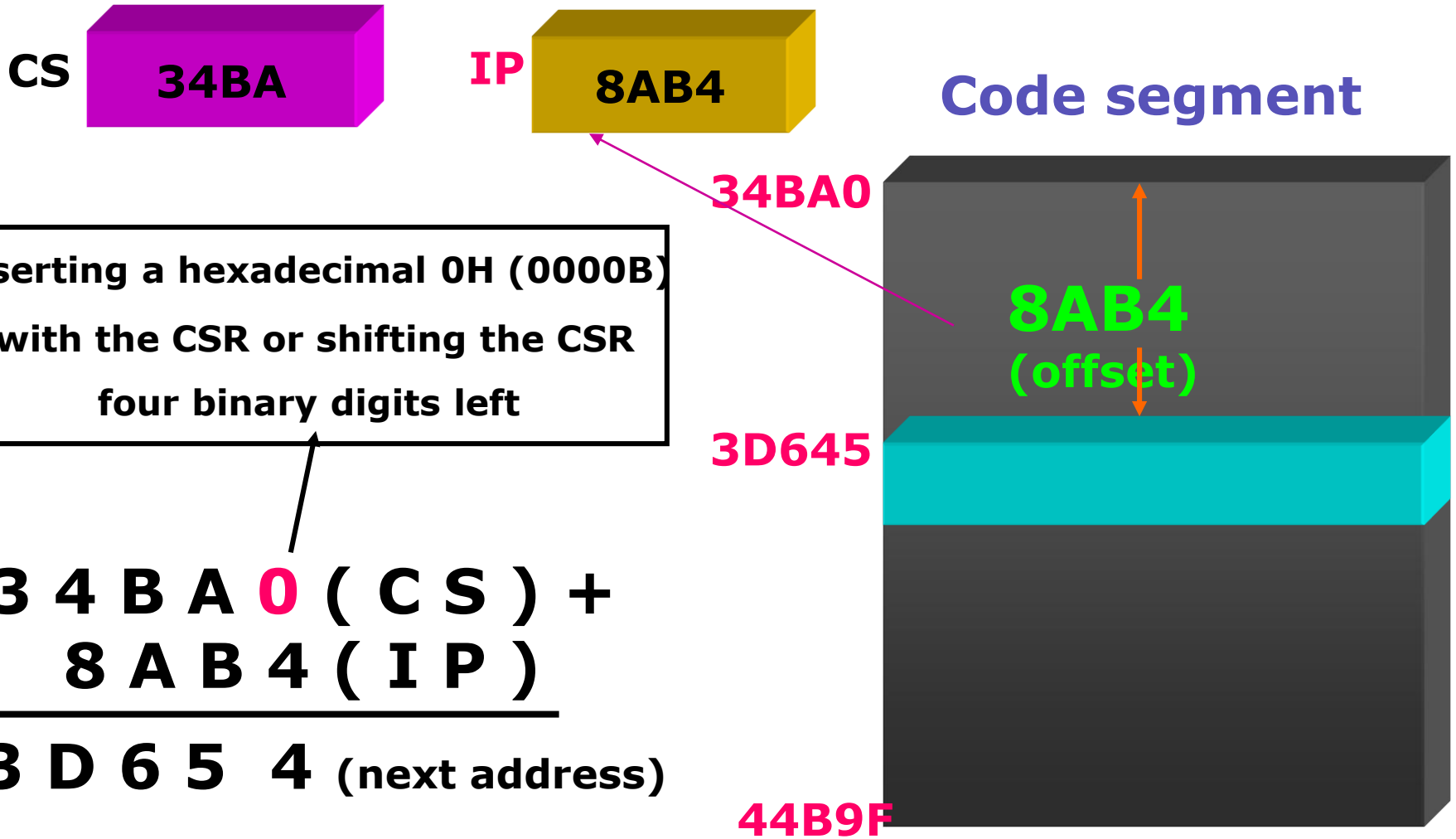


Instruction pointer & summing block

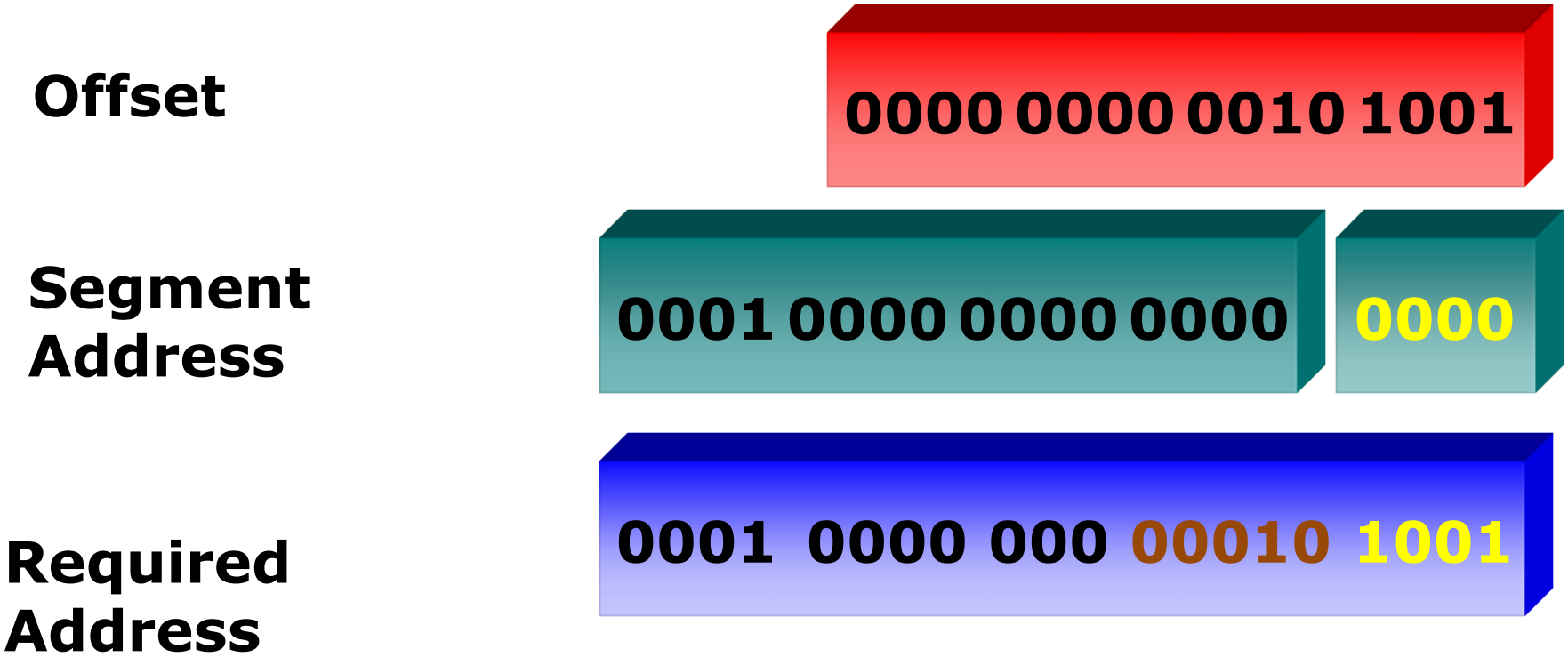
- **The instruction pointer register contains a 16-bit offset address of instruction that is to be executed next.**
- **The IP always references the Code segment register (CS).**
- **The value contained in the instruction pointer is called as an offset because this value must be added to the base address of the code segment, which is available in the CS register to find the 20-bit physical address.**
- **The value of the instruction pointer is incremented after executing every instruction.**
- **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.**


15
0
OFFSET VALUE
19
5
0
SEGMENT REGISTER
0h
ADDER
20 BIT PHYSICAL ADDRESS

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



- **Example For Address Calculation (segment: offset)**
- If the data segment starts at location 1000h and a data reference contains the address 29h where is the actual data?



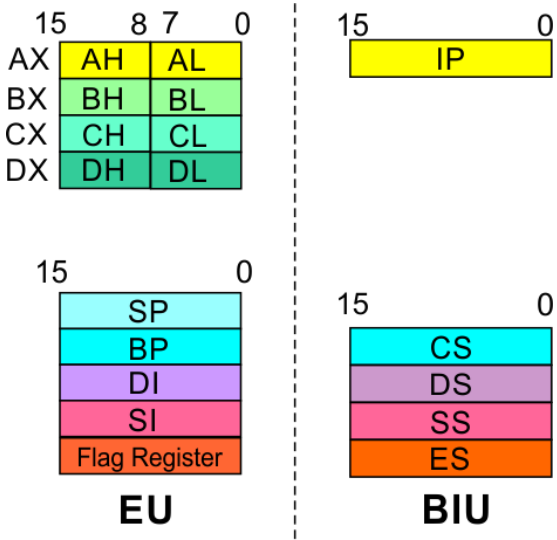
Segment and Address register combination

- **CS:IP**
- **SS:SP SS:BP**
- **DS:BX DS:SI**
- **DS:DI (for other than string operations)**
- **ES:DI (for string operations)**

EU Registers

Accumulator Register (AX)

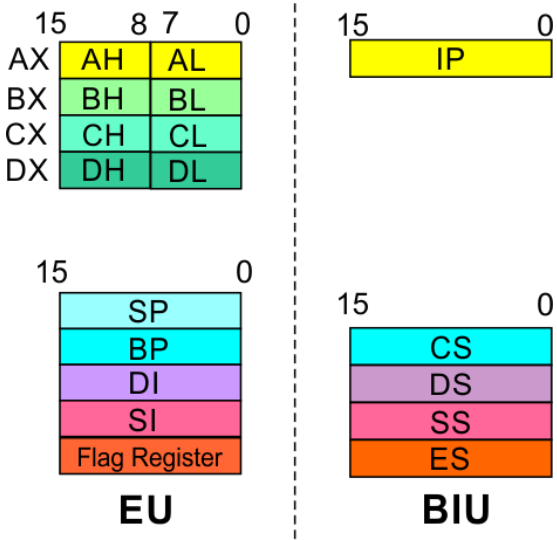
- Consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX.
- AL in this case contains the low order byte of the word, and AH contains the high-order byte.
- The I/O instructions use the AX or AL for inputting / outputting 16 or 8 bit data to or from an I/O port.
- Multiplication and Division instructions also use the AX or AL.



EU Registers

Base Register (BX)

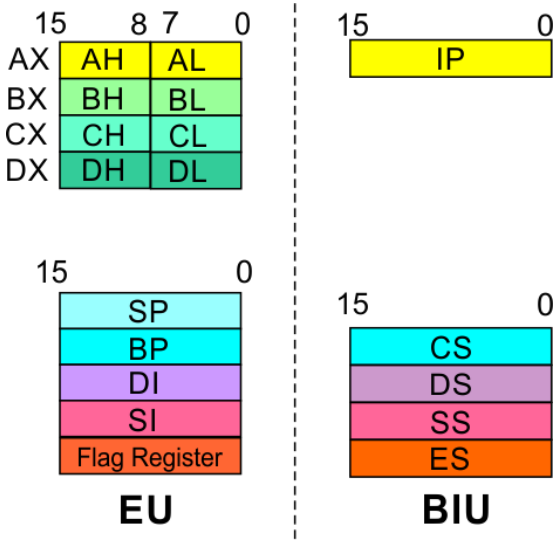
- Consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX.
- BL in this case contains the low-order byte of the word, and BH contains the high-order byte.
- This is the only general purpose register whose contents can be used for addressing the 8086 memory.
- All memory references utilizing this register content for addressing use DS as the default segment register.



EU Registers

Counter Register (CX)

- Consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX.
- When combined, CL register contains the low order byte of the word, and CH contains the high-order byte.
- Instructions such as SHIFT, ROTATE and LOOP use the contents of CX as a counter.



Example:

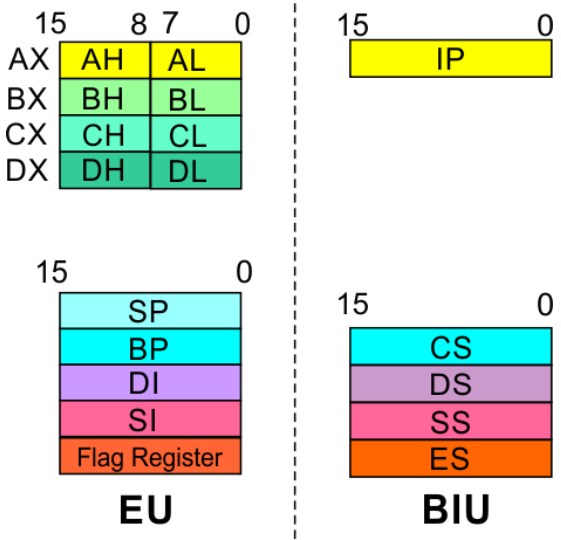
The instruction LOOP START automatically decrements CX by 1 without affecting flags and will check if [CX] = 0.

If it is zero, 8086 executes the next instruction; otherwise the 8086 branches to the label START.

EU Registers

Data Register (DX)

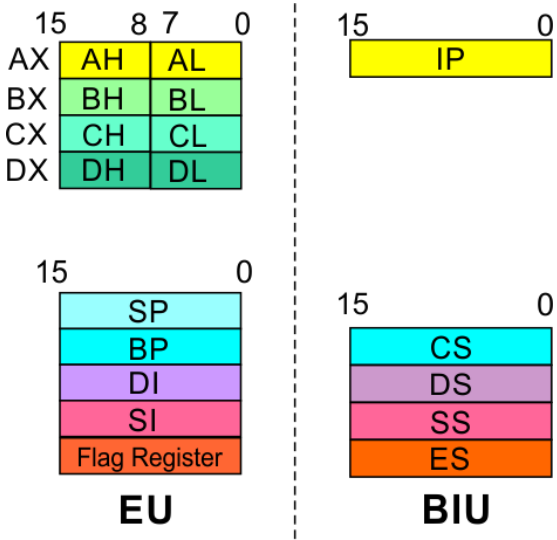
- Consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX.
- When combined, DL register contains the low order byte of the word, and DH contains the high-order byte.
- Used to hold the high 16-bit result (data) in 16 X 16 multiplication or the high 16-bit dividend (data) before a 32 ÷ 16 division and the 16-bit reminder after division.



EU Registers

Stack Pointer (SP) and Base Pointer (BP)

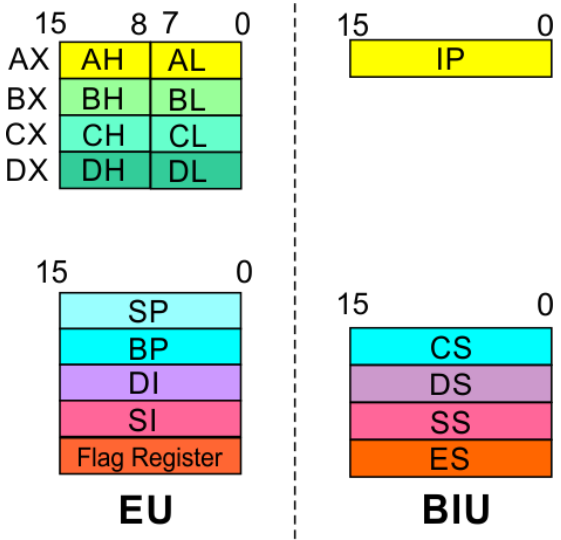
- **SP and BP are used to access data in the stack segment.**
- **SP is used as an offset from the current SS during execution of instructions that involve the stack segment in the external memory.**
- **SP contents are automatically updated (incremented/ decremented) due to execution of a POP or PUSH instruction.**
- **BP contains an offset address in the current SS, which is used by instructions utilizing the based addressing mode.**



EU Registers

Source Index (SI) and Destination Index (DI)

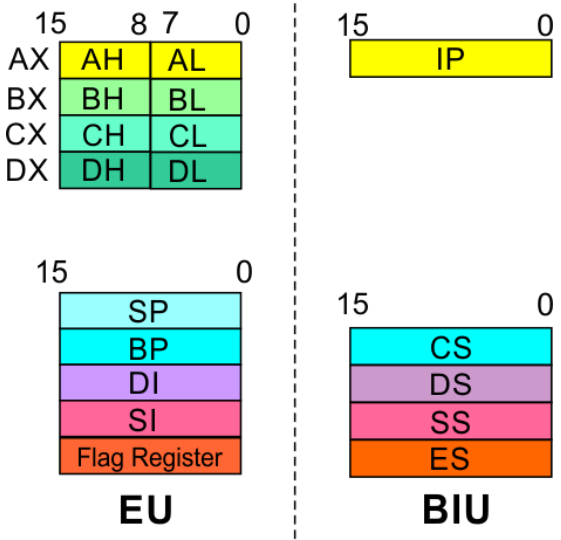
- **Used in indexed addressing.**
- **Instructions that process data strings use the SI and DI registers together with DS and ES respectively in order to distinguish between the source and destination addresses.**



EU Registers

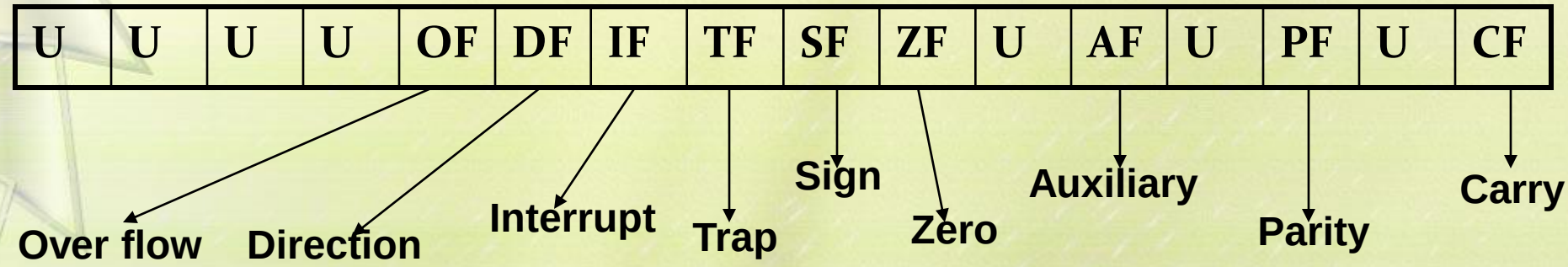
Source Index (SI) and Destination Index (DI)

- **Used in indexed addressing.**
- **Instructions that process data strings use the SI and DI registers together with DS and ES respectively in order to distinguish between the source and destination addresses.**



EXECUTION UNIT – 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 .
- 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 - Unused

Architecture

Execution Unit (EU)

Flag Register

Auxiliary Carry Flag

This is set, if there is a carry from the lowest nibble, i.e, bit three during addition, or borrow for the lowest nibble, i.e, bit three, during subtraction.

Carry Flag

This flag is set, when there is a carry out of MSB in case of addition or a borrow in case of subtraction.

Sign Flag

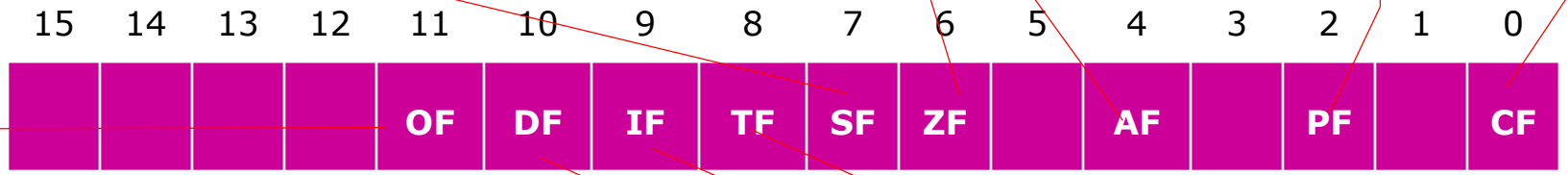
This flag is set, when the result of any computation is negative

Zero Flag

This flag is set, if the result of the computation or comparison performed by an instruction is zero

Parity Flag

This flag is set to 1, if the lower byte of the result contains even number of 1's ; for odd number of 1's set to zero.



Over flow Flag

This flag is set, if an overflow occurs, i.e, if the result of a signed operation is large enough to accommodate in a destination register. The result is of more than 7-bits in size in case of 8-bit signed operation and more than 15-bits in size in case of 16-bit sign operations, then the overflow will be set.

Tarp Flag

If this flag is set, the processor enters the single step execution mode by generating internal interrupts after the execution of each instruction

Direction Flag

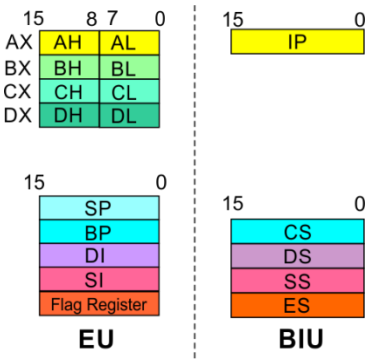
This is used by string manipulation instructions. If this flag bit is '0', the string is processed beginning from the lowest address to the highest address, i.e., auto incrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e., auto incrementing mode.

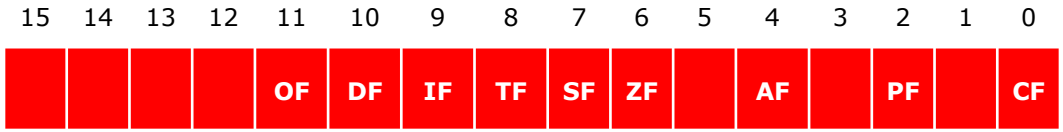
Interrupt Flag

Causes the 8086 to recognize external mask interrupts; clearing IF disables these interrupts.

Architecture

8086 registers categorized into 4 groups





Sl.No.	Type	Register width	Name of register
1	General purpose register	16 bit	AX, BX, CX, DX
		8 bit	AL, AH, BL, BH, CL, CH, DL, DH
2	Pointer register	16 bit	SP, BP
3	Index register	16 bit	SI, DI
4	Instruction Pointer	16 bit	IP
5	Segment register	16 bit	CS, DS, SS, ES
6	Flag (PSW)	16 bit	Flag register

Architecture

Registers and Special Functions

Register	Name of the Register	Special Function
AX	16-bit Accumulator	Stores the 16-bit results of arithmetic and logic operations
AL	8-bit Accumulator	Stores the 8-bit results of arithmetic and logic operations
BX	Base register	Used to hold base value in base addressing mode to access memory data
CX	Count Register	Used to hold the count value in SHIFT, ROTATE and LOOP instructions
DX	Data Register	Used to hold data for multiplication and division operations
SP	Stack Pointer	Used to hold the offset address of top stack memory
BP	Base Pointer	Used to hold the base value in base addressing using SS register to access data from stack memory
SI	Source Index	Used to hold index value of source operand (data) for string instructions
DI	Data Index	Used to hold the index value of destination operand (data) for string operations