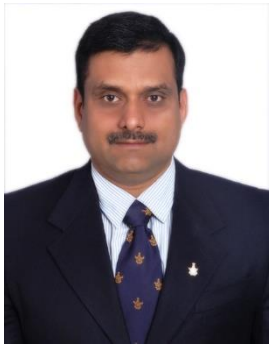
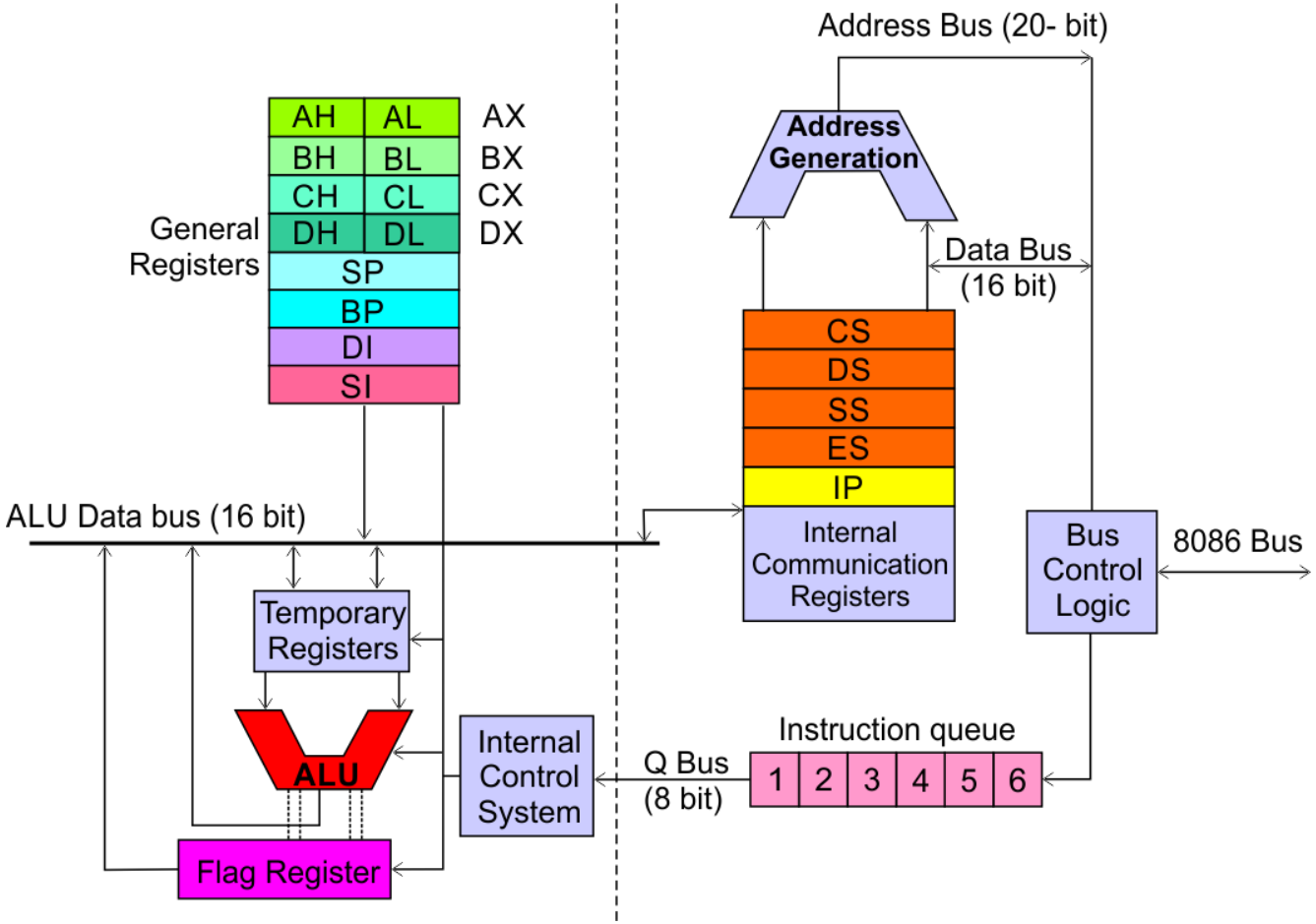
Microprocessors and Interfaces: 2021-22
Lecture 4
8086 Architecture

By Dr. Sanjay Vidhyadharan



8086 Architecture



Execution Unit (EU)

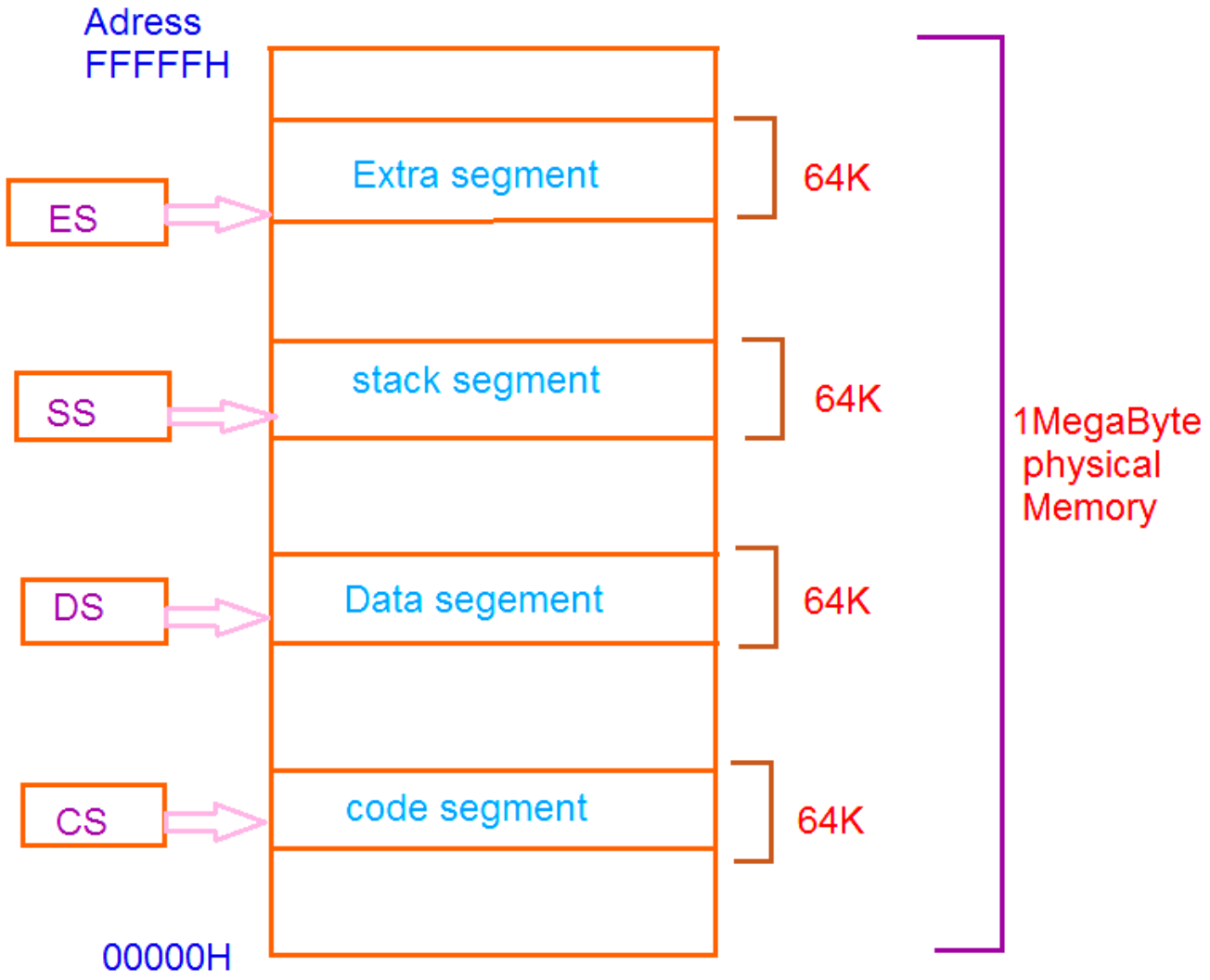
EU executes instructions that have already been fetched by the BIU.

BIU and EU functions separately.

Bus Interface Unit (BIU)

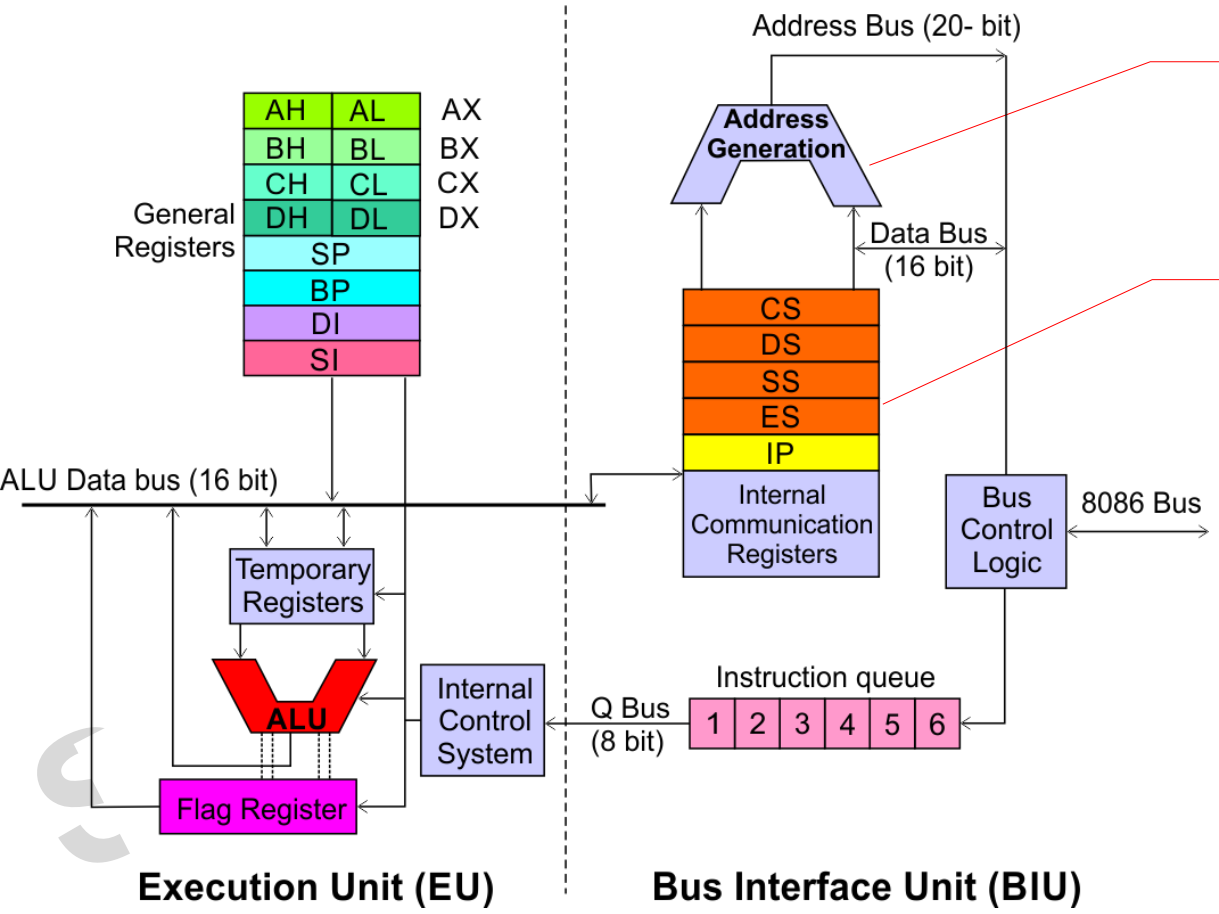
BIU fetches instructions, reads data from memory and I/O ports, writes data to memory and I/ O ports.

Memory Segmentation



8086 Architecture

Bus Interface Unit (BIU)



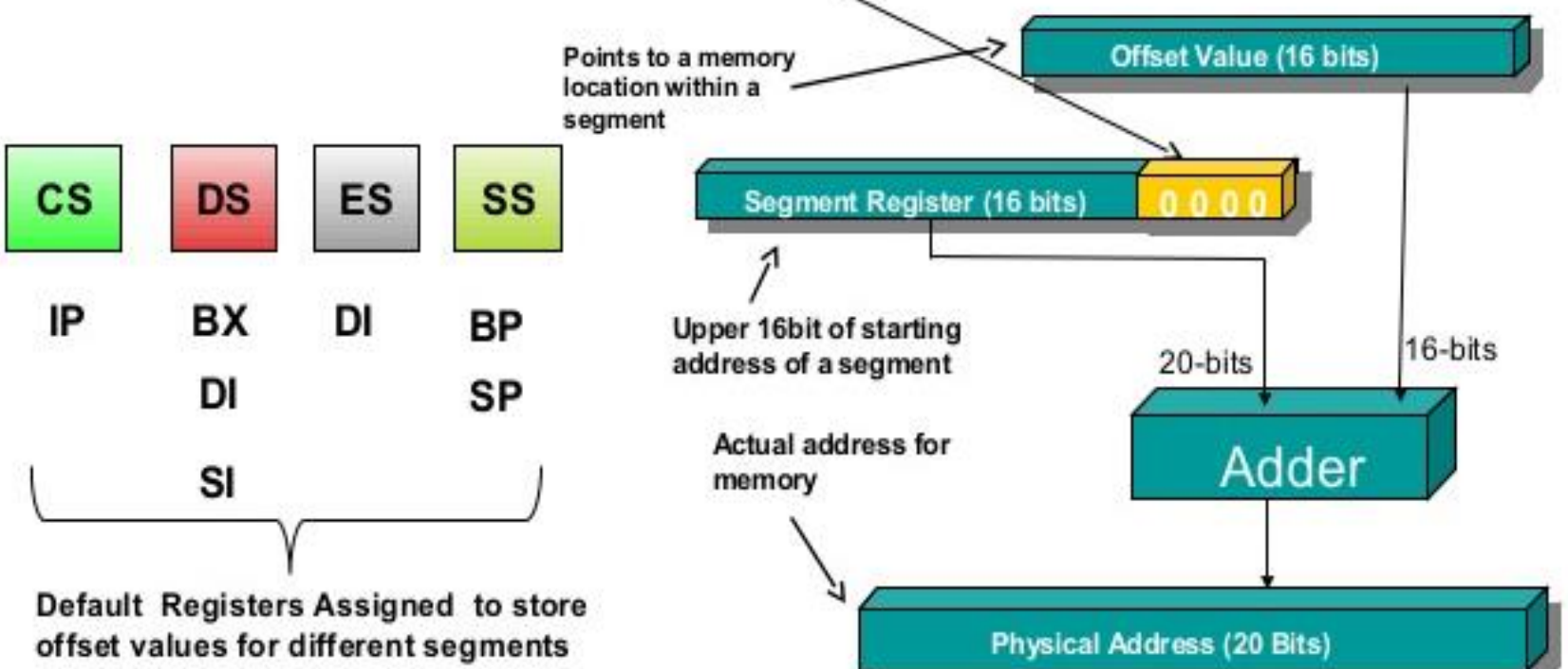
Dedicated Adder to generate 20 bit address

Four 16-bit segment registers

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

Physical Address Generation in 8086

- The 20-bit physical address is generated by adding 16-bit contents of a segment register with an 16-bit offset value (also called Effective Address) which is stored in a corresponding default register (either in IP, BX, SI, DI, BP or SP. Different segments have different default register for offset, for example IP is default offset register for Code Segment)
- BIU always appends 4 zeros automatically to the 16-bit address of a segment register (to make it 20-bit) because it knows the starting address of a segment always ends with 4 zeros



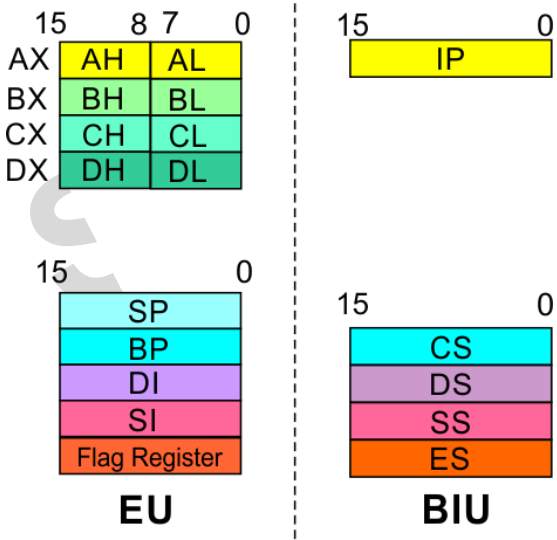
8086 Architecture

Bus Interface Unit (BIU)

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.**



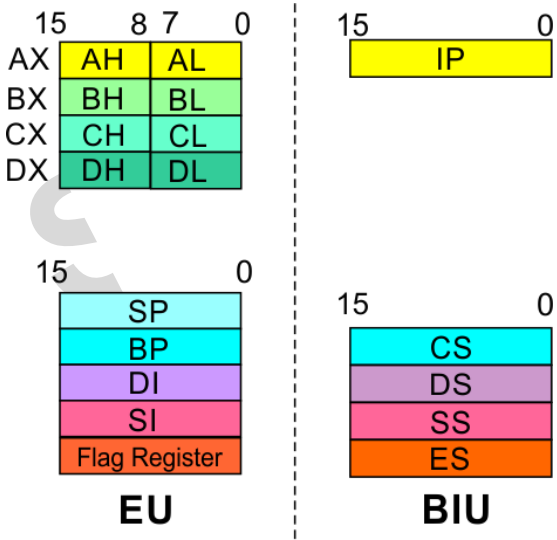
8086 Architecture

Bus Interface Unit (BIU)

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.**



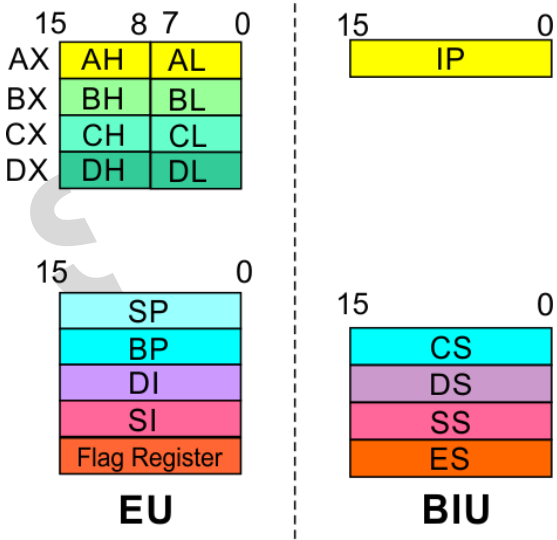
8086 Architecture

Bus Interface Unit (BIU)

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).**



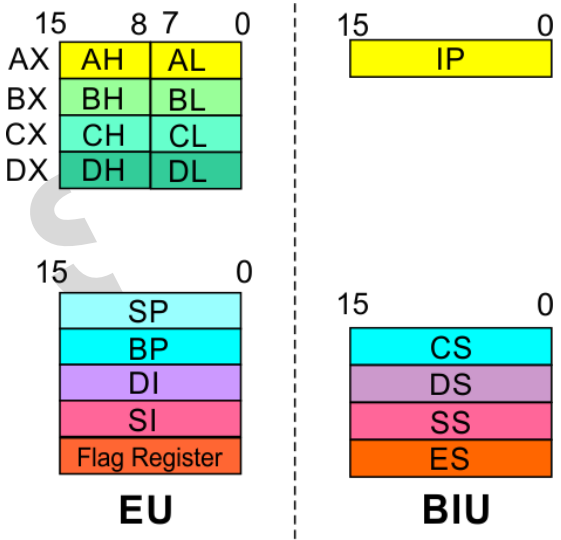
8086 Architecture

Bus Interface Unit (BIU)

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.**



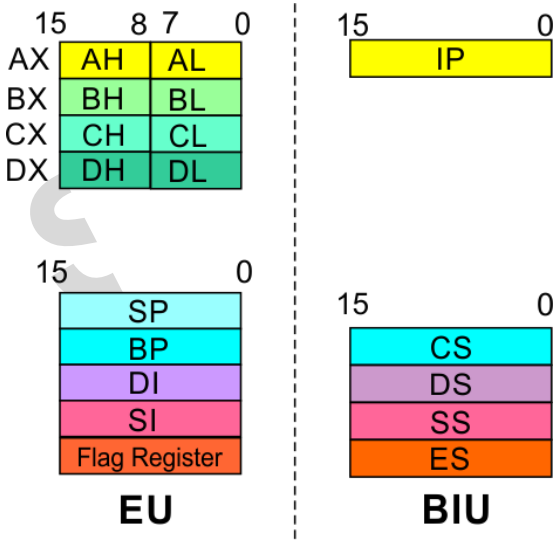
8086 Architecture

Bus Interface Unit (BIU)

Segment Registers

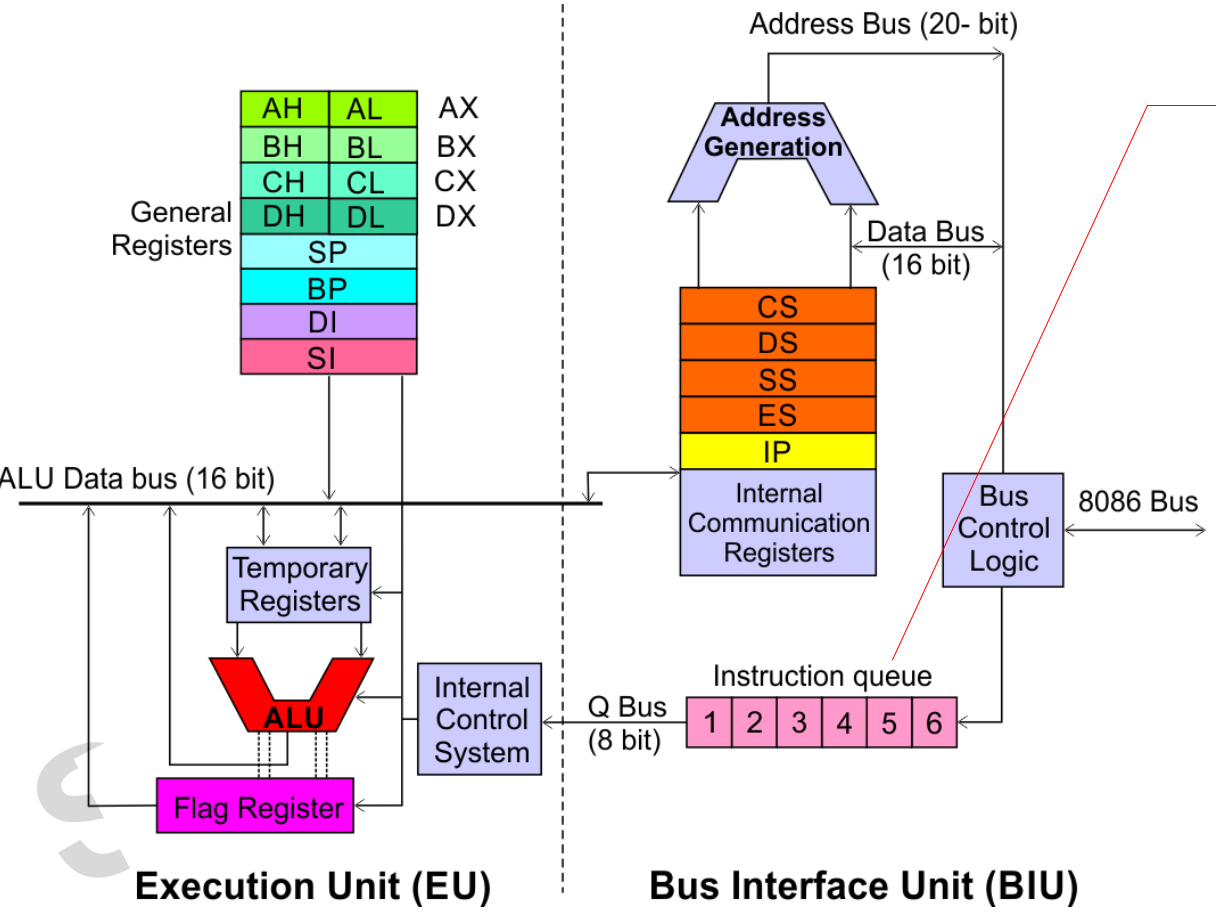
Instruction Pointer

- **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.**



8086 Architecture

Bus Interface Unit (BIU)

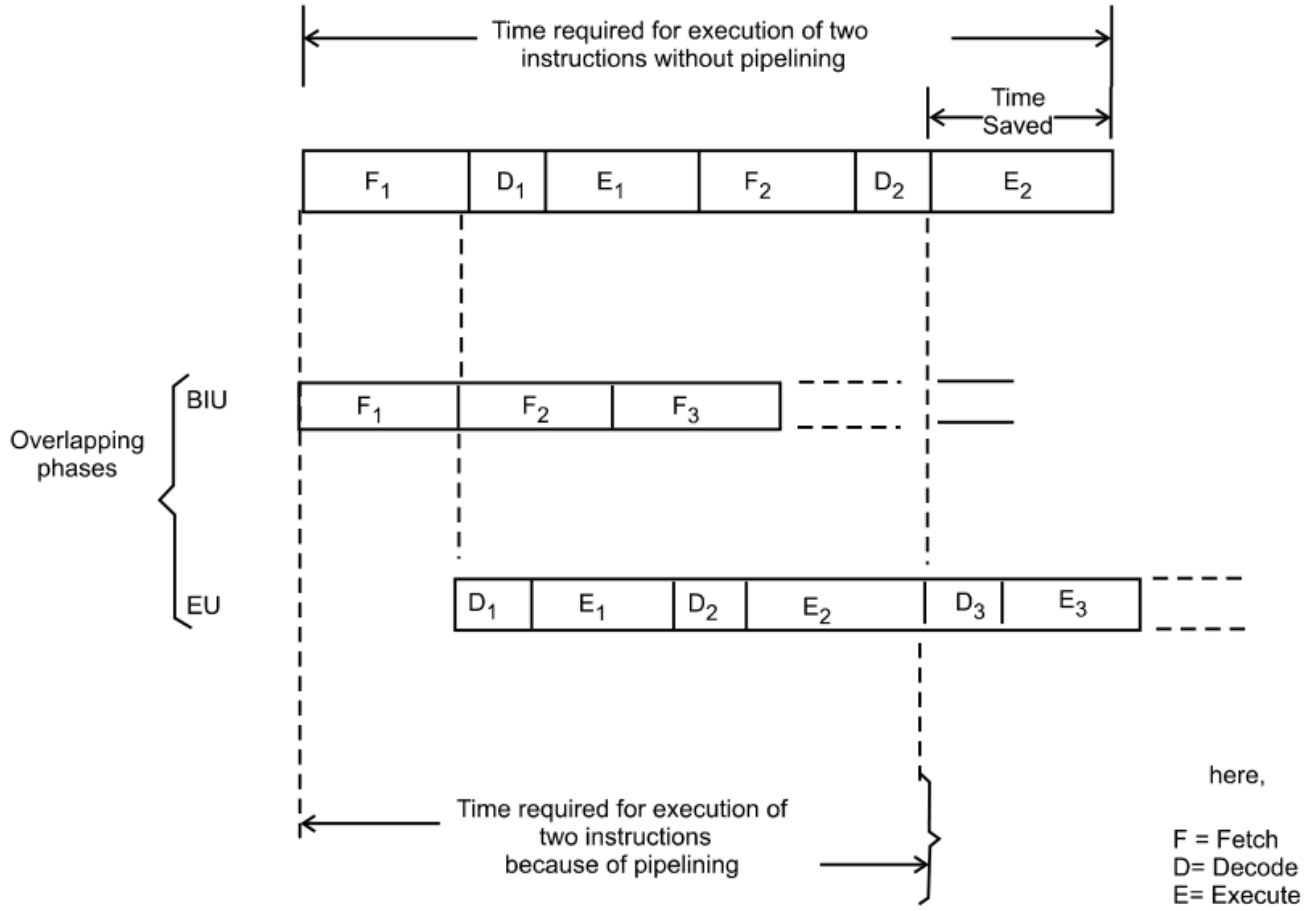


Instruction queue

- **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.**

8086 Architecture

Because of the instruction queue, there is an overlap between the instruction execution and instruction fetching. This feature of fetching the next instruction when the current instruction is being executed, is called *Pipelining*.



8086 Architecture

Execution Unit (EU)

EU decodes and executes instructions.

A decoder in the EU control system translates instructions.

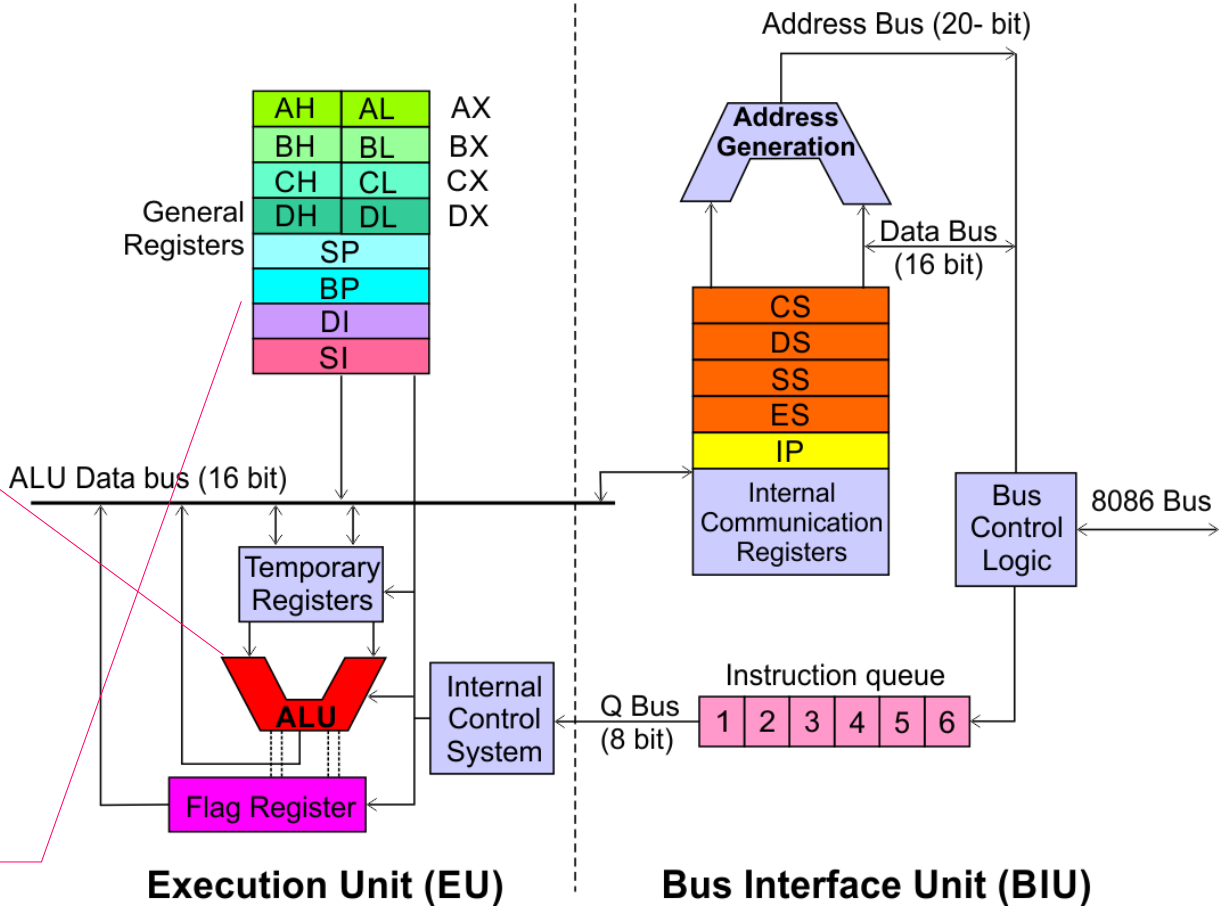
16-bit ALU for performing arithmetic and logic operation

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 :

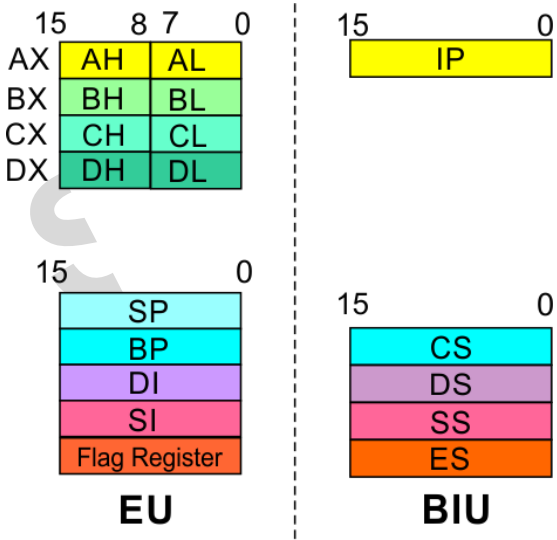
AX can be used as AH and AL
BX can be used as BH and BL
CX can be used as CH and CL
DX can be used as DH and DL

8086 Architecture

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.**



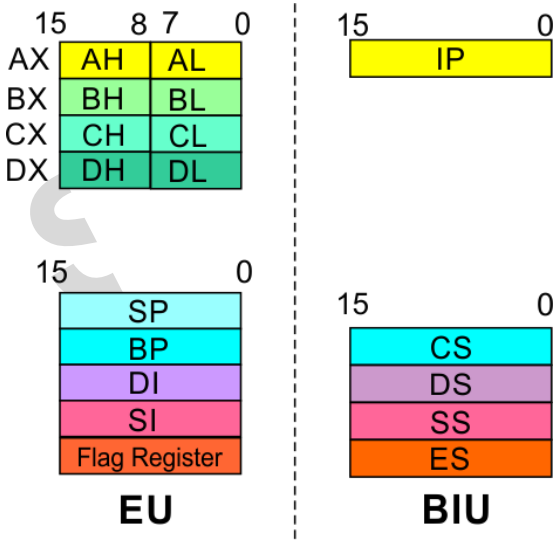
8086 Architecture

Execution Unit (EU)

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.**

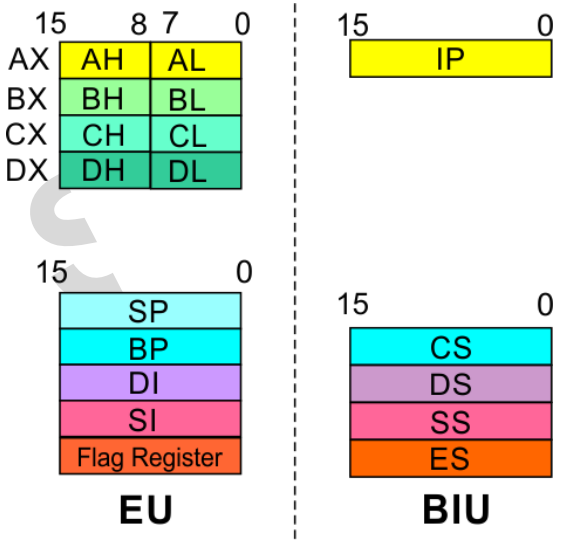


8086 Architecture

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.

8086 Architecture

Execution Unit (EU)

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.**

Data Register (DX)

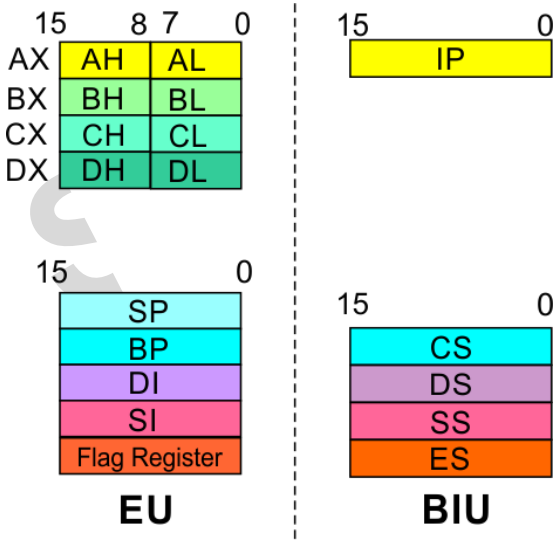
- Consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX.
- When combined, DL register contains the low order byte of the word, and DH contains the high-order byte.
- Used to hold the high 16-bit result (data) in 16 X 16 multiplication or the high 16-bit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

8086 Architecture

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.**

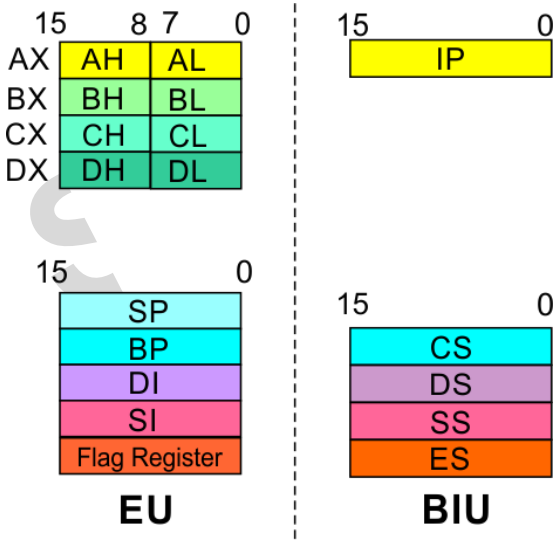


8086 Architecture

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.**



8086 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

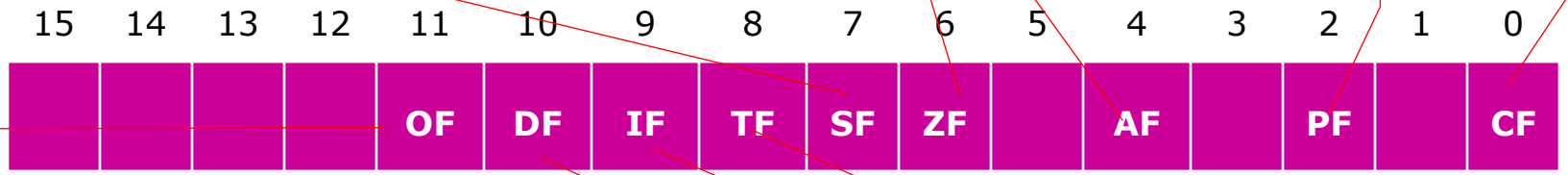
This flag is set, when the result of any computation is negative

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

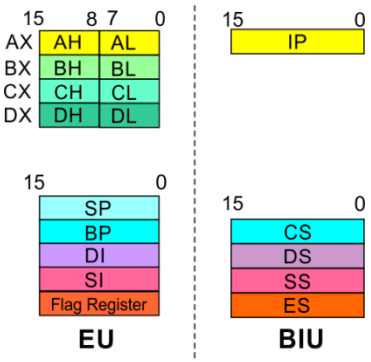
This is used by string manipulation instructions. If this flag bit is '0', the string is processed beginning from the lowest address to the highest address, i.e., auto incrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e., auto incrementing mode.

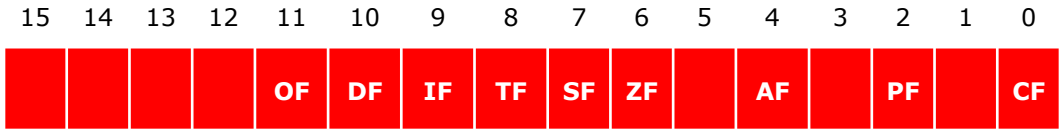
Interrupt Flag

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

8086 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

8086 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

Thankyou