CSC405 Microprocessor

By Rithesh Kini

Computer Eng Dept., TSEC

8086 Microprocessor

Features
Architecture
Programmer's Model

Contents

- 8086 Features
- 8086 – Architecture
  - Execution unit
  - Bus Interface Unit
  - Pipelining
- 8086 Programmer’s Model

8086 - Features

Basic Features:

1. 8086 is a **16 bit microprocessor.** Which means it has -
   - **16 bit ALU** that can perform 16-bit operation simultaneously
   - **16 bit internal registers** and **internal data bus**
   - **16-bit external data bus** and it can read from or write data to memory or I/O ports either 16 bits or 8 bits at a time

8086 - Features

2. It has **20 bits of address lines**

   ➢ It can access memory up to **2^{20}** memory locations

   i.e. **10,48,576** = **1Mbytes** locations. The address range is **00000h to FFFFFh**.

   ➢ **Each location** is **1 byte** wide. Therefor,16-bit numbers are stored in **consecutive memory locations**.

3. It has 16 address lines to access I/O devices, hence it can access 2^{16} = 65,536 I/O ports = **64K I/O locations**.

8086 - Features

4. 8086 has different **versions**.

	Versions of 8086 µP	**Working Frequency**
1	8086	5 MHz
2	8086-2	8 MHz
3	8086-1	10 MHz

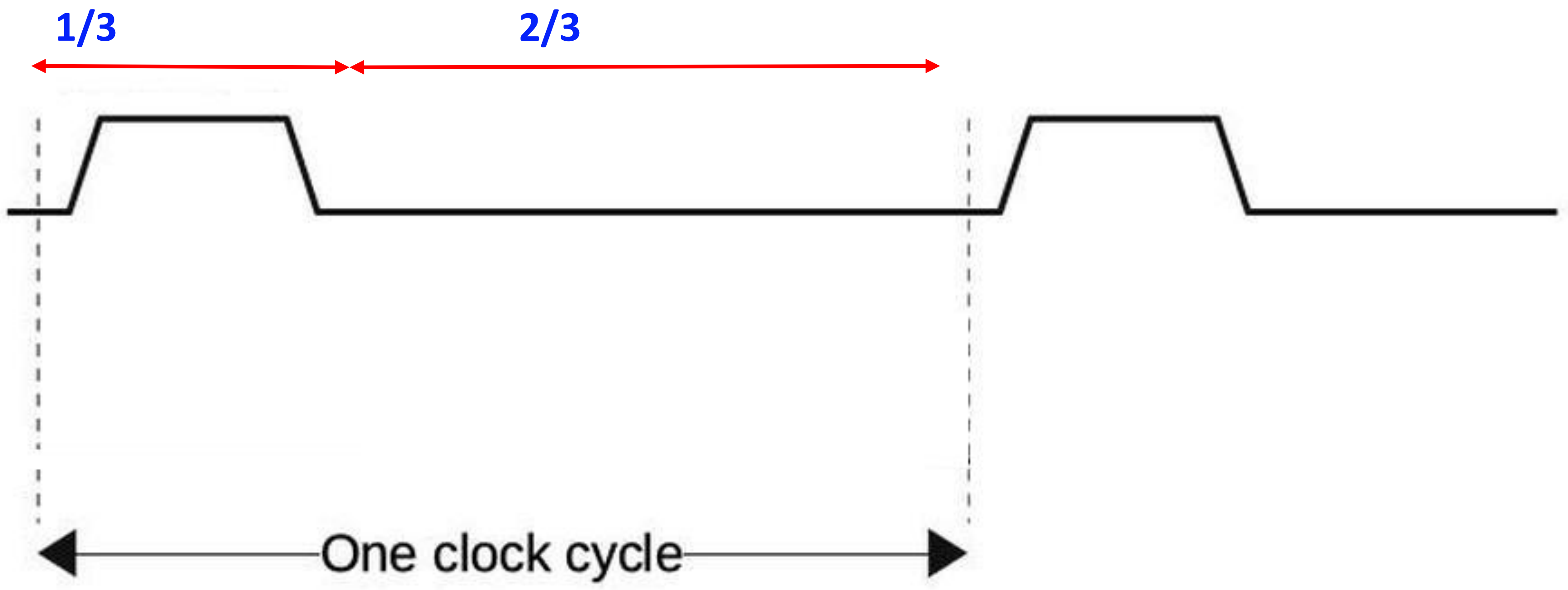
Working Frequency = maximum internal clock frequency

Intel 8086

8086 - Features

5. It requires a single phase **clock** with **33 % duty cycle**.
   - **No internal clock circuit**. So, clock is generated by a separate peripheral chip ***8284***.
6. It uses a **40-pin** DIP package.
7. It has **multiplexed** memory/IO address & data bus. This reduces number of pins needed.
8. It operates on **+5V** supply.

Clock Signal



Single phase (single line) clock with 33 % duty cycle

8086 - Features

Special Features:

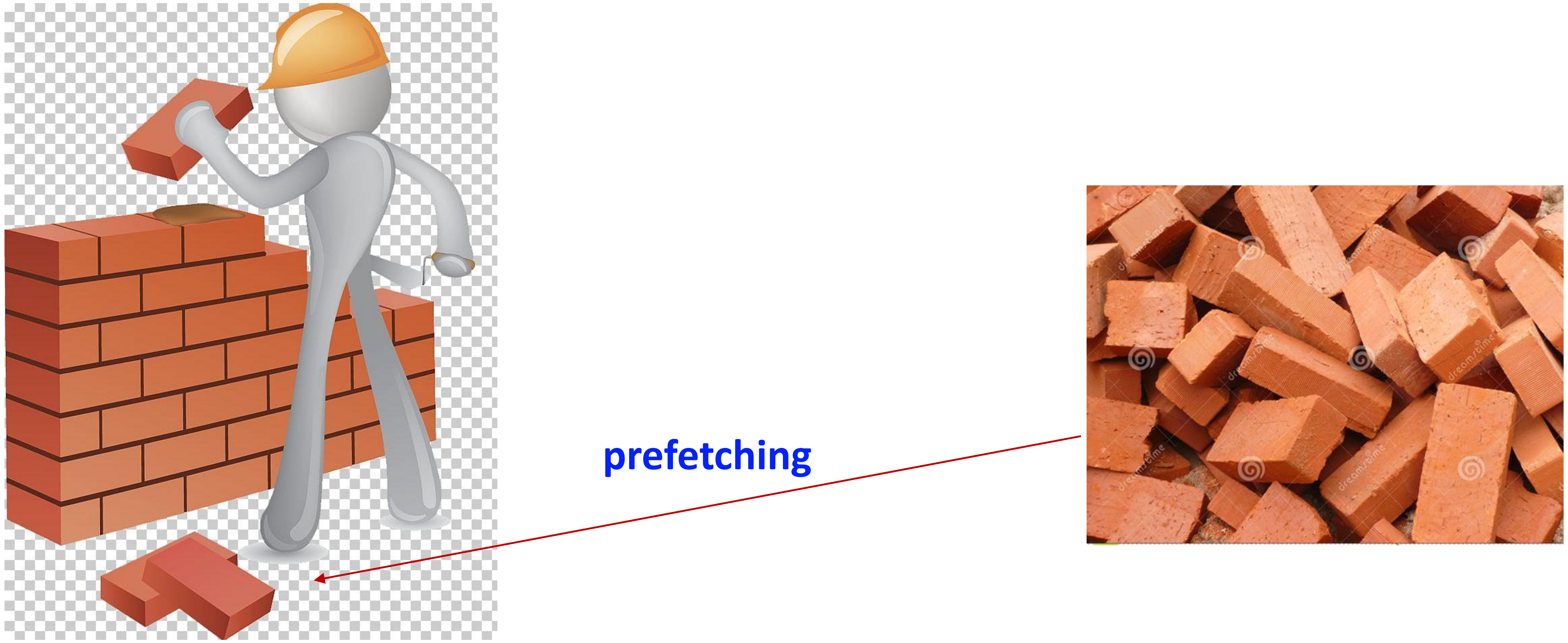
1. 8086 can do **only** **fixed-point arithmetic**.

   However, it can work **in conjunction with 8087** to do both fixed-point, floating-point & other complex mathematical functions.

8086 - Features

2. It is a **pipelined processor**: i.e., it supports 2 stage pipeline (fetch & execute)
   - **Pipelining : fetching** the next instruction while **executing** the current instruction.
   - It **prefetches** up to **6 instructions** bytes from memory & **queues** them in order to speed up execution process.
   - This improves the performance of the system i.e., the operations are faster => **Higher Throughput (Speed)**

Pipelining



8086 - Features

3. 8086 is designed to operate in 2 modes:
   - **Minimum mode** (**single processor** mode) : CPU generates control bus signals required by memory & I/O devices.
   - **Maximum mode** (**multiprocessor** mode) designed to be used to work with coprocessor 8087.
   - In multiprocessor mode the control signals for memory & I/O are generated with the help of an external BUS controller ***8288*.**

8086 - Features

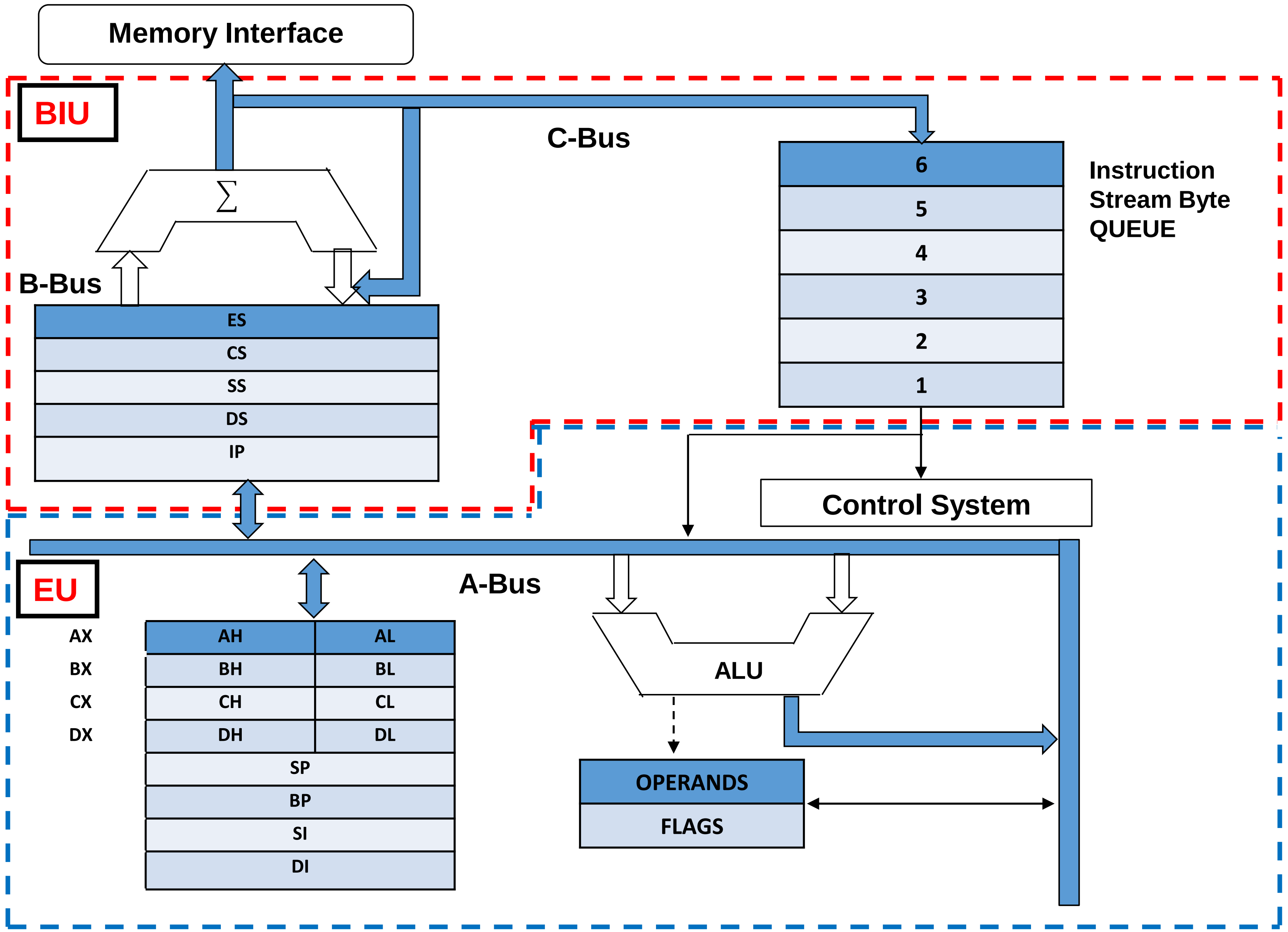
4. 8086 uses **Memory Banks** - 1MB is divided into 2 banks of 512KB – Lower (Even) and Higher (Odd) Banks.
5. It uses **memory segmentation**. i.e., divide memory into logical components. Divide 1Mbytes into 16 segments of 64Kbytes.
6. 8086 architecture has 2 separate units to fetch (BIU) and execute (EU) instructions.

8086 - Features

Miscellaneous Features:

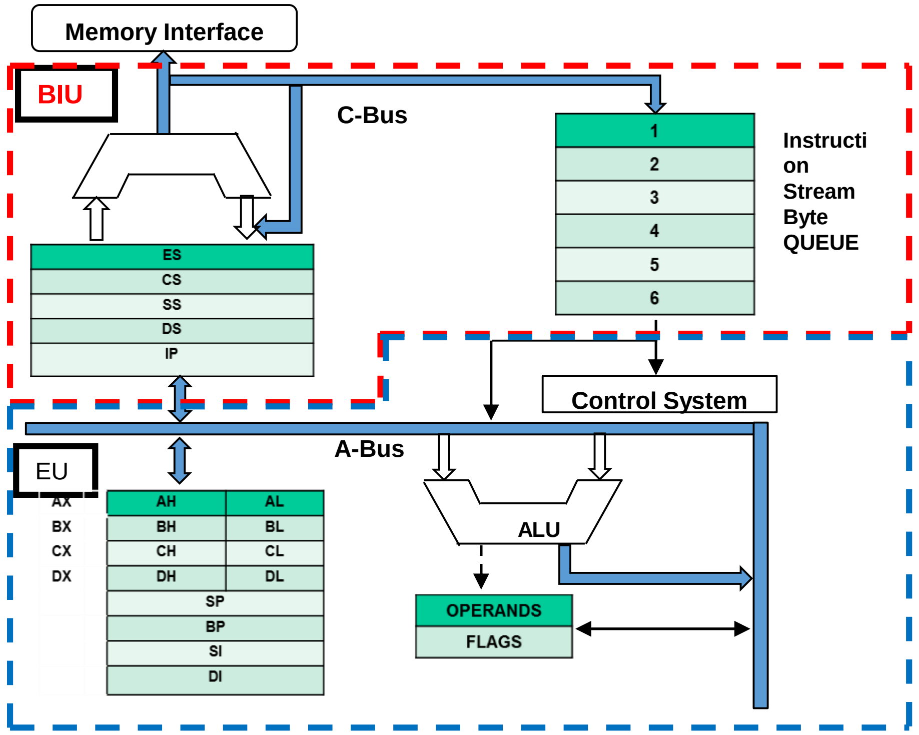
1. It has **256** types of **vectored interrupts**.
2. It has **fourteen 16-bit registers**.
3. Instruction set supports **MULTIPLY** and **DIVIDE** operations.
4. It can perform operations on **bit**, **byte(8-bits)**, **word(16-bits)** or a **string(block of data)**.

8086 - Architecture

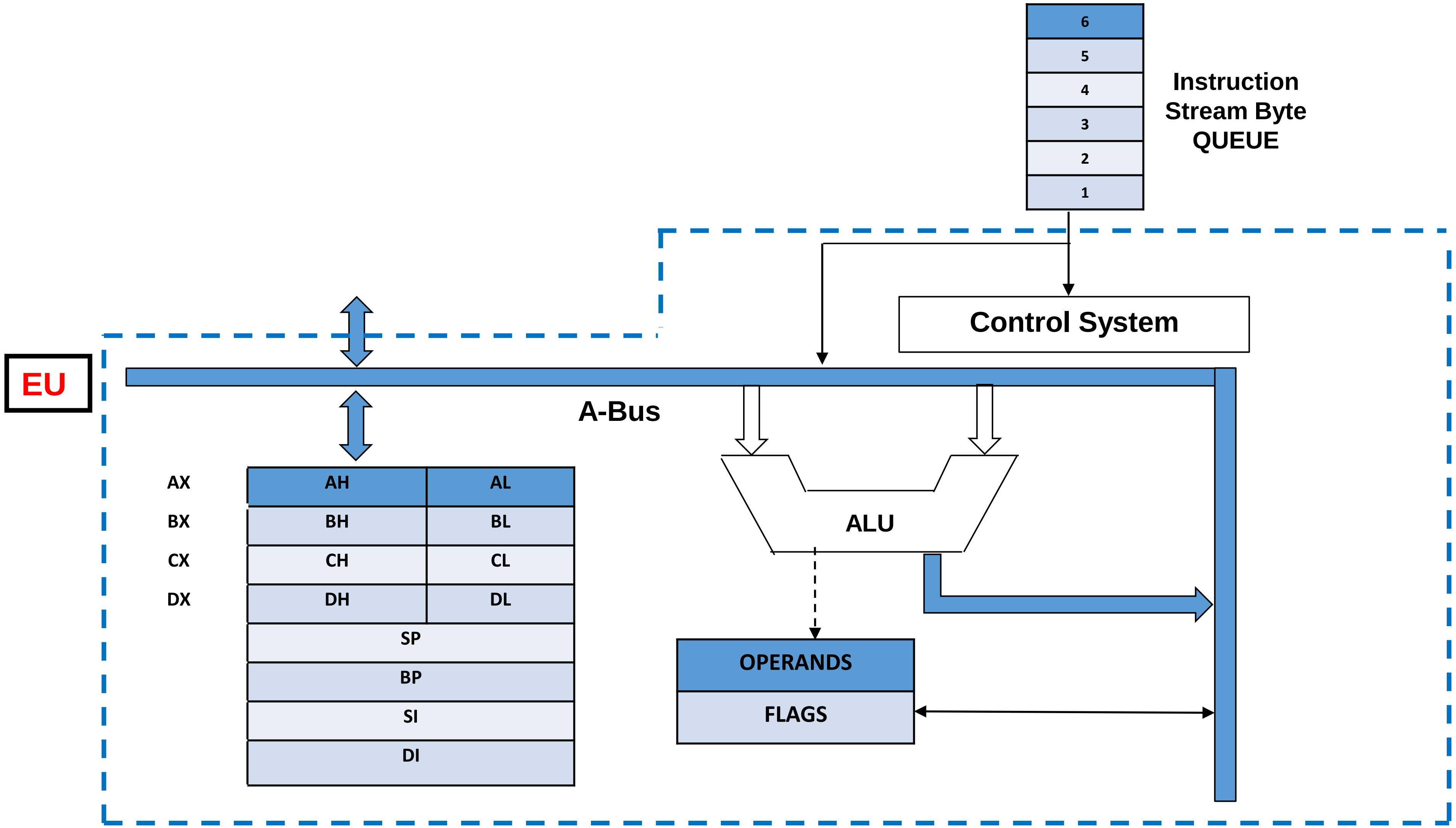


8086 - Architecture

- 8086 is divided into 2 independent functional parts:
  - I. Bus Interface Unit (BIU)
  - II. Execution Unit (EU)
- Dividing the work between these 2 units **speeds up processing**.



Execution Unit



Execution Unit

- The main function of EU is decoding and execution of the instructions already fetched by the BIU.
- EU also performs arithmetic, logic, decision making & data transfer operations.
- Operations of the EU are confined to µp. i.e., it does not directly perform any external (memory- or I/O-based) operation
- Sends request signals to BIU to access the external module.

Hence, EU operates w.r.t. T-states and not w.r.t. Bus Cycles

Execution Unit

The Execution Unit:

➢It has the following sub-units:

i. Arithmetic Logic unit (ALU)

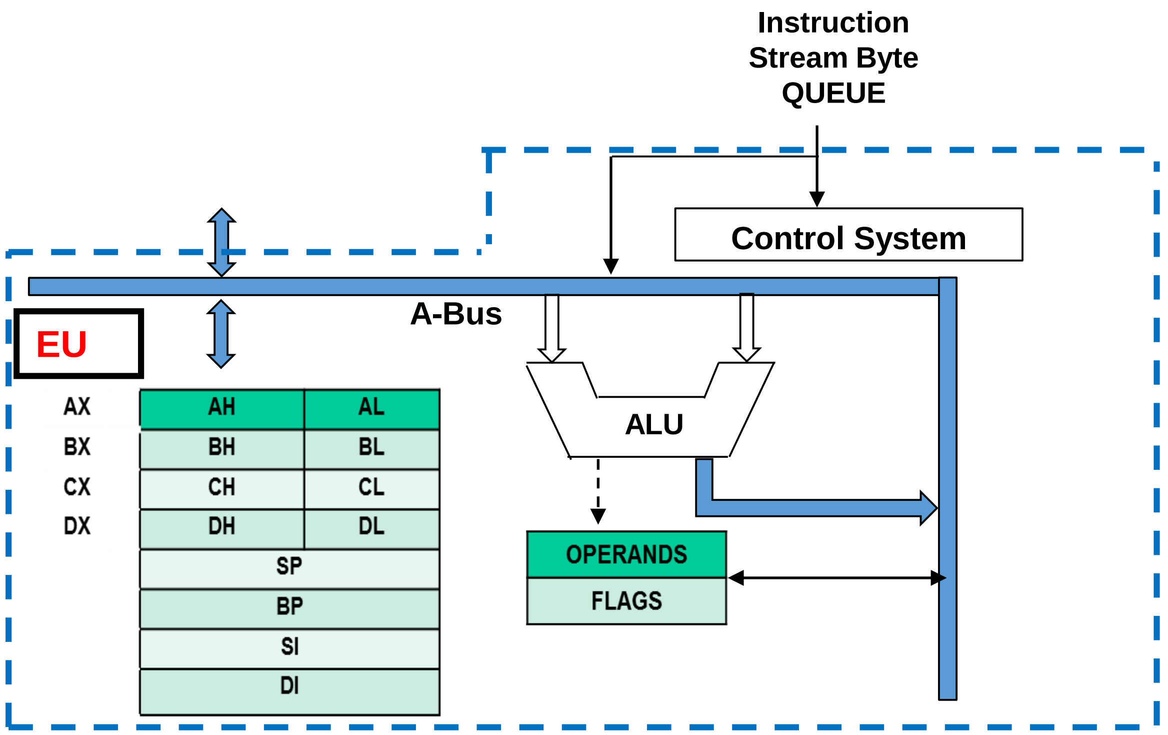
ii.Flag Register

iii.General Purpose Registers

iv.Control Unit

v. Decoder

vi.Pointer and Index Register



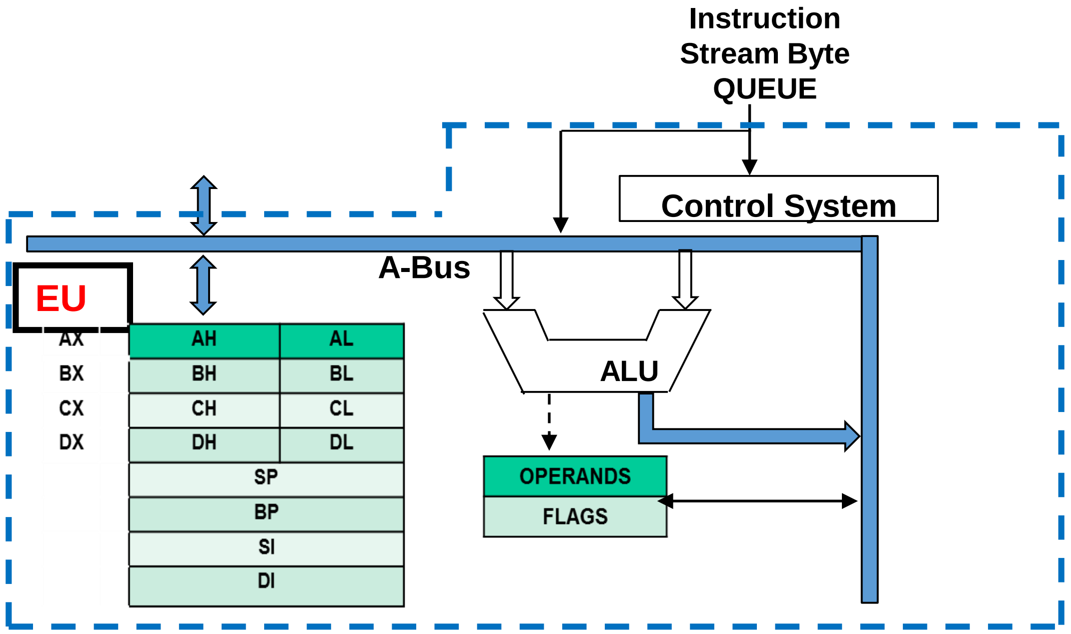
Execution Unit

i. **ALU (16-bit) :**

- Performs 8, 16-bit arithmetic & logical operations (AND, OR, XOR, increment, decrement, complement, or shift of bin nos.)

❑ 16 bit **Operand Register**

- Holds operands
- Not available to programmer



Execution Unit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

ii. **Flag Register (16-bit):**

- A Flag is a flip-flop
- 9 active flags
  - 6-Status flags &
  - 3-Control flags

Control Flag		Status Flag	
1	DF – Direction	1	SF - Sign
2	IF – interrupt Enable	2	ZF – Zero
3	TF – Trap	3	AF – Auxillary Carry
		4	PF – Parity
		5	CF – Carry
		6	OF - Overflow

Execution Unit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

ii. Flag Register (16-bit):

- **Status Flags** are set or reset **by EU**
- **Control flags** are used to control certain operations

Control Flag		Status Flag	
1	DF – Direction	1	SF - Sign
2	IF – interrupt Enable	2	ZF – Zero
3	TF – Trap	3	AF – Auxillary Carry
		4	PF – Parity
		5	CF – Carry
		6	OF - Overflow

Execution Unit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

ii. Flag Register (16-bit) : Status Flags

1. Carry Flag (CF) :

- **Carry** is set after an arithmetic operation results in a carry out of MSB.
  - Carry out of MSB after an addition operation or
  - Borrow out of MSB after a subtraction operation
- **CF =1 => Carry out of MSB**
- **CF =0 => NO Carry out of MSB**
- This flag is also **used** in some **shift** and **rotate** instructions.

Execution Unit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

ii. Flag Register (16-bit) : Status Flags

2. Parity Flag (PF):

- This is set to 1 if the result of byte operation or lower byte of the word operation contains an even number of ones; otherwise it is zero
- If PF=1 => Even parity (Even no of 1's in the result)
- If PF=0 => Odd parity (Odd no of 1's in the result)

Execution Unit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

ii. Flag Register (16-bit):

3. Auxiliary Carry Flag (AF):

- The flag is set if there is a carry out of the lower nibble to the higher nibble (D_3 to D_4 bit), during addition or borrow from higher nibble into the lower nibble (D_4 to D_3), during subtraction.
- **AC=1** => Carry from lower nibble to higher nibble.
- **AC=0** => NO Carry from lower nibble to higher nibble.
- It is **used** by **decimal (BCD) arithmetic** instructions.

Execution Unit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

ii. Flag Register (16-bit):

4. Zero Flag (ZF):

- The zero flag is set whenever the result of the operation is zero
- **ZF = 1 =>** zero result
- **ZF = 0 =>** non-zero result

Execution Unit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

ii. Flag Register (16-bit):

5. Sign Flag (SF):

- The sign flag is set if, after the arithmetic or logic operations, the MSB of the result is 1.
- For **signed numbers** MSB of the number indicated the sign.
- If SF=1 => MSB of the result is 1, hence the result is negative
- If SF=0 => MSB of the result is 0, hence the result is positive

Execution Unit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

ii. Flag Register (16-bit):

6. Overflow Flag (OF):

- This flag is used to detect magnitude overflow in signed arithmetic operation.
- Recollect 8-bit signed number range (-80 to +7F)
- When the result exceeds this range then it is said to have overflown
- For addition operation, the flag is set when there is a **carry into the MSB** and **no carry out of the MSB** or vice versa.
- For subtraction operation, the flag is set when the MSB needs a borrow and there is no borrow from MSB or vice versa.

Execution Unit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

ii. Flag Register (16-bit):

1. Trap Flag :

- One way to run a program is to run the program one instruction at a time & see the contents of used registers and memory variables after execution of every instruction.
- This process is called '**single stepping**' though a program.
- TF = 1, "start" Single stepping
- TF = 0, "stop" Single stepping
- This is very useful for debugging a program.
- Single stepping though a program is a tool to identify and solve "Logical Errors".

Execution Unit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

ii. **Flag Register (16-bit):**

2. **Interrupt Flag :**

- Used to allow/prohibit the interruption of a program.
- If IF = 1 (Set), a certain type of interrupts (a maskable interrupt) can be recognized by the 8086
- If IF = 0 (Reset), maskable interrupts are ignored
- IF has no effect on a non-maskable interrupt

Execution Unit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

ii. **Flag Register (16-bit):**

3. **Direction Flag:**

- Used with the string instructions which works on a continuous block of data
- If DF = 0, the string is processed from its beginning with the first element having the lowest address. i.e., from low address to high address
- If DF = 1, the string is processed from the high address towards the low address.

"Hello!" ← string is a series of characters

Mem loc	contents
1005H	'!'
1004H	'o'
1003H	'l'
1002H	'l'
1001H	'e'
1000H	'H'

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

Example1: Give the contents of the flag register after execution of the following addition:

$$\begin{array}{ll} & 0010\ 0001\ (21H) \\ + & \underline{0100\ 0001\ (41H)} \\ & 0110\ 0010\ (62H) \end{array}$$

Solution : SF = , ZF = , PF = , CF = , AF = , OF = .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

Example1: Give the contents of the flag register after execution of the following addition:

```
        0010 0001  (21H)
+       0100 0001  (41H)
        ----------------
        0110 0010  (62H)
```

Solution : SF = 0, ZF = 0, PF = 0, CF = 0, AF = 0, OF = 0.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	**X**	**X**	**X**	**OF**	**DF**	**IF**	**TF**	**SF**	**ZF**	**X**	**AF**	**X**	**PF**	**X**	**CF**

Example2: Give the contents of the flag register after execution of the following addition:

$$
\begin{array}{ll}
 & 0001\ 1001\ (19H) \\
+ & \underline{0101\ 1000\ (58H)} \\
 & 0111\ 0001\ (71H)
\end{array}
$$

Solution : SF = , ZF = , PF = , CF = , AF = , OF = .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

Example2: Give the contents of the flag register after execution of the following addition:

```
     0001 1001 (19H)
+    0101 1000 (58H)
     0111 0001 (71H)
```

Solution : SF = 0 , ZF = 0 , PF = **1** , CF = 0, AF = **1** , OF = 0 .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

Example3: Give the contents of the flag register after execution of the following addition:

```
     0100 0000 (40H)
+    0100 0101 (45H)
     1000 0101 (85H)
```

Solution : SF = , ZF = , PF = , CF = , AF = , OF = .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

Example3: Give the contents of the flag register after execution of the following addition:

```
      0100 0000 (40H)
+     0100 0101 (45H)
      1000 0101 (85H)
```

Solution : SF = 1, ZF = 0 , PF = 0, CF = 0, AF = 0, OF = 1.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

Example4: Give the contents of the flag register after execution of the following addition:

```
      0110 0101 1101 0001 (65D1H)
+     0010 0011 0101 1001 (2359H)
      -------------------
      1000 1001 0010 1010
```

Solution : SF = 1, ZF = 0, PF = 1, CF = 0, AF = 0, OF = 1.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

Example4: Give the contents of the flag register after execution of the following addition:

```
      0110 0101 1101 0001 (65D1H)
+     0010 0011 0101 1001 (2359H)
      1000 1001 0010 1010
```

Solution : SF = 1, ZF = 0, PF = 1, CF = 0, AF = 0, OF = 1.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

Example5: Give the contents of the flag register after execution of the following addition:

$$\begin{array}{l} 0110\ 0111\ 0010\ 1001 \\ \underline{0011\ 0101\ 0100\ 1010} \\ 1011\ 1100\ 0111\ 0011 \end{array}$$

Solution : SF = , ZF = , PF = , CF = , AF = , OF = .

Execution Unit

iii. General Purpose Registers (GPRs):

- Four 16 bit registers – **AX, BX, CX, DX**
- Available to the programmer for storing values during program
- Can be divided into 8 bit registers - **AH**,**AL**, **BH**,**BL**, **CH**,**CL**, **DH**,**DL**
- 32 bit registers – **DX:AX**

AX	**AH**	**AL**	Accumulator
BX	**BH**	**BL**	Base Register
CX	**CH**	**CL**	Count Register
DX	**DH**	**DL**	Data Register

Execution Unit

iii. **General Purpose Registers:**

Special functions of general purpose register:

1. ***AX reg*** : – 16 bit **accumulator**

   - **Holds 1st operand and the result** in complex arithmetic operation like **Multiply** & **Divide**.

     Ex: MUL BL; BL will be multiplied with AL & the result will be stored in AX.

   - Holds data during **I/O operation** such as **IN** & **OUT**
   - Holds data during **string operations** such as **LODS** & **STOS**

Execution Unit

2. ***BX reg*** : 16 bit **base** register

- BX is also called as a memory pointer.
- The **only GPR** which may be used for **indirect addressing**: Holds **memory addr** in Base Relative addressing mode
- **Ex**: MOV [BX],AX **=>** store the contents of AX in the memory location whose address is given in BX.

Execution Unit

3. ***CX reg***:– 16 bit count register

- It is the default counter register for 3 instructions :
  1. **Shift & Rotate** instructions : to indicate no. of shifts or rotations (only 8 bit **CL** used)
  2. In **Loop** instructions: to indicate iterations
  3. In **String** instructions : to indicates the size of the string block
- For Loop & String instructions the count is in **CX** register

Execution Unit

4. ***DX reg*****:**– 16 bit data register
   - Used with AX (as an extension) to hold result greater than 16 bit
   - Used to hold **port number** for **IN** and **OUT** instructions
   - Used as a **only pointer** for **16 bit address** accesses of **I/O port** in **Indirect addressing mode**

Execution Unit

iv. Control Circuitry & v. Decoder

- **Control circuitry** is used for directing the internal operation
- After the EU grabs the instruction from the prefetched queue, it goes into the **Instruction Register**
- Every instruction has an "**Opcode**" which is a unique binary code that represents the operation to be performed.
- The **Instruction Decoder** **analyzes the opcode** to understand the operation to be performed and sends the info to the control section
- The control section then generates internal control signals that inform all parts of the architecture, the actions to be taken for the execution of the instruction

Execution Unit

vi. Pointer and Index Register:

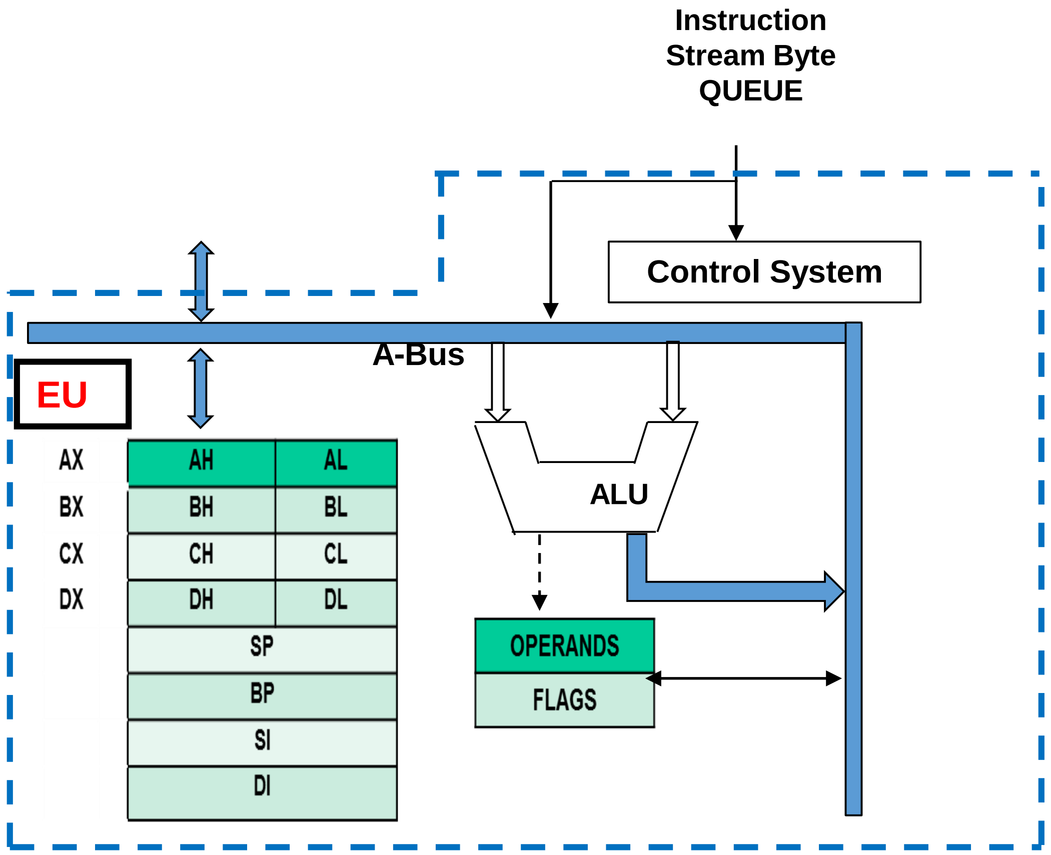
- Can be used as 16-bit GPRs.
- Used to hold 16 bit **offset** of data word in one of the segments:

  i. **Stack Pointer (SP)**

  ii. **Base Pointer (BP)**

  iii. **Source Index (SI)**

  iv. **Destination Index (DI)**



AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL
	SP	
	BP	
	SI	
	DI	

Execution Unit

vi. **Pointer and Index Register**:

1. **Base Pointer (BP 16 bit) : 16-bit wide**
   - Holds the 16 bit offset relative to the **stack segment** (**SS**) register.
   - Can hold offset address of any location in stack segment
   - Specifically used when we pass parameter by way of stack

Also used as an offset register in **base addressing mode**.

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL
	SP	
	BP	
	SI	
	DI	

Execution Unit

vi. **Pointer and Index Register**:

2. **Stack Pointer (SP) : 16-bit wide**

- Holds the 16 bit offset address relative to the stack segment (SS) register
- **SP always points to the top of the stack**
- Stack memory locations work in **LIFO** manner
- Stack is present in the stack segment of the memory
- SP is used in **sequential access** of stack segment
- Physical address of top of the stack = SS × 10H + SP
- SP is used during instructions such as PUSH, POP, CALL, RET…
- PUSH instruction will decrement SP by 2
- POP instruction will increment SP by 2

Execution Unit

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL
	SP	
	BP	
	SI	
	DI	

vi. Pointer and Index Register:

3. Source Index (SI) : 16-bit wide

- By default, holds the 16 bit offset of a data word in the data segment (DS)
- Can also be used with other segments using **segment overriding**.
- The physical address is generated by adding a **hard wired 0** to the segment base held by **DS reg** and then adding the **offset** in the **SI reg** to it.
- Why the name Source Index?
  - ✓ During string instructions like “**MOVSB**”, SI holds the **offset address** of the **source data** in the Data Segment

Execution Unit

vi. Pointer and Index Register:

3. Source Index (SI)

DS reg	2	0	0	0	0 ←	Hardwired 0
SI Reg.	+	2	F	3	2	
20-bit physical add in DS	2	2	F	3	2	

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL
	SP	
	BP	
	SI	
	DI	

Execution Unit

vi. **Pointer and Index Register**:

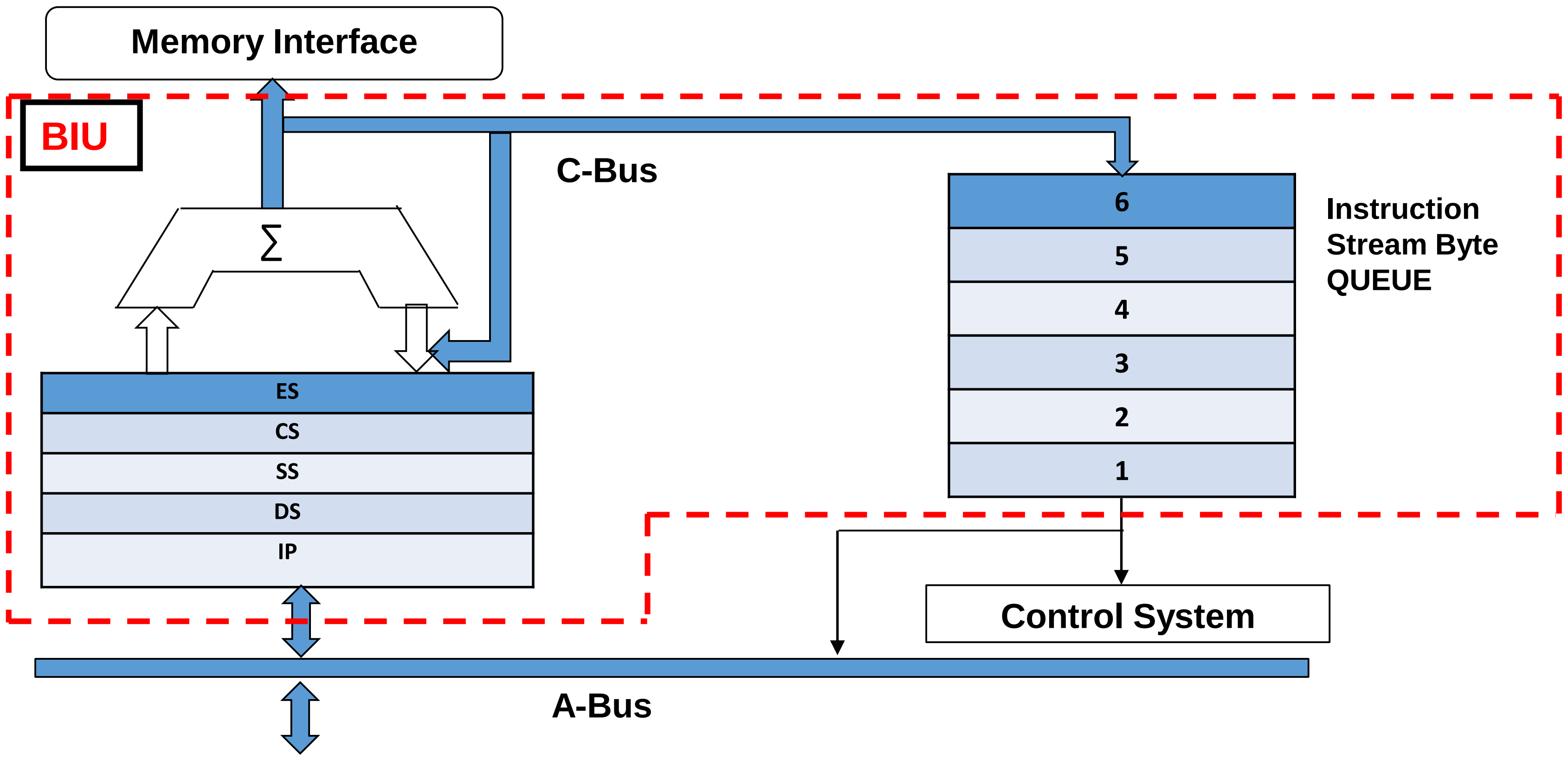
4. **Destination Index (DI)**

- Holds the 16 bit offset of a data word in the extra segment (ES)
- Can also be used with other segments using **segment overriding**.
- Like SI, DI is used for string related instructions
  - **Eg:** During string instructions like "**MOVSB**", DI holds the **offset address** of the **destination data** in Extra Segment

Execution Unit

Default Segment Register	**Pointer Register**
CS	**IP**
DS	**BX**
DS	**SI**
DS (**ES** in case of STRING Operation)	**DI**
SS	**SP**
SS	**BP**

BIU Unit


Memory Interface
BIU
C-Bus
∑
ES
CS
SS
DS
IP
6
5
4
3
2
1
Instruction Stream Byte QUEUE
Control System
A-Bus

BIU Unit

- Its main function is to fetch instructions from the primary memory.
- It interfaces to the outside world. i.e., all external data transfers with memory and I/O such as, Memory Read, Memory Write, I/O Read & I/O Write as well as the Instruction Fetch operations are performed by the BIU.
- These **µP initiated operations** are called as machine cycle or bus cycle
- Therefore, we say, the BIU operates w.r.t. to machine(or bus) cycles.
- To perform any memory operation the BIU has to calculate the 20-bit Physical Address using an arithmetic circuit (see the Diagram).

BIU Unit

- The formula used to calculate the Physical Address is :

  Physical Address = Segment Address × 10H + Offset Address

- As we know, the instructions are fetched from the **code segment**. To do this, CS reg. provides the segment address & IP reg. provides the offset address. Hence,

  Physical Address = CS × 10H + IP

Ex:

BIU Unit Functions

- Performs the following functions

  i. **Generates** 20-bit **physical address** for memory access.

  ii. **Fetches Instruction** from memory.

  iii. **Transfers data** to and from **memory and I/O**.

  iv. **Supports pipelining** using 6 byte instruction Queue.

BIU Unit

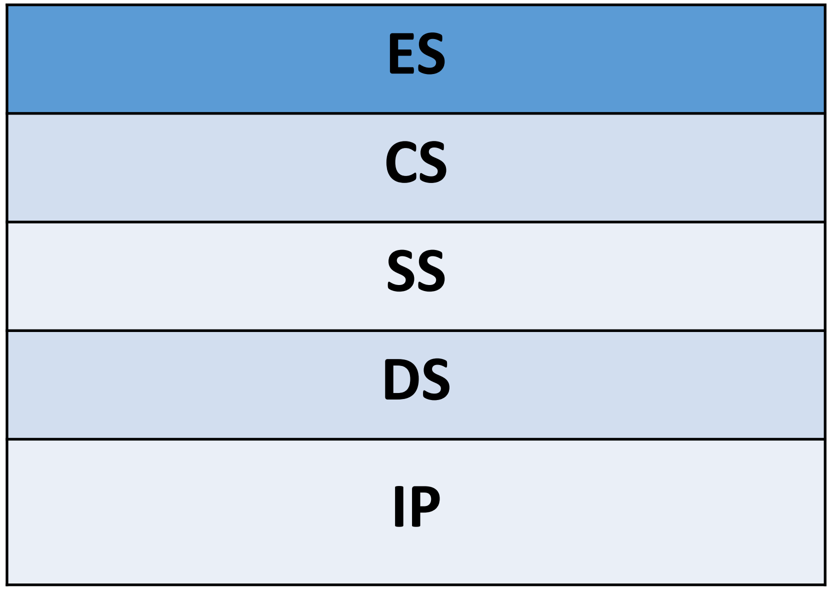
- BIU has 4 main components :
  1. Segment Registers: 4 **special purpose 16 bit** registers
  2. Instruction Pointer (IP): **16 bit** register
  3. Instruction Queue: **6 – Byte** Prefetch Queue.
  4. Address generation and bus control:
     i. The BIU contains a dedicated adder - Used to produce the 20-bit address.
     ii. The Bus Control Logic - Generates all the bus control signals such as read and write signals for memory and I/O.

8086 BIU - Internal Registers

BIU has

I. **4 16-bit Segment Registers:**
   1. Code Segment (CS) Register
   2. Data Segment (DS)
   3. Stack Segment (SS)
   4. Extra Segment (ES)

II. **Instruction Pointer (IP) Register**



8086 BIU - Internal Registers (cont..)

Code Segment (CS) - 16-bit register

- It contains the upper 16-bits of the starting (base) address of 64 KB code segment
- Code segment holds all the programs
- Therefore, the processor uses **CS segment** for accessing all instructions referenced by **instruction pointer** (IP) reg.
- To generate 20-bit physical address of the beginning of the code segment CS is multiplied by 10H (16_d).
- CS register cannot be changed directly.
- The CS register is automatically updated during far jump, far call and far return instructions.

8086 BIU - Internal Registers (cont..)

Stack Segment (SS) - 16-bit register

- It contains upper 16-bits of the starting (base) address of 64KB stack segment for the program stack.
- By default, all data referenced by **the stack pointer (SP) and base pointer (BP)** registers is located in the stack segment.
- To generate 20-bit physical address of the beginning of the stack segment SS is multiplied by 10H (16_d).
- SS register can be changed directly using POP instruction.

8086 BIU - Internal Registers (cont..)

Data segment (DS) - 16-bit register

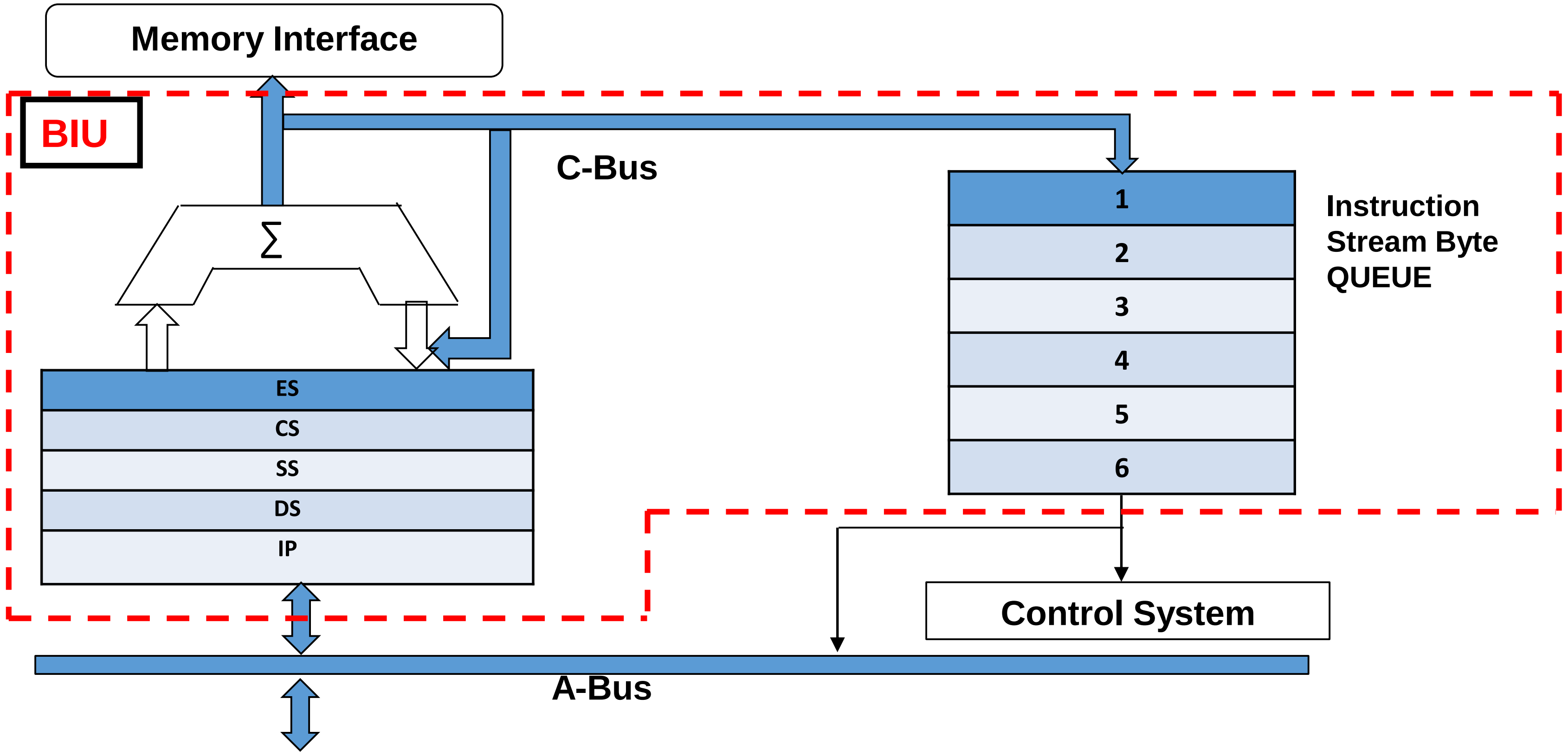
- It contains the upper 16-bits of the starting (base) address of 64KB Data segment with program data.
- By default, all data referenced **by general purpose registers (AX, BX, CX, DX) and index register (SI, DI)** is located in the data segment.
- To generate 20-bit physical address of the beginning of the data segment DS is multiplied by 10H (16_d).
- DS register can be changed directly using POP and LDS instructions.

8086 BIU - Internal Registers (cont..)

Extra segment (ES) - 16-bit register

- It contains the upper 16-bits of the starting (base) address of 64KB extra segment, usually with program data.
- By default, the processor assumes that the DI register references the ES segment in **string** manipulation instructions.
- To generate 20-bit physical address of the beginning of the Exata segment ES is multiplied by 10H (16_d).
- ES register can be changed directly using POP and LES instructions.

BIU Unit


Memory Interface
BIU
C-Bus
∑
ES
CS
SS
DS
IP
1
2
3
4
5
6
Instruction Stream Byte QUEUE
Control System
A-Bus

8086 BIU - Internal Registers (cont..)

Instruction Pointer (IP reg). 16 bit

- Hold the 16 bit addr of the next instruction byte within the code seg.
- The contents of IP is referred as **offset**.
- Used to calculate the addr. of the next instruction as

  CS × 10H + IP.
- It is incremented after every instruction byte fetch.
- IP gets a new value every time a branch occurs.

20-bit Physical Address Generation Circuit

It generates the **20-bit address** using the Segment & Offset addresses using the formula:

Physical address = Segment address X 10H + Offset address

- Basically the segment address is left shafted by 4 positions. This multiplies the number by 16_d (i.e., 10H)
- Then the Offset address is added to the shifted segment address.

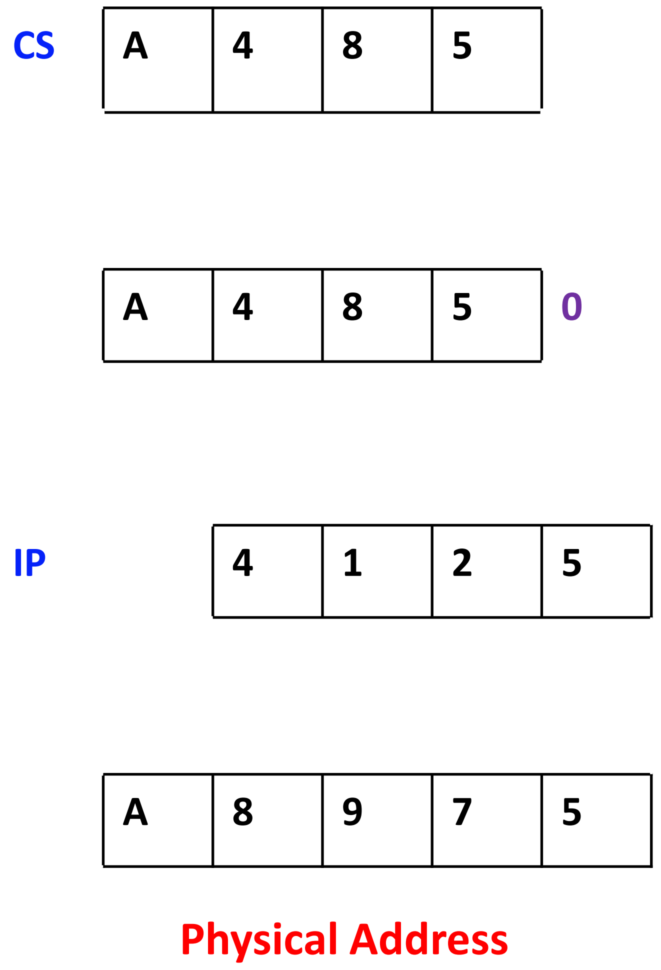
20-bit Physical Address Generation

The CS Reg. contains the upper 16 bits of the starting address of the Code segment

BIU will automatically insert “0” for the lowest 4 bits of the segment base address to get 20 bit physical address

The IP reg contains Offset

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 to be fetched



20-bit Physical Address Generation

Generation of Physical Address

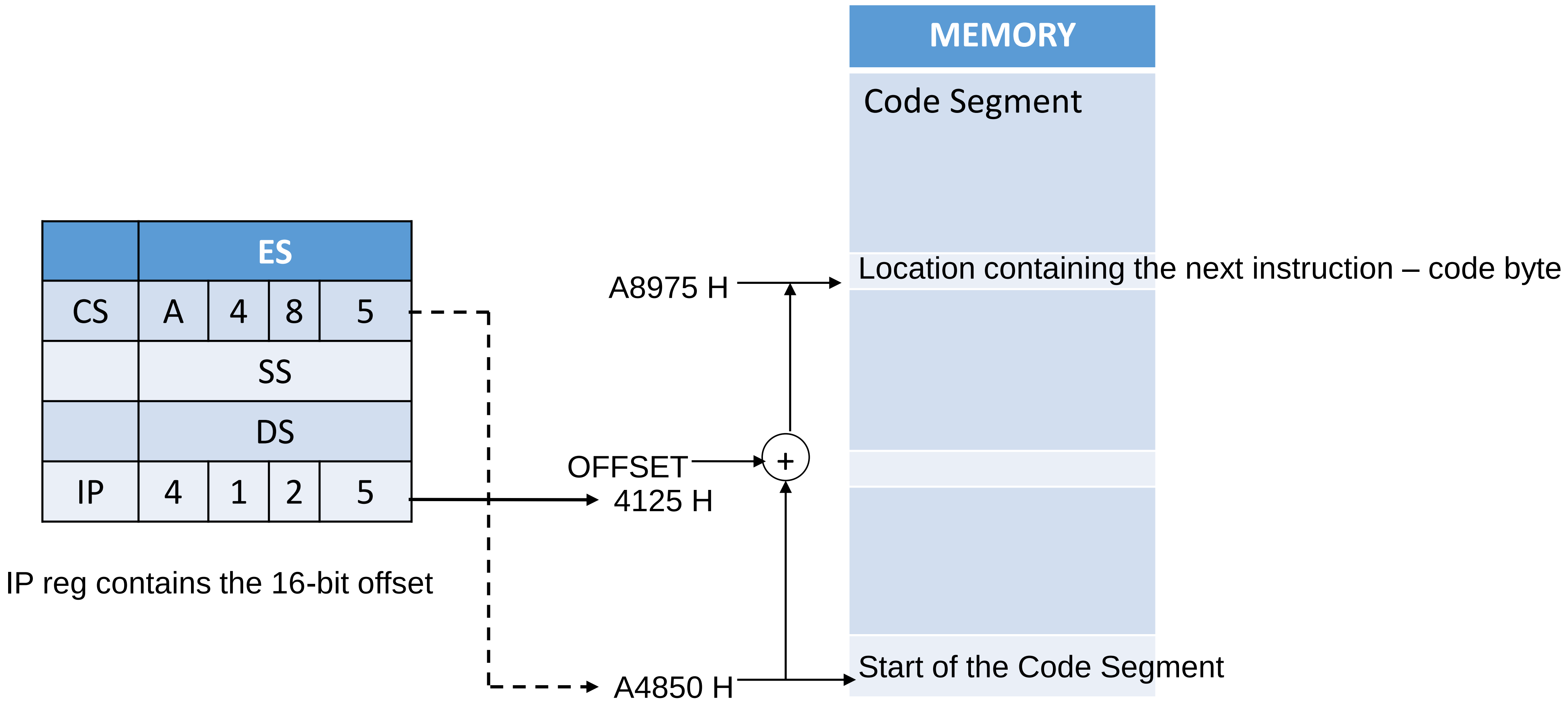
					Hardwired Zero
CS	3	4	8	A	**0**
IP +		4	2	1	4
Physical Address	**3**	**8**	**A**	**B**	**4**

20 bit physical address of the location containing the next code byte to be fetched

Representation of Physical Address

Segment Base : **Offset**

20-bit Physical Address Generation

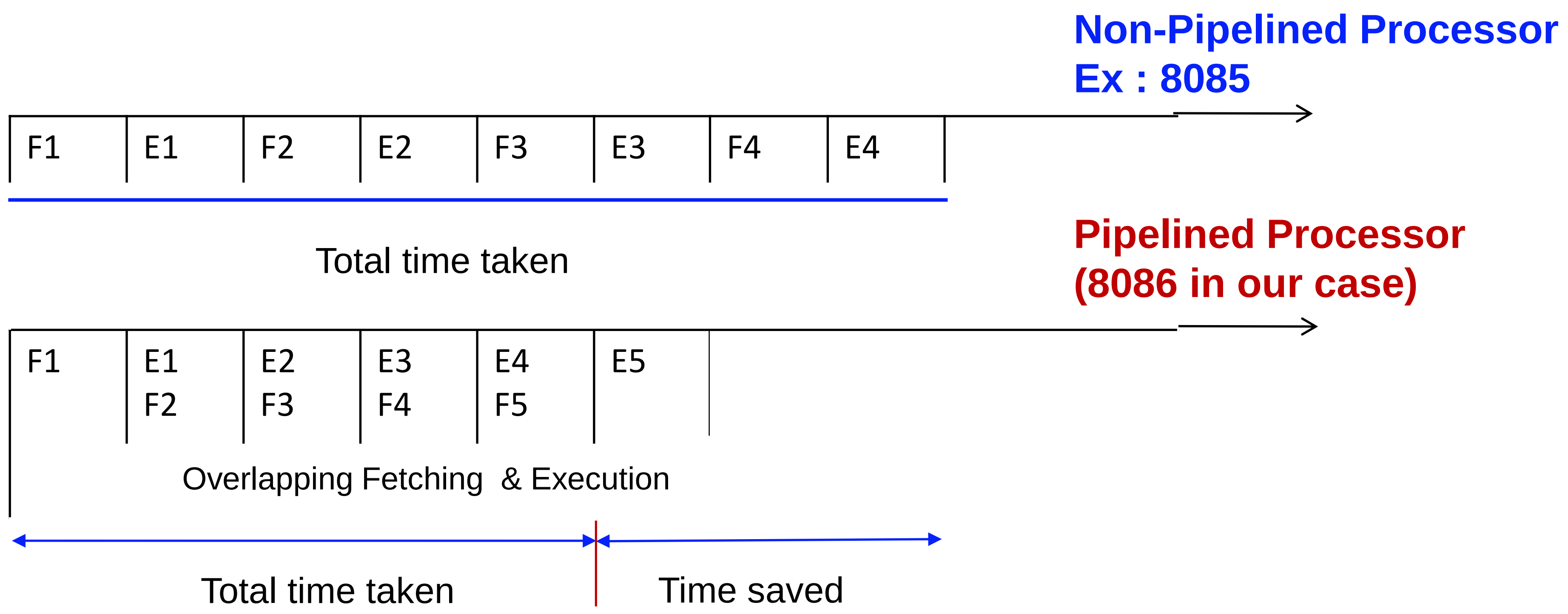


MEMORY
Code Segment
Location containing the next instruction – code byte
Start of the Code Segment
ES
CS
A
4
8
5
SS
DS
IP
4
1
2
5
A8975 H
OFFSET
4125 H
+
A4850 H
IP reg contains the 16-bit offset

BIU Unit - 6 Byte Pre-Fetch Queue

- The queue is 6 Byte First-In-First-Out (**FIFO) RAM** used to implement **pipelining**
- 8086 implements 2-stage pipelining. i.e., it fetches the next byte while executing the current instruction.
- When EU is decoding & executing an instruction, BIU fetches up to 6 instruction bytes from the code segment & stores them in the queue.
- These bytes are called as perfected bytes and are stored in the FIFO manner.
- EU takes instructions from the queue & executes them.
- The Queue is refilled when at least 2 bytes are empty as 8086 has 16-bit data bus.

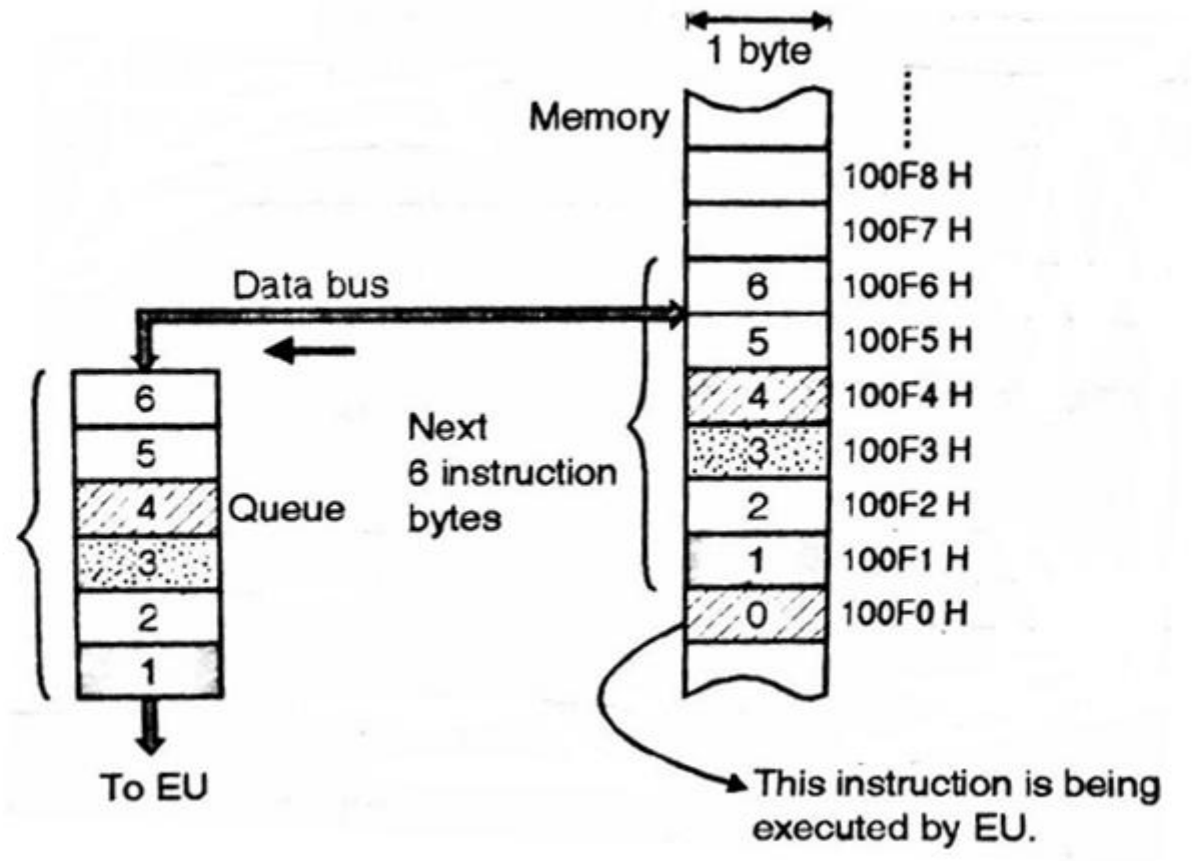
BIU Unit - 6 Byte Pre-Fetch Queue



F – FETCH IN BIU
E – EXECUTE IN EU

8086 - Pipelining

Queue prefetches the next 6 instruction bytes



8086 - Pipelining

Advantages of pipelining:

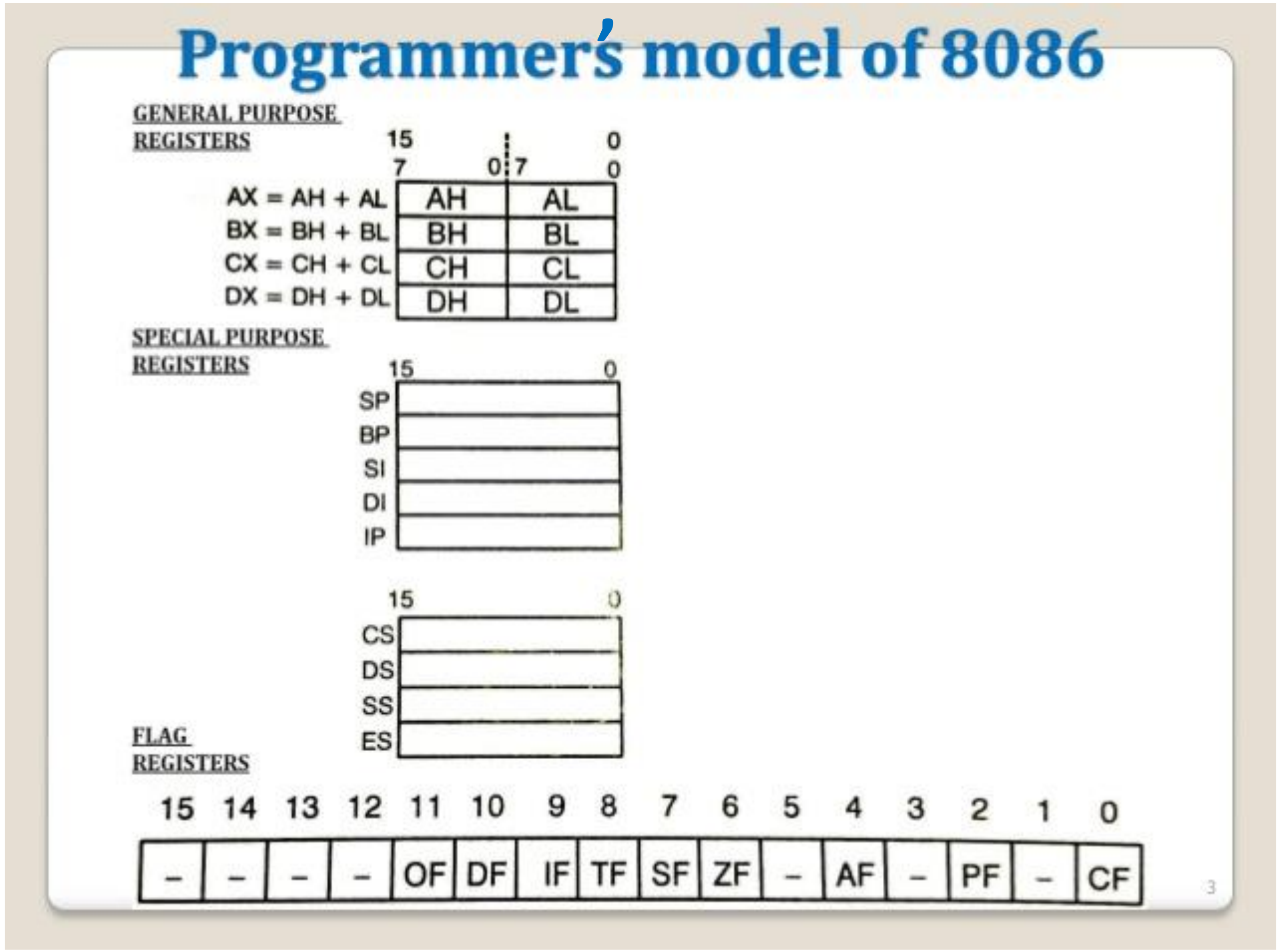
- EU reads the next instruction byte from queue in the BIU instead of sending out an address to the memory & then waiting for the next instruction byte to come in. Clearly, this increases the efficiency of **µp**.
- EU can execute instructions almost continually instead of having to wait for the BIU to fetch a new instruction. No separate time is consumed for fetching. It happens alongside execution.

8086 – Pipelining Drawbacks

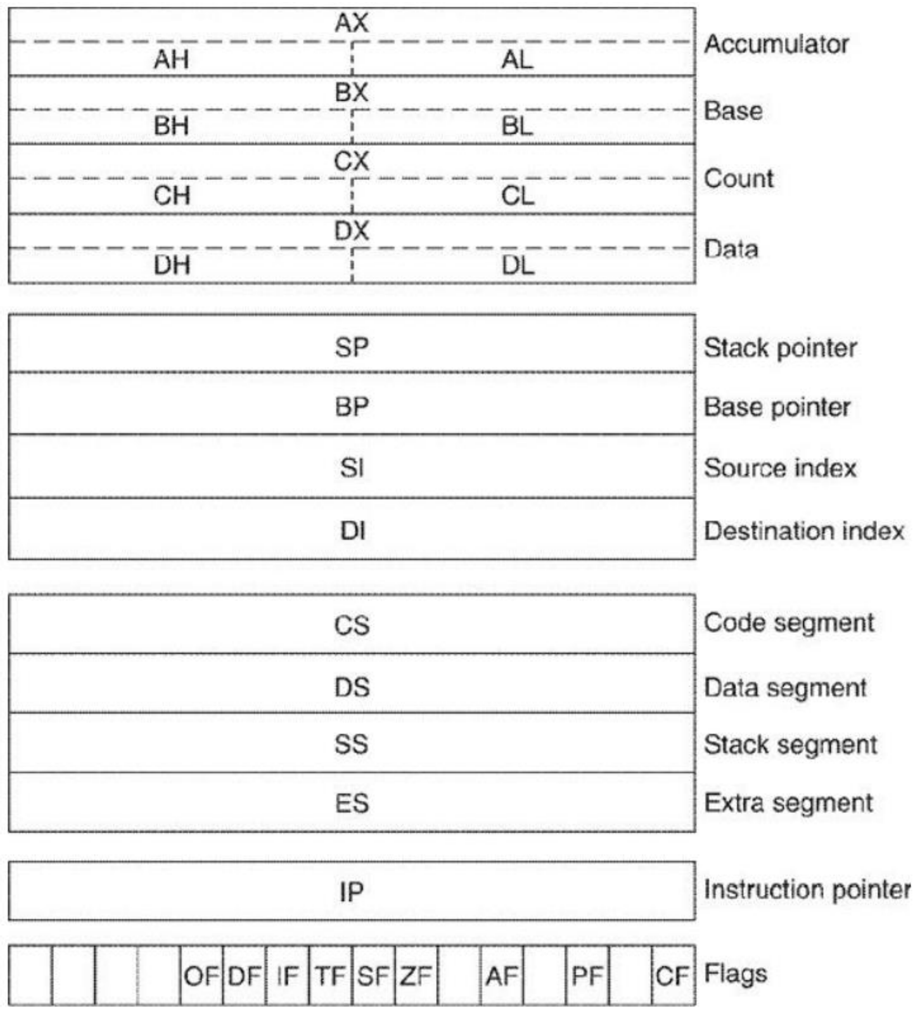
Exceptions when the queue must be dumped:

- There are conditions that will cause the EU to enter a "wait" mode.
- When JMP or CALL instruction appears in the main program :
  1. Queue is flushed out and is made empty
  2. It is then reloaded with the six instruction bytes from the new locations starting with the JMP or CALL address.

8086 Programmer’s Model



8086 Programmer's Model



8086 Programmer’s Model

- Programmer’s model is the information needed to write programs.
- It consists of some segments of the ALU & the registers.
- This model does not reflect the physical structure of the 8086 but includes information that is critical in writing assembly language programs.

8086 Programmer’s Model

- The programmer’s model includes:
  - All General Purpose Registers (GPRs) : AX, BX, CX, DX
  - Segment Registers : CS, SS, DS, ES
  - All Offset Registers : IP, SP, BP, SI, DI
  - The Flag Register

All these have been explained during the 8086 Architecture discussion.

END