8086 Microprocessor

Dr. Manju Khurana
Assistant Professor, CSED
TIET, Patiala
manju.khurana@thapar.edu

Architecture

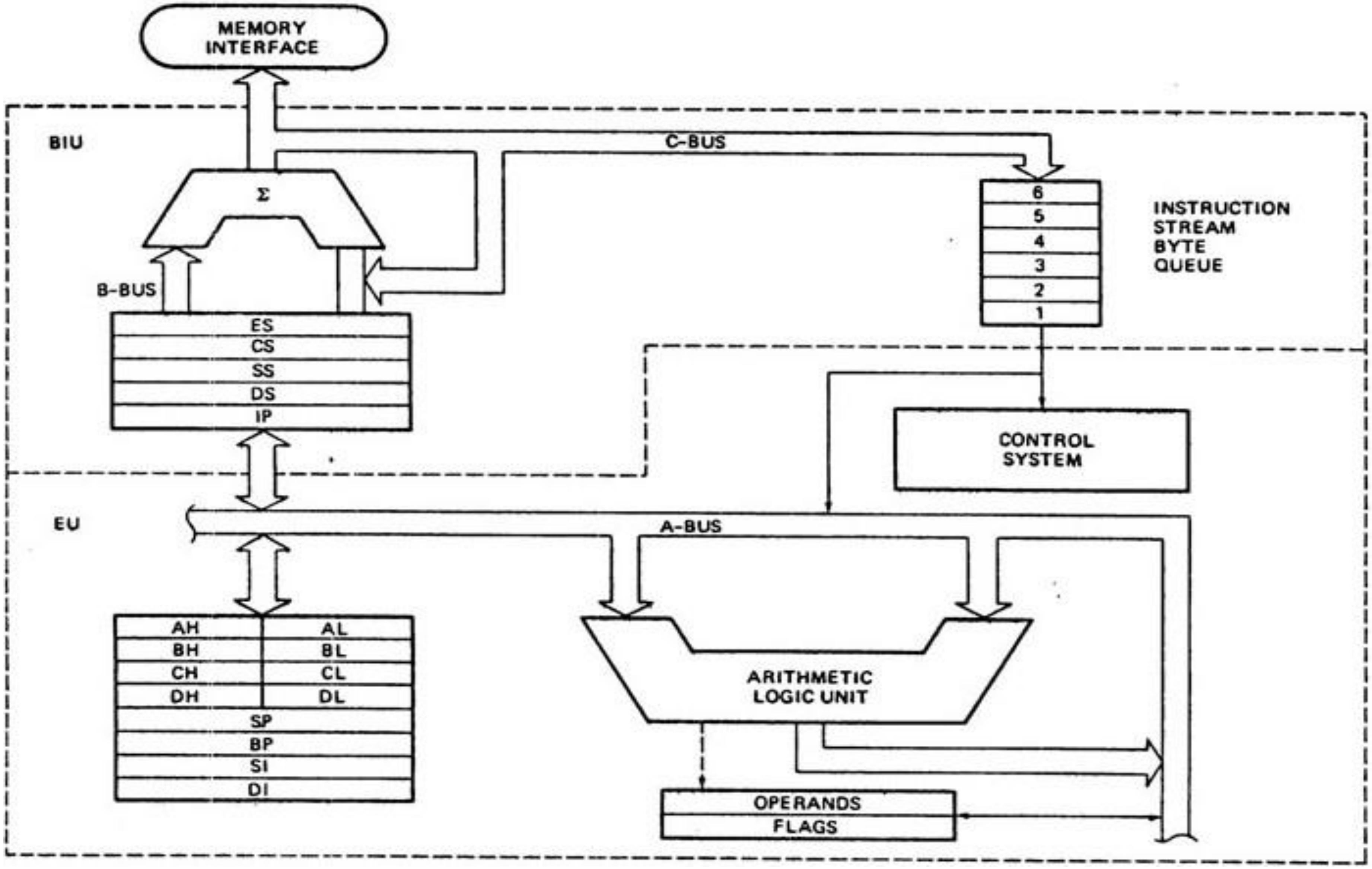
BIU is responsible for:-

- It handles transfer of data and addresses between the processor and memory/IO devices.
- It computes and sends out addresses, Fetches instruction codes, stores fetched instruction codes in a first in first out register called a Queue, Reads data from memory and I/O devices, Writes data to memory and I/O devices.

EU is responsible for:-

- EU receives opcode of an instruction from the queue, decodes it and then executes it.
- While EU is decoding an instruction or executing an instruction, the BIU fetches instruction codes from the memory and stores them in the queue.
- The BIU and EU operate in parallel independently.
- This type of overlapped operation of the functional units of a microprocessor is called pipelining.

Architecture



Segment Registers

- There are four segment registers in Intel 8086:
  - ⦿ Code Segment Register (CS)
  - ⦿ Data Segment Register (DS)
  - ⦿ Stack Segment Register (SS)
  - ⦿ Extra Segment Register (ES)

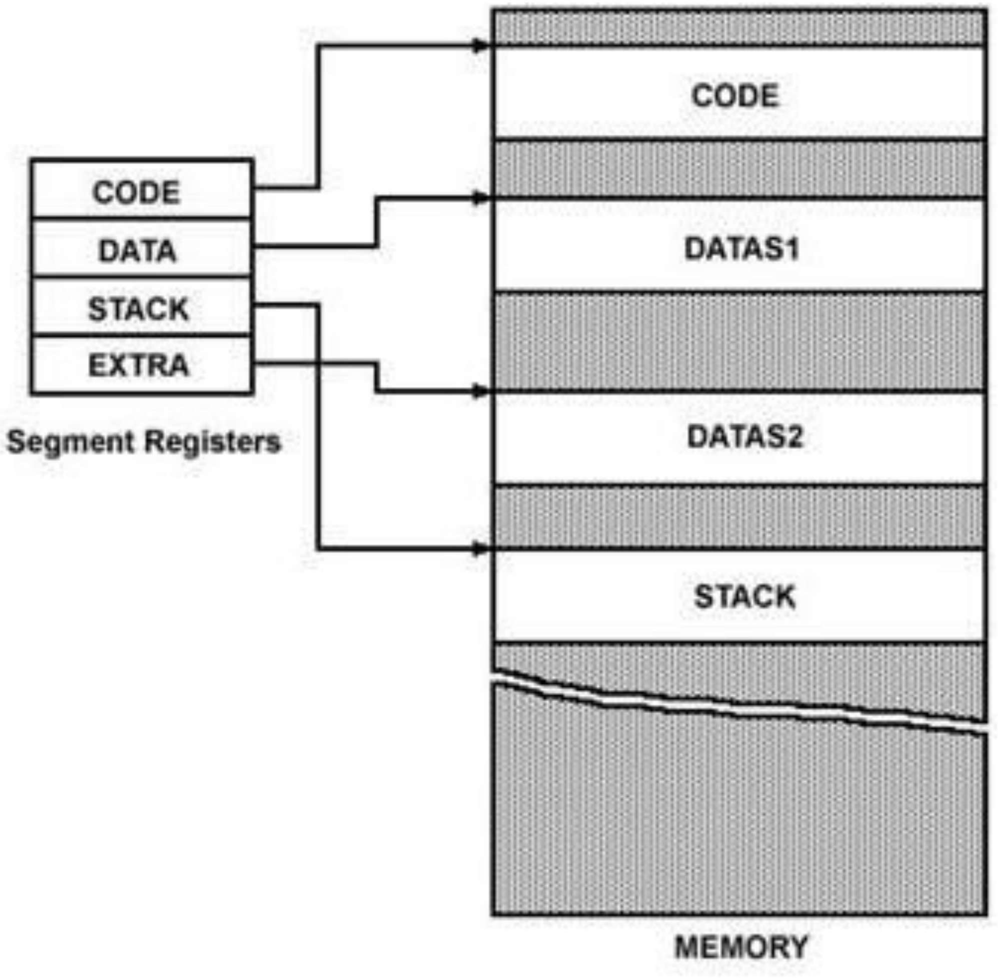
Segment Registers

- In 8086, memory is divided into four segments:
  - ⦿ Code Segment
  - ⦿ Data Segment
  - ⦿ Stack Segment
  - ⦿ Extra Segment

Segment Registers

- A segment register points to the starting address of a memory segment.
- For e.g.:
  - The code segment register points to the starting address of the code segment.
  - The data segment register points to the starting address of the data segment, and so on.
- The maximum capacity of a segment may be up to 64 KB.

Segment Registers



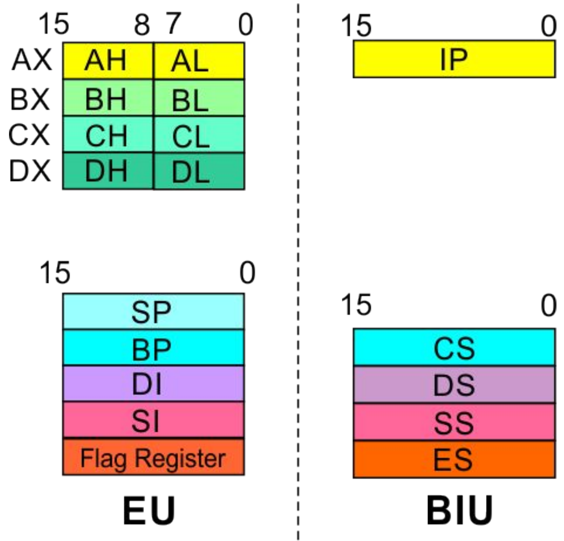
Segment Registers

- The instructions of 8086 specify 16-bit address.
- But the actual physical addresses are of 20-bit.
- Therefore, they are calculated using the contents of the segment registers and the effective memory address.

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.**



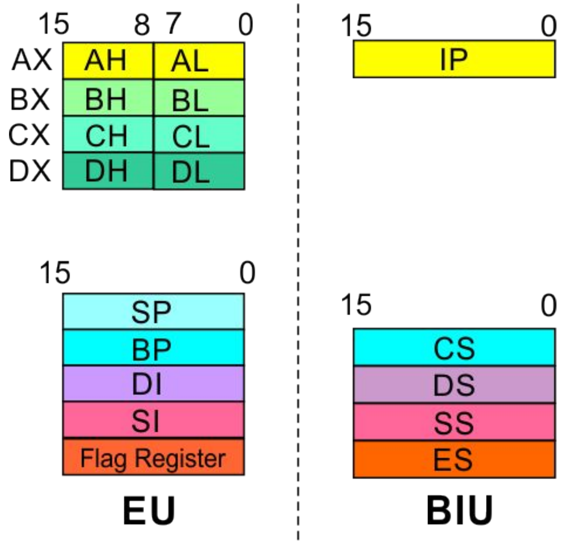
Architecture

Bus Interface Unit (BIU)

Segment Registers

Data Segment Register

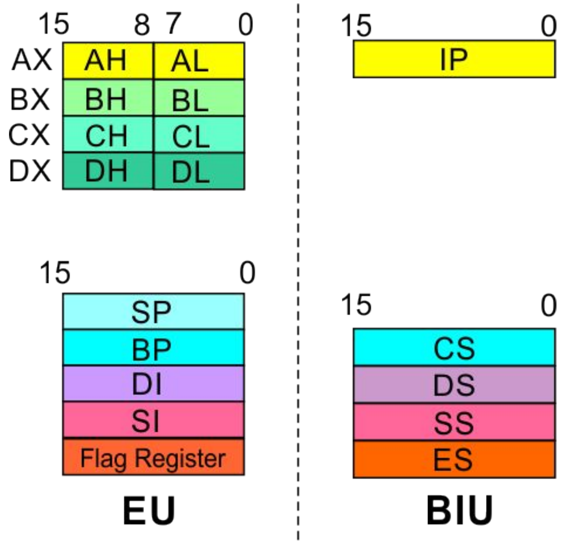
- **16-bit**
- **Hold data, variables and constants of program.**
- **Points to the current data segment; operands for most instructions are fetched from this segment.**
- **The 16-bit contents of the Source Index (SI) or Destination Index (DI) or a 16-bit displacement are used as offset for computing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



Segment Registers

Stack Segment Register

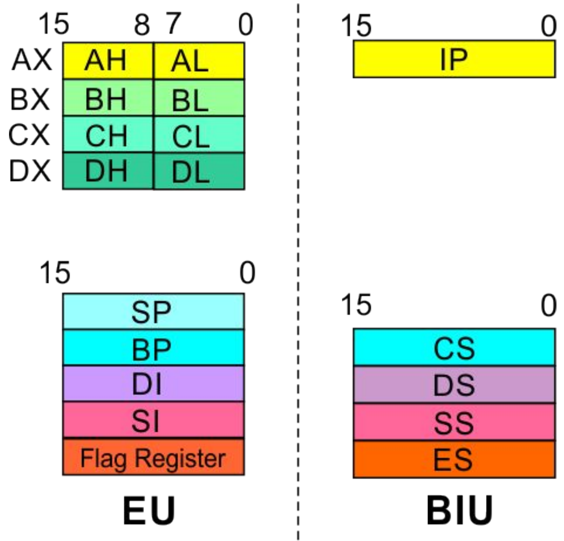
- **16-bit**
- **Points to the current stack.**
- **The 20-bit physical stack address is calculated from the Stack Segment (SS) and the Stack Pointer (SP) for stack instructions such as PUSH and POP.**
- **In based addressing mode, the 20-bit physical stack address is calculated from the Stack segment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



Segment Registers

Extra Segment Register

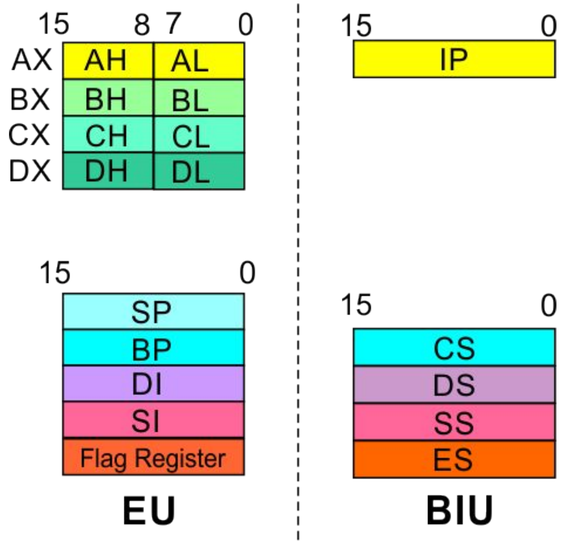
- **16-bit**
- **Points to the extra segment in which data (in excess of 64K pointed to by the DS) is stored.**
- **String instructions use the ES and DI to determine the 20-bit physical address for the destination.**



Segment Registers

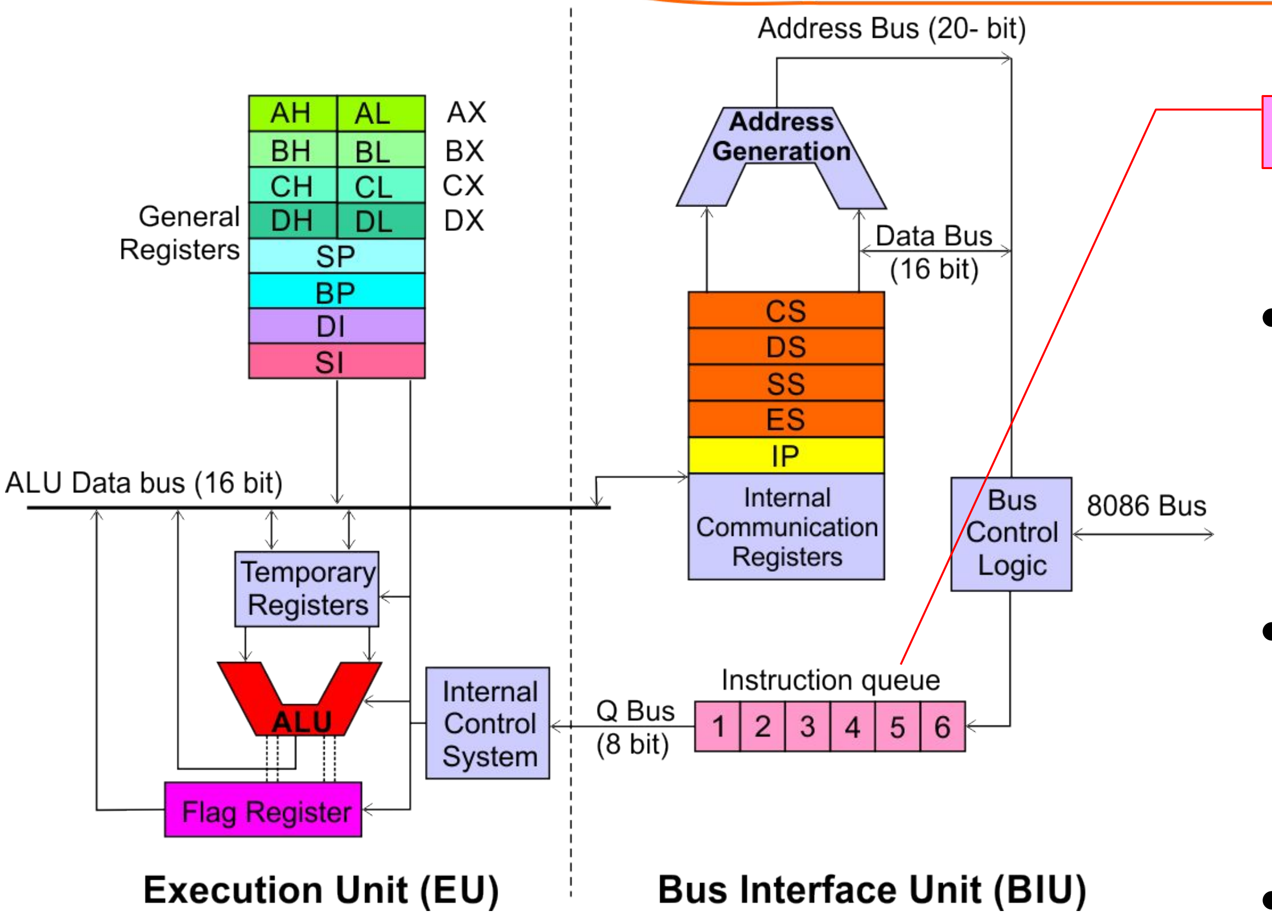
Instruction Pointer

- **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.**
- The contents of the IP and Code Segment Register are used to compute the memory address of the instruction code to be fetched.
- This is done during the Fetch Cycle.



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 <u>pipelining</u>.**

Architecture

Execution Unit (EU)

EU decodes and executes instructions.

A decoder in the EU control system translates instructions.

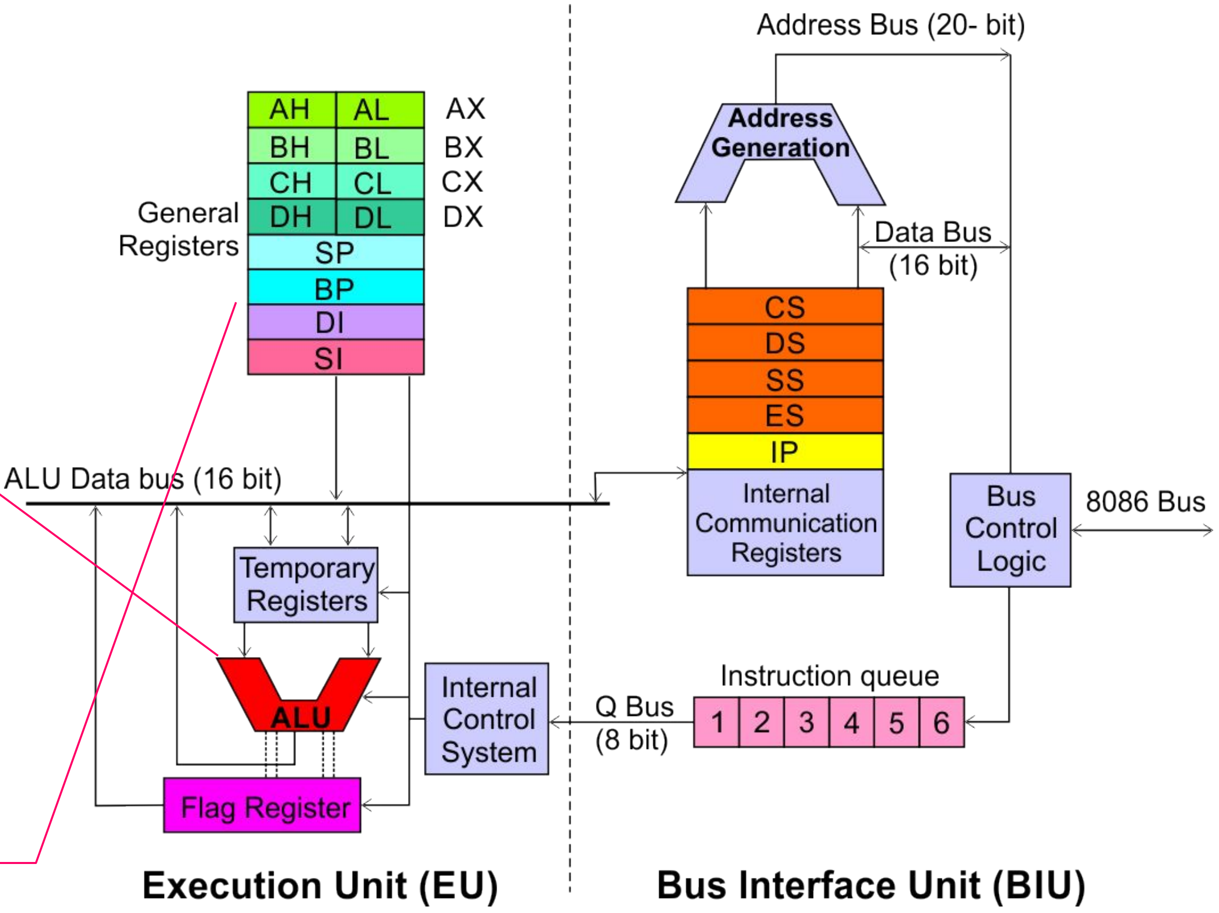
16-bit ALU for performing arithmetic and logic operation

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



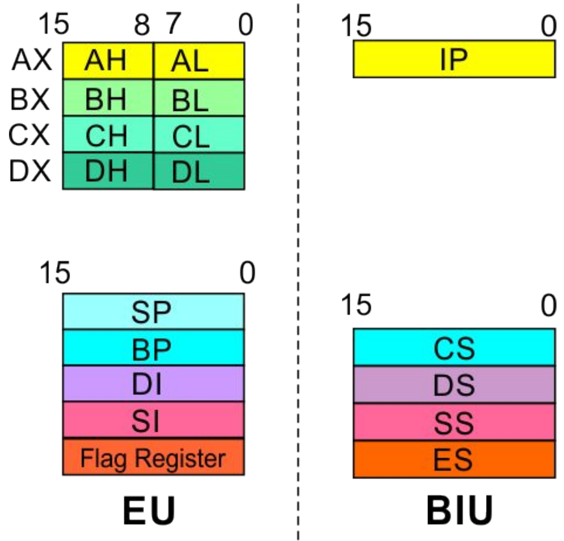
Some of the 16 bit registers can be used as two 8 bit registers as :

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

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.**



Architecture

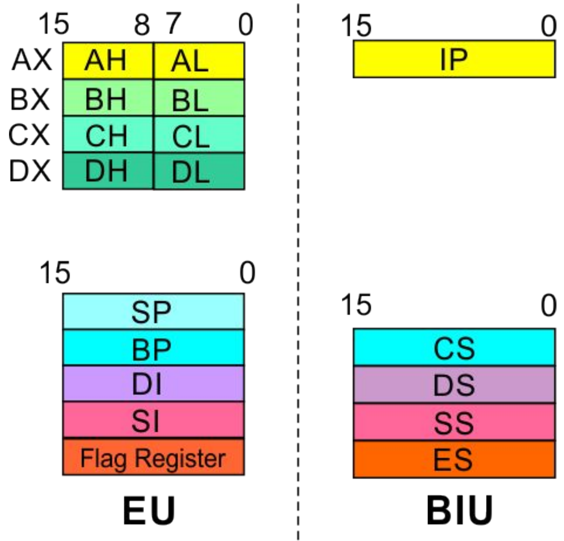
Execution Unit (EU)

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.**
- **As a special purpose reg.,BX serve as a base register for the computation of mem. Address.**

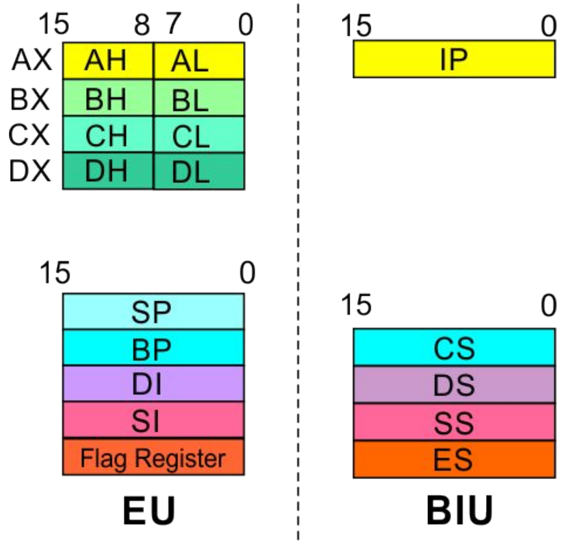
All memory references utilizing this register content for addressing use DS as the default segment register.



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.
- As a special purpose, used as a counter in case of multi-iteration instruction.
- Instructions such as SHIFT, ROTATE and LOOP use the contents of CX as a counter.



Example:

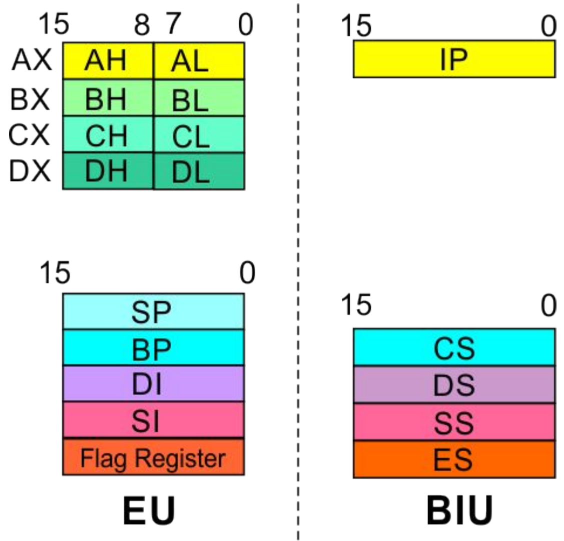
The instruction LOOP START automatically decrements CX by 1 without affecting flags and will check if [CX] = 0.

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

EU Registers

Data Register (DX)

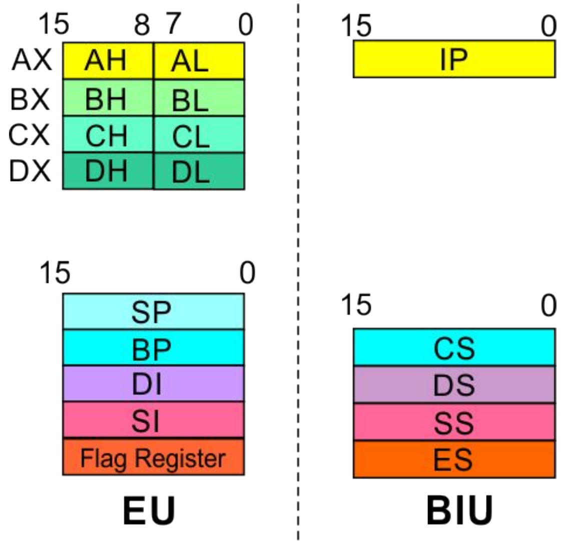
- **Consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX.**
- **When combined, DL register contains the low order byte of the word, and DH contains the high-order byte.**
- **Used to hold the high 16-bit result (data) in 16 X 16 multiplication or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

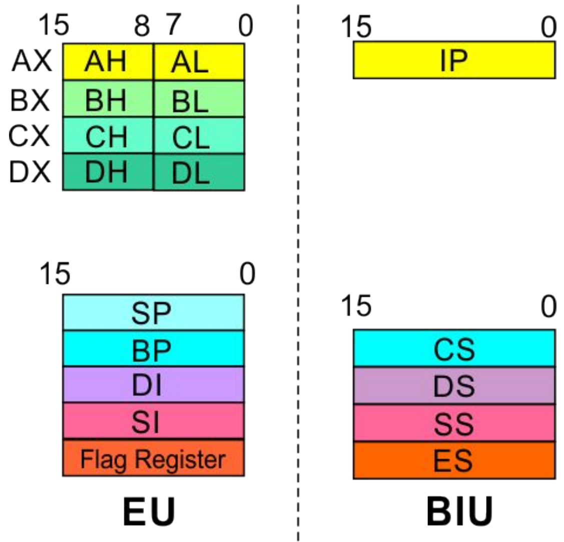
- **SP and BP are used to access data in the stack segment.**
- **SP is used as an offset from the current SS during execution of instructions that involve the stack segment in the external memory.**
- **SP contents are automatically updated (incremented/ decremented) due to execution of a POP or PUSH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



EU Registers

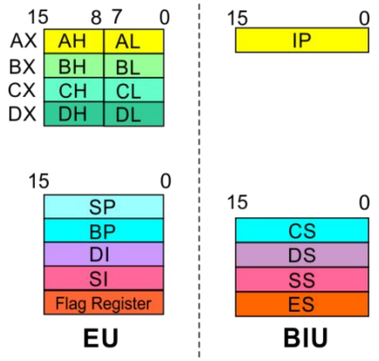
Source Index (SI) and Destination Index (DI)

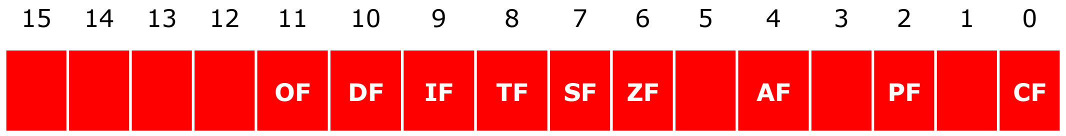
- Used in indexed addressing.
- Instructions that process data strings use the SI and DI registers together with DS and ES respectively in order to distinguish between the source and destination addresses.



Architecture

8086 registers categorized into 4 groups





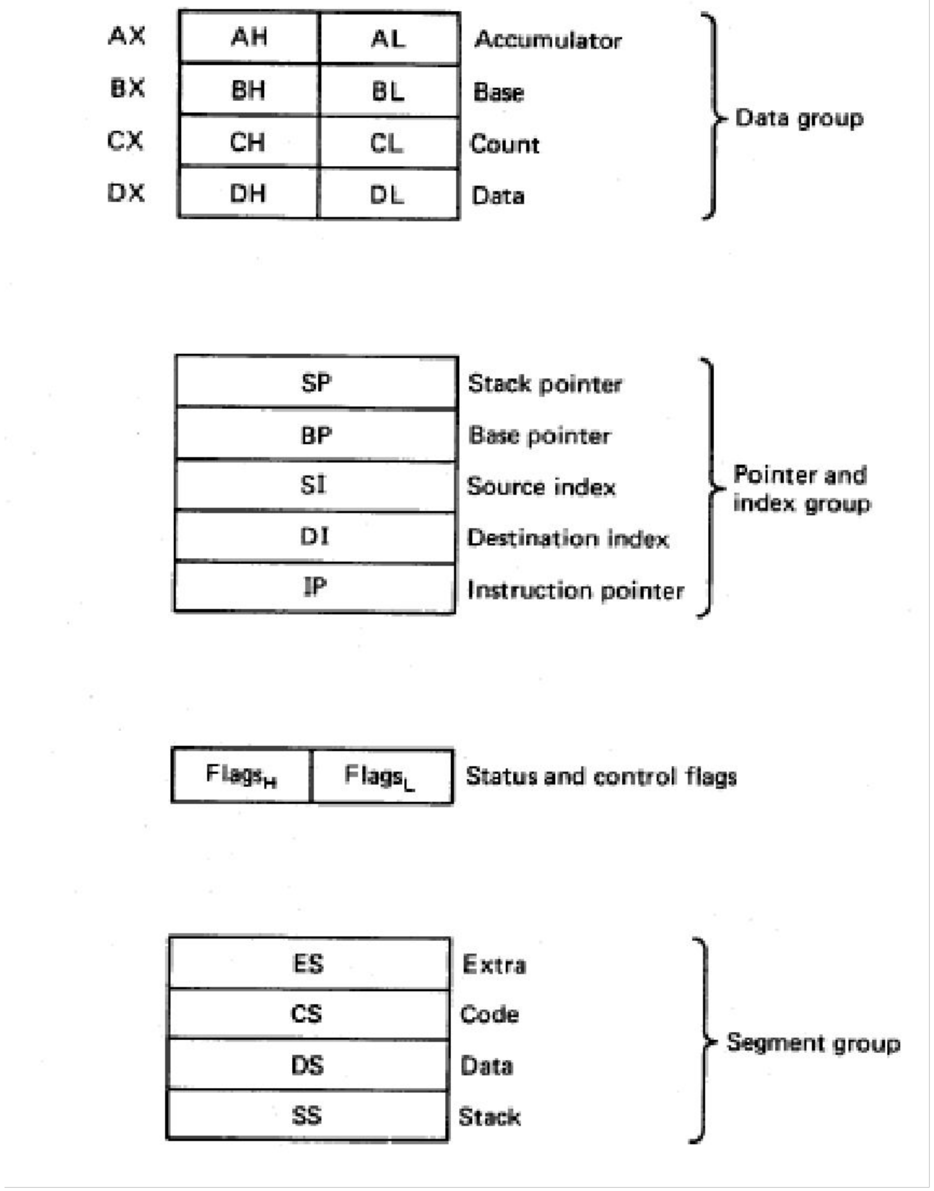
Sl.No.	Type	Register width	Name of register
1	General purpose register	16 bit	AX, BX, CX, DX
		8 bit	AL, AH, BL, BH, CL, CH, DL, DH
2	Pointer register	16 bit	SP, BP
3	Index register	16 bit	SI, DI
4	Instruction Pointer	16 bit	IP
5	Segment register	16 bit	CS, DS, SS, ES
6	Flag (PSW)	16 bit	Flag register

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

8086/88 internal registers 16 bits (2 bytes each)



AX, BX, CX and DX are two bytes wide and each byte can be accessed separately

These registers are used as memory *pointers*.

Flags will be discussed later

Segment registers are used as base address for a segment in the 1 M byte of memory

Status Flags

- 8086 has 9 flags and they are divided into two categories:

  - ⦿ Condition Flags
  - ⦿ Control Flags

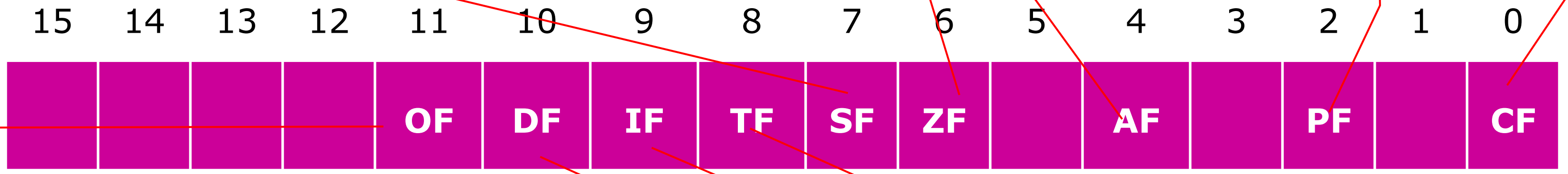
Status Flags

- Following are the nine flags:

Condition Flags	Control Flags
1. Carry Flag	1. Trap Flag
2. Auxiliary Carry Flag	2. Interrupt Flag
3. Zero Flag	3. Directional Flag
4. Sign Flag	
5. Parity Flag	
6. Overflow Flag	

Architecture(Flags) 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.

Trap 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.

Control Flags

- Control flags are set or reset deliberately to control the operations of the execution unit. Control flags are as follows:

- **Trap Flag (TP):**
  - It is used for single step control.
  - It allows user to execute one instruction of a program at a time for debugging.
  - When trap flag is set, program can be run in single step mode.

Control Flags

- **Interrupt Flag (IF):**
  - ⦿ It is an interrupt enable / disable flag.
  - ⦿ If it is set, the INTR interrupt of 8086 is enabled and if it is reset then INTR is disabled.
  - ⦿ It can be set by executing instruction STI and can be cleared by executing CLI instruction.

Control Flags

- **Directional Flag (DF):**
  - ⦿ It is used in string operation.
  - ⦿ If it is set, string bytes are accessed from higher memory address to lower memory address.
  - ⦿ When it is reset, the string bytes are accessed from lower memory address to higher memory address.
  - ⦿ It is set with STD instruction and cleared with CLD instruction.

Segment:Offset Address

- Logical Address is specified as segment:offset
- Physical address is obtained by shifting the segment address 4 bits to the left and adding the offset address
- Thus the physical address of the logical address A4FB:4872 is

$$\begin{array}{r} \text{A4FB0} \\ +\ \underline{\ 4872} \\ \text{A9822} \end{array}$$

Example

- If DS=7FA2H and the offset is 438EH

  a) Calculate the physical address

  7FA20 + 438E = 83DAE

  b) calculate the lower range

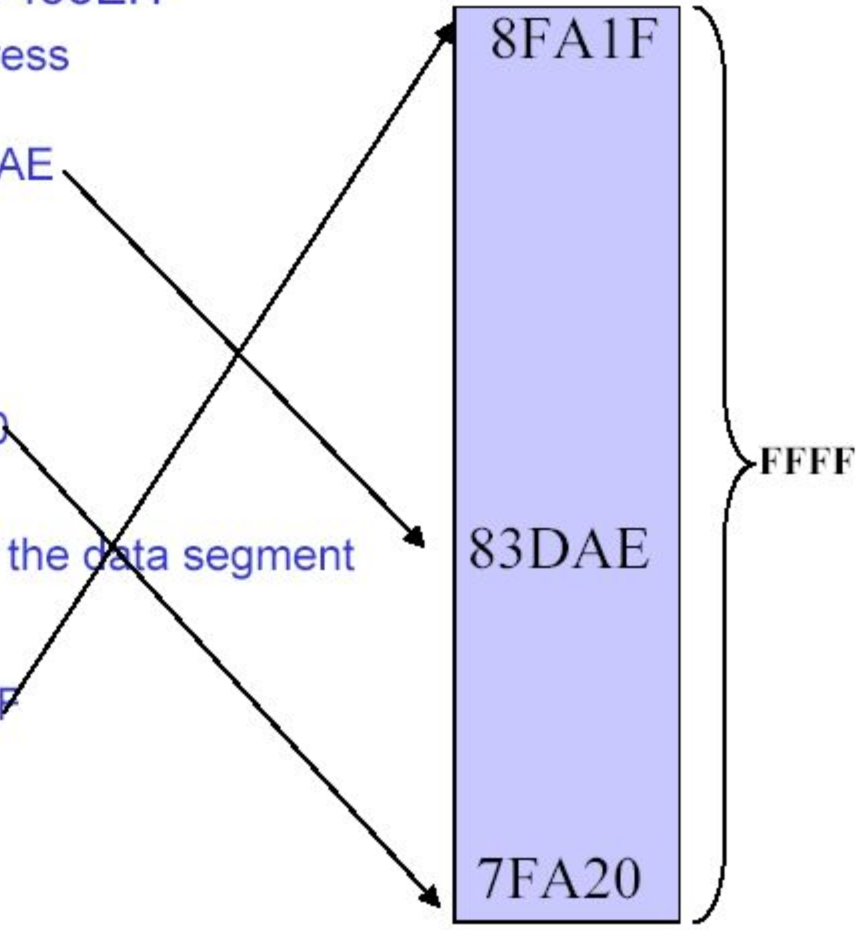
  7FA20 + 0000 = 7FA20

  c) Calculate the upper range of the data segment

  7FA20 + FFFF = 8FA1F

  d) Show the logical Address

  7FA2:438E



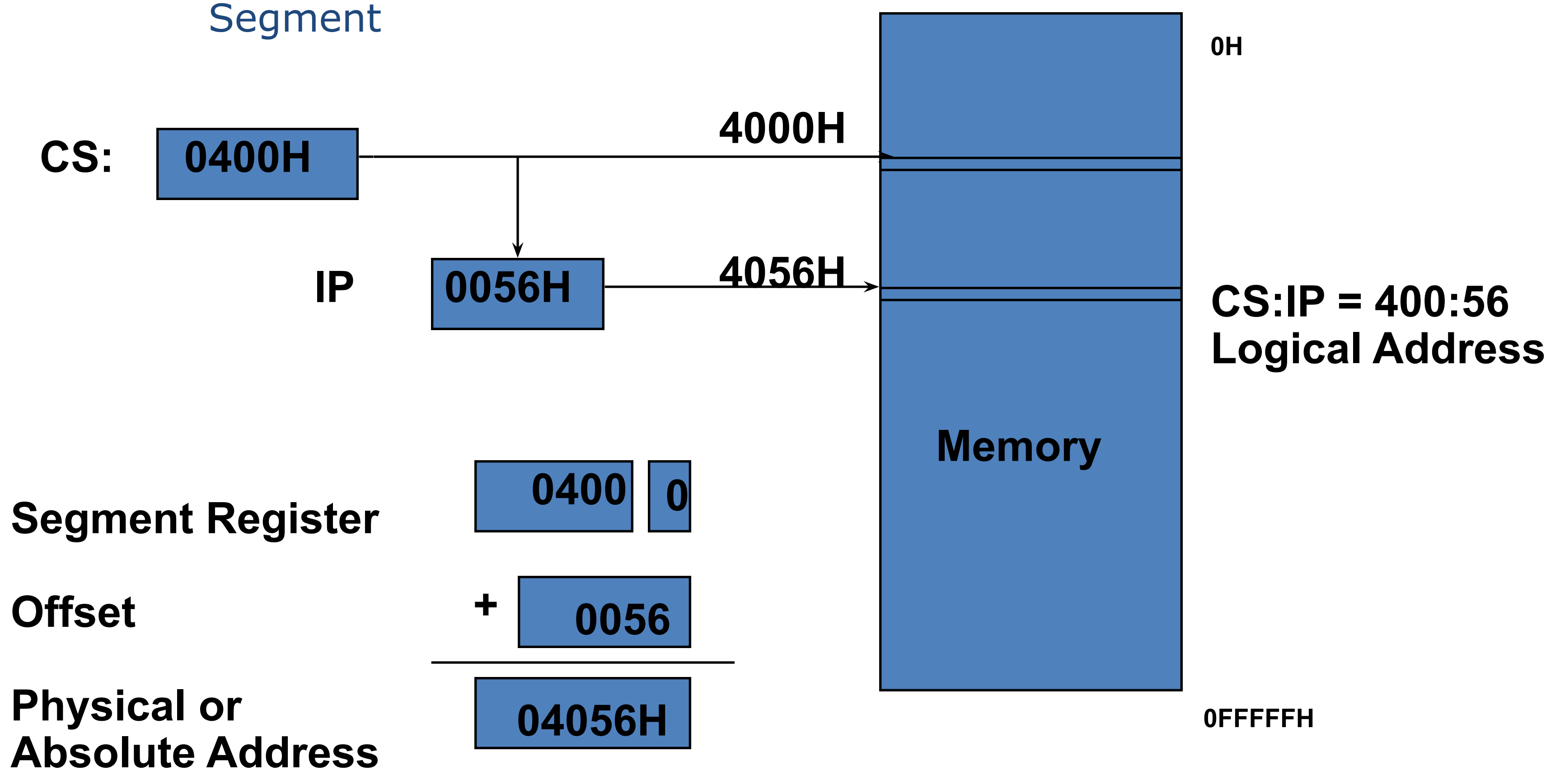
Your turn . . .

What linear address corresponds to the segment/offset address 028F:0030?

028F0 + 0030 = 02920

Always use hexadecimal notation for addresses.

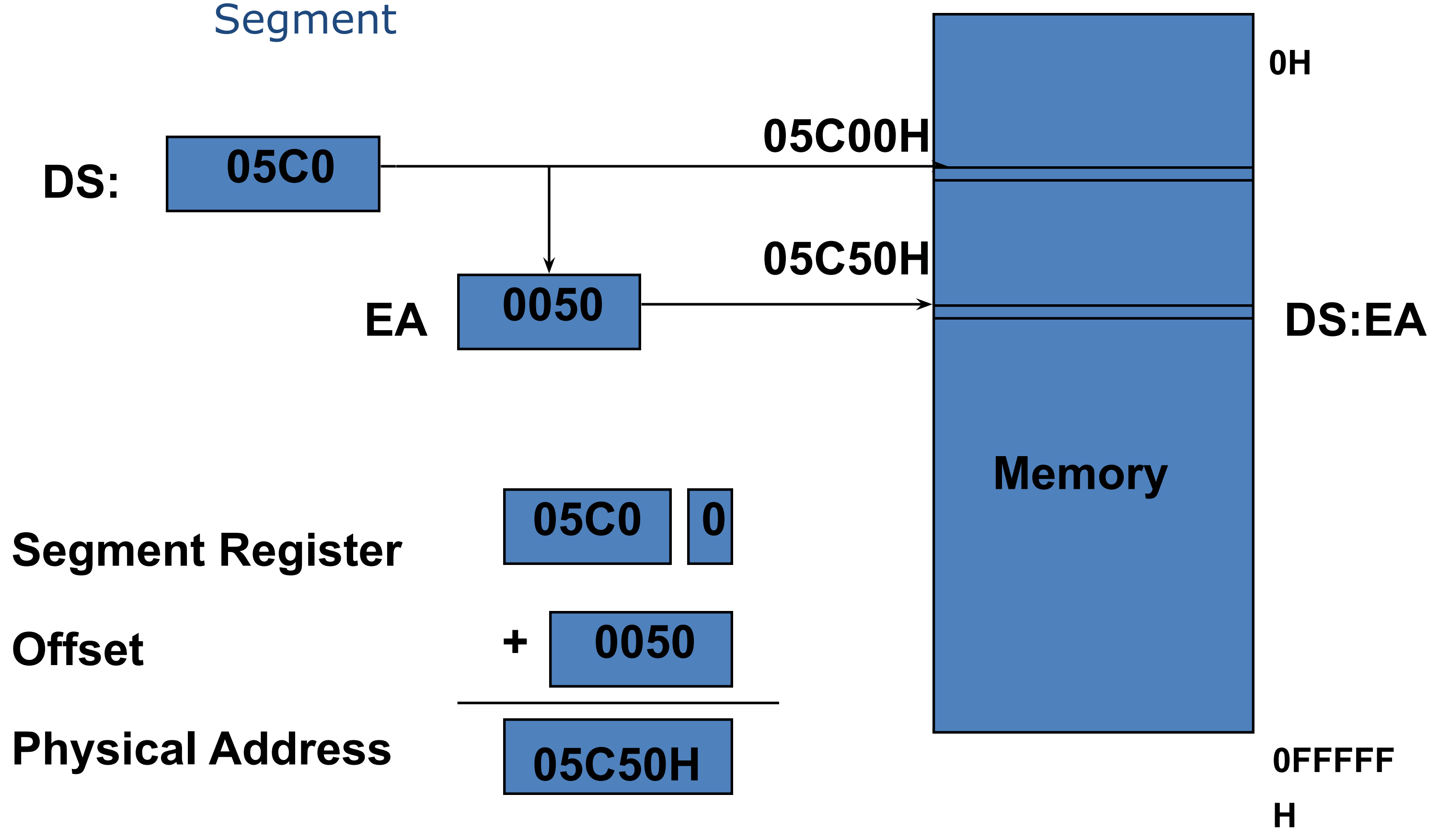
The Code Segment



The offset is the distance in bytes from the start of the segment.
The offset is given by the IP for the Code Segment.
Instructions are always fetched with using the CS register.

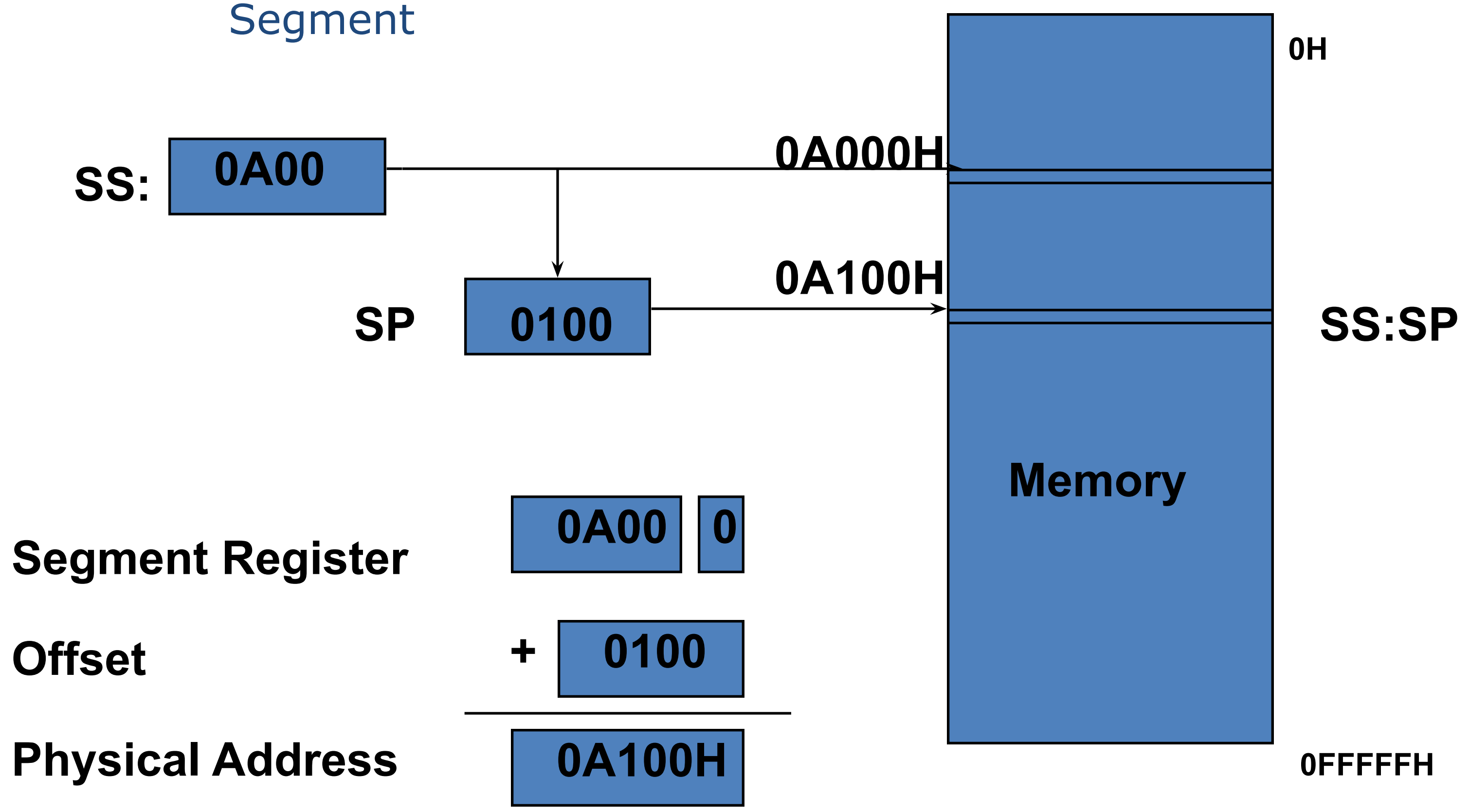
The ~~physical address~~ is also called the ~~absolute address~~.

The Data Segment



Data is usually fetched with respect to the DS register.
The effective address (EA) is the offset.
The EA depends on the addressing mode.

The Stack Segment



The offset is given by the SP register.
The stack is always referenced with respect to the stack segment register.
The stack grows toward decreasing memory locations.
The SP points to the last or top item on the stack.

PUSH - pre-decrement the SP
POP - post-increment the SP

Thank you!