Lecture 11:
Introduction to 8086 Microprocessor

Features of 8086

- 8086 is a 16bit processor. It's ALU, internal registers works with 16bit binary word
- 8086 has a 16bit data bus. It can read or write data to a memory/port either 16bits or 8 bit at a time
- 8086 has a 20bit address bus which means, it can address upto $2^{20} = 1MB$ memory location
- Frequency range of 8086 is 6-10 MHz

<u>**Data Read/Write process from /To Memory**</u>

<u>**Word Read**</u>

- Each of 1 MB memory address of 8086 represents a byte wide location
- 16bit words will be stored in two consecutive Memory location

- If first byte of the data is stored at an <u>**even address**</u> **,** 8086 can read the entire word in one operation.
    - For example if the 16 bit data is stored at even address 00520_H is <u>2607</u>

      MOV BX, [00520]
      8086 reads the first byte and stores the data in BL and reads the 2^{nd} byte and stores the data in BH

      BL← (00520)
      BH← (00521)

- If the first byte of the data is stored at an ODD address, 8086 needs two operation to read the 16 bit data
    - For example if the 16 bit data is stored at even address 00521_H is <u>F520</u>

      MOV BX, [00521]
      In first operation , 8086 reads the 16 bit data from the 00520 location and stores the data of 00521 location in register BL and discards the data of 00520 location

      In 2^{nd} operation, 8086 reads the 16 bit data from the 00522 location and stores the data of 00522 location in register BH and discards the data of 00523 location

      BL$\leftarrow$ (00521)
      BH$\leftarrow$ (00522)

Byte Read:

MOV BH, [Addr]

For Even Address:

Ex: MOV BH, [00520]

8086 reads the first byte from 00520 location and stores the data in BH and reads the 2nd byte from the 00521 location and ignores it

BH ← [00520]

For Odd Address

MOV BH, [Addr]

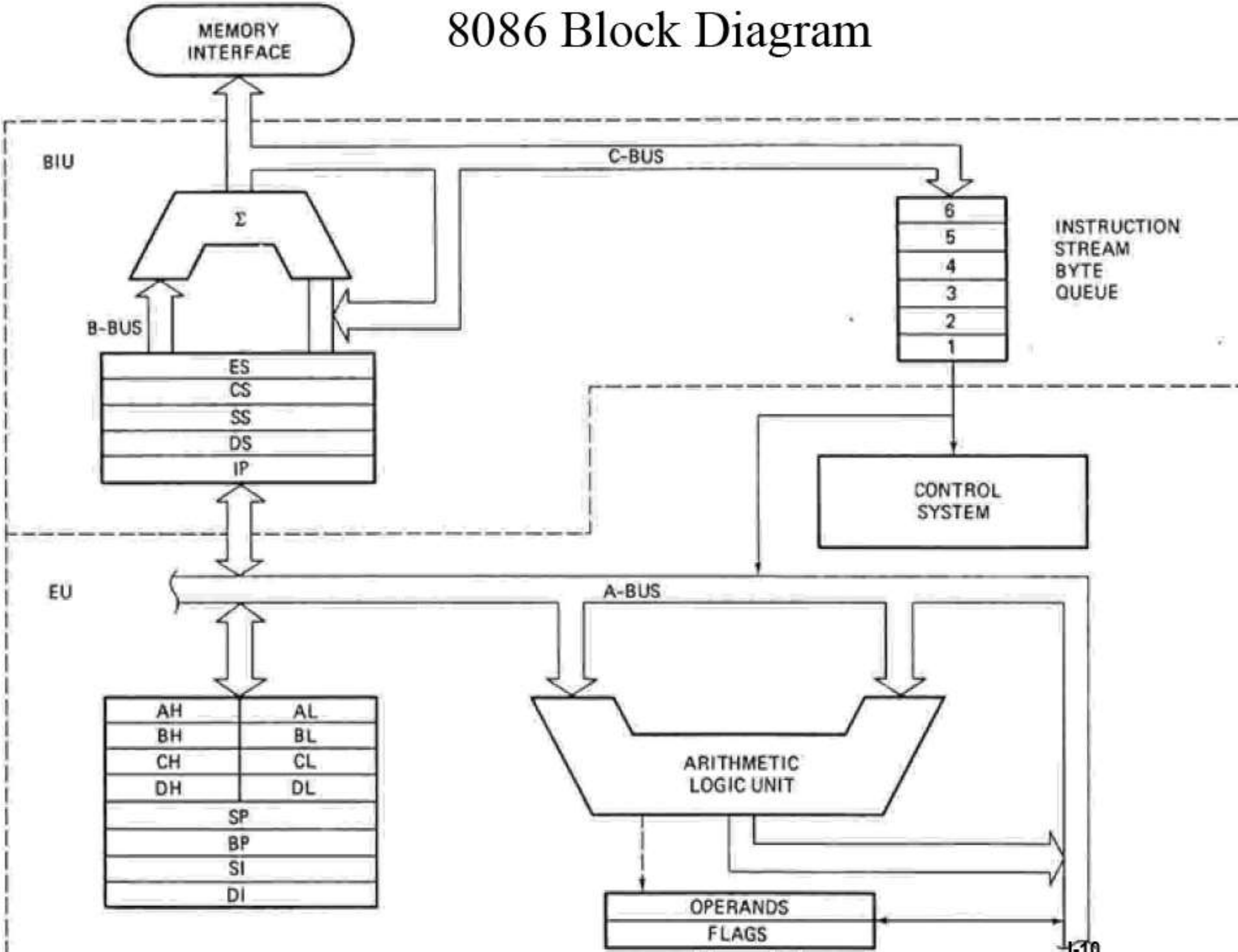
Ex: MOV BH, [00521]

8086 reads the first byte from 00520 location and ignores it and reads the 2nd byte from the 00521 location and stores the data in BH

BH ← [00521]

Architecture of 8086

8086 Block Diagram



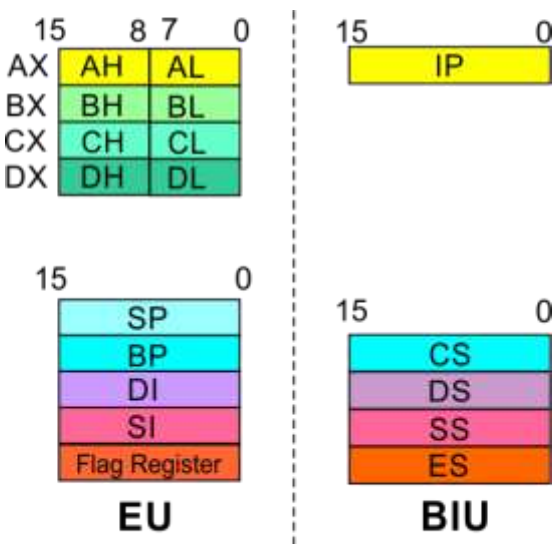
8086 has two parts:

- BUI (Bus Interface Unit)
  - BIU fetches instructions,
  - Reads data from memory and I/O ports,
  - Writes data to memory and I/ O ports,
  - Computes the 20-bit address.
- EU (Execution Unit)
  - EU executes instructions that have already been fetched by the BIU.
  - BIU and EU functions separately.

Segment Registers

Code Segment Register

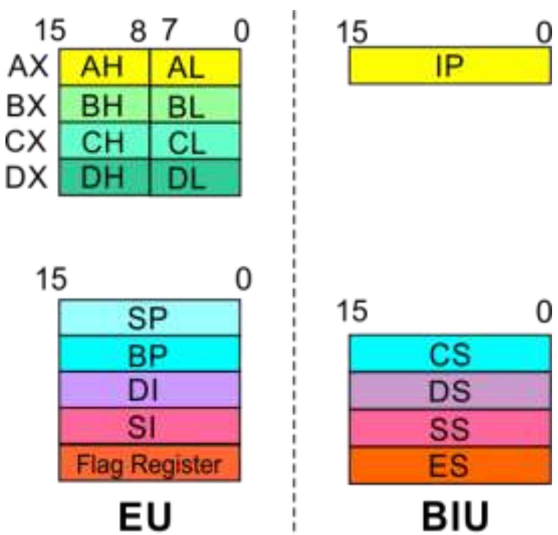
- 16-bit register containing address of 64KB segment.
- CS contains the base or start of the current code segment; IP contains the distance or offset from this address to the next instruction byte to be fetched.
- BIU computes the 20-bit physical address by logically shifting the contents of CS 4-bits 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

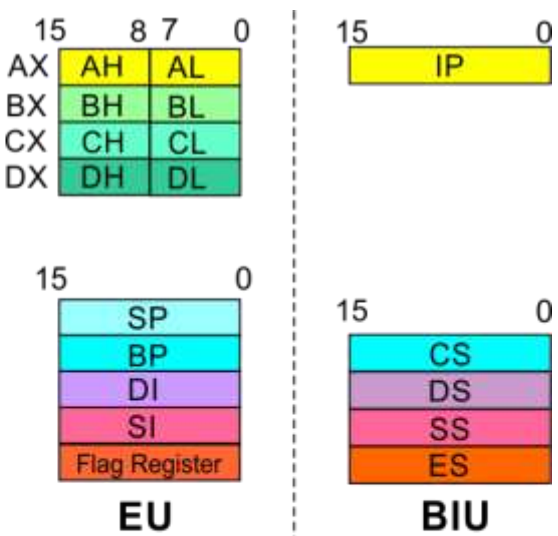
- 16-bit register containing address of 64KB segment.
- Points to the current data segment; operands for most instructions are fetched from this segment.
- The 16-bit contents of the Source Index (SI) or Destination Index (DI) or a 16-bit displacement are used as offset for computing the 20-bit physical address.



Segment Registers

Stack Segment Register

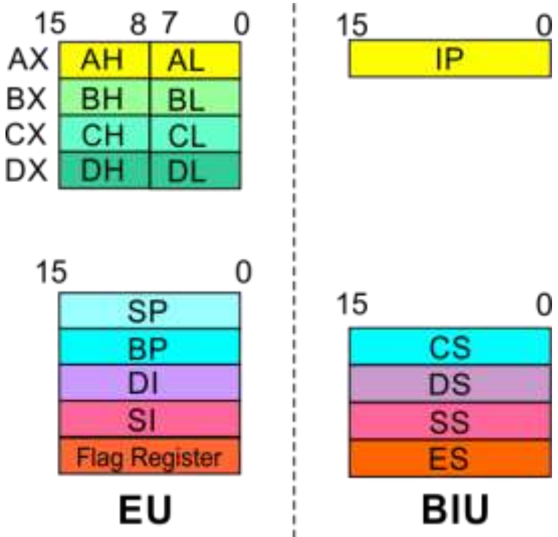
- 16-bit register containing address of 64KB segment.
- Points to the current stack.
- The 20-bit physical stack address is calculated from the Stack Segment (SS) and the Stack Pointer (SP) for stack instructions such as PUSH and POP.
- In based addressing mode, the 20-bit physical stack address is calculated from the Stack segment (SS) and the Base Pointer (BP).



Segment Registers

Extra Segment Register

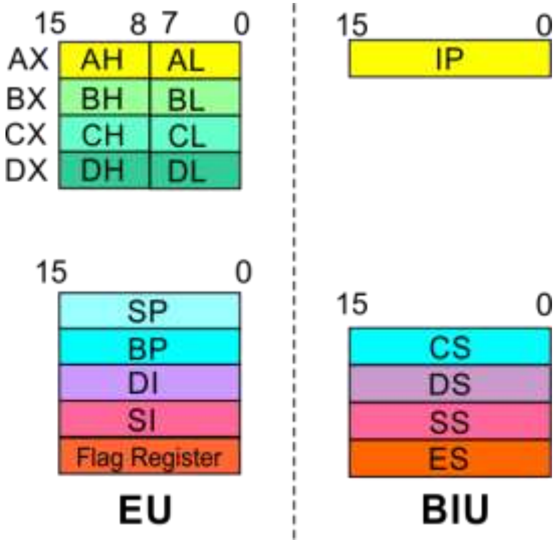
- 16-bit register containing address of 64KB segment.
- Points to the extra segment in which data (in excess of 64K pointed to by the DS) is stored.
- String instructions use the ES and DI to determine the 20-bit physical address for the destination.



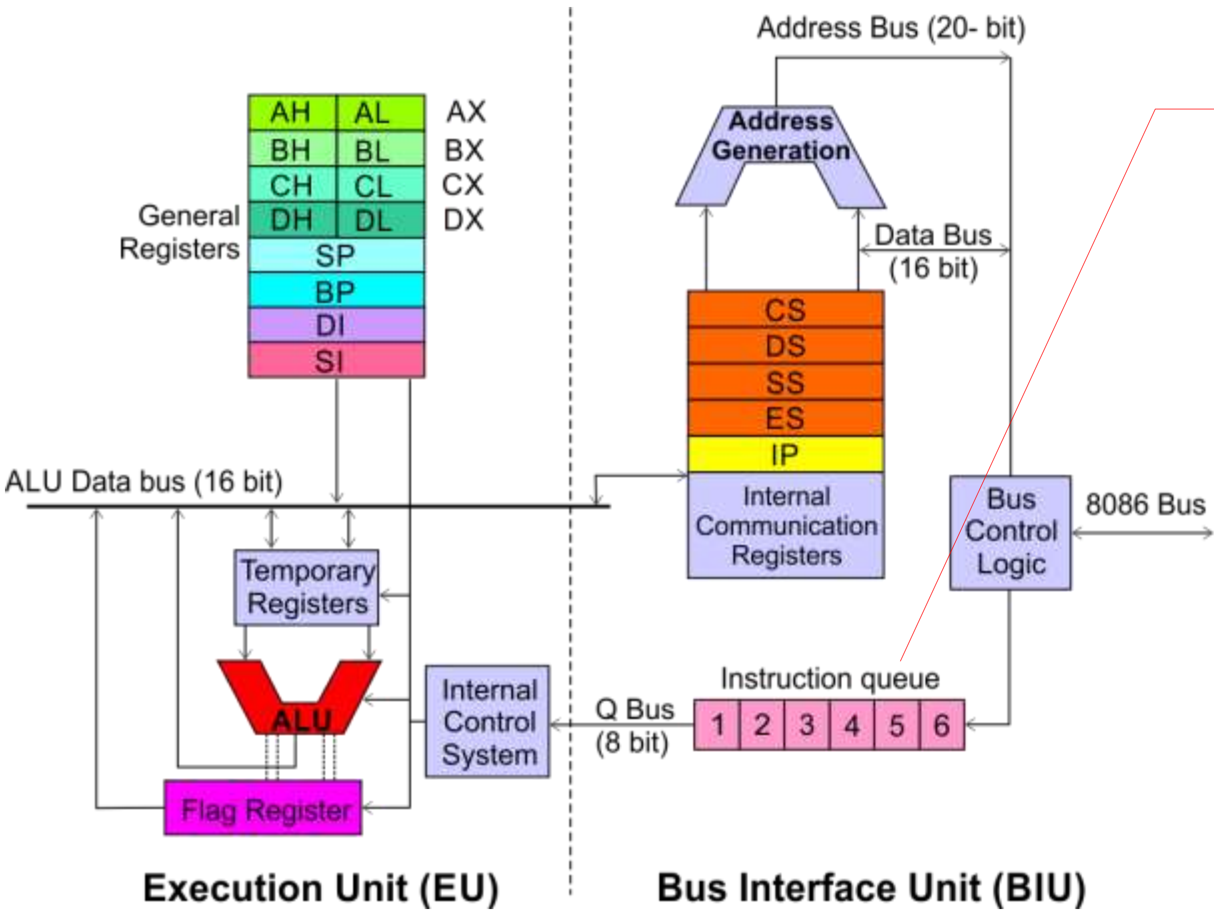
Segment Registers

Instruction Pointer

- 16-bit register.
- **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.**



Bus Interface Unit (BIU)



Instruction queue

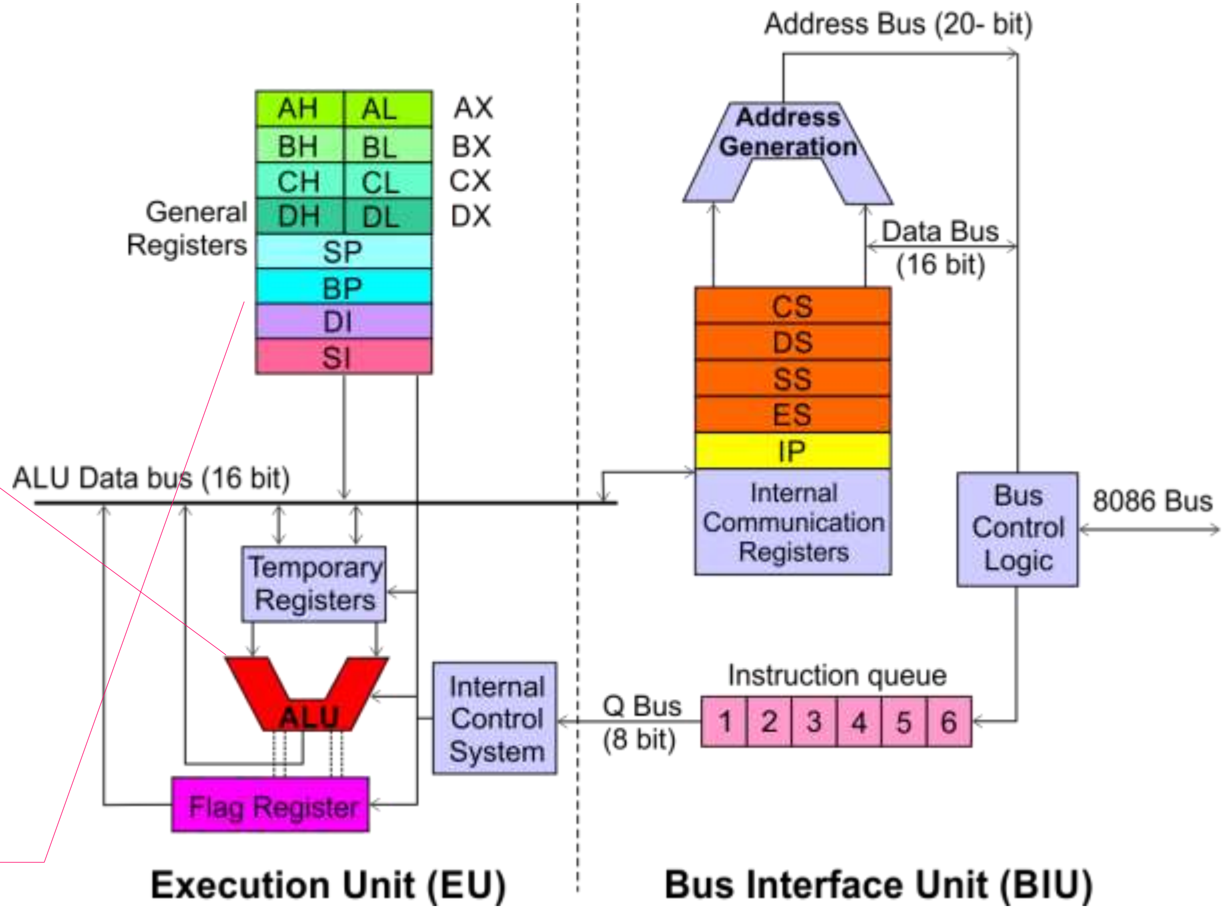
- **A group of First-In-First-Out (FIFO) in which up to 6 bytes of instruction code are pre fetched from the memory ahead of time.**
- **This is done in order to speed up the execution by overlapping instruction fetch with execution.**
- **This mechanism is known as <u>pipelining</u>.**

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

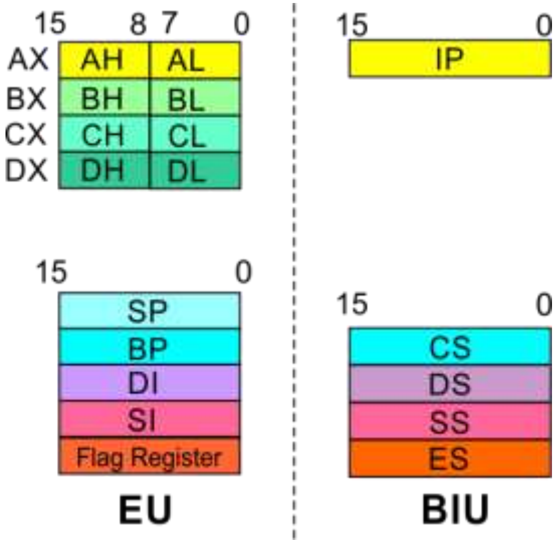
Some of the 16 bit registers can be used as two 8 bit registers as :

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)

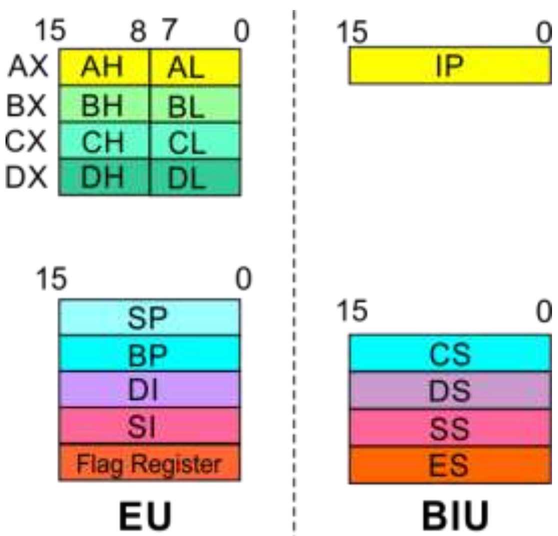
- **Consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX.**
- **AL in this case contains the low order byte of the word, and AH contains the high-order byte.**
- **The I/O instructions use the AX or AL for inputting / outputting 16 or 8 bit data to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



EU Registers

Base Register (BX)

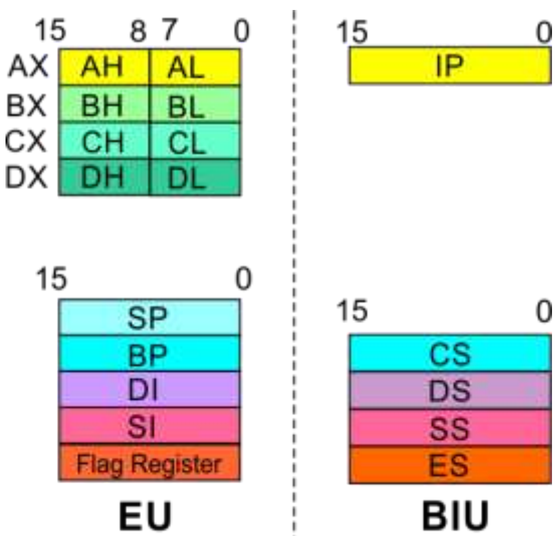
- **Consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX.**
- **BL in this case contains the low-order byte of the word, and BH contains the high-order byte.**
- **This is the only general purpose register whose contents can be used for addressing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



EU Registers

Counter Register (CX)

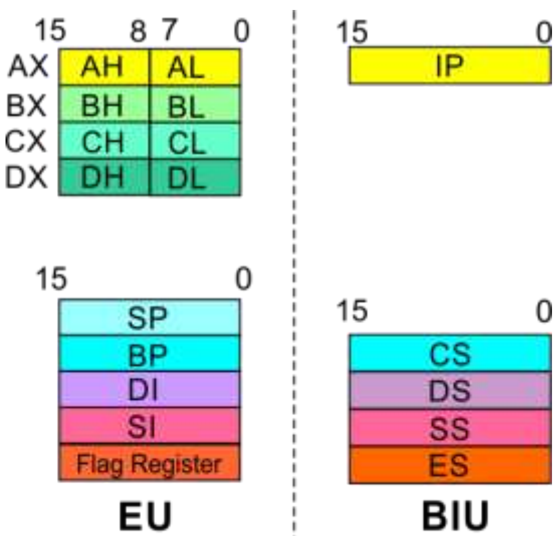
- **Consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX.**
- **When combined, CL register contains the low order byte of the word, and CH contains the high-order byte.**
- **Instructions such as SHIFT, ROTATE and LOOP use the contents of CX as a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



EU Registers

Data Register (DX)

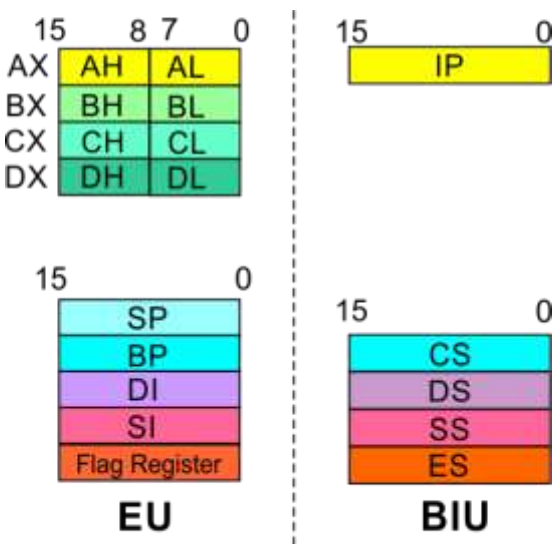
- Consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX.
- When combined, DL register contains the low order byte of the word, and DH contains the high-order byte.
- Used to hold the high 16-bit result (data) in 16 X 16 multiplication or the high 16-bit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



EU Registers

Stack Pointer (SP) and Base Pointer (BP)

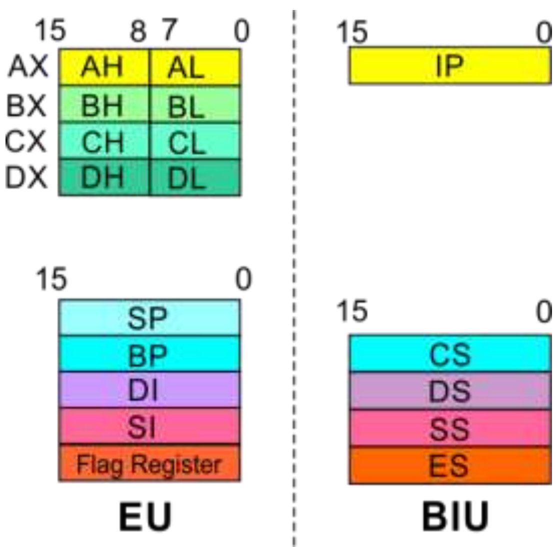
- **SP and BP are used to access data in the stack segment.**
- **SP is used as an offset from the current SS during execution of instructions that involve the stack segment in the external memory.**
- **SP contents are automatically updated (incremented/ decremented) due to execution of a POP or PUSH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



Flag Register

- Flag register contains information reflecting the current status of a microprocessor. It also contains information which controls the operation of the microprocessor.

15														0
—	NT	IOPL	OF	DF	IF	TF	SF	ZF	—	AF	—	PF	—	CF

- Control Flags

IF:	Interrupt enable flag
DF:	Direction flag
TF:	Trap flag

- Status Flags

CF:	Carry flag
PF:	Parity flag
AF:	Auxiliary carry flag
ZF:	Zero flag
SF:	Sign flag
OF:	Overflow flag
NT:	Nested task flag
IOPL:	Input/output privilege level

Flags Commonly Tested During the Execution of Instructions

- There are five flag bits that are commonly tested during the execution of instructions
  - Sign Flag (Bit 7), SF: 0 for positive number and 1 for negative number
  - Zero Flag (Bit 6), ZF: If the ALU output is 0, this bit is set (1); otherwise, it is 0
  - Carry Flag (Bit 0), CF: It contains the carry generated during the execution
  - Auxiliary Carry, AF: (Bit 4) Depending on the width of ALU inputs, this flag bit contains the carry generated at bit 3 (or, 7, 15) of the 8088 ALU
  - Parity Flag (bit2), PF: It is set (1) if the output of the ALU has even number of ones; otherwise it is zero

Control Flags

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

Trap Flag (TF): 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.

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

Direction 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.

Memory Address Calculation

- In 8086, **logical address**isdescribed by combining two parts: **Segment address** and **offset**.
- **Segment address** is 16-bit data from one of the segment registers (CS, SS, DS and ES).
- **Offset address** is 16-bit data from one of the index and pointer registers (DI, SI, SP and BP). Also it could be base register BX.
- To express the 20-bit **PhysicalAddress** of memory
  1. Multiply Segment register by **10H** (or shift it to left by four bit)
  2. Add it to the offset(see Fig 9)

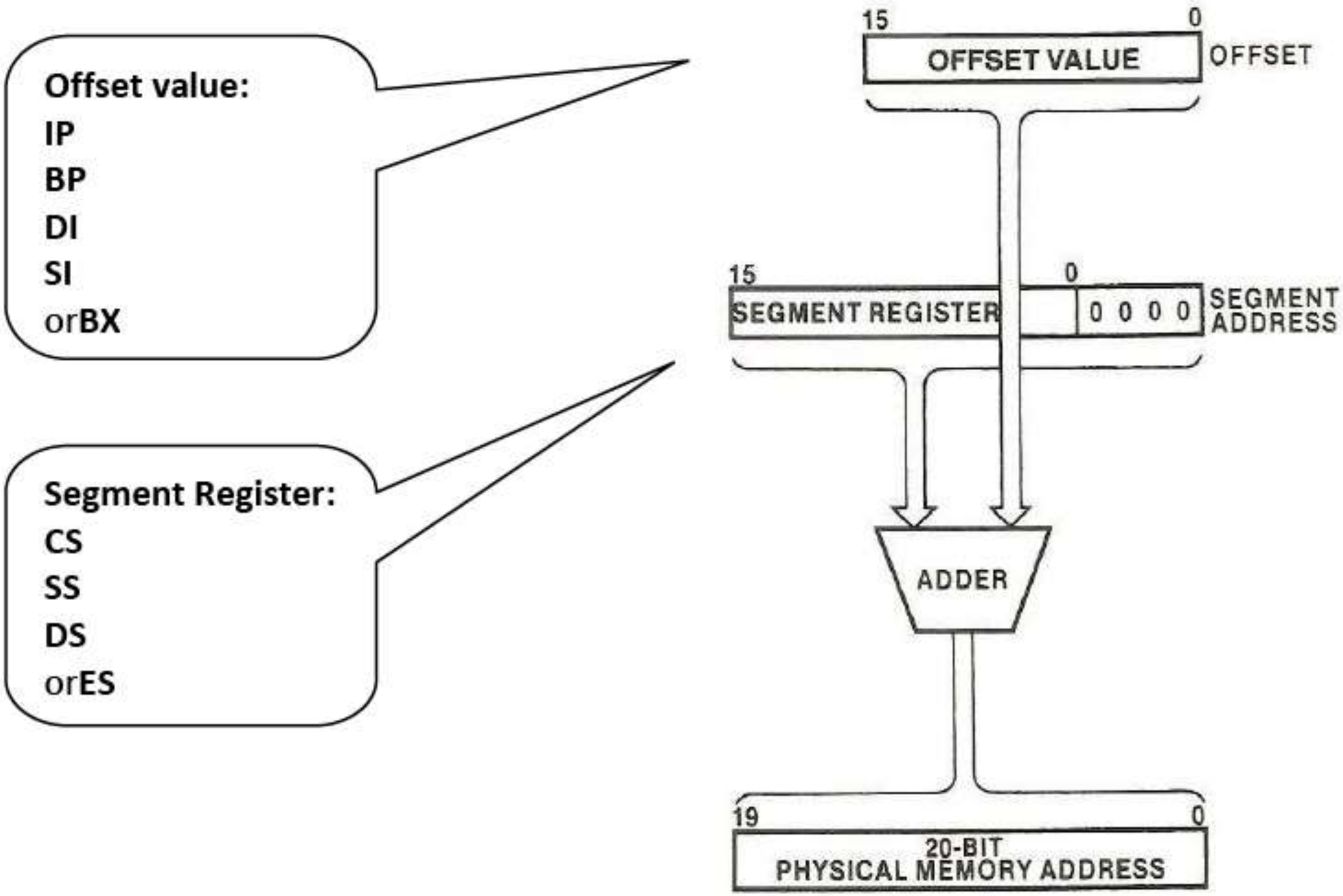


Fig 9: Generating a Memory Address

Segment	Offset Registers	Function
CS	IP	Address of the next instruction
DS	BX, DI, SI	Address of data
SS	SP, BP	Address in the stack
ES	BX, DI, SI	Address of destination data (for string operations)

Example 3: if **CS** = 002AH, and **IP** = 0023H, write the **logical address**that they represent, then map it to **Physical address.**

Solution:

Logical address = **CS:IP**

002A : 0023

Physical address = (**CS** X 10H) + **IP** = 002A0 +0023 = 002C3

Example 4: if **CS** = 002BH, and **IP** = 0013H, write the **logical address** that they represent, then map it to **Physical address.**

Solution:

Logical address = **CS:IP**

002B : 0013

Physical address = (**CS** X 10H) + **IP** = 002B0 +0013 = 002C3

Physical addresses are identical here !

The Stack

The stack is implemented in the memory and it is used for temporary storage of information such as data and addresses. The stack is 64Kbytes long and is organized from a software point of view as 32Kwords (see **Fig 10**).

- SS register points to the lowest address word in the stack
- SP and BP points to the address within stack
- Data transferred to and from the stack are **word-wide**, not **byte-wide**.
- The first address in the Stack segment (SS : 0000) is called *End of Stack.*
- The last address in the Stack segment (SS : FFFE) is called *Bottom of Stack.*
- The address (SS:SP) is called *Top of Stack*.
- POP instruction is used to read **word**from the stack.
- PUSH instruction is used to write **word** to the stack.
- When a word is to be pushed onto the top of the stack:
  - the value of SP is first automatically decremented by two
  - and then the contents of the register written into the stack.
- When a word is to be popped from the top of the stack the
  - the contents are first moved out the stack to the specific register
  - then the value of SP is first automatically incremented by two.

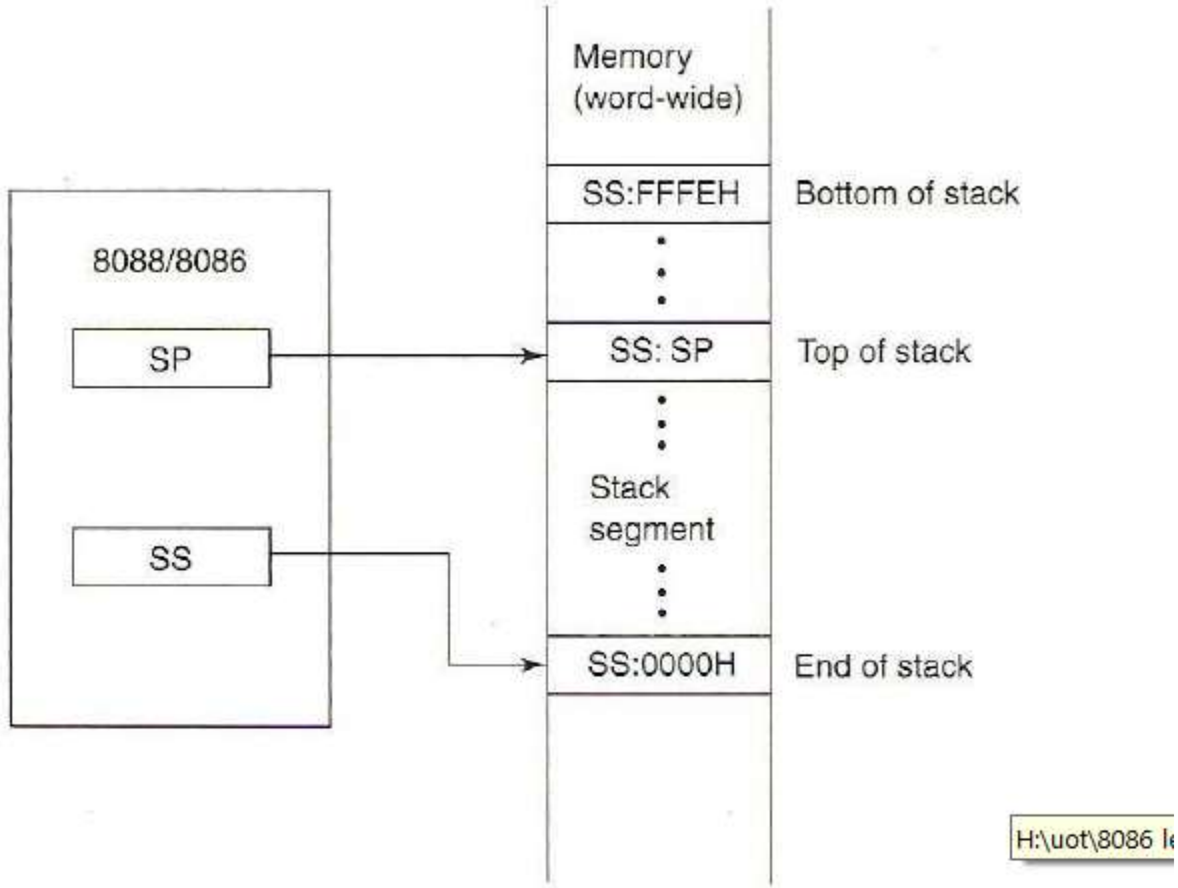
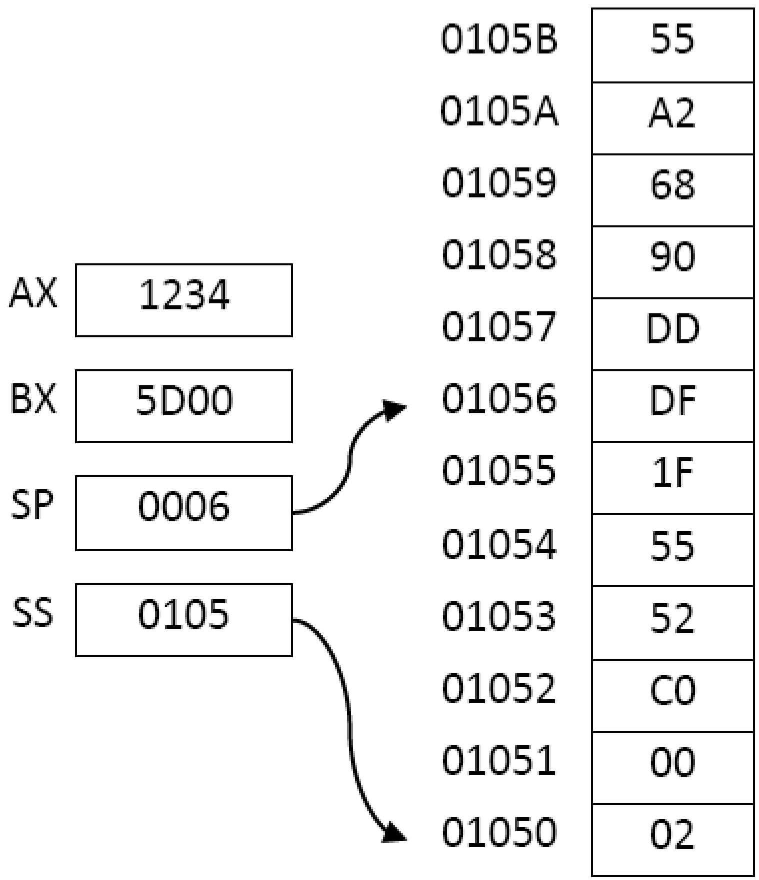


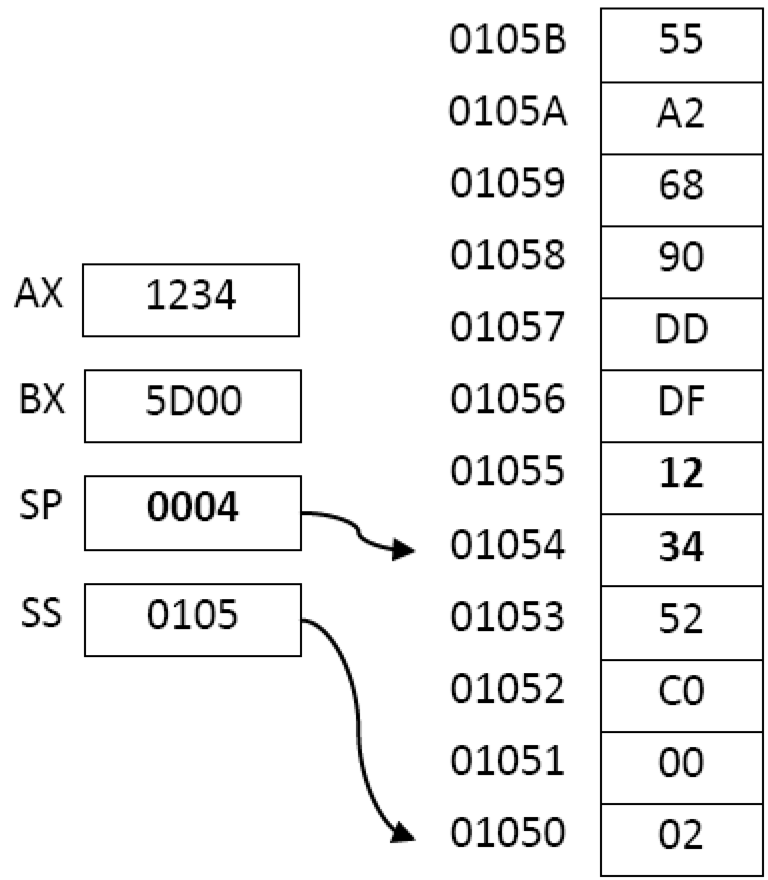
Fig 10: Stack segment of memory

Example 5: let **AX**=1234H ,**SS**=0105H and **SP**=0006H. Fig 11 shows the state of stack prior and after the execution of next program instructions:

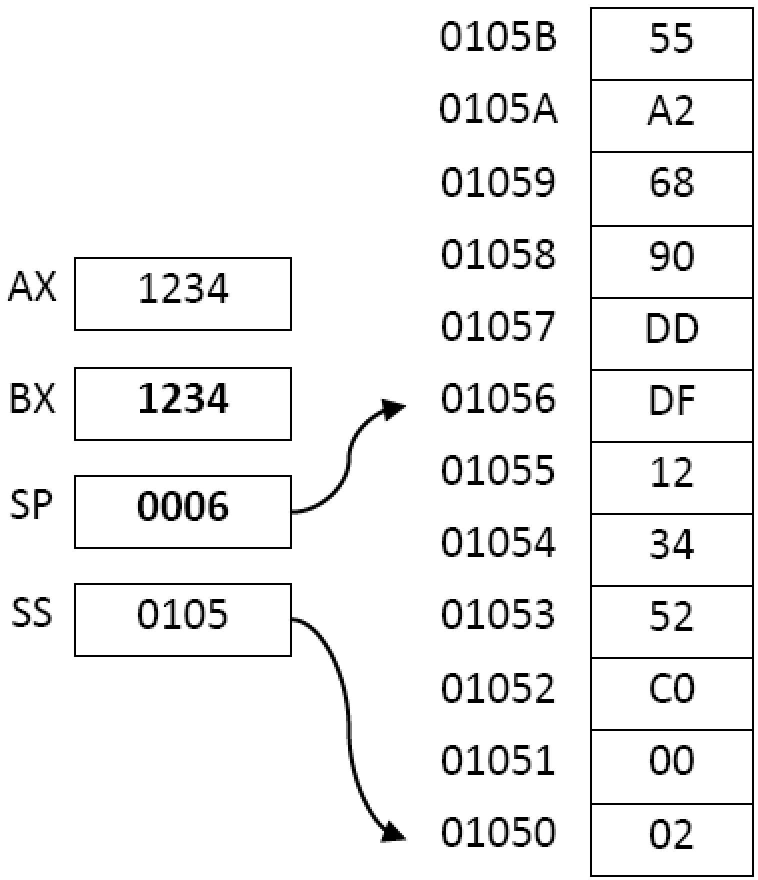
```
PUSH AX
POP BX
POP AX
```



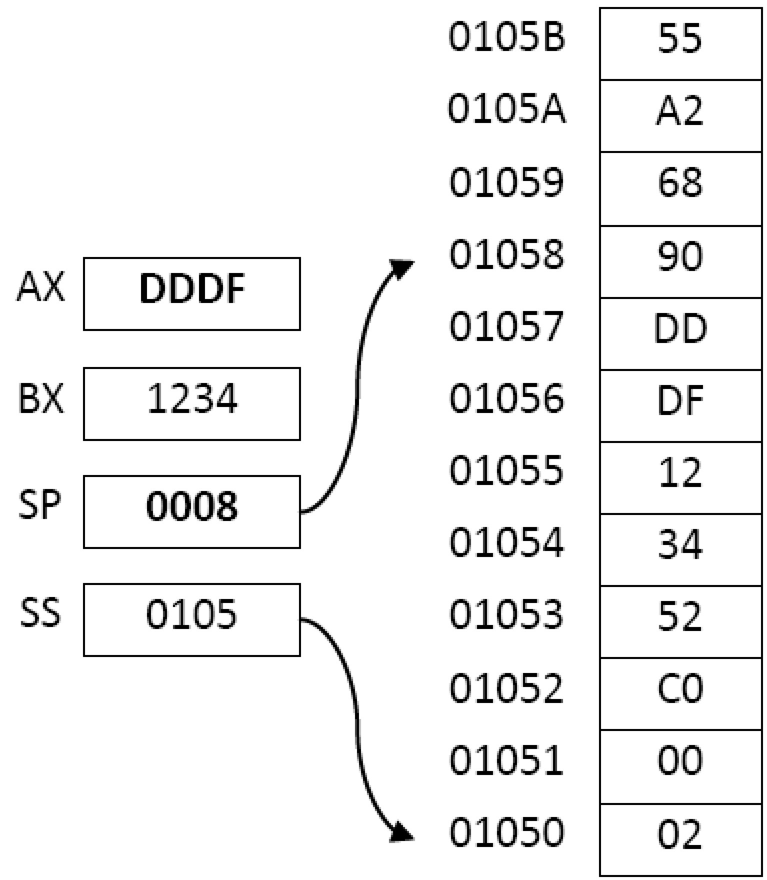
(a) Initial state



(b) After execution of PUSH AX



(c) After execution of POP BX



(d) After execution of POP AX

Fig 11PUSH and POP instruction