Segment Override

MOV CX , SS: [BX]

Here, the concept of Segment Override Prefix is being used.

Although the default segment for the offset BX is DS, as the SS is mentioned in the instruction, it is overriding the default segment.

Hence, the Stack Segment (SS) register is being used here.

Chapter 5

8086 Microprocessor

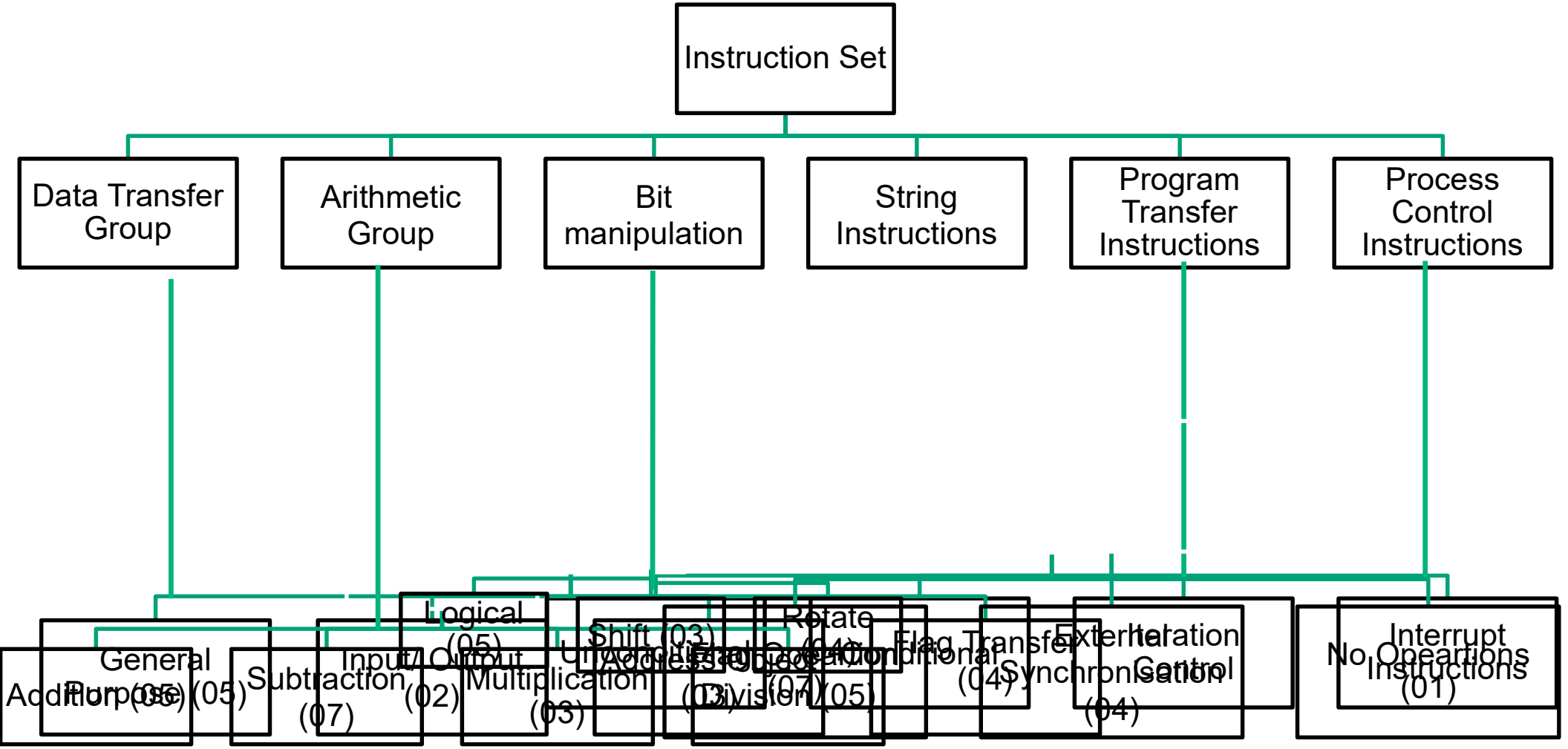
Instruction Set

Bharati Ingale (bharati_ingale@yahoo.com)

Classification of Instruction Set

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Bit Manipulation Instructions
4. String Instructions
5. Program Execution Transfer Instructions
6. Processor Control Instructions

8086


Instruction Set
Data Transfer Group
Arithmetic Group
Bit manipulation
String Instructions
Program Transfer Instructions
Process Control Instructions
General Purpose (05)
Addition (06)
Subtraction (07)
Input/Output (02)
Logical (05)
Multiplication (03)
Shift (03)
Rotate (04)
Flag Transfer (04)
Unconditional
Synchronisation (04)
Interrupt Control (01)
No Opeartions
Interrupt Instructions (01)

Data Transfer Instructions

- These instructions are used to transfer data from source to destination.

- The operand can be a constant, memory location, register or I/O port address.

8086 - Instruction Set-**Data Transfer Instructions**

General purpose	
MOV	:Move byte or word
PUSH	: Push word onto stack
POP	: Pop word off stack
XCHG	: Exchange byte or word
XLAT	: Translate byte
Input/ Output	
IN	: Input byte or word
OUT	: Output byte or word

Address Object	
LEA	:Load effective address
LDS	:Load pointer using DS
LES	: Load pointer using ES
Flag Transfer	
LAHF	: Load AH register from flags
SAHF	:Store AH register in flags
PUSHF	:Push flag onto stack
POPF	:pop flags off stack

8086 - Instruction Set

General Purpose Byte or Word Transfer Instruction

1. MOV : copy a Word or a Byte

MOV destination, source: destination ⟵ source

MOV operand 1, operand 2: operand 1 ⟵ operand 2

Sr.No	Destination	Source
1	Memory	Accumulator
2	Accumulator	Memory
3	Register	Register
4	Register	Memory
5	Memory	Register

Sr.No	Destination	Source
6	Register	Immediate
7	Memory	Immediate
8	Seg – Reg	Reg – 16
9	Seg – Reg	Mem - 16
10	Reg – 16	Seg – Reg
11	Mem - 16	Seg – Reg

8086 - Instruction Set

General Purpose Byte or Word Transfer Instruction

1. MOV memory, accumulator:

- Copies the contents of accumulator to memory location specified in the instruction.
- E.g.: MOV [SI], AL

2. MOV accumulator, memory:

- Copies the contents of memory location specified in the instruction to accumulator
- E.g.: MOV AX, TEMP_RESULT; TEMP_RESULT ⟵ Memory location

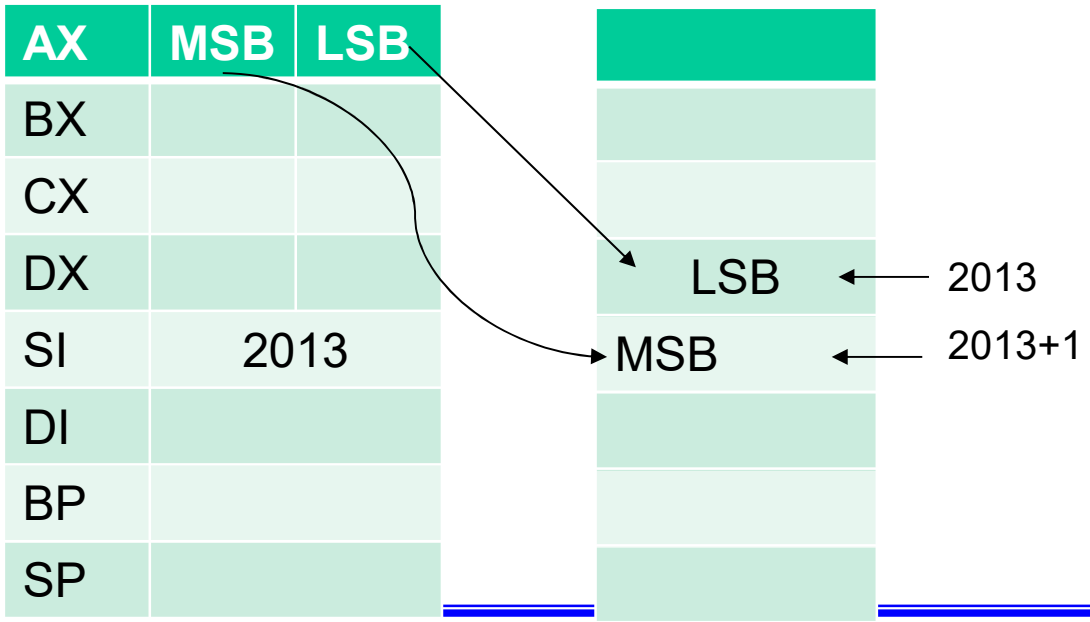
8086 - Instruction Set

General Purpose Byte or Word Transfer Instruction

1. MOV memory, accumulator:

- Copies the contents of accumulator to memory location specified in the instruction.

- E.g.: MOV [SI], AX if SI = 2013, CS= 4205, IP = 187A, DS =2314, AX= 4220



Physical Address =
2 3 1 4 0 H → contents of DS
+ 2 0 1 3 H → offset in SI
2 5 1 5 3 H → PA of mem location

8086 - In

E.g.: MOV [SI], AX if SI = 2013, CS= 4205, IP = 187A, DS =2314, AX= 4220

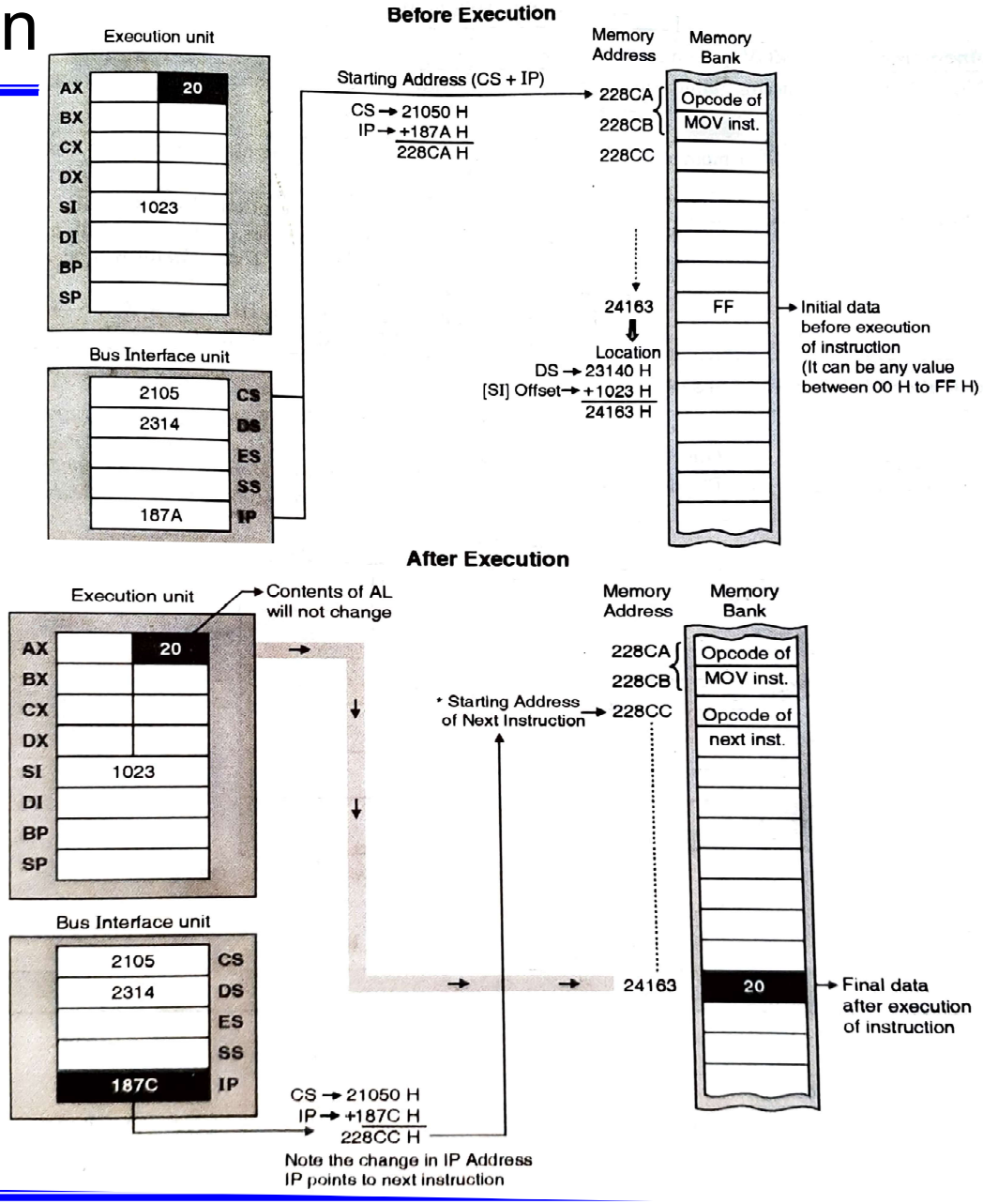
Physical Address =

2 3 1 4 0 H → contents of DS

+ 2 0 1 3 H → offset in SI

2 5 1 5 3 H → PA of mem location

	Before Execution	After Execution
Code Segment (CS)	2105H	2105H
Instruction Pointer (IP)	187AH	187CH
Data Segment (DS)	2314H	2314H
Register AL	20H	20H
Mem Location-24163H	FFH	20H



Bharati Ingale (bharati_ingale@yahoo.com)

8086 - Instruction Set

General Purpose Byte or Word Transfer Instruction

3. MOV destination reg, source reg:

- Copies the contents of source reg. to the destination reg.
- E.g.: MOV AX, BX

4. MOV register, memory:

- Copies the contents of memory location specified in the instruction to register
- E.g.: MOV CX, COUNT [DI]; COUNT is a variable defined while programming Memory location

8086 - Instruction Set

General Purpose Byte or Word Transfer Instruction

5. MOV memory , register:

- Copies the contents of the source register to the specified memory location in the instruction.
- This can be 8/16 bit transfer.
- Cannot be used to access the value of CS & IP.
- E.g.: MOV COUNT [DI], CX

8086 - Instruction Set

General Purpose Byte or Word Transfer Instruction

6. MOV register, immediate:

- Copies the immediate data to the destination reg.
- E.g.: MOV CL, 02H

7. MOV memory , immediate :

- Copies the immediate data to the memory location specified in the instruction.
- E.g.: MOV COUNT [DI], 2D H; COUNT is a variable defined while programming Memory location

8086 - Instruction Set

General Purpose Byte or Word Transfer Instruction

8. MOV seg. reg., reg-16:

- Copies the 16-bit source register contents to the destination segment segment register.
- It cannot be used to copy the contents of CS reg
- Only 16 bit transfer. 8 bit iilegal
- E.g.: MOV DS, AX

8086 - Instruction Set

General Purpose Byte or Word Transfer Instruction

9. MOV seg. reg., mem-16:

- Copies the contents of specified memory location into the desired segment register.
- E.g.: MOV DS,[SI]

10. MOV reg-16, seg. reg. :

- Copies the contents of 16 bit segment register into the destination register specified in the instruction.
- E.g.: MOV AX, DS

2. PUSH Operand: Push Word onto Stack

- It pushes the operand into top of stack.

i. This instruction decrements the stack pointer by 2 & copies a word from a specified source to the location in the stack segment where the stack pointer points

ii. The source of the operand:
   - general purpose reg,
   - segment reg,
   - flag reg or
   - memory

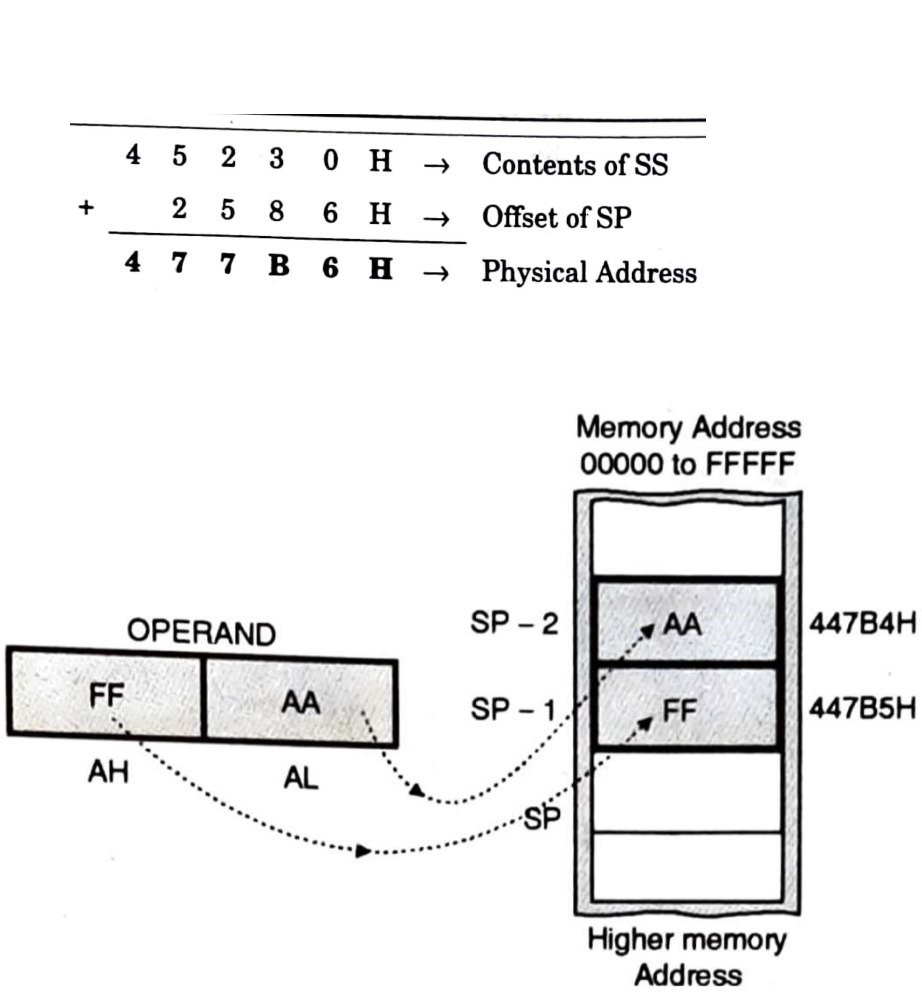
iii. The stack segment reg and stack pointer must be initialised before using this instruction.

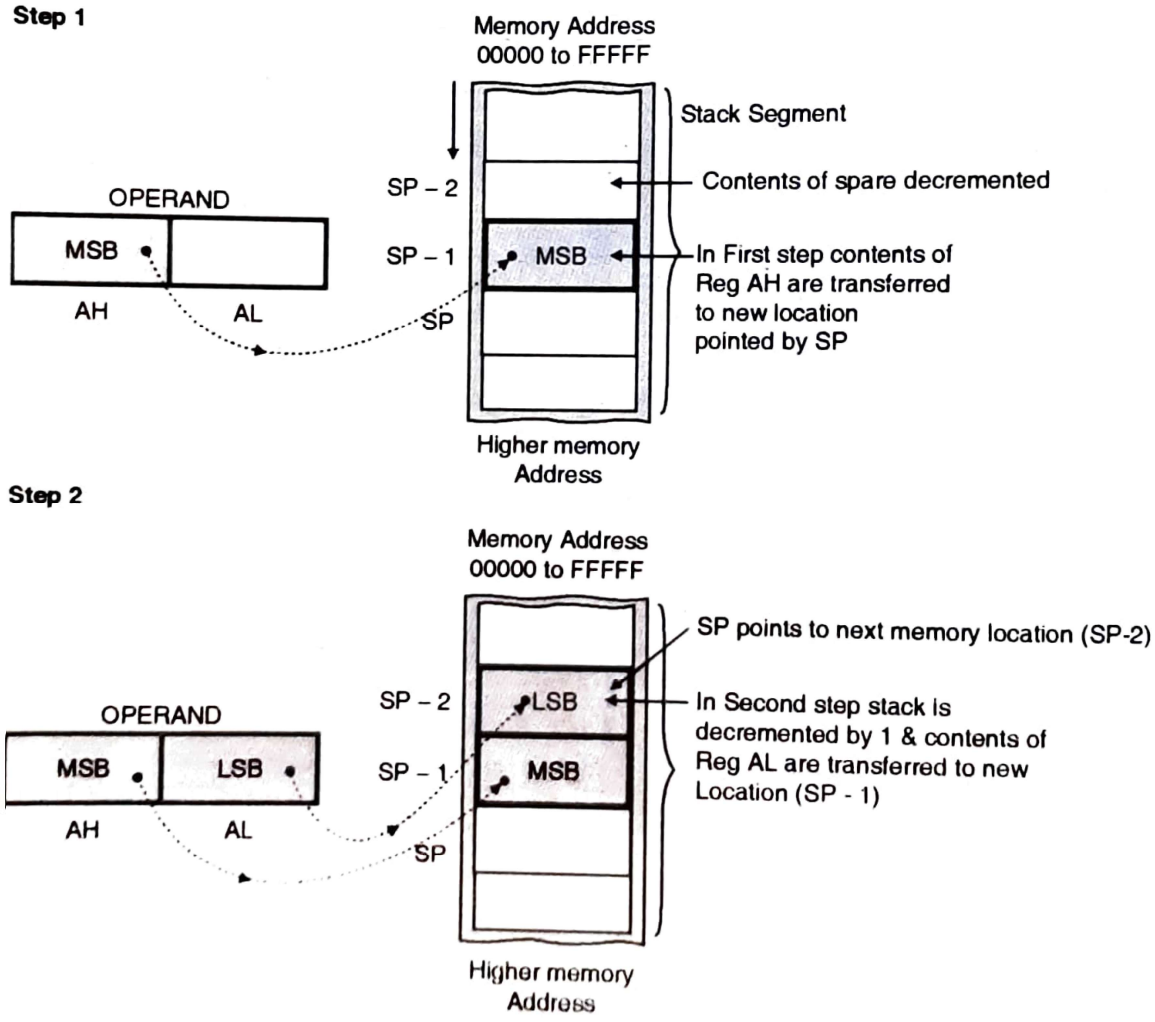
E.g.: PUSH BX; SS:[SP-1] ← BH, SS:[SP-2] ← BL, SP ← [SP-2]

PUSH AX

If SS = 4523H; SP = 2586H; CS = 2105H; IP = 187AH

	4	5	2	3	0	H	→ Contents of SS
+		2	5	8	6	H	→ Offset of SP
	4	**7**	**7**	**B**	**6**	**H**	→ Physical Address





3. POP Des: Pop Word off Stack

- It pops the operand from top of stack to Des.
- Des can be a general purpose register, segment register (except CS) or memory location.

E.g.: POP AX; AL ← SS:[SP], AH ← SS:[SP+1]

SP ← [SP+2]

Physical Address =

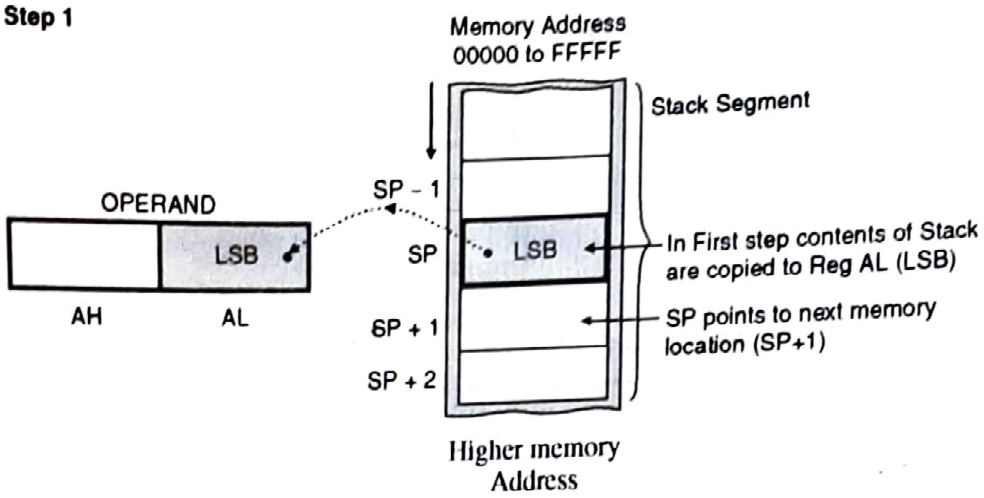
4 5 2 3 0 H → contents of SS

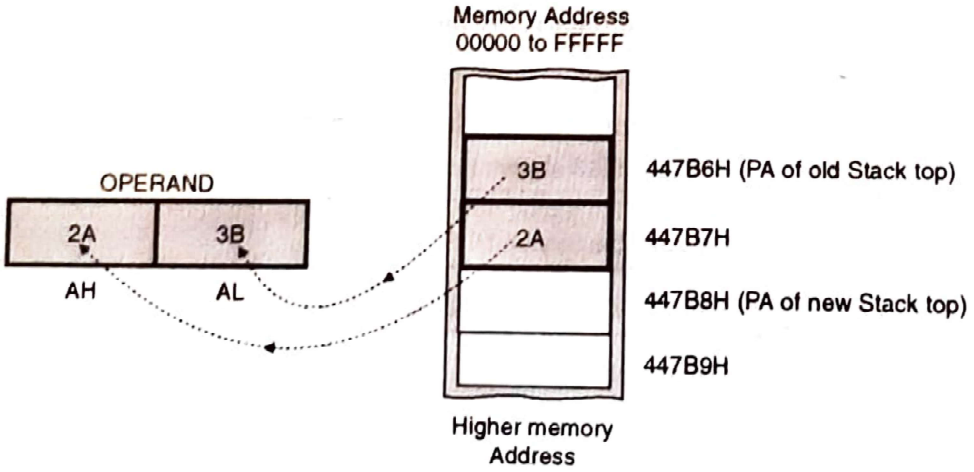
+ 2 5 8 6 H → offset in SI

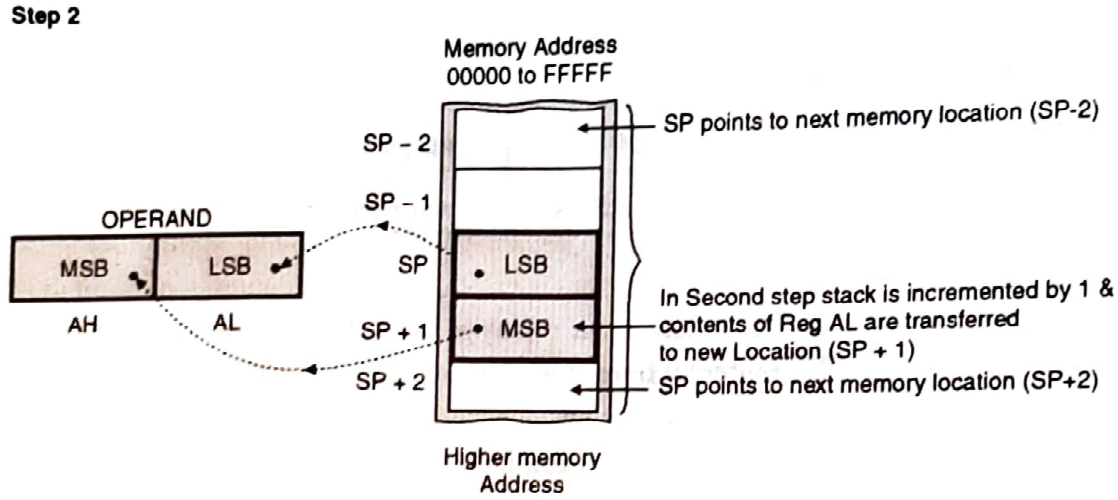
4 7 7 B 6 H → PA Top of Stack

- If SS = 4523H; Offset of SP = 2586H; CS = 2105H and IP = 187AH
- Contents of 477B6 = 3BH and 477B7 = 2AH

- If SS = 4523H; Offset of SP = 2586H; CS = 2105H and IP = 187AH
- Contents of 477B6 = 3BH and 477B7 = 2AH







Physical Address =

4 5 2 3 0 H → contents of SS

+ 2 5 8 6 H → offset in SI

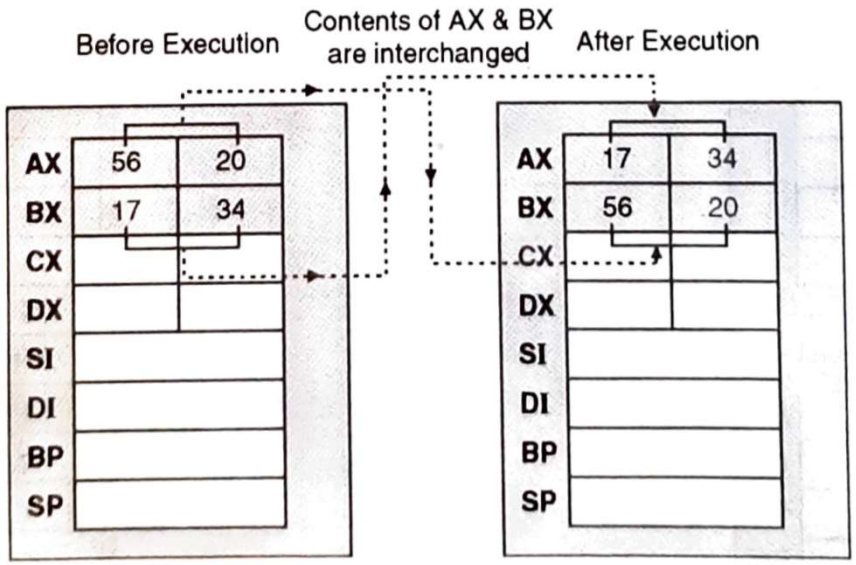
4 7 7 B 6 H → PA Top of Stack

8086 - Instruction Set

4. XCHG Des, Src: Exchange byte or word

- This instruction exchanges Src with Des.
- Src: Register, Memory Location
- Des: Register, Memory Location
- It cannot exchange two memory locations directly.
- E.g.: XCHG DX, AX ; DX ⟷ AX

Segment reg. cannot be used as reg



5. XLAT: Translate or Replace byte

- XLAT/ XLATB
- AL = DS:[BX+ unsigned AL]
- This instruction replaces a byte in AL register with a byte from a look up table in the memory i.e it copies the value of memory byte at location DS : [BX + unsigned AL]
- Here the contents of AL acts as index to the desired location in lookup table

Segment reg. cannot be used as reg

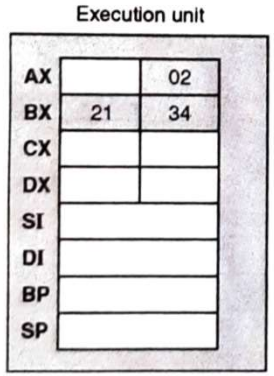
8086 - Instruction Set

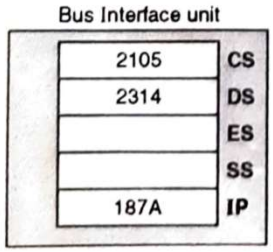
5. XLAT

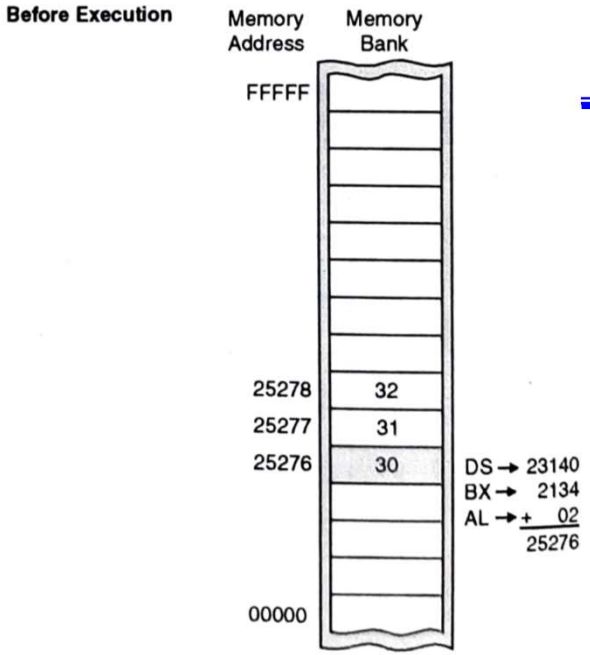
- **Translate** byte in AL reg with a byte from a lookup table in memory.
- BX stores the offfset of the starting address of the lookup table
- AL reg stores the byte no. from the lookup table.
- This instruction copies byte from address ponted by [BX+AL] back into AL or Move into AL the contents of memory location in Data Segment, whose offset address is formed by [BX+AL]

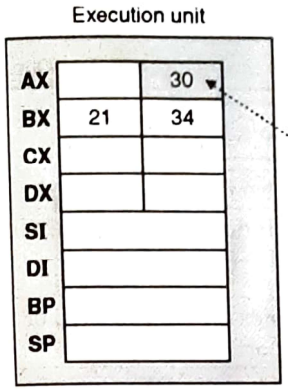
AL = DS:[BX+ unsigned AL]

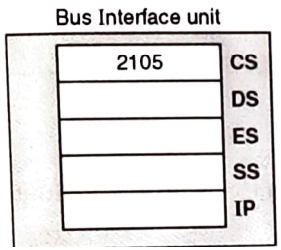
	23140	H	→	Address of DS
+	2134	H	→	Offset of BX
+	02	H	→	Contents of AL
	25276	**H**	→	Physical address

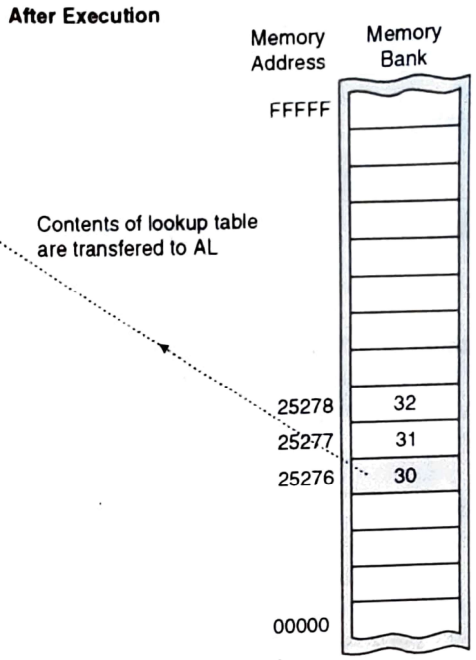












6. Input/ Output

- These instructions are basically related to communication with I/O devices.
- 2 instructions: IN & OUT

8086 - Instruction Set

6.1. IN Accumulator, Port Address

- It transfers the operand from specified port to accumulator register.
- E.g.: IN AX, 28 H; $AX \leftarrow [28H]_{I/O}$

6.2. OUT Port Address, Accumulator:

- It transfers the operand from accumulator to specified port.
- E.g.: OUT 28 H, AX; $[28H]_{I/O} \leftarrow AX$

Port can be a 8 bit or 16 bit port.

7. Address Object

- These instructions manipulate the addresses of the variables, rather than the contents or values of the variables.
- These are mainly used for list processing, based variables and string operation.
- 3types:
  - LEA: Load Effective Address
  - LDS: Load pointer with DS
  - LES: Load pointer using ES

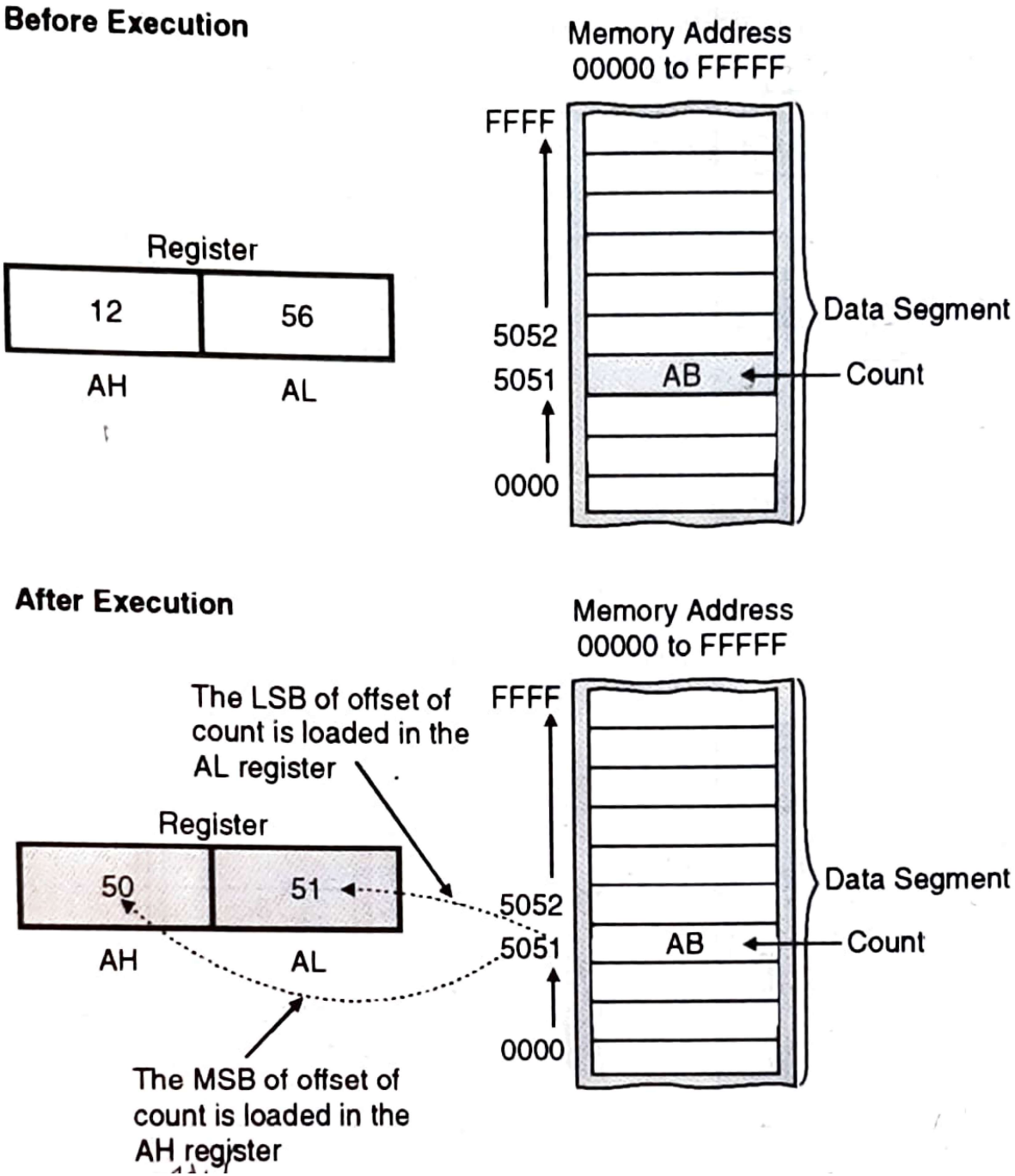
7.1 LEA Register, Src:

- Load Effective Address.(offset addr.)
- It loads a 16-bit register with the offset address of the data specified by the Src.
- The offset is loaded into index reg. or base pointer reg- SI,DI,BX,BP
- E.g.: LEA AX, COUNT
- This instruction loads AX with the offset of COUNT in DS.

  AX ⟵ offset of COUNT

LEA AX, count

If Count = 5051



7.2. LDS reg, Src: Load pointer with DS

- The **source is always memory location** & **DS is used as segment reg** for memory.
- It is a **2 byte** instruction.
  i. It copies a word from 2 memory locations into reg specified in the instruction.
  ii. It copies a word from the next 2 memory locations into DS reg
- It loads 32-bit pointer from memory source to destination register and DS.

7.2. LDS reg, Src: Load pointer with DS

- The offset addr. is placed in the destination register and the segment addr. is placed in DS.
- To use this instruction the word at the lower memory address must contain the offset addr and the word at the higher address must contain the segment addr.
- E.g.: LDS BX, [0301 H]

  BX ⟵ {DS:[0301H], DS:[0301H +1]}

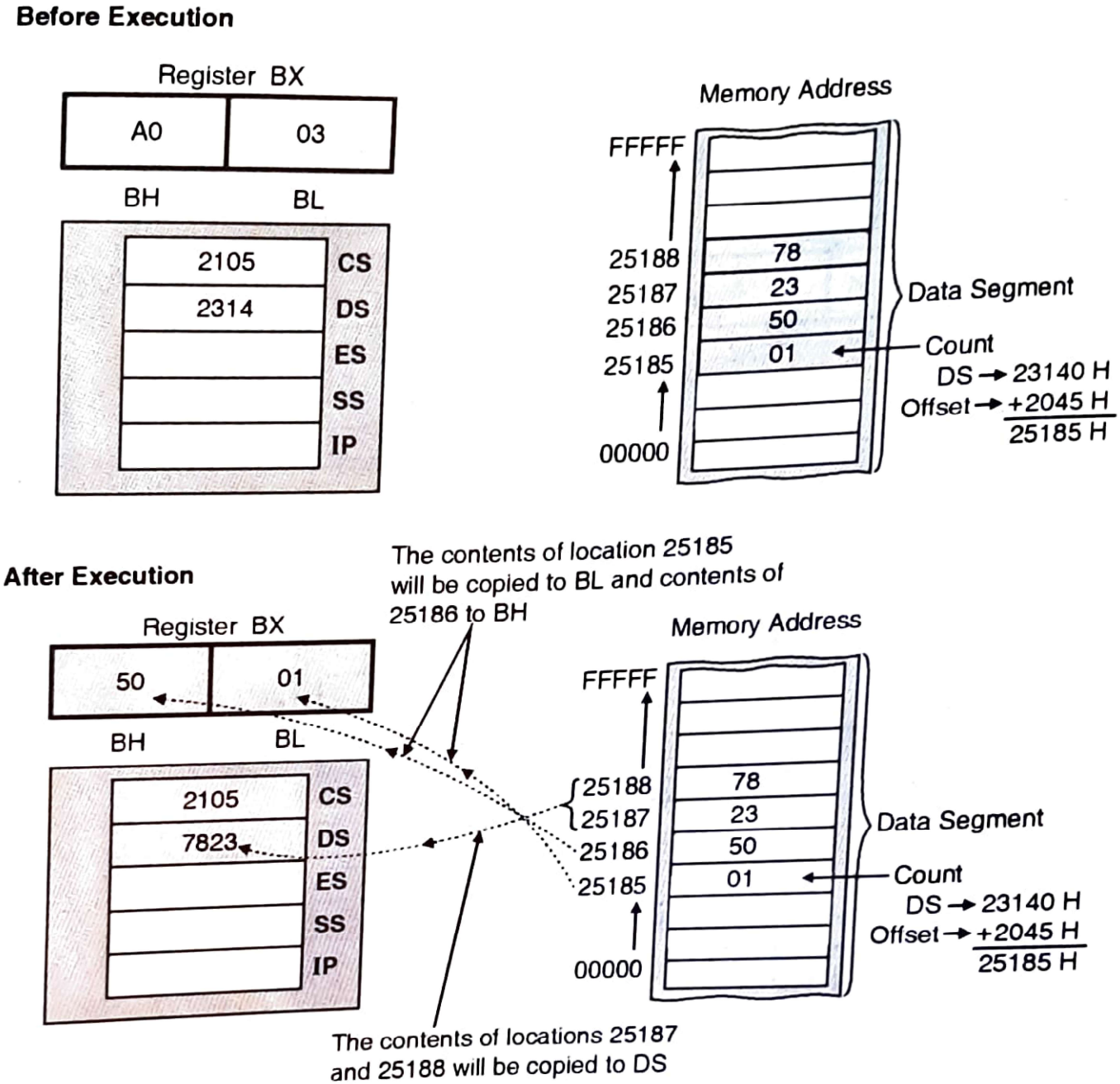
  DS ⟵ {DS:[0301H+2], DS:[0301H +3]}

LDS BX, count

- Count = 2045
- DS = 2314; CS = 2105H and IP = 187AH; ES = 2010H

Physical Address

2 3 1 4 0 H
+ 2 0 4 5 H
2 5 1 8 5 H



Bharati Ingale (bharati_ingale@yahoo.com)

7.3. LES reg, Src: Load pointer using ES

- The source is always memory location.
- It is a 2 byte instruction.
  i. It copies a word from 2 memory locations into reg specified in the instruction.
  ii. It copies a word from the next 2 memory locations into ES reg
- It loads 32-bit pointer from memory source to destination register and ES.

7.3. LES reg, Src: Load pointer using ES

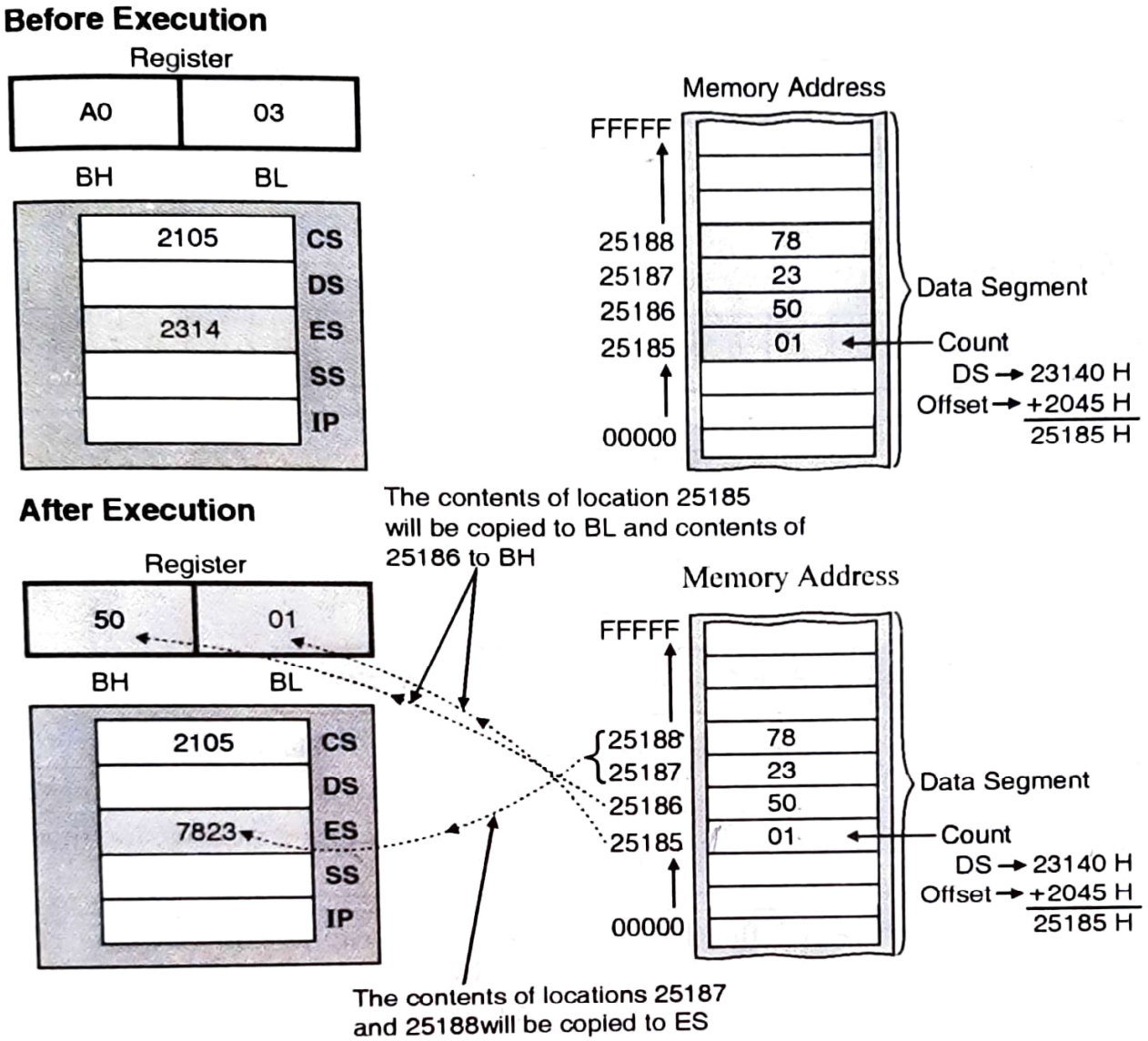
- The offset addr. is placed in the destination register and the segment addr is placed in ES.
- This instruction is very similar to LDS except that it initializes ES instead of DS.
- E.g.: LES BX, [0301 H]
- BX ⟵ {DS:[0301H], DS:[0301H +1]}
- ES ⟵ {DS:[0301H+2], DS:[0301H +3]}

LES BX, count

- Count = 2045
- ES = 2314; CS = 2105H and IP = 187AH; DS = 2010H; BH = A0H; BL = 03H

Physical Address

2 3 1 4 0 H
+ 2 0 4 5 H
2 5 1 8 5 H

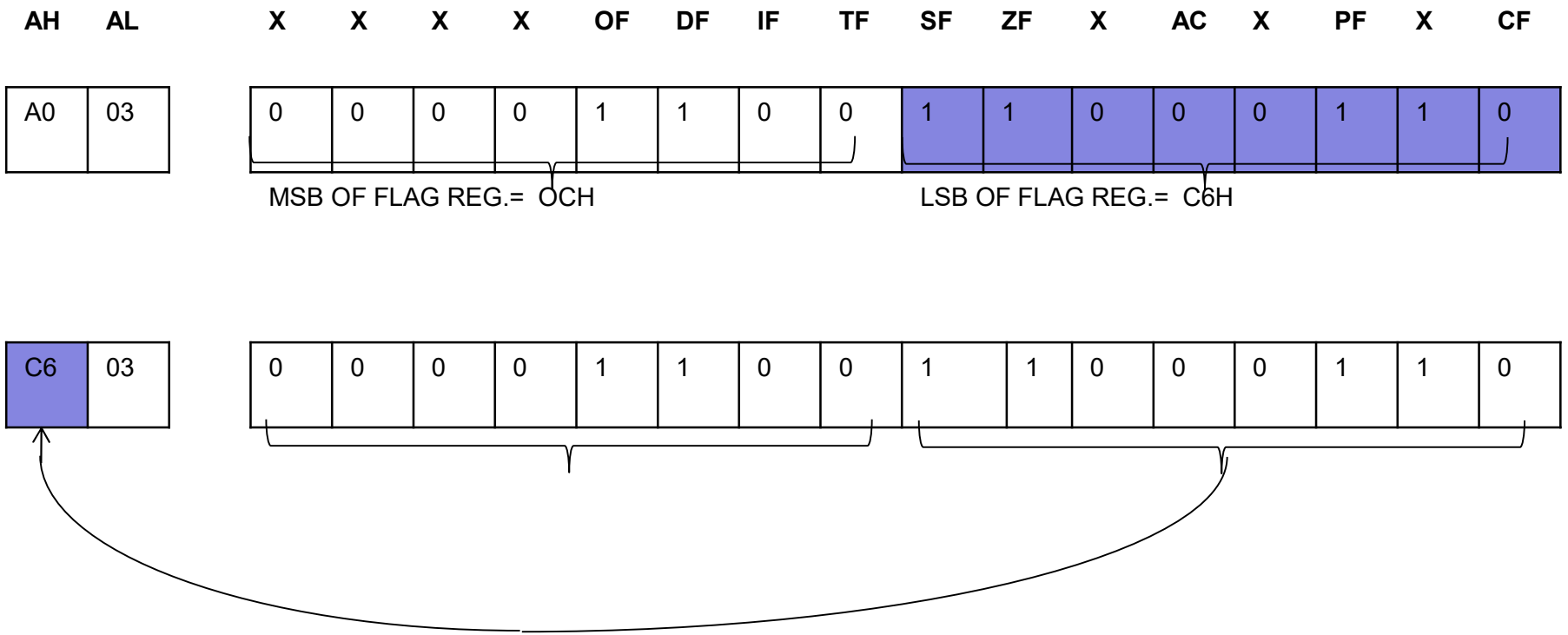


8. Flag Instructions

- These instructions are related to movement of flag register to/from a register & memory
- 4 instructions:
  - LAHF
  - SAHF
  - PUSHF
  - POPF

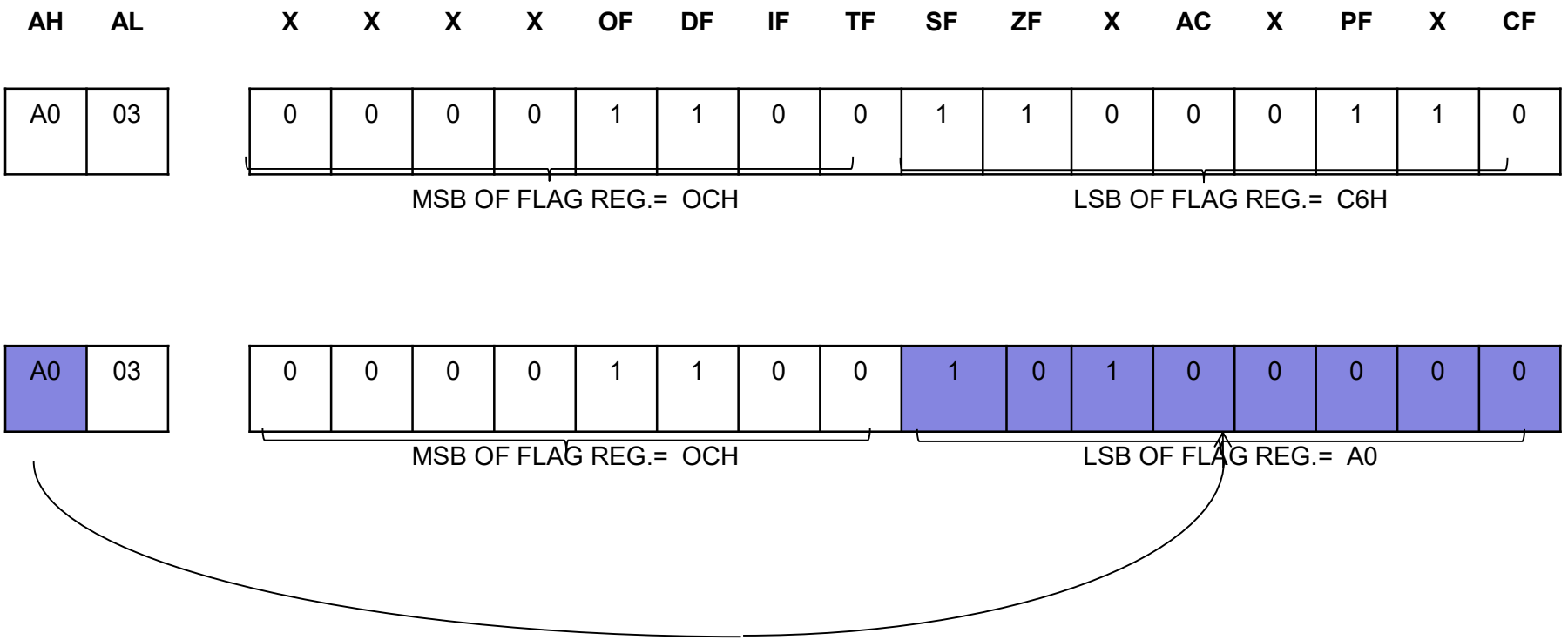
8.1. LAHF

- The lower byte of flag register is copied to the AH reg.



8.2. SAHF

- It stores the contents of AH to lower byte of flag register.

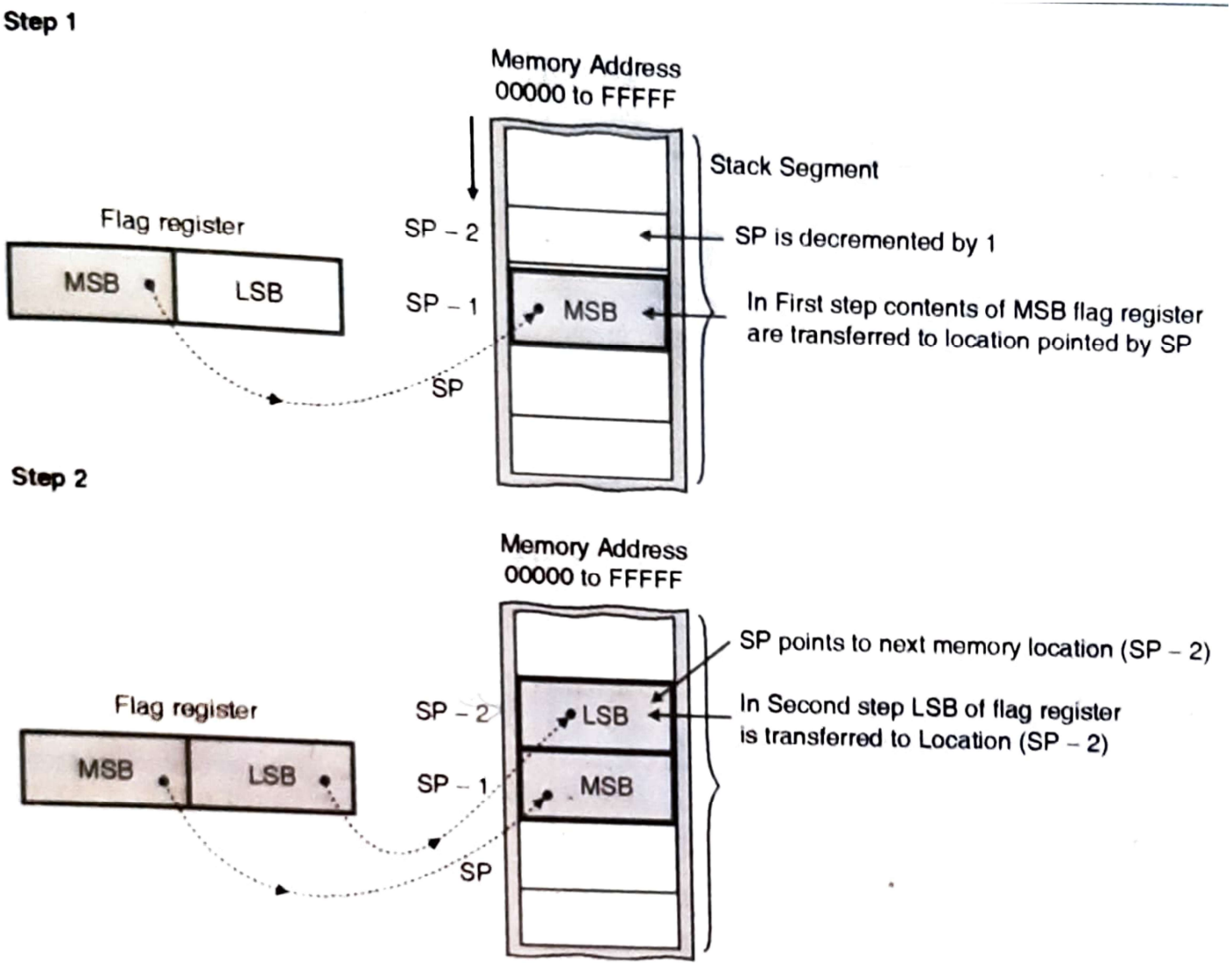


8.3 PUSHF: Push flags onto stack

- Pushes flag register to top of stack.
  - i. This instruction decrements the stack pointer by 2
  - ii. Copies word in the flag reg to the memory location pointed by stack pointer

SS:[SP-1]$\leftarrow$ $Flag_H$, SS:[SP-2]$\leftarrow$ $Flag_L$, SP$\leftarrow$ SP-2

PUSHF


Step 1
Memory Address
00000 to FFFFF
Stack Segment
Flag register
MSB
LSB
SP – 2
SP – 1
SP
SP is decremented by 1
MSB
In First step contents of MSB flag register
are transferred to location pointed by SP
Step 2
Memory Address
00000 to FFFFF
SP points to next memory location (SP – 2)
Flag register
MSB
LSB
SP – 2
SP – 1
SP
LSB
MSB
In Second step LSB of flag register
is transferred to Location (SP – 2)

8.4 POPF : Pop flags off stack

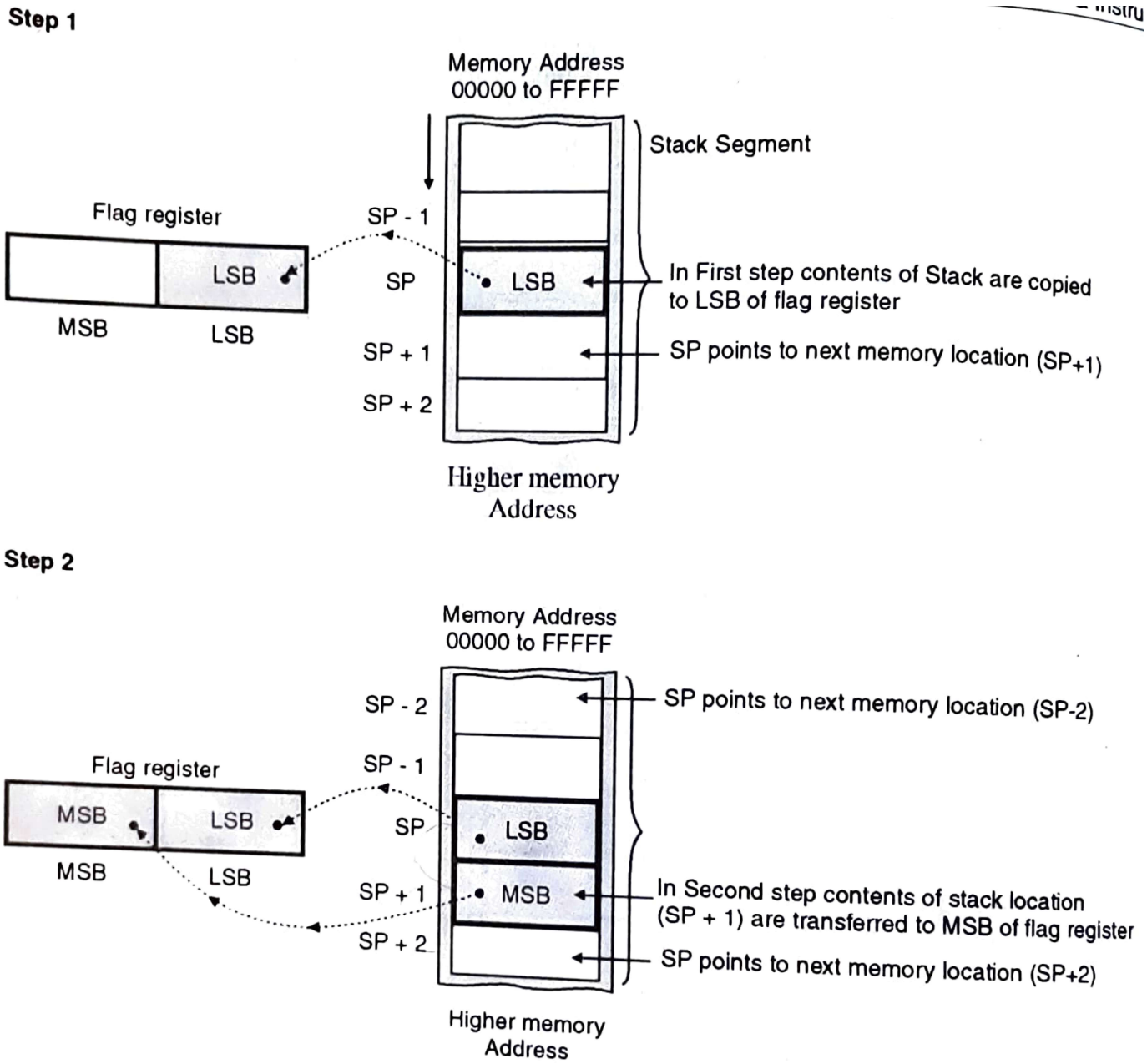
- Pops the stack top to flag register.

  i. This instruction copies a word from the 2 memory locations at the top of the stack to the flag register.

  ii. it increments the stack pointer by 2

$Flag_H \leftarrow SS:[SP]$, $Flag_L \leftarrow SS:[SP+1]$, $SP \leftarrow SP+2$

POPF


Step 1
Memory Address
00000 to FFFFF
Stack Segment
Flag register
SP - 1
LSB
SP
LSB
In First step contents of Stack are copied to LSB of flag register
MSB
LSB
SP + 1
SP points to next memory location (SP+1)
SP + 2
Higher memory Address
Step 2
Memory Address
00000 to FFFFF
SP - 2
SP points to next memory location (SP-2)
Flag register
SP - 1
MSB
LSB
SP
LSB
MSB
LSB
SP + 1
MSB
In Second step contents of stack location (SP + 1) are transferred to MSB of flag register
SP + 2
SP points to next memory location (SP+2)
Higher memory Address

8086 - Instruction Set

Arithmetic Instructions

Addition	
ADD	:Add byte or word
ADC	: Add byte or word with carry
INC	: Increment byte or word by 1
AAA	: ASCII adjust for addition
DAA	: Decimal adjustment for addition

Arithmetic Instructions

Subtraction	
SUB	:Subtract byte or word
SBB	:Subtract byte or word with borrow
DEC	: Decrement byte or word by1
NEG	: Negate byte or word
CMP	:Compare byte or word
AAS	: ASCII adjust for subtraction
DAS	: Decimal adjustment for subtraction

Arithmetic Instructions

Multiplication	
MUL	:Multiply byte or word unsigned
IMUL	: Multiply byte or word signed/ Integer Multiply byte or word
AAM	: ASCII adjust for multiply

Arithmetic Instructions

Division	
DIV	:Divide byte or word unsigned
IDIV	:integer divide byte or word
AAD	: ASCII adjust for division
CBW	: Convert byte to word
CWD	:Convert word to double word

Arithmetic Instructions

I) Addition Instructions

1. **ADD Des, Src:**

- It adds a byte to byte or a word to word.
- Adds a number form src to des and the result is in des.
- Src: Register ,Memory Location or immediate number.
- Des: Register ,Memory Location
- **Flags affected**: AF, CF, OF, PF, SF, ZF

E.g.: ADD AL, 74H
ADD DL, CL
ADD DX, AX
ADD AX, [BX]

8086 - Instruction Set

2) ADC Des, Src:

- It adds the two operands with CF.
- Src: Register ,Memory Location or immediate number.
- Des: Register ,Memory Location
- **Flags affected**: AF, CF, OF, PF, SF, ZF

E.g.: ADC AL, 74H
ADC DX, AX
ADC AX, [BX]

- ADD BL, CL: CL = 04H and BL = 07H

8086 - Instruction Set

3. INC Src

- It increments the byte or word by one. Or add 1 to specified Src.
- **Src:** Register or Memory location.
- **Flags affected:** AF, OF, PF, SF, ZF

  CF is not affected.

E.g.: INC AX

INC BL

4. AAA (ASCII Adjust after Addition):

- The data entered from the terminal is in ASCII format.
- In ASCII, 0 – 9 are represented by 30H – 39H.
- This instruction allows us to add the ASCII codes.
- **It result is obtained in unpacked BCD form.**
- This instruction does not have any operand.
- AAA updates the AF & CF
- But OF,PF,SF,ZF are undefined

4. AAA (ASCII Adjust after Addition):

- AAA instrcution woks only on AL reg
- If lower nibble of AL > 9 or AF=1

  then AL =AL + 6

  AH = AH + 1

  AF =1, CF=1 else AF=0, CF=0

Eg: ADD CL ; if AL=34 ASCII & CL= 38 ASCII

AL= 6CH Incorrect

AAA ;AL= 0 2 Unpacked BCD

Carry = 1 Ans= 12 decimal

- AL=35 ASCII ==5 5+ 9= 14==E
- BL= 39 ASCII ==9

```
  0 0 1 1   0 1 0 1 ➔ AL
+ 0 0 1 1   1 0 0 1 ➔ BL
-----------------------
  0 1 1 0   1 1 1 0 ➔ 6E
+ 0 0 0 0   0 1 1 0 ➔ 06
-----------------------
    0 0 0 0   0 1 0 0
+   0 0 0 1           ➔1
-----------------------
  0 0 0 1  0 1 0 0 ==14
```

5. DAA (Decimal Adjust after Addition)

- It is used to make sure that the result of adding two BCD numbers is adjusted to be a correct BCD number.
- It only works on AL register.
- Flags updated: AF, CF, PF & ZF. OF is undefined

5. DAA (Decimal Adjust after Addition)

- If the value of low order 4 bits (D0-D3) in the AL reg is greater than 9 or if AF is set, the instruction adds 06 to the result
  - If lower nibble of AL > 9 or AF=1

    then AL =AL + 6

- If the value of higher order 4 bits (D4-D7) in the AL reg is greater than 9 or if CF is set, the instruction adds 60 to the result
  - If higher nibble of AH > 9 or CF=1

    then AH =AH + 6

- AL =59 BL =34 ADD AL,BL DAA

ADD AL, BL

```
  0101  1001
+ 0011  0100
-----------
  1000  1101
```

AL = 8DH invalid

DAA

```
  0011  0100
+ 0000  0110
-----------
  1001 001 1 ➔93
```

AL = 93H

8086 - Instruction Set

II Substraction Instructions:

6. SUB Des, Src:

- It subtracts a byte from byte or a word from word.
- Subtracts a number in the src from des and the result is in des.
- Src: Register ,Memory Location or immediate number.
- Des: Register ,Memory Location
- **Flags affected**: AF, CF, OF, PF, SF, ZF flags.
- For subtraction, CF acts as borrow flag.
- E.g.: SUB AL, 74H
  SUB DX, AX
  SUB AX, [BX]

7. SBB Des, Src:

- It subtracts the two operands and also the borrow from the result.
- Src: Register ,Memory Location or immediate number.
- Des: Register ,Memory Location
- **Flags affected**: AF, CF, OF, PF, SF, ZF flags.
- E.g.: SBB AL, 74H
  SBB DX, AX
  SBB AX, [BX]

8. DEC Src:

- It decrements the byte or word by one.
- **Src:** Register or Memory location.
- **Flags affected:** AF, OF, PF, SF, ZF
  CF is not affected.

E.g.: DEC AX
DEC BL

9. NEG Src:

- It creates 2's complement of a given number and stores it back in the Src.
- That means, it changes the sign of a number.
- **Src:** Register, Memory location
- **Flags affected:** AF, OF, PF, SF, ZF, CF
- Assume AL= 0011 0101 = 35H then

NEG AL; AL ⟵ 1100 1011 = CBH

- NEG AX
- AX = 00A3H = 0000 0000 1010 0011

2's complement= 1111 1111 0101 1100 ---1's complement

\+ 1

1111 1111 0101 1101 –2's complement

Contents of AX after Execution on NEG AX

= FF5DH

8086 - Instruction Set

10. CMP Des, Src:

- It compares two specified bytes or words.
- **Src:** Immediate numebr, register or memory location.
- **Des:** Register or memory location.
- Both operands cannot be a memory location at the same time.
- The comparison is done simply by internally subtracting the source from destination.
- The value of source and destination does not change, but the flags are modified to indicate the result.
- **Flags affected:** AF, OF, PF, SF, ZF, CF
- Eg: CMP BL,55H BL=55

	CF	ZF	SF
CMP CX,BX SRC>DES	1	0	1
CMP AL, [5000H] SRC<DES	0	0	0
SRC=DES	0	1	0

11. AAS (ASCII Adjust after Subtraction):

- Similar like AAA
- It result is obtained in unpacked BCD form.
- This instruction does not have any operand.
- **Flag Affected:** AF & CF But OF,PF,SF,ZF are undefined

Let AL = 0011 1001 ➔ 39H ➔ ASCII 9

Let BL = 0011 0101 ➔ 35H ➔ASCII 5

SUB AL, BH =➔ 0011 1001

- 0011 0101

0000 0100 ➔ BCD = 04

CF = 0 No borrow required

12. DAS (Decimal Adjust after Subtraction)

- It is used to make sure that the result of subtracting two BCD numbers is adjusted to be a correct BCD number.
- It only works on AL register.
- **Flags updated:** AF, CF, PF & ZF. OF is undefined
- If the value of low order 4 bits (D0-D3) in the AL reg is greater than 9 or if AF is set, the instruction subtracts 06 to the result
- If the value of higher order 4 bits (D4-D7) in the AL reg is greater than 9 or if CF is set, the instruction subtracts 60 to the result

AL = 1000 0110 ➔ 86 BCD

BH = 0101 0111 ➔ 57 BCD

SUB AL, BH

DAS

AL = 1000 0110

BH = - 0101 0111

0010 1111 ➔ 2F, CF = 0

Lower Nibble = FH - 0000 0110 ➔ 06

➔ >9 subtract 06 0010 1001 ➔ 29H BCD

III Multiplication Instruction

13. MUL Src: (unsigned multiplication) 8/16 bit

- It multiplies two bytes to produce a word or two words to produce a double word. **AX = AL * Src (8 bit)**
- **DX : AX = AX * Src (16 bit)**
- This instruction assumes one of the operand in AL or AX.
- **Src:** register or memory location.
- It affects AF,PF,SF & ZF
- Eg: MUL BL; AX = AL * BL

  MUL BX; DX : AX = AX * BX

IV Division

15. DIV Src: (unsigned division) 8/16 bit reg – divisor

- It divides word by byte or double word by word.
- If the divisor is 8 bit then dividend is stored in AX, and the result is stored as: **AH = remainder AL = quotient**
- If the divisor is 16 bit then dividend is stored in DX: AX, and the result is stored as: **DX = remainder AX = quotient**
- **ALL flags are undefined after DIV instruction**
- Eg: DIV BL;AX / BL

8086 - Instruction Set

15. **IMUL Src: (**signed multiplication **) 8/16 bit**

17. **IDIV Src: (**signed division**) 8/16 bit**

V Sign Extension Instruction

11. CBW (Convert signed Byte to signed Word):

- This instruction converts byte in AL to word in AX
- This instruction copies the sign of the byte in AL to all bits in AH
- AH is said to be sign extension of Al or DX is said to be sign extension of AX
-
- Eg: Assume AX = XXXX XXXX **1**001 1000

  Then CBW gives AX= **1111 1111 1**001 0001

12. CWD (Convert Word to Double Word):

- This instruction converts word in AX to double word in DX : AX.
- It copies sign of the word in AX into all the bits of DX.
- DX is then called sign extension of AX
- The conversion is done by extending the sign bit of AX throughout DX.
- Eg: Assume AX = **1**000 0000 1001 0001

  DX = xxxx xxxx xxxx xxxx
- Then CWD gives AX = **1**000 0000 1001 0001

  DX = **1111 1111 1111 1111**

8086 - Instruction Set

Bit Manipulation Instructions:

- These instructions are used at the bit level.
- These instructions can be used for:
    i. Testing a zero bit
    ii. Set or reset a bit
    iii. Shift bits across registers

8086 - Instruction Set

Bit Manipulation Instructions

LOGICALS	
NOT	:Not byte or word
AND	:And byte or word
OR	: Inclusive or byte or word
XOR	: Exclusive byte or word
TEST	:Test byte or word

8086 - Instruction Set

Bit Manipulation Instructions

SHIFTS	
SHL/SAL	: Shift logical/ arithmetic left byte or word
SHR	: Shift logical right byte or word
SAR	: Shift arithmetic right byte or word
ROTATES	
ROL	: Rotate left byte or word
ROR	: Rotate right byte or word
RCL	: Rotate through carry left byte or word
RCR	: Rotate through carry right byte or word

8086 - Instruction Set

1. NOT Src:

- It complements each bit of Src to produce 1's complement of the specified operand.
- **Src:** register or memory location.
- Eg: NOT AL

  NOT CX

8086 - Instruction Set

2. **AND Des, Src:**

- It performs AND operation of Des and Src.
- **Src:** Immediate number, register or memory
- location.
- **Des:** register or memory location.
- Both operands cannot be memory locations at the same time.
- **Flags Affected:** CF and OF become zero after the operation and PF, SF and ZF are updated. AF is undefined
- Eg: AND BL,AL

  AND CX, 00F0H

8086 - Instruction Set

3. **OR Des, Src:**

- It performs OR operation of Des and Src.
- **Src:** immediate number, register or memory
- location.
- **Des:** register or memory location.
- Both operands cannot be memory locations at the same time.
- **Flags Affected:** CF and OF become zero after the operation and PF, SF and ZF are updated. AF is undefined
- Eg: OR BL,AL

  OR CX, 00F0H

4. XOR Des, Src:

- It performs XOR operation of Des and Src.
- **Src:** immediate number, register or memory
- location.
- **Des:** register or memory location.
- Both operands cannot be memory locations at the same time.
- **Flags Affected:** CF and OF become zero after the operation and PF, SF and ZF are updated. AF is undefined
- Eg: XOR BL,AL

  XOR CX, 00F0H

5. TEST des, src

- This instruction logically AND's the source with the destination but the result is not stored anywhere.
- Both the Src and Des should be of same size.
- **Src:** immediate number, register or memory location.
- **Des:** register or memory location.
- Only flag bits are affected.
- Flags are affected, PF, SF and ZF are updated to show results.
- Eg: TEST BL,AL

  TEST CX, BX

Shift Instructions

SAL/SHL Des, Count:

- It shift bits of byte or word left, by count
- It puts **zero(s) in LSBs** and **MSB is shifted into carry flag**.
- Des: Register or Memory Location
- Bits are shifted 'count' no. of types

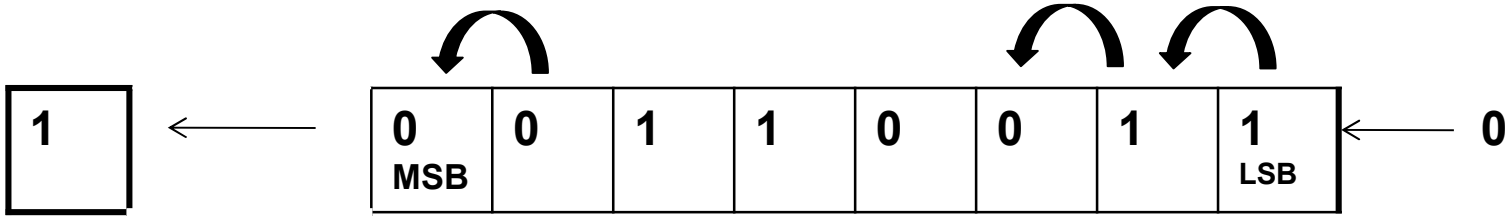
  If count = 1,it is directly specified in the instruction as count

  If count > 1, count is put in CL register & CL gives the count in the instruction.

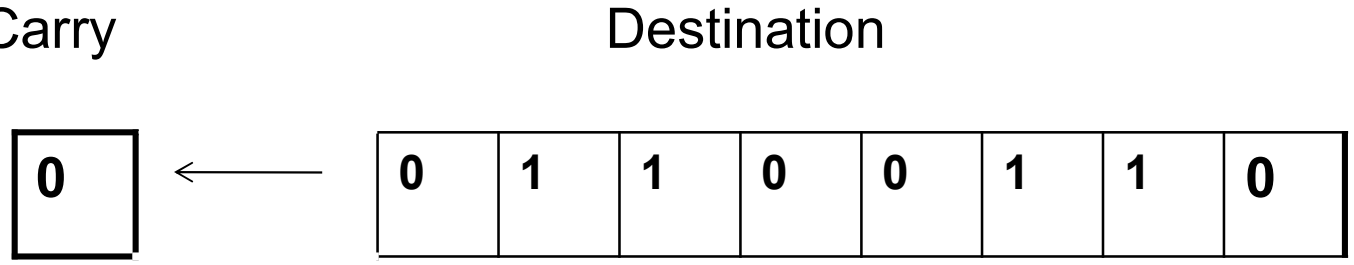
8086 - Instruction Set

Eg: Count = 1 ;SAL BL,1

Assume before operation BL=0011 0011 & CF=1



After operation BL=0110 0110 & CF=0



Eg: Count >1

```
MOV CL, 05H
SAL BL,CL
```

SHR Des, Count:

- It shift bits of byte or word right, by count
- It puts **zero(s) in MSBs** and **LSB is shifted into carry flag**.
- Des: Register or Memory Location
- Bits are shifted 'count' no. of types

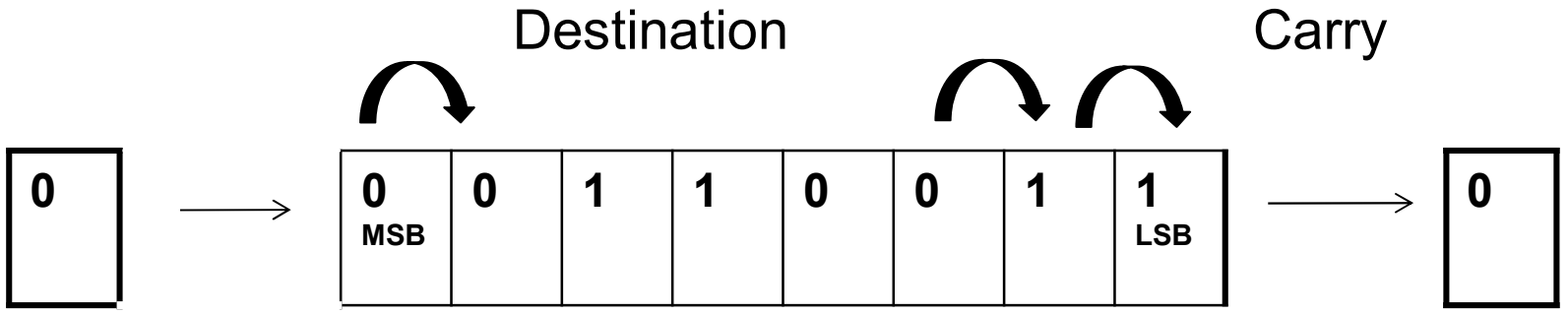
  If count = 1,it is directly specified in the instruction as count

  If count > 1, count is put in CL register & CL gives the count in the instruction.

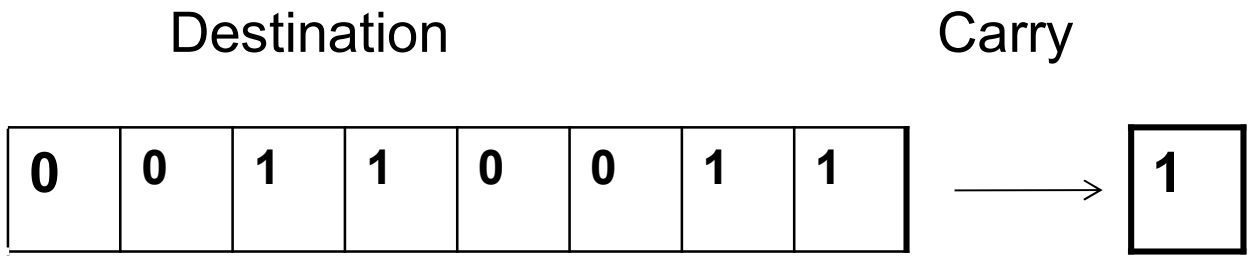
8086 - Instruction Set

Eg: Count = 1 ;SHR BL,1

Assume before operation BL=0011 0011 & CF=0



After operation BL=0001 1001 & CF=1



Eg: Count >1

MOV CL, 05H

SHR BL,CL

SAR Des, Count:

- It right shifts bits of destination byte or word right, by count
- A copy of **old MSB is placed in MSB** itself and **LSB is shifted into carry flag.**
- Des: Register or Memory Location
- Bits are shifted 'count' no. of types

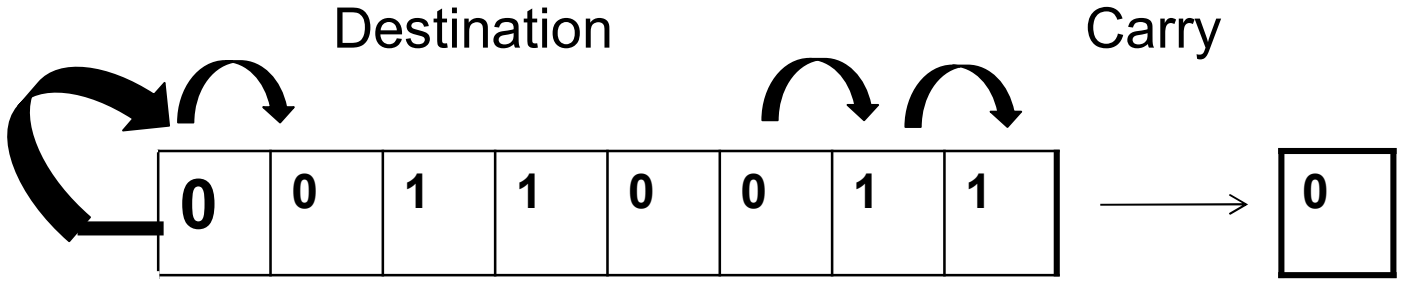
  If count = 1,it is directly specified in the instruction as count

  If count > 1, count is put in CL register & CL gives the count in the instruction.

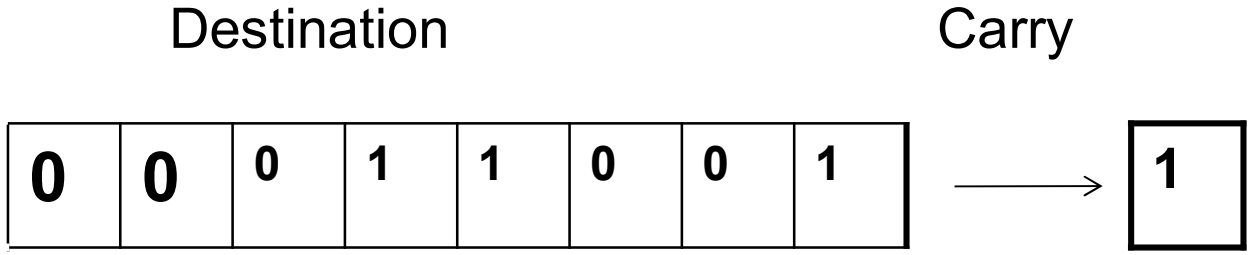
8086 - Instruction Set

Eg: Count = 1 ;SHR BL,1

Assume before operation BL=0011 0011 & CF=0



After operation BL=0001 1001 & CF=1



Eg: Count >1

```
MOV CL, 05H
SHR BL,CL
```

Rotate Instructions

1. ROL Des, Count

- The instruction rotates all bits of destination byte or word to the left, by count
- **MSB is shifted into carry flag and also goes to LSB**
- Des: Register or Memory Location
- Bits are shifted ‘count’ no. of types

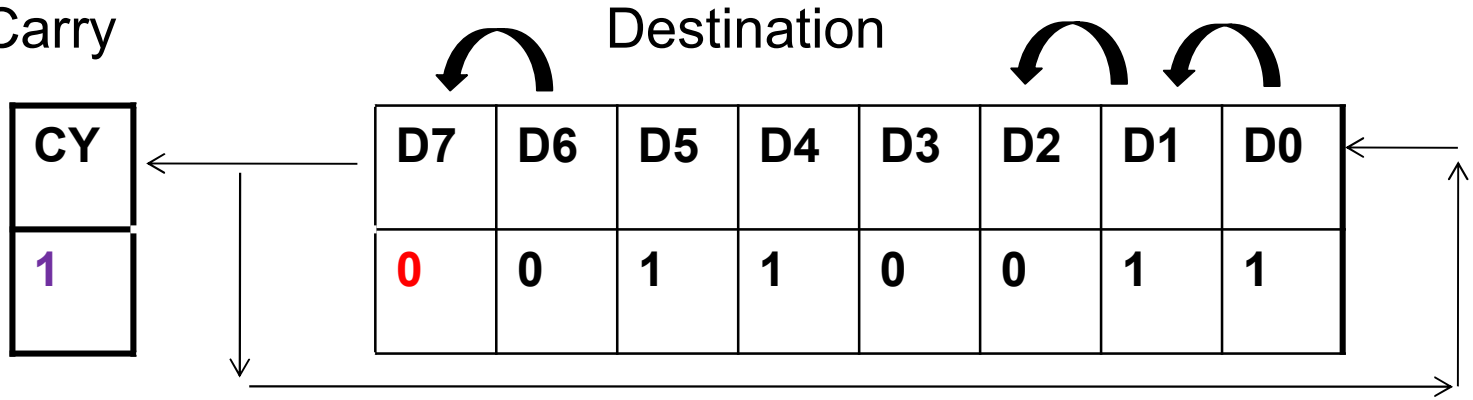
If count = 1,it is directly specified in the instruction as count

If count > 1, count is put in CL register & CL gives the count in the instruction.

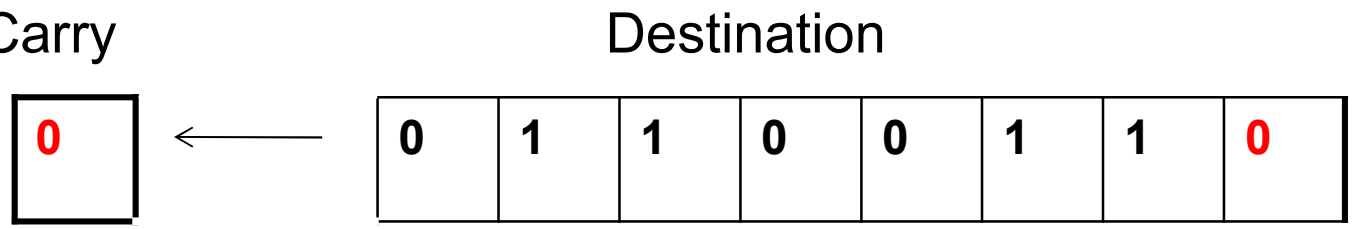
8086 - Instruction Set

Eg: Count = 1 ;ROL BL,1

Assume before operation BL=0011 0011 & CF=1



After operation BL=0110 0110 & CF=0



Eg: Count >1

```
MOV CL, 05H
ROL BL,CL
```

2. ROR Des, Count:

- The instruction rotates all bits of destination byte or word right, by count
- **LSB is shifted into carry flag and also goes to MSB**
- Des: Register or Memory Location
- Bits are shifted 'count' no. of types

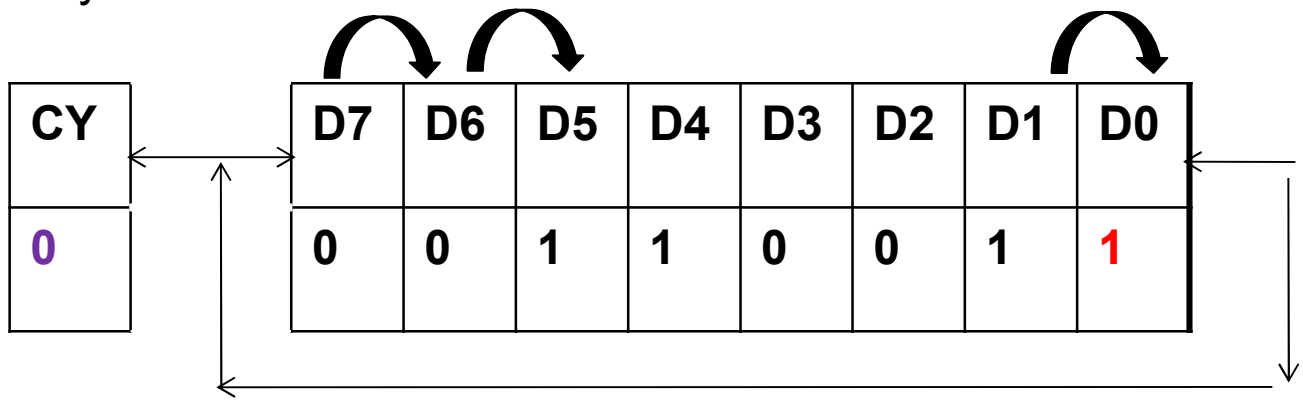
If count = 1,it is directly specified in the instruction as count

If count > 1, count is put in CL register & CL gives the count in the instruction.

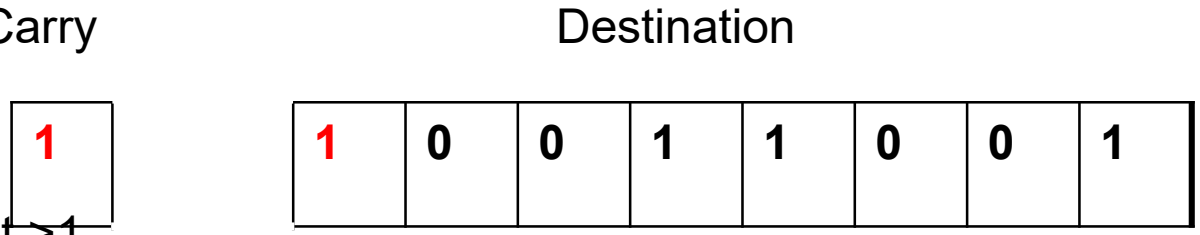
8086 - Instruction Set

Eg: Count = 1 ;ROR BL,1

Assume before operation BL=0011 0011 & CF=0



After operation BL=0001 1001 & CF=1



Eg: Count >1

```
MOV CL, 05H
ROR BL,CL
```

3.RCL Des, Count

- The instruction rotates all bits of destination byte or word to the left, by count
- **MSB is shifted into carry flag (CF) and CF goes to LSB**
- Des: Register or Memory Location
- Bits are shifted ‘count’ no. of types

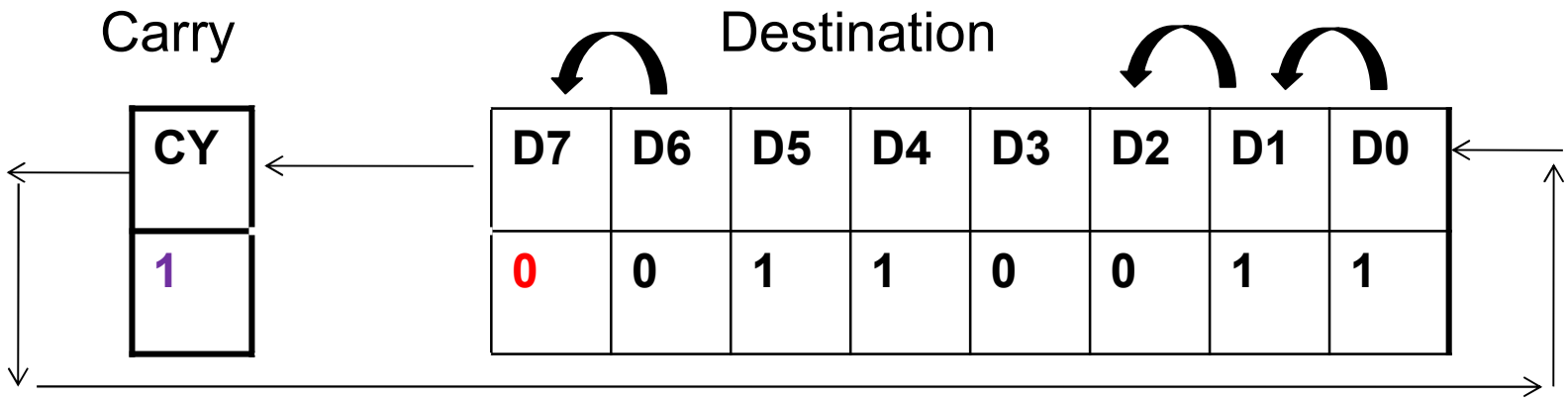
If count = 1,it is directly specified in the instruction as count

If count > 1, count is put in CL register & CL gives the count in the instruction.

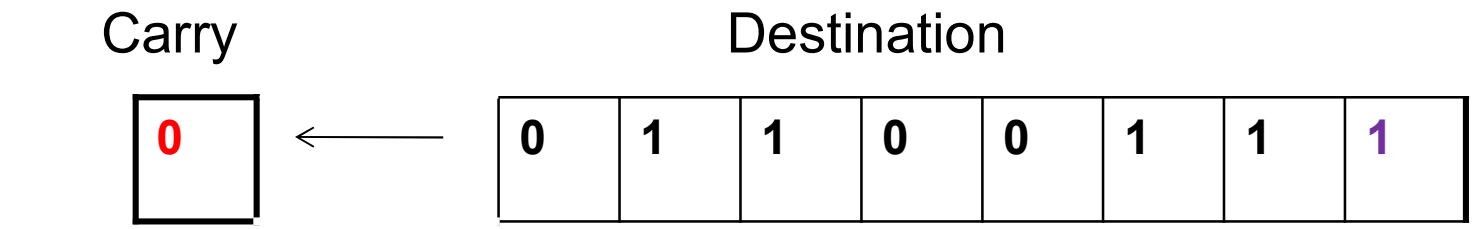
8086 - Instruction Set

Eg: Count = 1 ;RCL BL,1

Assume before operation BL=0011 0011 & CF=1



After operation BL=0110 0110 & CF=0



Eg: Count >1

```
MOV CL, 05H
RCL BL,CL
```

4. RCR Des, Count:

- The instruction rotates all bits of destination byte or word right, by count
- **LSB is shifted into carry flag and CF goes to MSB**
- Des: Register or Memory Location
- Bits are shifted 'count' no. of types

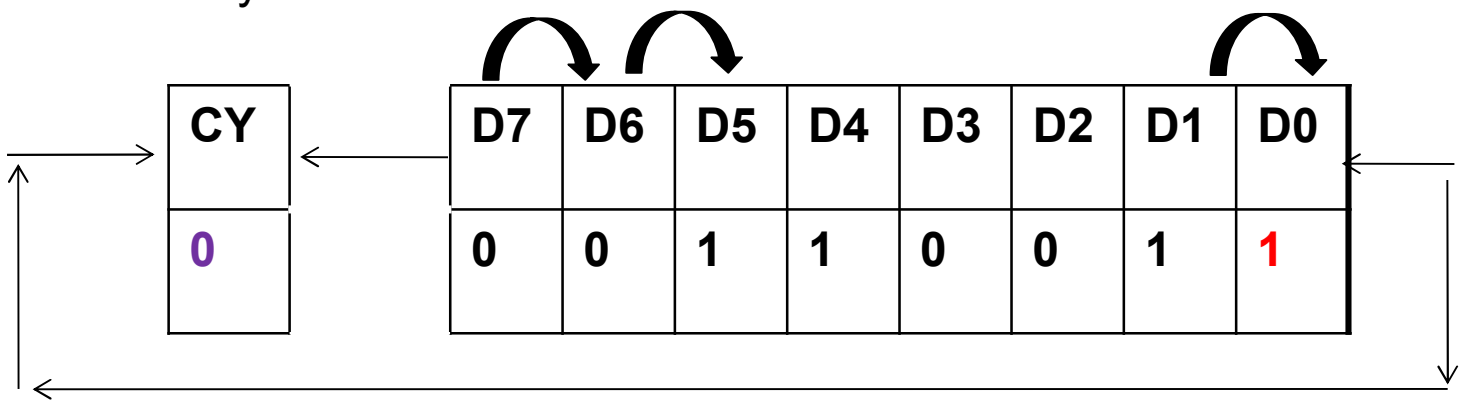
If count = 1,it is directly specified in the instruction as count

If count > 1, count is put in CL register & CL gives the count in the instruction.

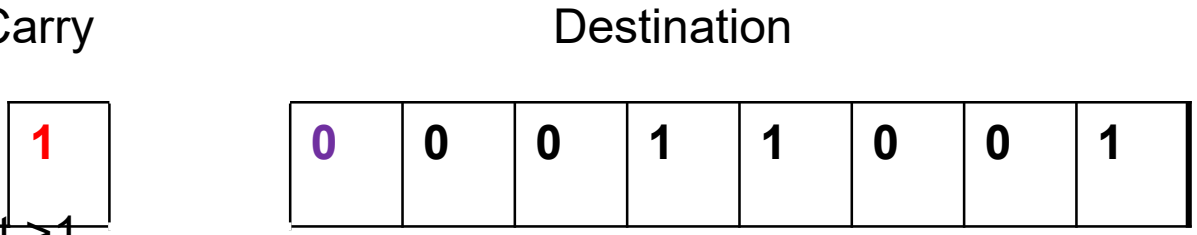
8086 - Instruction Set

Eg: Count = 1 ;RCR BL,1

Assume before operation BL=0011 0011 & CF=0



After operation BL=0001 1001 & CF=1



Eg: Count >1

```
MOV CL, 05H
RCR BL,CL
```

Processor Control Instructions

- These instructions control the processor itself.
- 8086 allows to control certain control flags that:

  i. Causes the processing in a certain direction
  ii. Processor synchronization if more than one microprocessor attached.

8086 - Instruction Set

Processor Control Instructions

FLAG OPERATIONS	
STC	: Set carry flag
CLC	: Clear carry flag
CMC	: Complement carry flag
STD	: Set Direction Flag
CLD	: Clear direction flag
STI	: Set Interrupt enable flag
CLI	: Clear interrupt enable flag

8086 - Instruction Set

Processor Control Instructions

NO OPERATION	
NOP	: No Opeartion
EXTERNAL SYNCHRONIZATION	
HLT	: Halt until interrupt or reset
WAIT	: Wait for TEST pin active
ESC	: Escape to external coprocessor
LOCK	: Lock bus during next instruction

Processor Control Instructions

1. For Carry Flag:

- **STC:**

  This instruction sets the carry flag to 1.

- **CLC:**

  This instruction clears the carry flag to 0.

- **CMC:**

  This instruction complements the carry flag.

NO OTHER FLAGS ARE AFFECTED

Processor Control Instructions

2. For Direction Flag:

- **STD:**

  This instruction sets the direction flag to 1.

- **CLD:**

  This instruction clears the direction flag to 0.

NO OTHER FLAGS ARE AFFECTED

Processor Control Instructions

2. For Interrupt Enable Flag:

- **STI:**

  This instruction sets the Interrupt Enable flag to 1.

- **CLI:**

  This instruction clears the Interrupt Enable flag to 0.

NO OTHER FLAGS ARE AFFECTED

External Hardware Synchronization Instructions

1. ESC
   - Instruction – prefix used to indicate that the current instruction is for the 8087 NDP.
   - When an instruction with ESC prefix is encountered, 8087 is activated and hence it executes the instruction.

External Hardware Synchronization Instructions

2. WAIT

- Instruction is used to synchronize 8086 with 8087 Co-Processor via $\overline{\text{TEST}}$ input pin of 8086.
- This instruction makes the µP check the TEST pin.

External Hardware Synchronization Instructions

3. LOCK

- This is 8086 instruction – prefix
- It prevents any external bus master from taking control of the system bus during execution of the instruction, which has a LOCK prefix.
- It causes 8086 to activate the $\overline{\text{LOCK}}$ signal so that no other bus master takes control of the system bus.

External Hardware Synchronization Instructions

4. NOP

- No operation is performed
- Requires 3 Tstates
- Used toinsert time delays

5. HLT

- Causes 8086 to stop fetching any more instructions.
- 8086 enters Halt State.

Program Execution Transfer Instructions

- These instructions cause **change in the sequence of the execution of instruction** or **cause a branch** in the program sequence.
  i. Unconditional Call
  ii. JMP – Conditional & Unconditional
  iii. INT – Software Interrupt

Program Execution Transfer Instructions

- This change can be through a **condition** or sometimes **unconditional**.
- The conditions are **represented by flags**.
- There are 2 main types of branching:
    i. Near Branch
    ii. Far Branch

Program Execution Transfer Instructions

I. Near Branch

- This is a **Intra- Segment branch**
- Needs **only IP** to be changed
- If the Near branch is in the **range of –128 to 128**, it is called as **short jump**

II. Far Branch

- This is a **Inter- Segment branch**
- ∴ **both CS & IP** needs to be changed

Program Execution Transfer Instructions

- The way of calculating the branch address provides us 4 addressing modes for indicating branch addresses

i. Intrasegment Direct
ii. Intrasegment Indirect
iii. Intersegment Direct
iv. Intersegment Indirect

8086 - Addressing modes for Program Memory

1. Intra-segment direct mode:

- The address to where the control is transferred is specified directly in the instruction as an immediate displacement value.

- The address to which the control is to be transferred lies in the **same segment** in which the control transfer instruction lies

8086 - Addressing modes for Program Memory

1. Intra-segment direct mode:

- Only **IP changes**, **CS does not change**

- The **effective address to which the control will be transferred is given by the sum of 8 or 16 bit displacement and current content of IP**.

Eg: JMP Label1

8086 - Addressing modes for Program Memory

2. Intra-segment indirect mode:

- The branch address is **specified in a register or a memory location** (in DS only)
- As intra seg only IP changes, CS does not change
- Eg: JMP Label1 [BX]
- IP ⟵ {DS:[BX], DS:[BX+1]}

8086 - Addressing modes for Program Memory

3. Inter-segment direct mode:

- The **new branch address** is **specified in the instruction**.
- In this mode, the address to which the control is to be **transferred is in a different segment.**
- This addressing mode provides a means of branching from one code segment to another code segment.

8086 - Addressing modes for Program Memory

3. Inter-segment direct mode:

- CS and IP of the destination address are specified directly in the instruction.
- Eg: JMP Label
- IP ⟵ offset address of Label
- CS ⟵ Segment address of Label

1. **JMP Des (Unconditional Jump):**

Intra Segment (Near) Jump:

- Jump address specified in two ways:

  i. Intra Segment Direct Jump: **Relative Jump**

  ii. Intra Segment Indirect Jump

1. JMP Des (Unconditional Jump):

Intra Segment (Near) Jump:

i. Intra Segment Direct Jump: **Relative Jump**

- The new branch location is **specified directly** in the instruction.
- The new addr is calculated by **adding 8/16 bit displacement to IP.**
- CS does not change

Eg: JMP Prev; JMP Next

1. JMP Des (Unconditional Jump):

Intra Segment (Near) Jump:

i. Intra Segment Direct Jump: **Relative Jump**

- A **+ve displacement** means that the **JUMP is ahead (forward)** of the program
- A **-ve displacement** means that the **JUMP is behind (backward)** of the program

Intra Segment (Near) Jump

ii. Intra Segment Indirect Jump:

- The new branch location is **specified indirectly through a register or a memory location (in DS only).**
- The value in **IP is replaced with a new value.**
- CS does not change

Eg: JMP WORD PTR [BX]; IP ⟵ *{DS:[BX], DS:[BX+1]}*

Inter Segment (Far) Jump

Jump address specified in two ways:

i. Intra Segment Direct Jump:

ii. Intra Segment Indirect Jump:

Inter Segment (Far) Jump

Jump address specified in two ways:

i. Intra Segment Direct Jump:

- The new branch location is **specified directly** in the instruction.
- Both **IP and CS gets new values.**

Eg: Assume NextSeg; Label pointing to an instruction in a different segment.

```
JMP NextSeg;
```

Inter Segment (Far) Jump

ii. Intra Segment Indirect Jump:

- The new branch location is **specified indirectly through a register or a memory location (in DS only).**
- Both **IP and CS gets new values.**

Eg: JMP DWORD PTR [BX];

IP *{DS:[BX], DS:[BX+1]}*, CS *{DS:[BX+2], DS:[BX+3]}*

2. JCondition Des (Conditional Jump):

- This is a conditional branch instruction.
- All the conditional jumps follow some conditional statements or any instruction that affects the flag.

Sr. No.	DESCRIPTION	Jump Condition
1.	Carry	CF =1
2.	Not Carry	CF = 0
3.	Equal or Zero	ZF = 1
4.	Not Equal or Not Zero	ZF = 0
5.	Parity or Even Parity	PF = 1
6.	Not Parity or Odd Parity	PF = 0

2. JCondition Des (Conditional Jump):

- If the **condition is TRUE**,then it is similar to an **INTRA- Segment Direct Jump**.
- If the **condition is FALSE**, then the branch does not take place and the **next sequential instruction is executed**.
- The destination must be in the range of –128 to 128 from the address of the instruction (ie only **SHORT Jump)**
- **Eg: JNC Next**

8086 - Instruction Set

Conditional Jump Table

Sr. No.	Mnemonic	DESCRIPTION	Jump Condition
COMMON OPERATIONS			
1.	JC	Carry	CF =1
2.	JNC	Not Carry	CF = 0
3.	JE/JZ	Equal or Zero	ZF = 1
4.	JNE/JNZ	Not Equal or Not Zero	ZF = 0
5.	JP/JPE	Parity or Even Parity	PF = 1
6.	JNP/JPO	Not Parity or Odd Parity	PF = 0

8086 - Instruction Set

Conditional Jump Table

Sr. No.	Mnemonic	Meaning	Jump Condition
SIGNED OPEARTIONS			
7.	JO	Overflow	OF = 1
8.	JNO	Not Overflow	OF = 0
9.	JS	Sign	SF = 1
10.	JNS	Not Sign	SF = 0
11.	JL/JNGE	Less	(SF ExOr OF) = 1
12.	JGE/JNL	Greater or Equal	(SF ExOr OF) = 0
13.	JLE/JNG	Less or Equal	((SF ExOr OF) + ZF) = 1
14.	JG/JNLE	Greater	((SF ExOr OF) + ZF) = 0

8086 - Instruction Set

Conditional Jump Table

Sr. No.	Mnemonic	Meaning	Jump Condition
UNSIGNED OPERATIONS			
15.	JB/JNAE	Below	CF = 1
16.	JAE/JNB	Above or Equal	CF = 0
17.	JBE/JNA	Below or Equal	(SF ExOr ZF) = 1
18.	JA/JNBE	Above	(SF ExOr OF) = 0

8086 - Instruction Set

3. CALL Des: Unconditional Call

- This instruction is used to call a subroutine or function or procedure with intention of coming back to the main program.

- 8086 saves the address of the next instruction after CALL onto stack before branching to the subroutine

- Two types of Call: i) Near CALL

  ii) Far CALL

Intra Segment (Near) CALL

- The new subroutine called must be in the same segment (hence intra - segment)
- The CALL address: Specified directly in the instruction or through registers or memory location
- Sequence executed for a Near CALL
  i. 8086 will **PUSH Current IP** into the Stack.
  ii. **Decrement SP by 2**
  iii. **New Value** loaded into **IP**
  iv. **Control transferred** to a subroutine within the same segment.

Eg: CALL subADD ***;{SS:[SP-1], SS::[SP-2]}*** ← ***IP,***
; SP ← ***SP-2***
; IP ← ***New Offset Address of subAdd***

Intra Segment (Near) CALL

- Sequence executed for a Near CALL

i. 8086 will **PUSH Current IP** into the Stack.

ii. **Decrement SP by 2**

iii. **New Value** loaded into **IP**

iv. **Control transferred** to a subroutine within the same segment.

Eg: CALL subADD ***;{SS:[SP-1], SS::[SP-2]}*** ***← IP,***
; SP ← SP-2
; IP ← New Offset Address of subAdd

Intra Segment (Far) CALL

- The new subroutine called is in another segment (hence inter - segment)
- CS & IP both get new values
- The CALL address: Specified directly in the instruction or through registers or memory location

8086 - Instruction Set

Intra Segment (Far) CALL

- Sequence executed for a Far CALL Eg: CALL subADD

i. **PUSH CS** into the Stack. ;*{SS:[SP-1], SS::[SP-2]}* ~~← *CS ,*~~

ii. **Decrement SP by 2** *; SP* ~~← *SP-2*~~

iii. **PUSH Current IP** into the Stack. ;*{SS:[SP-1], SS::[SP-2]}* ~~← *IP,*~~

iv. **Load CS** with new segment address; *CS* *New ~~Offset~~ Address of subAdd*

v. **Load IP** with new segment address ~~←; *IP*~~ *New Offset Address of subAdd*

vi. **Control transferred** to a subroutine in the new segment

4. RET:

- Causes the control to return to the main program from subroutine
- Every CALL instruction should have a RET
- **Intrasegment RET**

  Eg: RET ;IP ← *SS:[SP], SS::[SP+1]*

  ;SP ← *SP+2*

- **Intersegment RET**

  Eg: RET ;IP ← *SS:[SP], SS::[SP+1]*

  ;CS ← *SS:[SP+2], SS::[SP+3]*

  ;SP ← *SP+4*

Iteration Control Instructions

- These instructions cause a series of instructions to be executed repeatedly
- The no.of iterations is loaded in CX reg.
- CX is decremented by 1, after every iteration
- Iterations occur until CX = 0
- The max difference between the address of the instruction and the address of the Jump can be 127.

Iteration Control Instructions

1) LOOP Label:

- This instructionis used to repeat a series of instructions CX no of times.
- Jump to specified label if CX not equal to 0; and decrement CX

Eg:
```
        MOV CX,40H
BACK:   MOV AL,BL
        ADD AL, BL
          "
          "
           "
        MOV BL, AL
        LOOP BACK;  Repeat from the instruction having
                    BACK label if  CX not equal to 0
                ;   CX  <---- CX-1
```

Iteration Control Instructions

2) LOOPE/LOOPZ Label (Loop on Equal/ Loop on Zero)
Jump to specified label if CX not equal to 0 & only if ZF = 1; and decrement CX

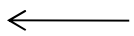
Eg:
```
        MOV CX,40H
BACK:   MOV AL,BL
        ADD AL, BL
          "
          "
          "
        MOV BL, AL
        LOOPZ BACK;  Repeat from the instruction having BACK
                     label if CX not equal to 0 & ZF = 1
                   ;  CX <---- CX-1
```

Iteration Control Instructions

3) LOOPNE/LOOPNZ Label (Loop on NOT Equal/ Loop on NO Zero)

Jump to specified label if CX not equal to 0 & only if ZF = 0; and decrement CX

```
Eg:        MOV CX,40H
   BACK:   MOV AL,BL
           ADD AL, BL
                "
                "
                "
           MOV BL, AL
           LOOPNZ BACK;  Repeat from the instruction having BACK label
                         if CX not equal to 0 & ZF = 1
                    ;    CX          CX-1
```



Interrupt Control Instructions

1. INT Type

- This instruction causes an interrupt of the given type.
- Causes 8086 to CALL a far procedure
- The “Type” is a no between 0 ……255
- Software interrupts
- Address of the procedure is taken from memory (IVT Table) whose address is 4* Type no

Interrupt Control Instructions

1. INT Type

- Following action takes place
  - i. **PUSH Flag** Register onto the Stack & SP decremented by 2
  - ii. **IF &TF cleared**
  - iii. **PUSH CS** onto the Stack & SP decremented by 2
  - iv. **PUSH IP** onto the Stack & SP decremented by 2
  - v. **New value of IP** taken from location type x4
  - vi. **New value of CS** taken from location (type x4) + 2
  - vii. **Execution of the ISR** begins from the address formed by the new values of CS & IP

- Eg: INT 1 ;IP = {[00004] & [00005]}; CS = {[00006] & [00007]};
  as 1x4=00004H

Interrupt Control Instructions

2. INTO (Interrupt on Overflow)

- This instruction causes an interrupt type 4, only if Overflow Flag (OF) is set
- Same action takes place and ctrl transferred to location pointed by 00010H

Interrupt Control Instructions

3. IRET (Return from ISR)

- This instruction causes 8086 to return to the main program.
- Used at the end of the ISR
- The following action takes place
  i. POP IP from the stack: SP incremented by 2
  ii. POP CS from the stack: SP incremented by 2
  iii. POP Flag Register from the stack: SP incremented by 2.

  In all SP incremented by 6
  - Execution of the Main Program continues from the address formed by CS & IP

String Instructions

- String is a **series of bytes** stored **sequentially in the memory location**.
- String instructions operate on strings.
- By using these string instructions, the **size of the program is considerably reduced.**

String Instructions

- The **Source element** is taken from **Data Segment using** the **SI register**.
- The **Desitination element** is in **Extra Segment pointed** by the **DI register.**
- **SI &/or DI** are **incremented /decremented** after each operation depending upon the **direction flag "DF" in Flag Register**

String Instructions

1. MOVS: MOVSB/MOVSW (Move String)

- Used to **transfer a word/byte** from **data segment to extra segment**
- Offset : **source is in SI and destination is in DI**
- **SI &/or DI** are **incremented /decremented** after each operation depending upon the **direction flag "DF" in Flag Register**
- Eg: MOVSB ; *ES:[DI]* ⟵ *DS:[SI]*

  ; *SI* ⟵ *SI* ± *1* & *DI* ⟵ *DI* ± *1: depending on DF*

  MOVSW ;{*ES:[DI], ES:[DI+1]*} ⟵ *DS:[SI], DS:[SI+1]*

  ; *SI* ⟵ *SI* ± *2* & *DI* ⟵ *DI* ± *2: depending on DF*

String Instructions

2. LODS: LODSB/LODSW (Load String)

- Load AL or AX Reg with a byte or word from data segment
- Offset of the source is in SI
- **SI** is **incremented /decremented** after each operation depending upon the **direction flag "DF" in Flag Register**
- Eg: LODSB ; AL ⟵ DS:[SI]

  ; SI ⟵ SI ± 1 : depending on DF

  LODSW ;AL ⟵ DS:[SI], AH ⟵ DS:[SI+1]

  ; SI ⟵ SI ± 2 : depending on DF

String Instructions

3. STOS: STOSB/ STOSW (Store String)

- Store AL or AX Iinto a byte or word in Extra segment
- Offset of the source in Extra Segemnt is in DI
- **DI** is **incremented /decremented** after each operation depending upon the **direction flag "DF" in Flag Register**
- Eg: STOSB, STOSW

String Instructions

4. CMPS: CMPSB/ CMPSW (Compare String)

- Compare a byte or word in the Data Segment with a byte or word in the Extra Segemnt
- Offset of the byte or word in Data Segment is in SI
- Offset of the byte or word in Extra Segemnt is in DI
- **SI &/or DI** is **incremented /decremented** after each operation depending upon the**direction flag "DF" in Flag Register**
- Comparision done by substraction ([ES] – [DS]), result not stored; flag bits affected
- Eg: CMPSB, CMPSW

String Instructions

5. SCAS: SCASB/ SCASW (String Compare Accumulator)

- Compare the contents of AL or AX with a byte or word in the Extra Segemnt
- Offset of the byte or word in Extra Segemnt is in DI
- **DI** is **incremented /decremented** after each operation depending upon the **direction flag "DF" in Flag Register**
- Comparision done by substracting byte or word from AL or AX; result not stored; flag bits affected
- Eg: SCASB, SCASW

String Instructions

6. REP(Repeat)

- Instruction – prefix used in string instructions
- Canbe used only with string instructions
- Causes the instruction to be repeated CX number of times
- After every execution, SI and **DI reg are incremented /decremented** after each operation depending upon the**direction flag "DF" in Flag Register and CX is decremented**

String Instructions

7. REPZ/REPE (Repeat on ZERO/ Equal)
   - Conditional repeat Instruction – prefix
   - Causes the instruction to be repeated CX number of times till ZF = 1

8. REPNZ/REPNE (Repeat on No ZERO/ Equal)
   - Conditional repeat Instruction – prefix
   - Causes the instruction to be repeated CX number of times till ZF = 0

Thank You

8086 - Instruction Set

Intra Segment (Far) CALL

- The new subroutine called is in another segment (hence inter - segment)
- CS & IP both get new values
- The CALL address: Specified directly in the instruction or through registers or memory location
- Sequence executed for a Far CALL Eg: CALL subADD

i. **PUSH CS** into the Stack. ;*{SS:[SP-1], SS::[SP-2]}* ← *CS ,*

ii. **Decrement SP by 2** *; SP* ← *SP-2*

iii. **PUSH Current IP** into the Stack. ;*{SS:[SP-1], SS::[SP-2]}* ← *IP,*

iv. **Decrement SP by 2** *; SP* ← *SP-2*

v. **Load CS** with new segment address ; *CS* ← *New Offset Address of subAdd*

vi. **Load IP** with new segment address ; *IP* ← *New Offset Address of subAdd*

vii. **Control transferred** to a subroutine in the new segment.