Instruction set

- Data transfer Instructions
- Arithmetic Instructions
- Logical Instructions

Instruction set

- Data transfer Instructions

1. General Purpose grp
2. Input/Output
3. Address Object
4. Flag transfer

1)General Purpose grp

MOV

PUSH

POP

XCHG

XLAT

Data transfer group

MOV: Copy a word or a byte

MOV destination,source

1. Memory,accumulator eg mov [SI],AL
2. Accumulator,memory
3. Register,register
4. Register,memory
5. Memory,register
6. Register,immediate
7. memory.,immediate
8. Seg-reg,reg-16 e.g. mov DS,AX
9. Seg-reg,mem-16 e.g. mov DS,[SI]
10. Reg-16,seg-reg
11. Memory,seg-reg

Data transfer group

MOV: Copy a word or a byte

MOV destination,source

1. Memory,accumulator eg mov [SI],AL

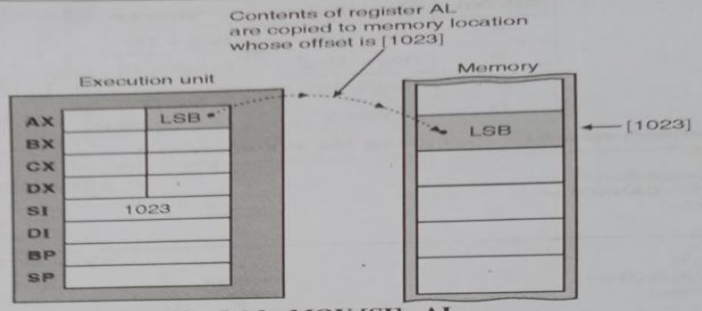


Fig. 5.6.2 : MOV [SI], AL

In case of 16-bit transfer the contents of register AL are copied to memory location whose offset is [1023] and contents of register AH are copied to location whose offset is [1024]. This is indicated in Fig 5.6.3.

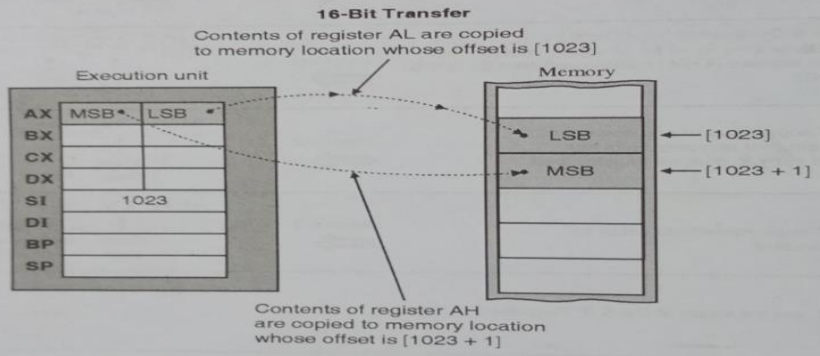


Fig. 5.6.3 : MOV [SI], AX

Let the contents of CS → 2105 H, IP → 187A H, DS → 2314 H and AL → 20 H

Generation of physical address (PA)

2 3 1 4 0 H	→	Contents of DS
+ 1 0 2 3 H	→	Offset of memory location
2 4 1 6 3 H	→	Physical address

The contents of Reg. AL will be transferred at the location whose physical address is given by 24163 H.

	Before execution	After execution	
Code Segment (CS)	2105 H	2105 H	
Instruction Pointer (IP)	**187A H**	**187C H**	→ Note the change
Data Segment (DS)	2314 H	2314 H	
Register AL	20 H	20 H	
Memory location 24163 H	**FF H**	**20 H**	→ Note the change
Figure	Refer Fig. 5.6.4	Refer Fig. 5.6.5	

Data transfer group

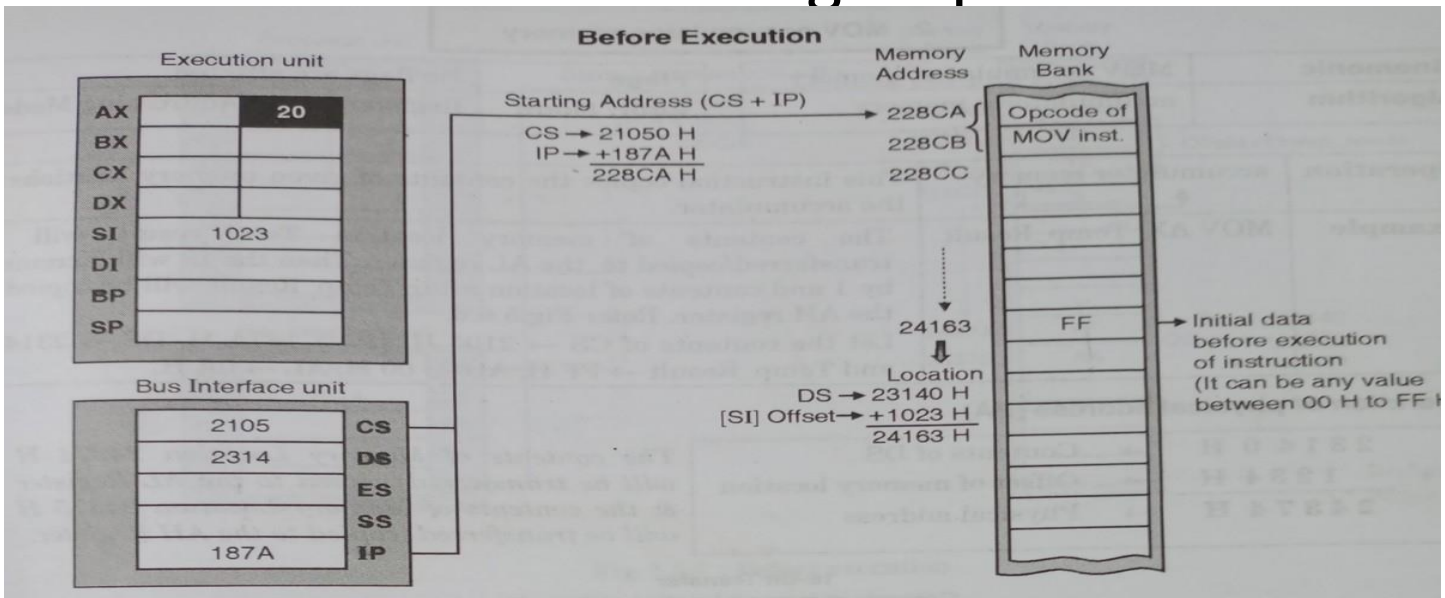


Fig. 5.6.4 : Before execution

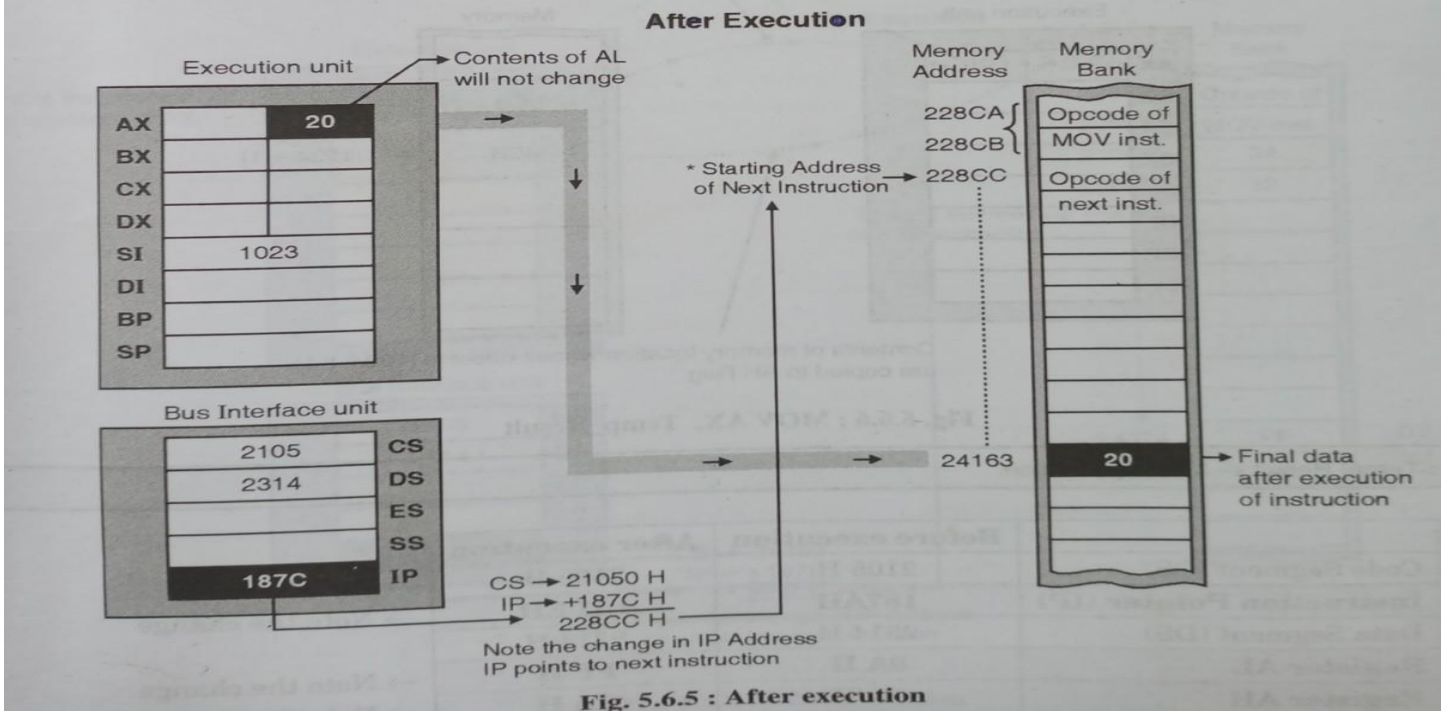


Fig. 5.6.5 : After execution

Data transfer group

MOV destination,source

2. MOV accumulator,memory eg mov AX,Temp_Result

and Temp_Result → FF H, AH → 00 H, AL → 0A

Generation of physical address (PA)

	23140 H	→	Contents of DS
+	1234 H	→	Offset of memory location
	24374 H	→	Physical address

The contents of Memory Locati
will be transferred/copied to the
& the contents of Memory Locat
will be transferred/copied to the A

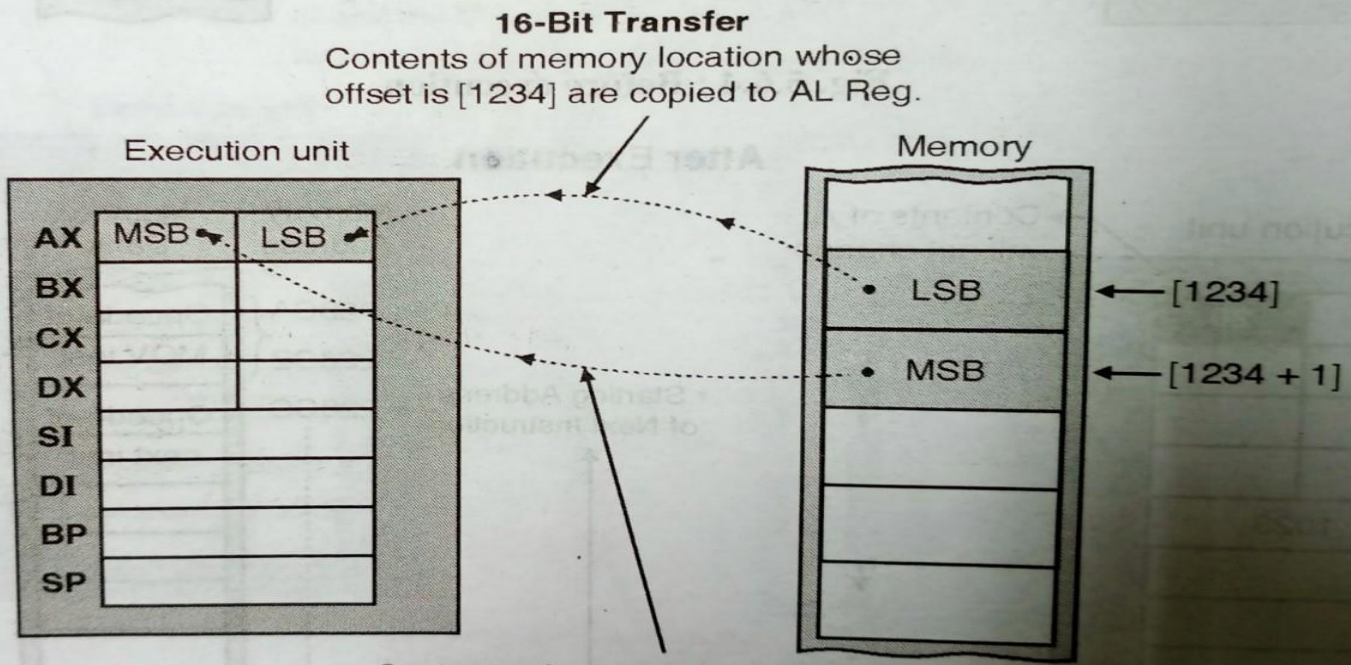


Fig. 5.6.6 : MOV AX, Temp_Result

Data transfer group

MOV destination,source

3. MOV destination register,source register

eg mov AX,BX

Data transfer group

MOV destination,source

4. MOV register,memory eg mov CX,COUNT[DI]

	MOV CX, COUNT [DI]	This instruction copies the contents of the CX register. Contents of first location are copied to CL and contents of the next location to CH register.

Note : Count is any variable which is defined while programming.

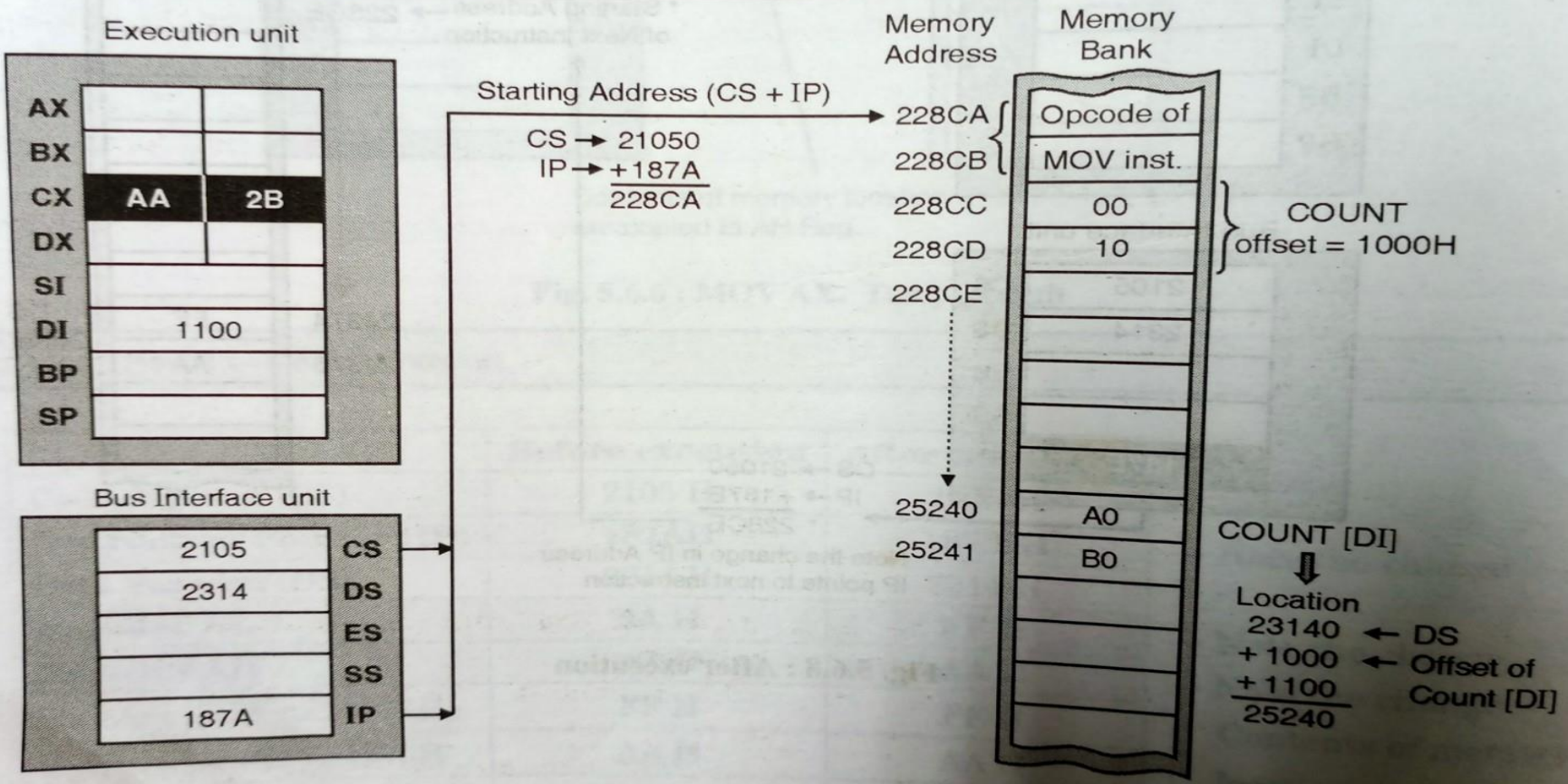


Fig. 5.6.9 : Before execution

Data transfer group

MOV destination,source

4. MOV register,memory eg mov CX,COUNT[DI]

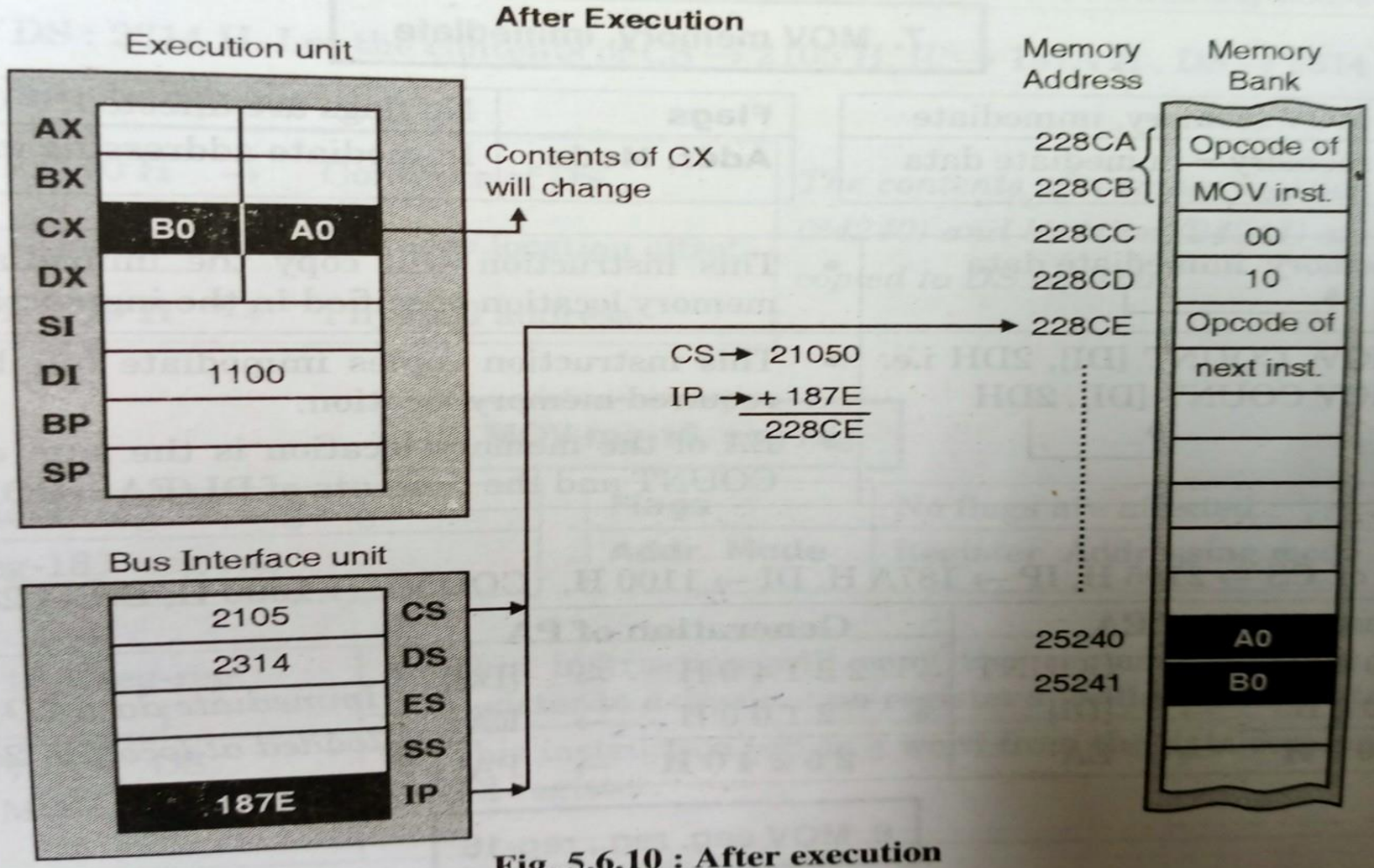


Fig. 5.6.10 : After execution

Data transfer group

MOV destination,source

5. MOV memory, register eg mov COUNT[DI], CX

Data transfer group

MOV destination,source

6. MOV register,immediate eg MOV CL,02h

7. MOV memory, immediate eg MOV COUNT[DI],2DH

8.MOV seg reg.,reg-16 eg MOV DS,AX

9. MOV seg reg.,mem-16 eg MOV DS,[SI]

10. MOV reg-16, seg reg. eg MOV AX,DS

11.MOV memory, seg reg. eg MOV count,DS

Data transfer group

PUSH:Push word onto stack

PUSH Source

SP=SP-2

SS:[SP] (Top of the stack)=Operand

- This instruction decrements the stack pointer by 2 and copies a word from a specified source to the location in the stack segment where stack pointer points.
- The source of operand (16 bit data to be stored on stack) can be a general purpose register, flag register, segment register or memory.
- The stack segment register and stack pointer must be initialised before using this instruction.
- PUSH can be used to save data on the stack so that it will not be destroyed by a execution of successive instructions.

Data transfer group

PUSH:Push word onto stack

Example : PUSH AX

Now we will see detailed analysis of what exactly happens when instruction PUSH AX ex

PUSH AX

SP → SP – 2

- SS → Copy word from AX register to location in Stack segment (SS) where SP points.
- Let the address of SS = 4523 H, SP = 2586 H , Let contents of CS → 2105 H and o 187A H. Fig. 5.6.11 shows this

eneration of phy sical address

lote : In this case physical address is the location of top of stack, where SP points before execution.

	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

The contents of stack pointer is decremented b The contents of AH register will be copie location (477B5). Then Stack pointer decremented by 1 and contents of AL register copied to memory locations (477B4).

Let the contents of AH = FF H and contents of AL = AA H.

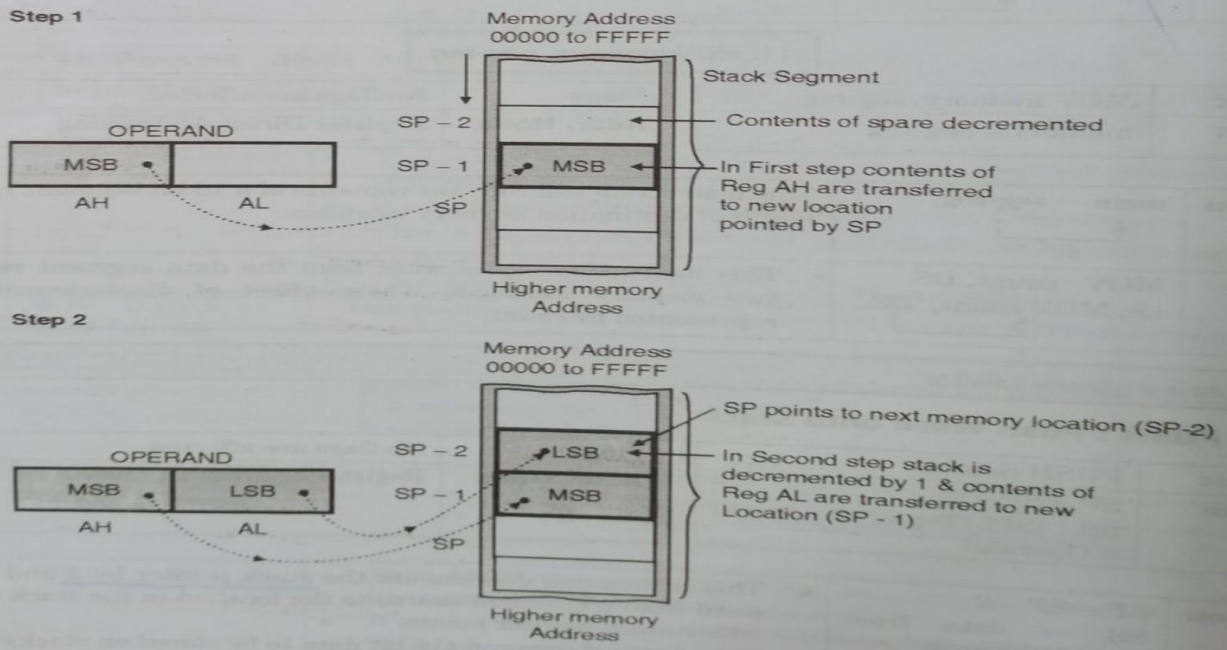


Fig. 5.6.11

Data transfer group

PUSH:Push word onto stack

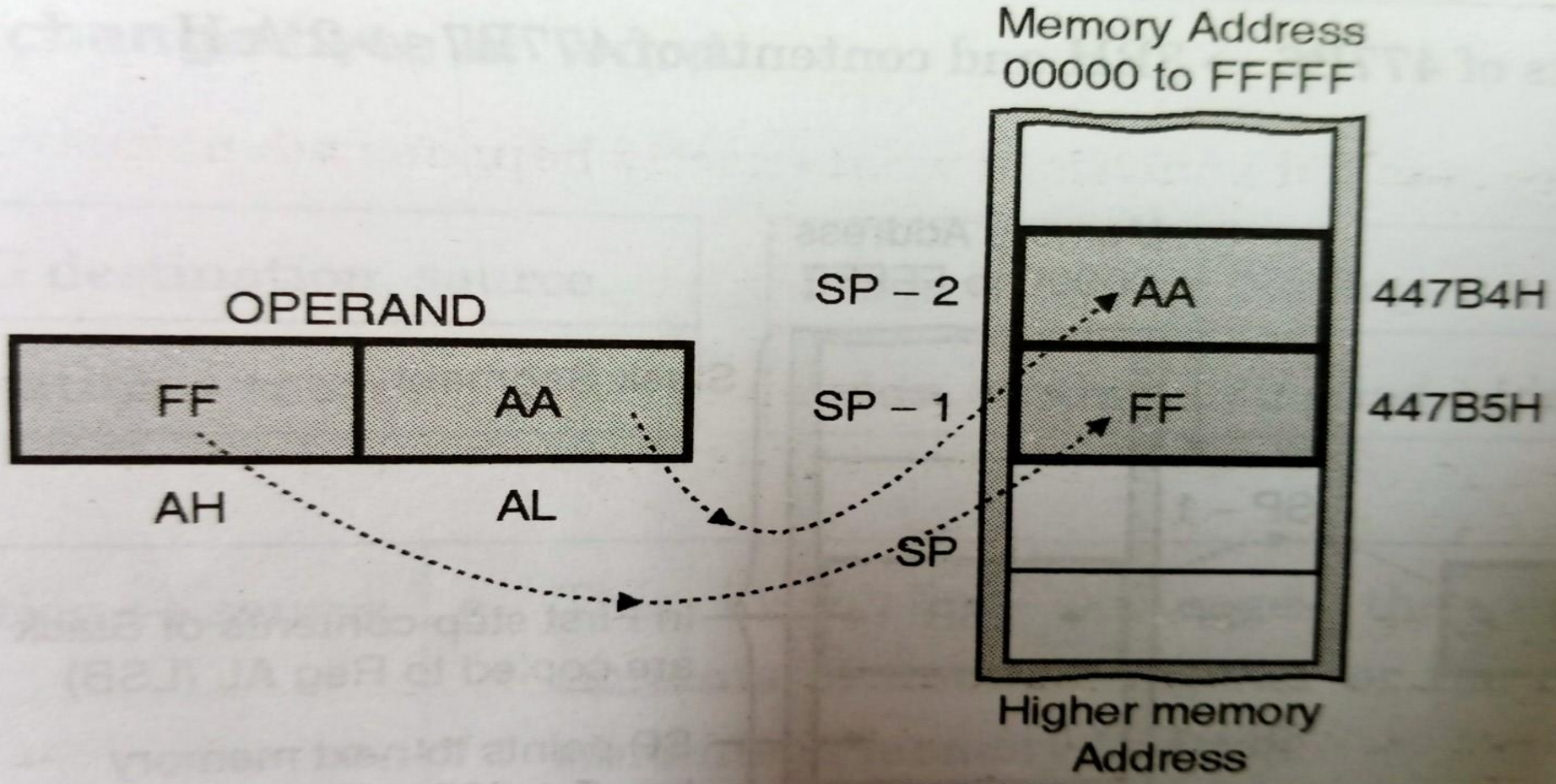


Fig. 5.6.12 : PUSH AX

Data transfer group

POP:Pop word off stack

POP destination or POP Operand

Operand=SS:[SP](top of stack)SP=SP+2

- This instruction copies a word from stack segment in memory to a destination specified in the instruction.
- The destination can be a general purpose register,flag register,a segment register or a memory location.
- The data in the stack pointer is not changed.
- After the word is copied to specified destination the stack pointer is automatically incremented by 2 to point to next word on stack.

POP:P

Let the contents of 477B6 → 3BH and contents of 477B7 → 2 A H.

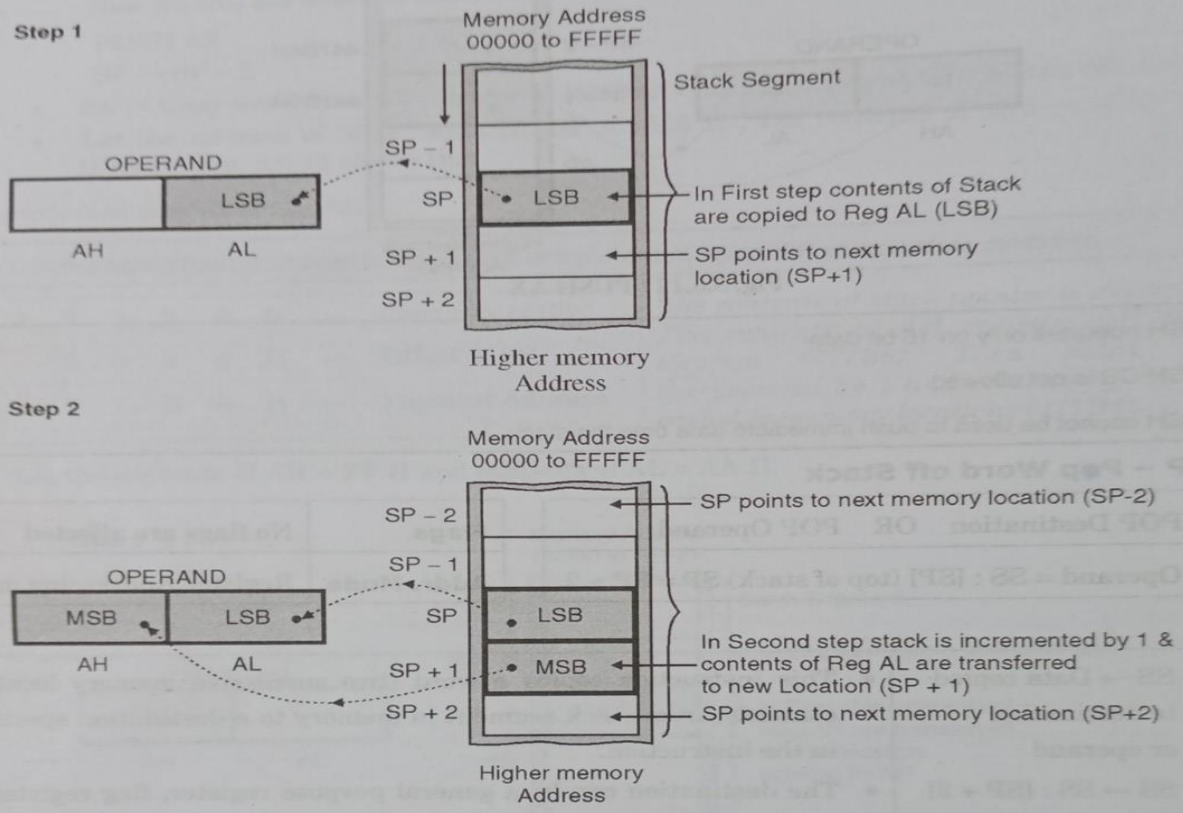


Fig. 5.6.13

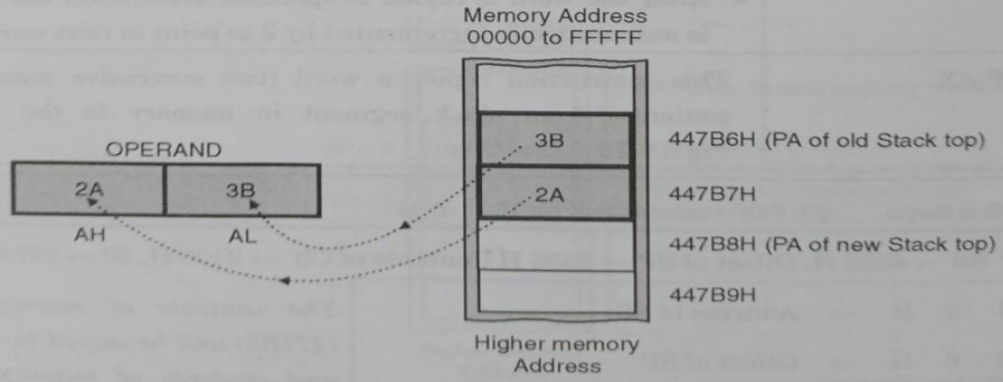


Fig. 5.6.14 : POP AX

Data transfer group

XCHG: exchange byte or word

XCHG destination,source

XCHG AX,BX

- This instruction exchanges the contents of a register with contents of another register or the contents of a register with contents of memory location.
- The register can be 8/16 bit .
- XCHG cannot directly exchange the contents of two memory locations.

  eg XCHG [1234],[5068] this transfer not possible
- The source and destination must both be words or they both must be bytes.

Data transfer group

XCHG: exchange byte or word

	AX ↔ BX	This instruction will exchange the word in AX word in BX register. Contents of AL ↔ Contents of BL, Contents of AH

te : 1) The segment registers cannot be used as registers in this instruction. The memory as segment registers.

2) Since immediate operands cannot be changed neither operand in this instruction ca

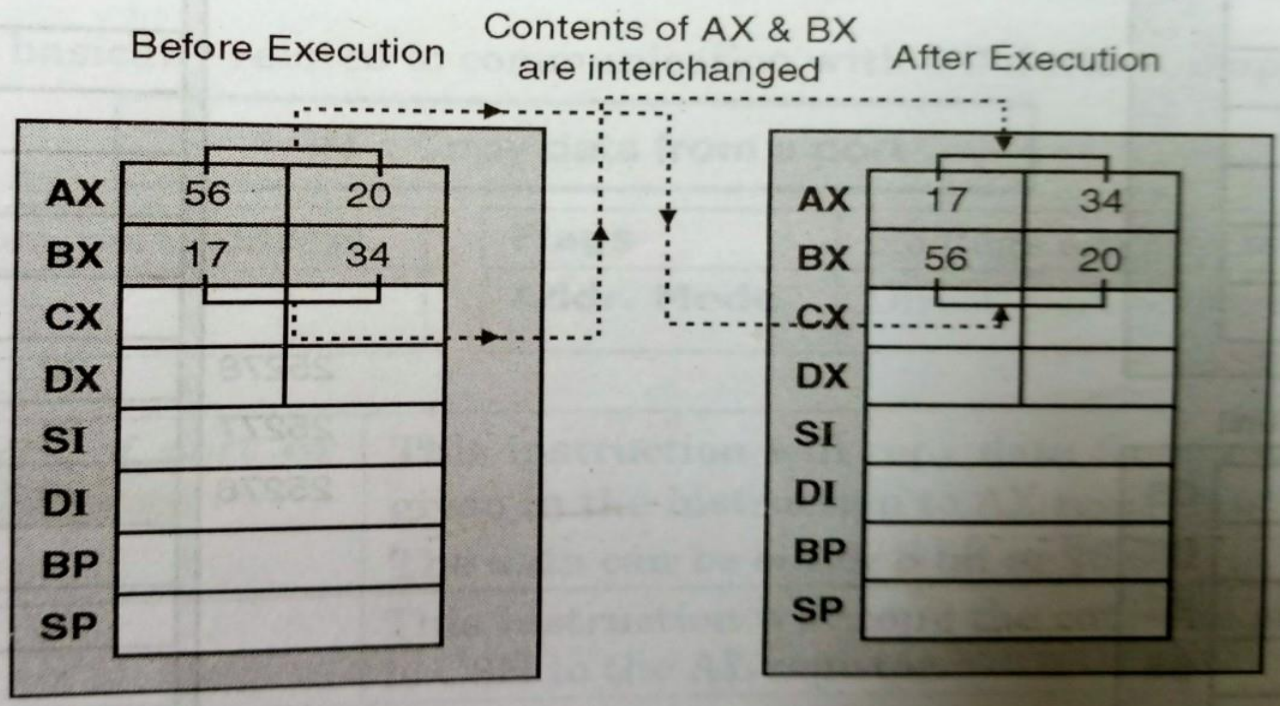


Fig. 5.6.15 : XCHG AX, DX

Data transfer group

XLAT/XLATB: Translate or replace byte

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] to AL register.
- This instruction is used to translate a byte from one code to another code.
- This concept is used to convert BCD to seven segment code.

XLAT/

eration	AL ← DS : [BX + 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] to the AL register. • Here contents of AL before execution acts as index to the desired location in lookup table. • This instruction is used to translate a byte from one code to another code. • Mostly this concept is used to convert BCD to seven segment code or ASCII to EDCBIC code conversion

the address of DS be 2314 H, Let the offset of BX be 2134 H, Let the contents of CS be 2105 H and of e 187A H, Let the contents of AL be 05 H.

eration of physical address

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

25276 H is the base of look up table. The contents at this location are transferred to AL as shown in 5.6.16(b).

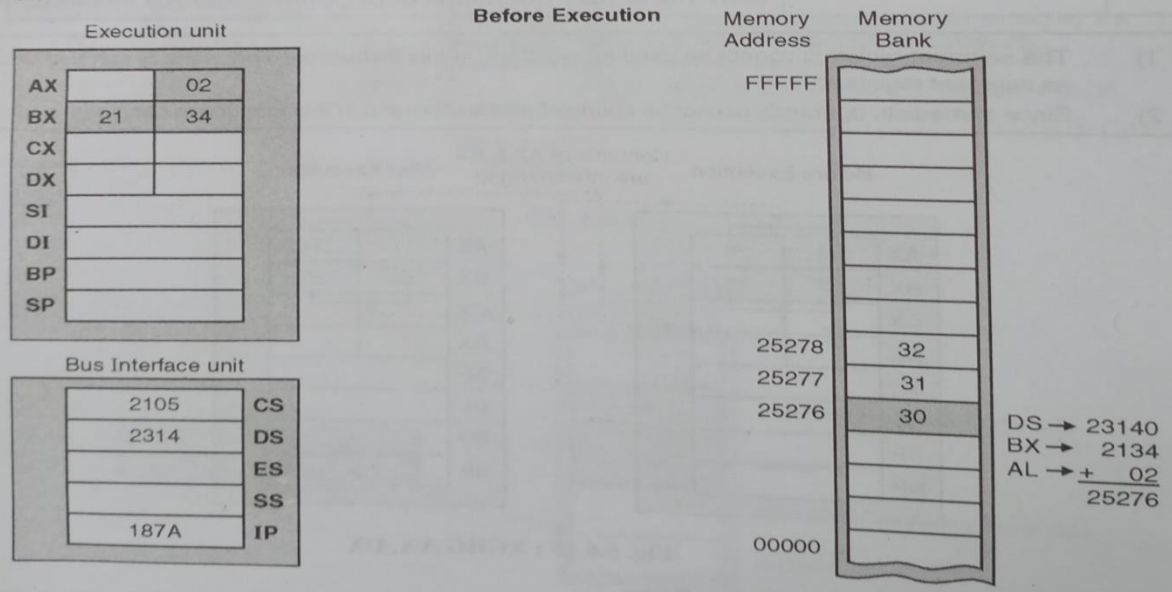


Fig. 5.6.16(a) : XLAT

Data transfer group

XLAT/XLATB: Translate or replace byte

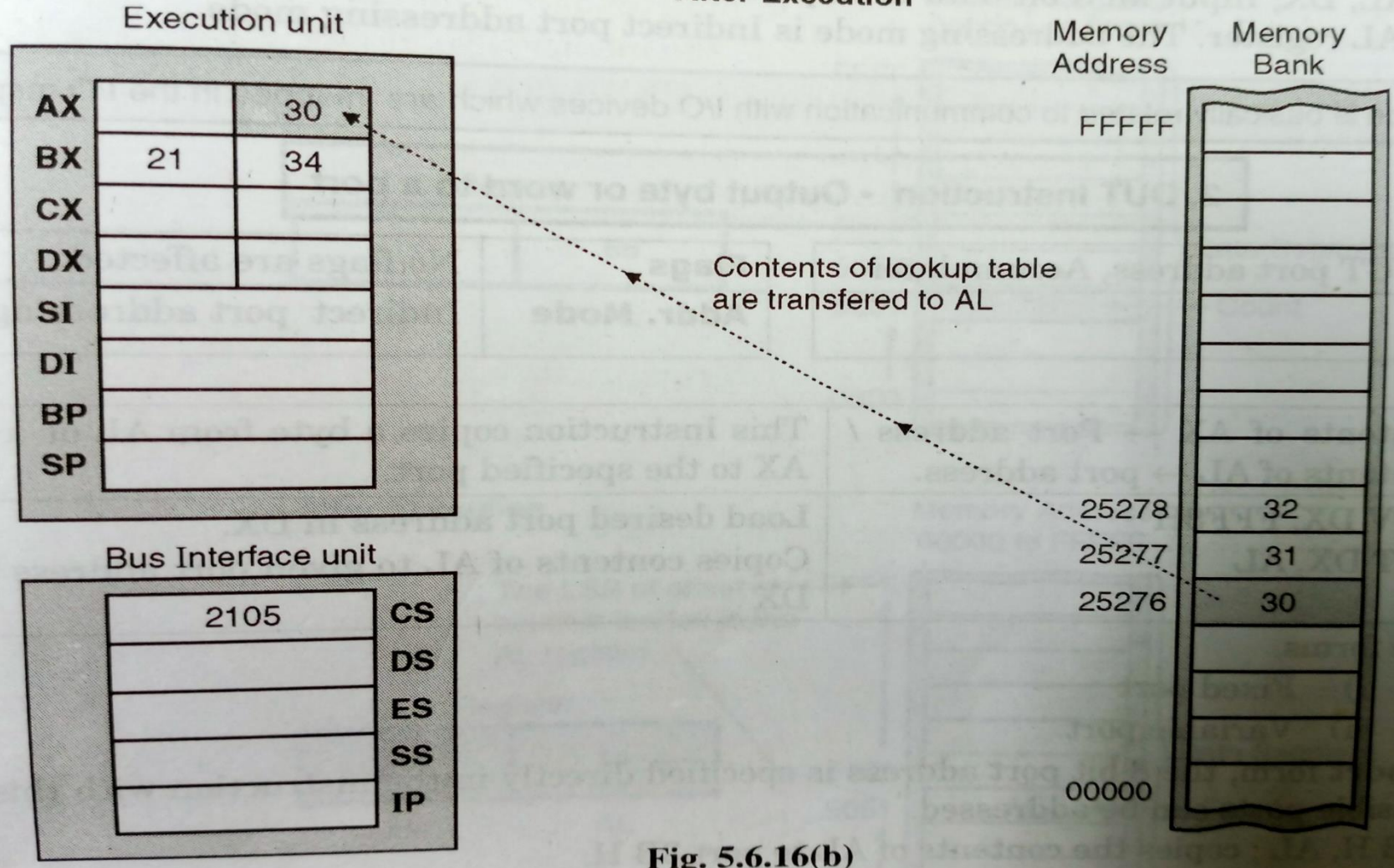


Fig. 5.6.16(b)

Data transfer group:Input/Output

IN: Copy data from a port

IN accumulator, port address

e.g. IN Al,C8h

- This instruction will copy data from a port whose address is given in instruction to AX register or AL register. The data can be either 8 bit or 16 bit.
- The IN register has 2 possible formats.

  i) Fixed port

  ii) variable port
- For **Fixed port**, the port address is directly specified in the instruction.
- For **Variable port,** the port address is loaded into DX register before IN instruction.Since,DX is a 16 bit register, the port address can be any no between 0000H to FFFFH, so total 65,536 ports are possible.

  e.g. MOV DX, 0FF0H

  IN AL,DX

Data transfer group:Input/Output

OUT: Output byte or word to a port

OUT port address, accumulator

e.g. MOV DX,FFF8h

OUT DX,AL

- This instruction copies a byte from AL or a word from AX to the specified port.
- OUT has 2 possible forms.
  - i) Fixed port
  - ii) variable port
- For **Fixed port**, the 8 bit port address is directly specified in the instruction.

  e.g. OUT 3Bh,AL
- For **Variable port,** the contents of AL or AX will be copied to the port at an address contained in DX. The DX register should be loaded with the desired port address.

  e.g. MOV DX, 0FF0H

Data transfer group:Address object

1) **LEA: Load Effective Address**

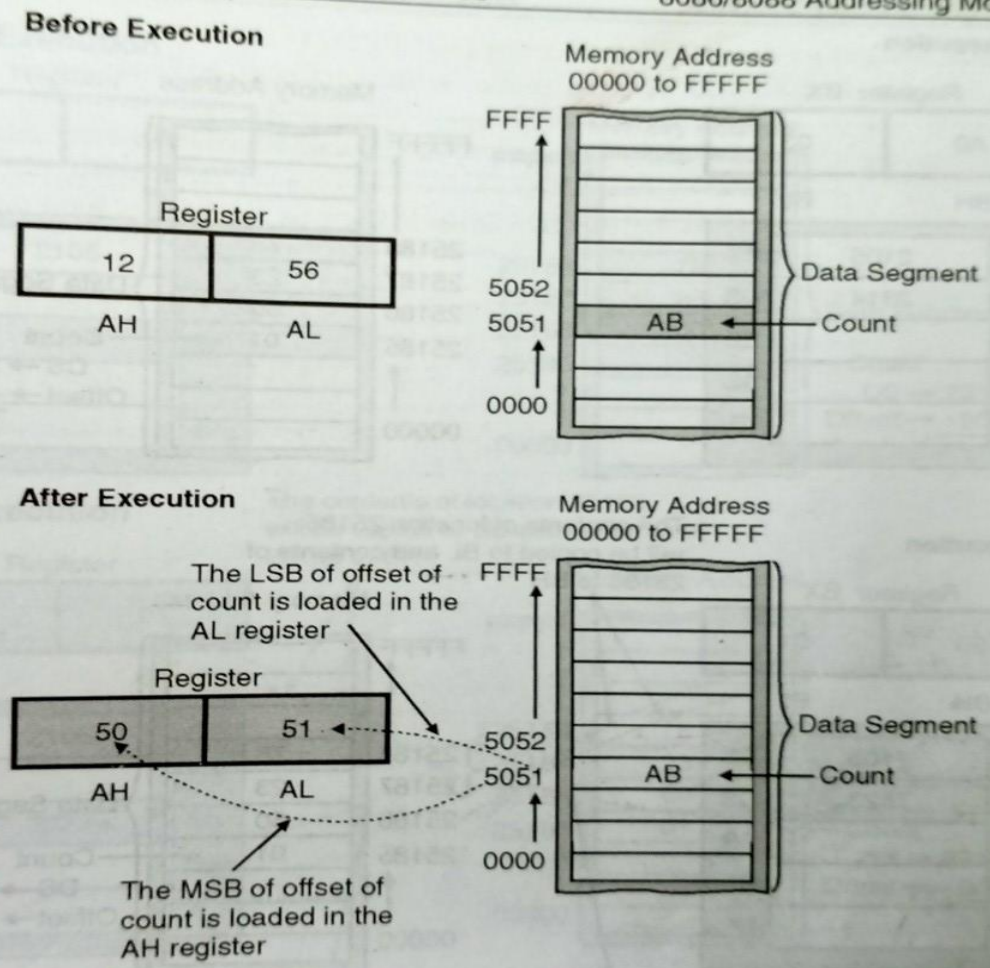
LEA register,source

e.g. LEA AX, COUNT

- This instruction determines the offset of variables or memory location named as source and puts the offset in the indicated 16 bit register.

Data transfer group:Address object

1) **LEA: Load Effective Address**



Note : The Offset of variable count is loaded in the AX register, not its contents

Fig. 5.6.17 : LEA AX, count

Data transfer group:Address object

2) LDS : Load pointer with DS(Load register and DS with words from memory)

LDS register,source

e.g. LDS BX, COUNT

- This instruction is 2 byte instruction which copies a word from two memory locations into register specified in the instruction. It then copies a word from the next two memory locations into the DS register.

Data transfer group:Address object

2) LDS : Load pointer with DS(Load register and DS with words from

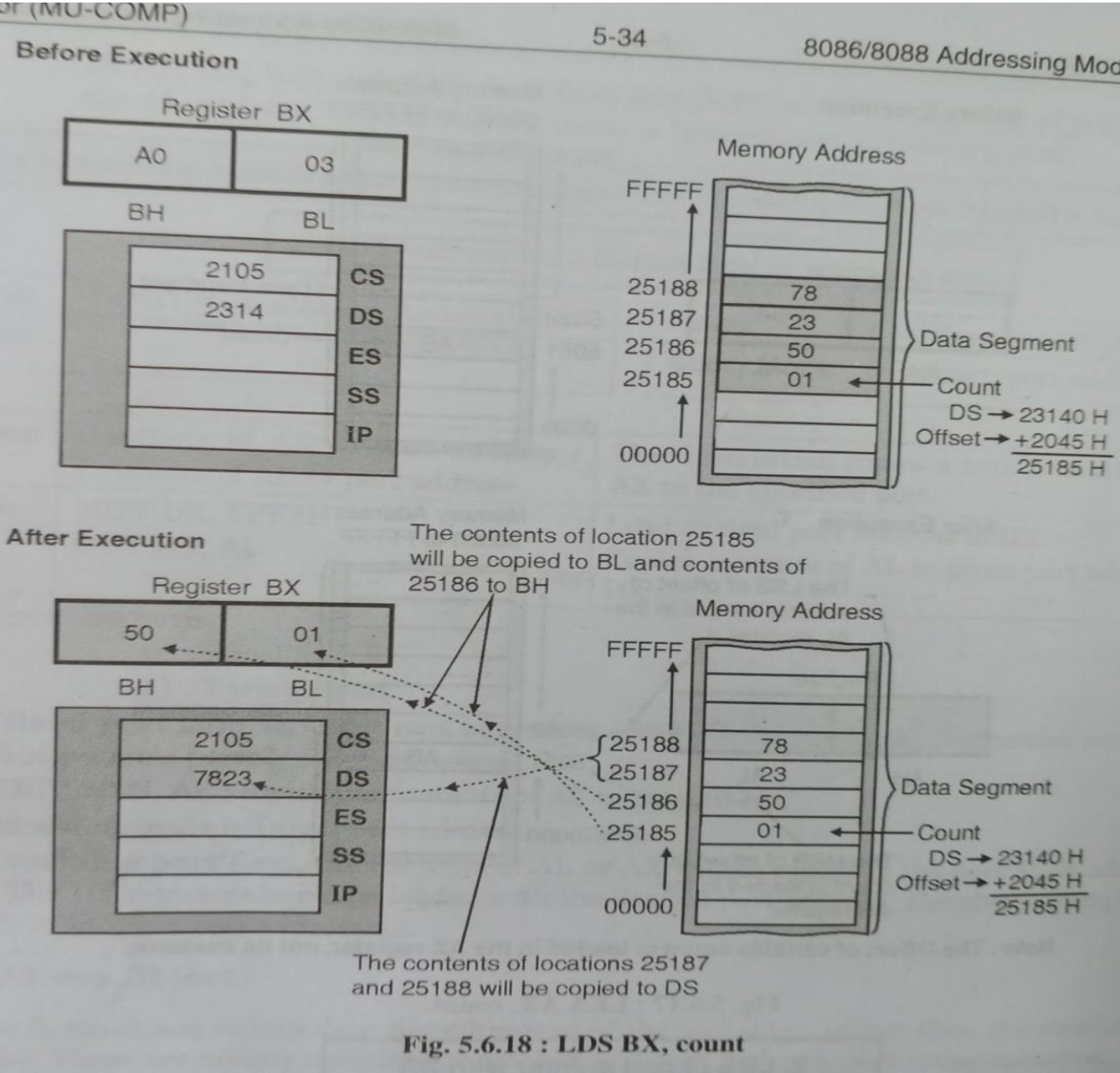


Fig. 5.6.18 : LDS BX, count

Data transfer group:Address object

3) LES : Load pointer with ES(Load register and ES with words from memory)

LES register,source

e.g. LES BX, COUNT

- The Source is always a memory location.
- This instruction is 2 byte instruction which copies a word from two memory locations into register specified in the instruction. It then copies a word from the next two memory locations into the ES register.

NOTE: The source cannot be a register.

Data transfer group:Address object

3) LES : Load pointer with ES(Load register and ES with words from memory)

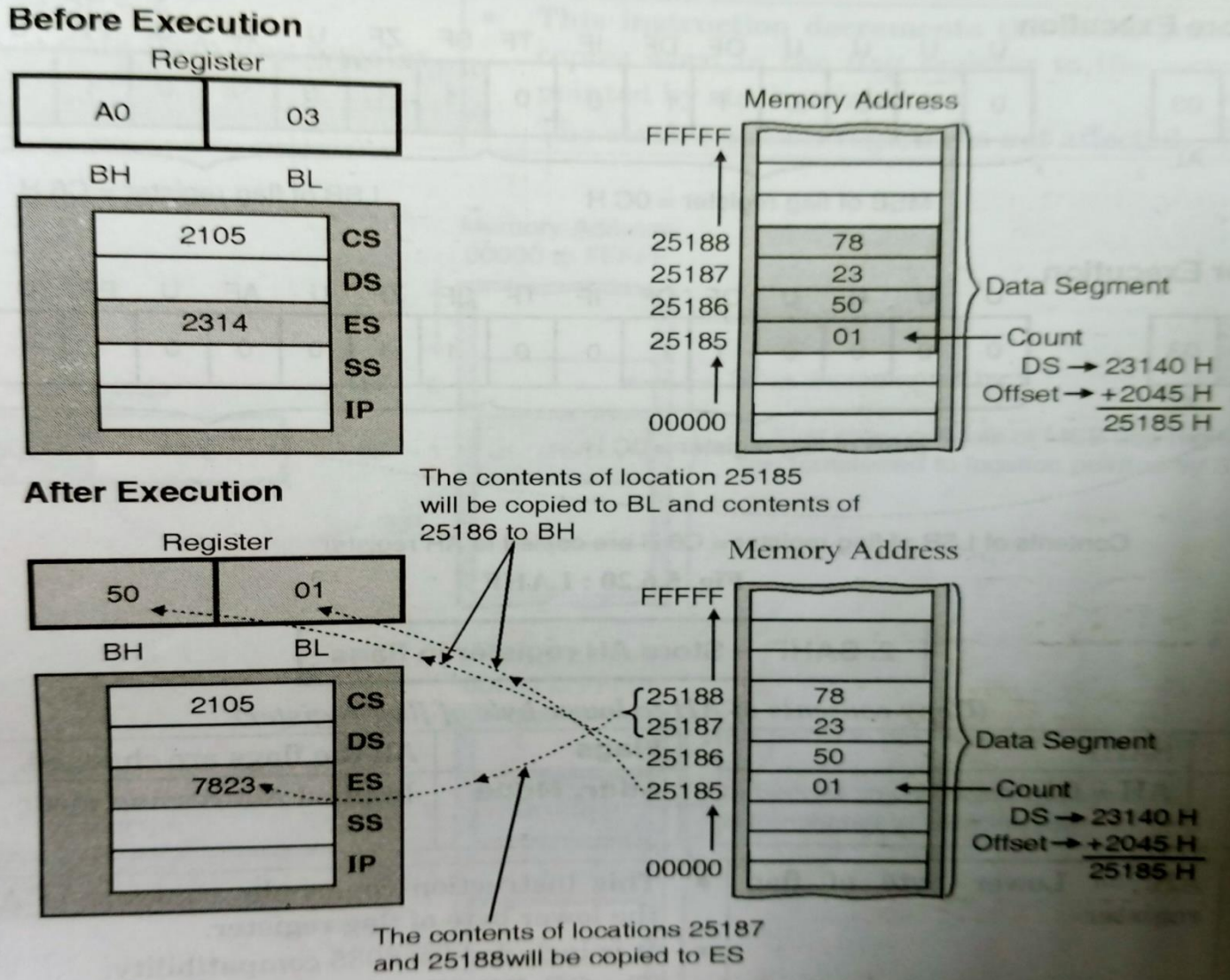


Fig. 5.6.19 : LES BX, count

Data transfer group:Flag transfer

These instructions are related to movement of flag register to/from a register and memory.

1) LAHF : Load AH register from flags(Copy lower byte of flag register to AH)

LAHF

e.g. LAHF

- The lower byte of 8086 flag register is copied to the AH register.
- U indicates undefined i.e. value of that bit is in determined (it may be 0 or 1 anything)

Data transfer group:Flag transfer

1) LAHF : Load AH register from flags(Copy lower byte of flag register to AH)

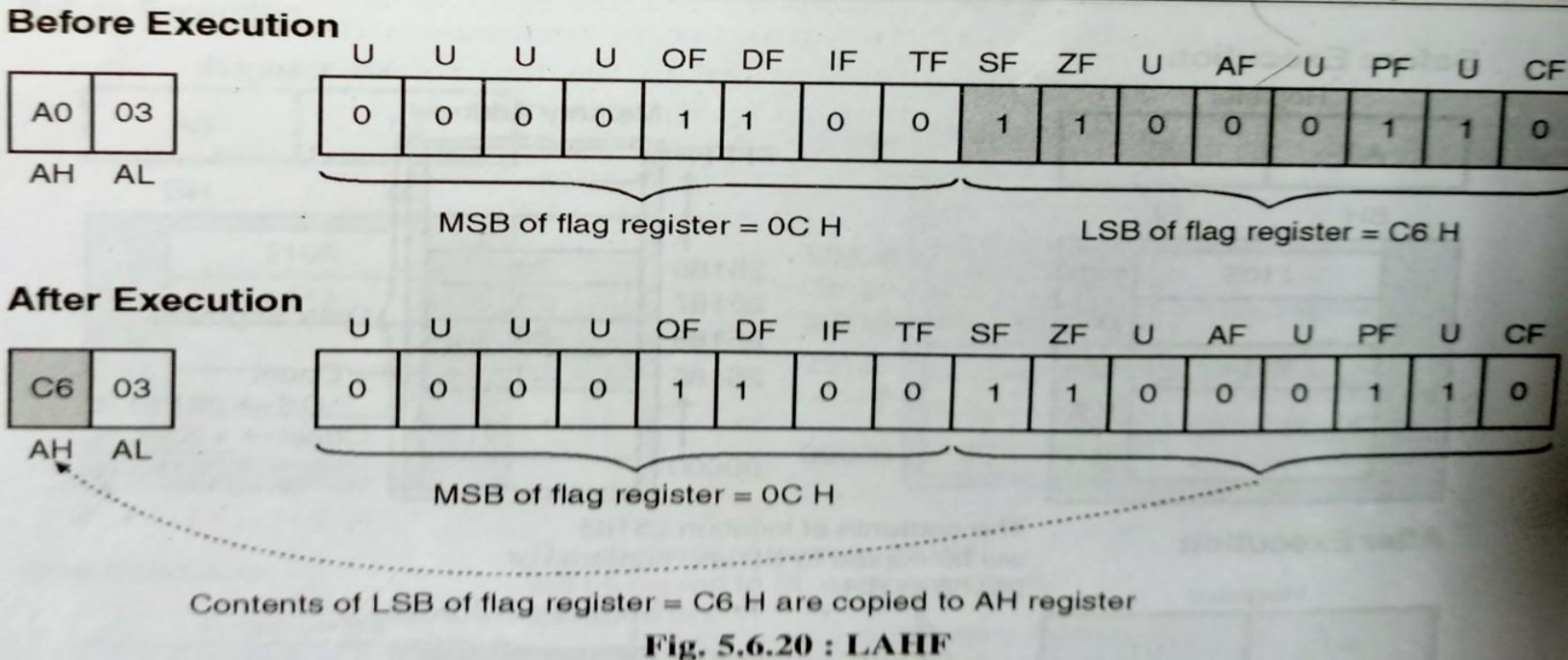


Fig. 5.6.20 : LAHF

Data transfer group:Flag transfer

2) SAHF : Store AH register in flags(Copy contents of AH to lower byte of flag register)

SAHF

e.g. SAHF

- This instruction copies the contents of AH register to the lower byte of flag register.
- U indicates undefined i.e. value of that bit is in determined (it may be 0 or 1 anything)

Data transfer group:Flag transfer

2) SAHF : Store AH register in flags(Copy contents of AH to lower byte of

(Copy contents of AH to lower byte of flag register)

Mnemonic	SAHF
Algorithm	AH = flag register

Flags	All the flags are changed.
Addr. Mode	Implied Addressing mode

Operation	AH → Lower byte of flag register	• This instruction copies the contents of AH the lower byte of flag register. • It is included for 8085 compatibility. • The OF, DF, IF and TF are not affected.

Let the contents of Flag register be 0CC6 H

Before Execution

A0	03
AH	AL

U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF
0	0	0	0	1	1	0	0	1	1	0	0	0	1	1	0

MSB of flag register = 0C H

LSB of flag register = C6 H

After Execution

A0	03
AH	AL

U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF
0	0	0	0	1	1	0	0	1	0	1	0	0	0	0	0

MSB of flag register = 0C H

LSB of flag register = A0 H

Contents of AH register are copied to LSB of flag register

Fig. 5.6.21 : SAHF

Data transfer group:Flag transfer

3) PUSHF : Push flags onto stack(Push flag register onto stack)

PUSHF

e.g. PUSHF

- This instruction decrements the stack pointer by 2 and copies word in the flag register to the memory location pointed by stack pointer.
- MSB to SP-1 and LSB to SP-2.

Data transfer group:Flag transfer

3) PUSHF : Push flags onto stack(Push flag register onto stack)

data from flag register.

instruction decrements the stack pointer by 2
copies word in the flag register to the memory loc
pointed by stack pointer.

- The stack segment register is not affected.

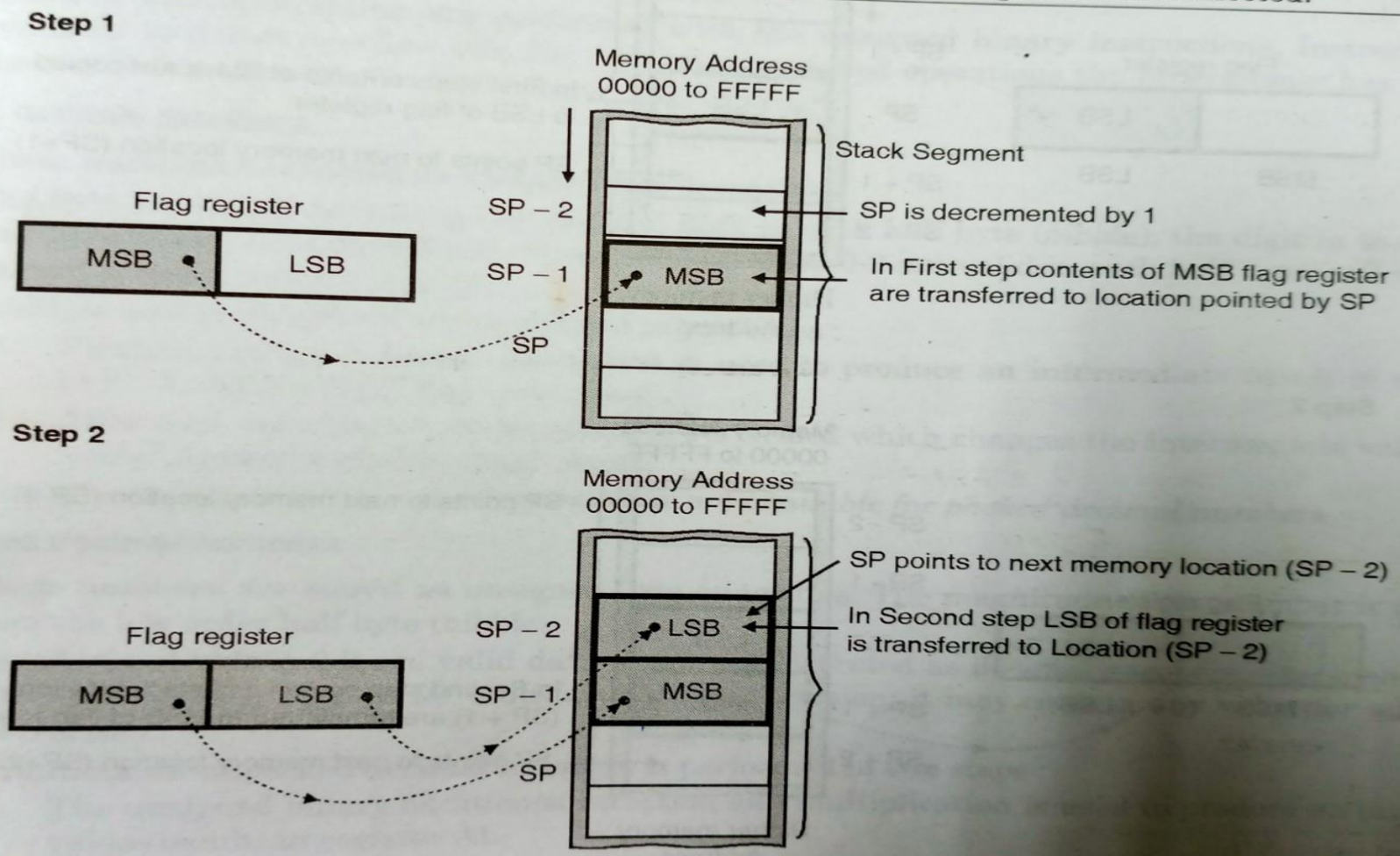


Fig. 5.6.22 : PUSHF

Data transfer group:Flag transfer

4) **POPF : Pop flags off stack.**

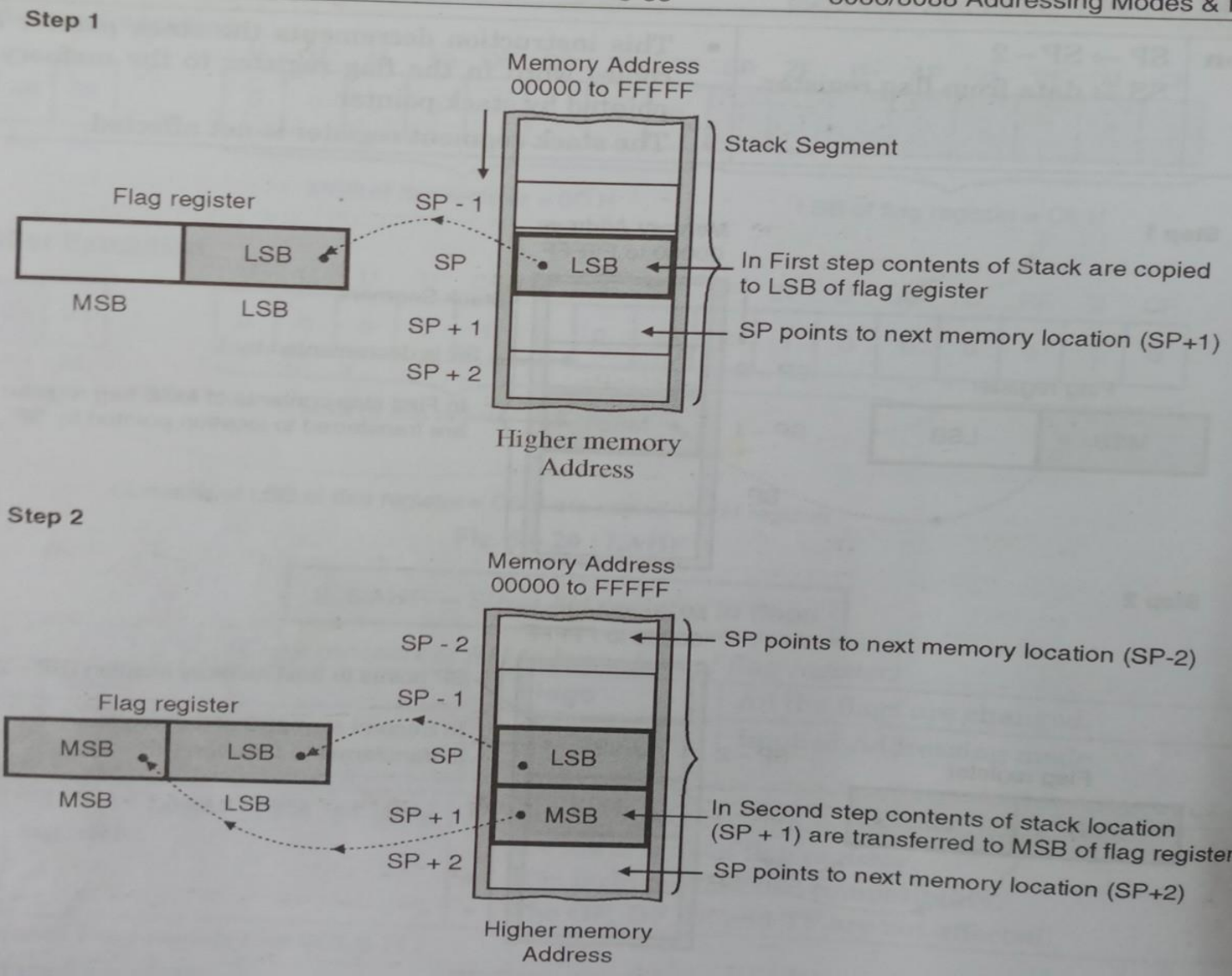
POPF

e.g. POPF

- This instruction copies a word from the two memory locations at the top of the stack to flag register and increments the stack pointer by 2.
- SP to LSB and SP+1 to MSB.

Data transfer group:Flag transfer

4) POPF : Pop flags off stack.



Arithmetic group: Addition

1. **ADD**
2. **ADC**
3. **INC**
4. **AAA**
5. **DAA**

1. **ADD: Add byte or word**

   ADD destination,source

- This instruction adds a number from source to number from destination and puts the result to specified destination.

  register,register

  register,memory

  memory,register

  register,immediate

  memory,immediate

  accumulator,immediate

Arithmetic group: Addition

i) ADD register ,register e.g. ADD BL,CL

Example	ADD BL, CL	• This instruction can be 8/16 bit . • This instruction adds the data in register CL The result is stored in BL register. It can be 8 instruction BL ← BL + CL

Note : The register cannot be a segment register.

Let CL = 04 H and BL = 07 H

```
CL      04 H
BL  +   07 H
-------------
        0B H   →   Result stored in BL
```

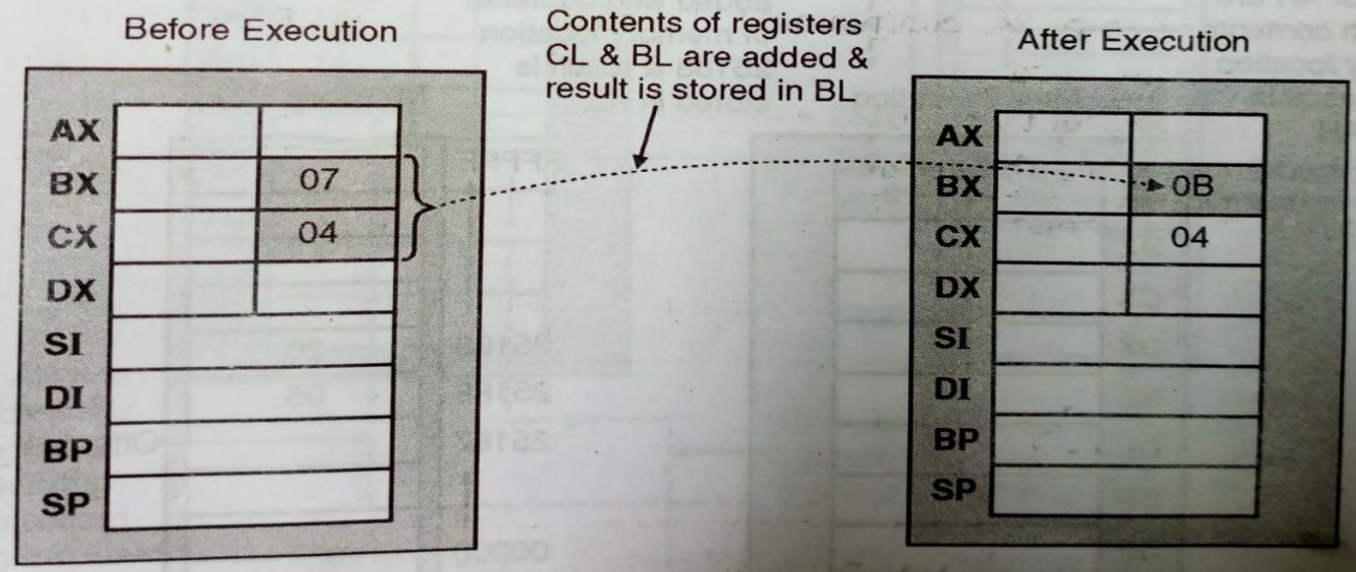


Fig. 5.7.1 : ADD BL, CL

Arithmetic group: Addition

ii) ADD register ,memory e.g. ADD AX,[2048]

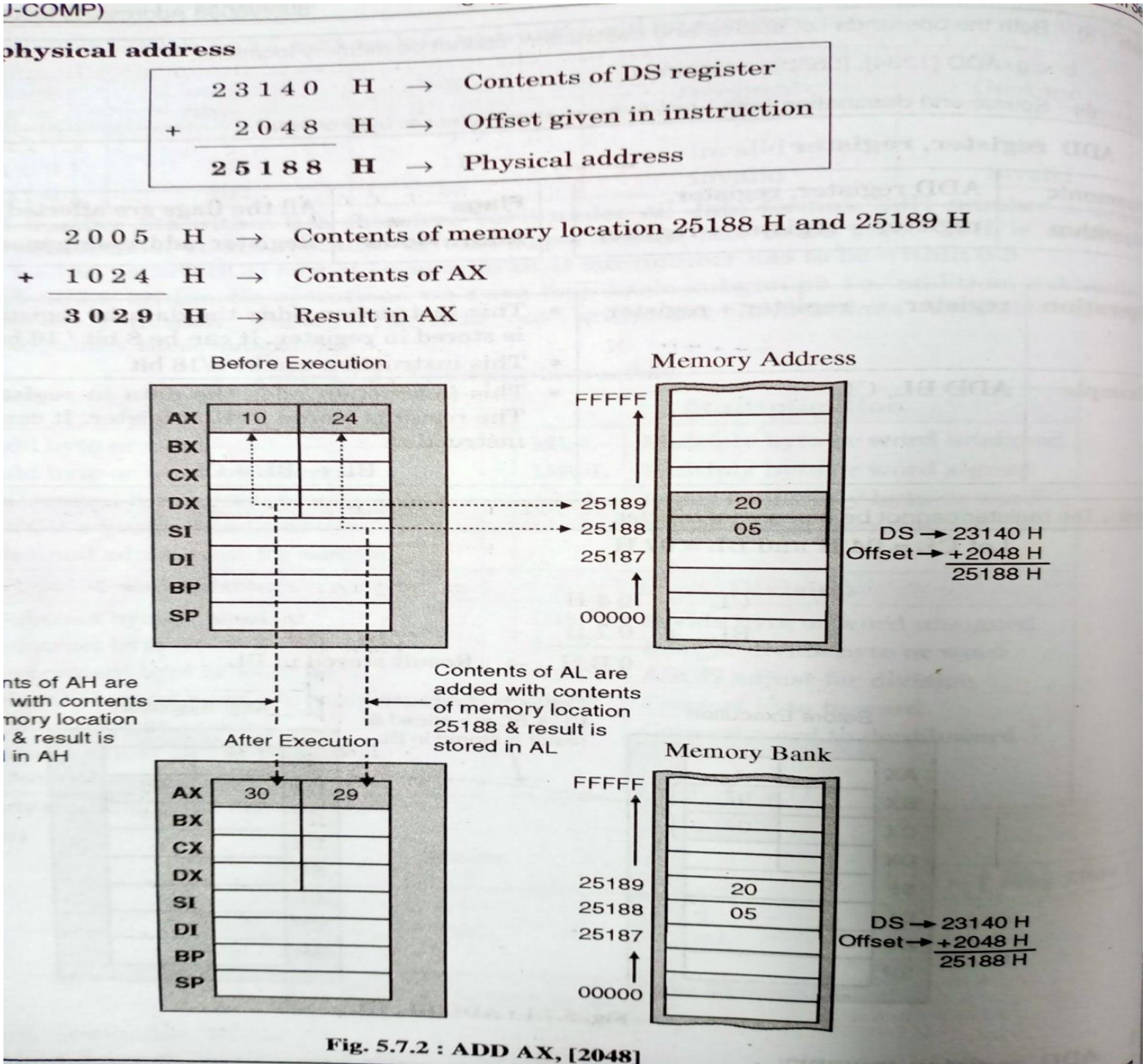


Fig. 5.7.2 : ADD AX, [2048]

Arithmetic group: Addition

iii) ADD memory, register e.g. ADD [2048],AX

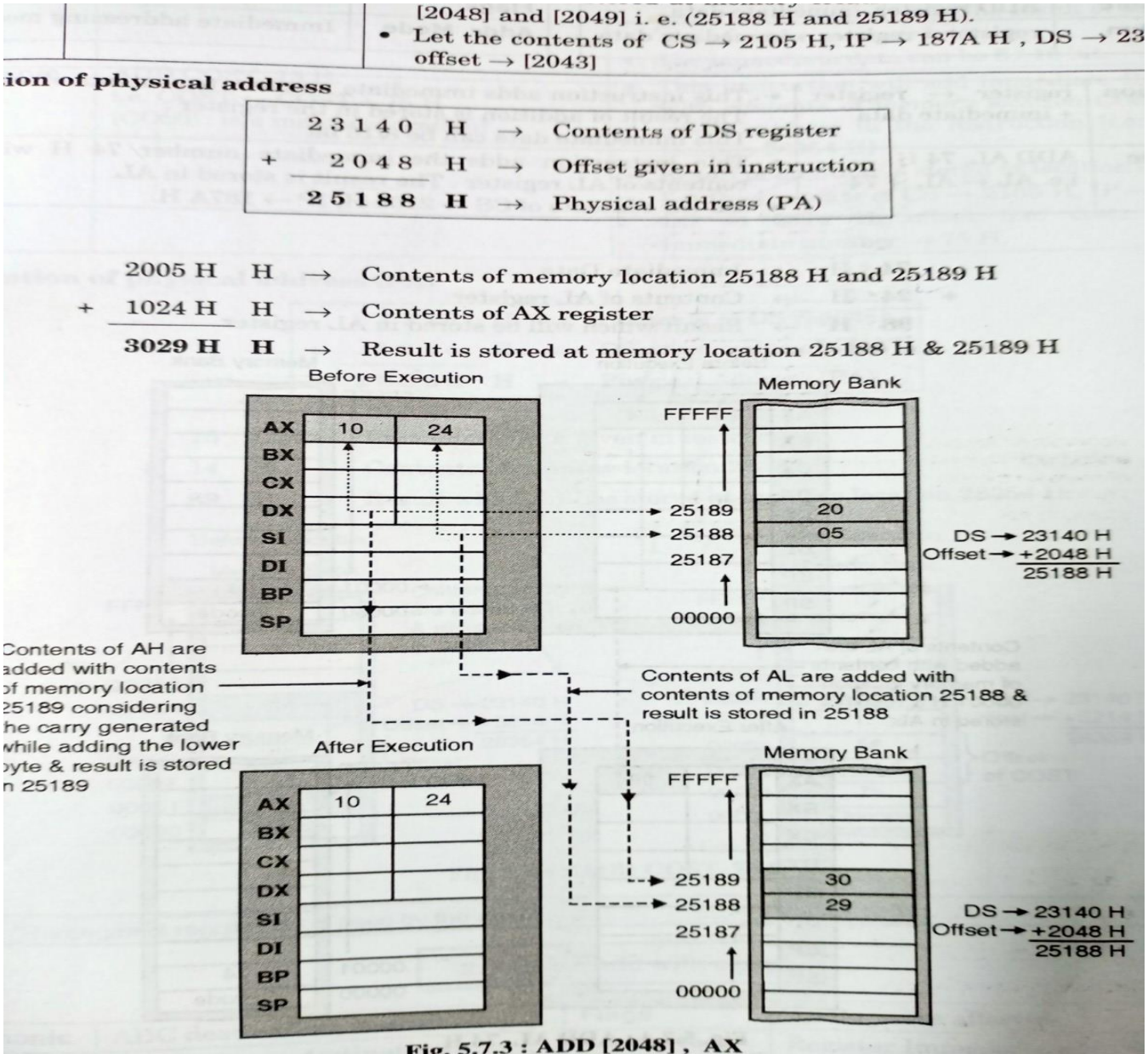


Fig. 5.7.3 : ADD [2048] , AX

Arithmetic group: Addition

iv) ADD register ,immediate/ADD accumulator, immediate

e.g. ADD AL,74H

← register iate data	• This instruction … The result of addition is stored in the … • This immediate data can be 8/16 bit .
, 74 H - AL + 74	• This instruction adds the immediate number 74 H with the contents of AL register . The result is stored in AL. • Let the contents of CS → 2105 H, IP → 187A H.

74 H → Immediate Data
\+ 24 H → Contents of AL register
98 H → Result which will be stored in AL register

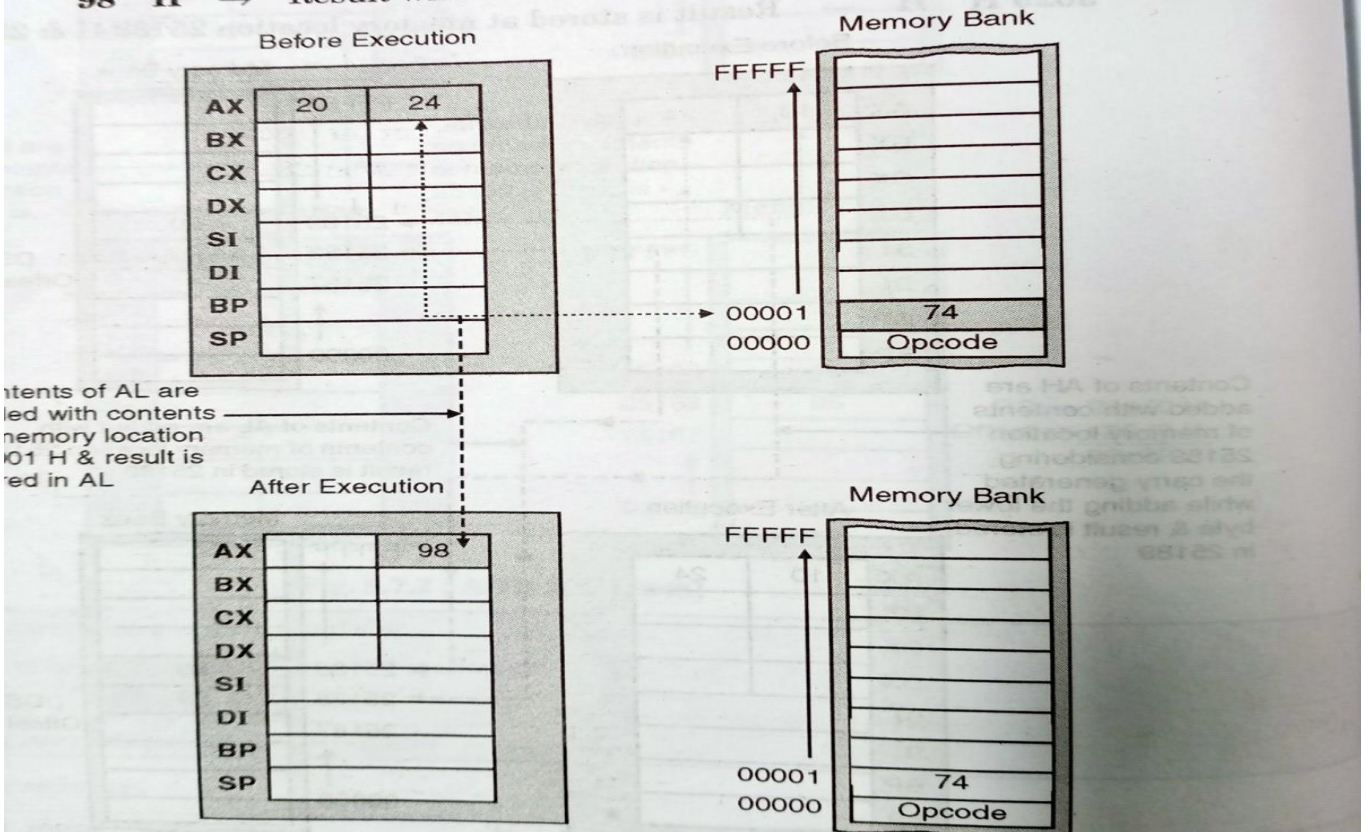


Fig. 5.7.4 : ADD AL, 74 H

Arithmetic group: Addition

v) ADD memory ,immediate e.g. ADD COST,75H

DS → 2314 H, offset (i.e. cost)
immediate number → 75 H.

Generation of physical address (PA)

	23140	H →	Contents of DS register
+	5214	H →	Offset i.e. value of COST
	28354	**H** →	Physical Address (PA)

	75	H →	Immediate Data given in instruction .
+	14	H →	Contents of memory location 28354 H
	89	**H** →	Result which will be stored at memory location 28354 H.

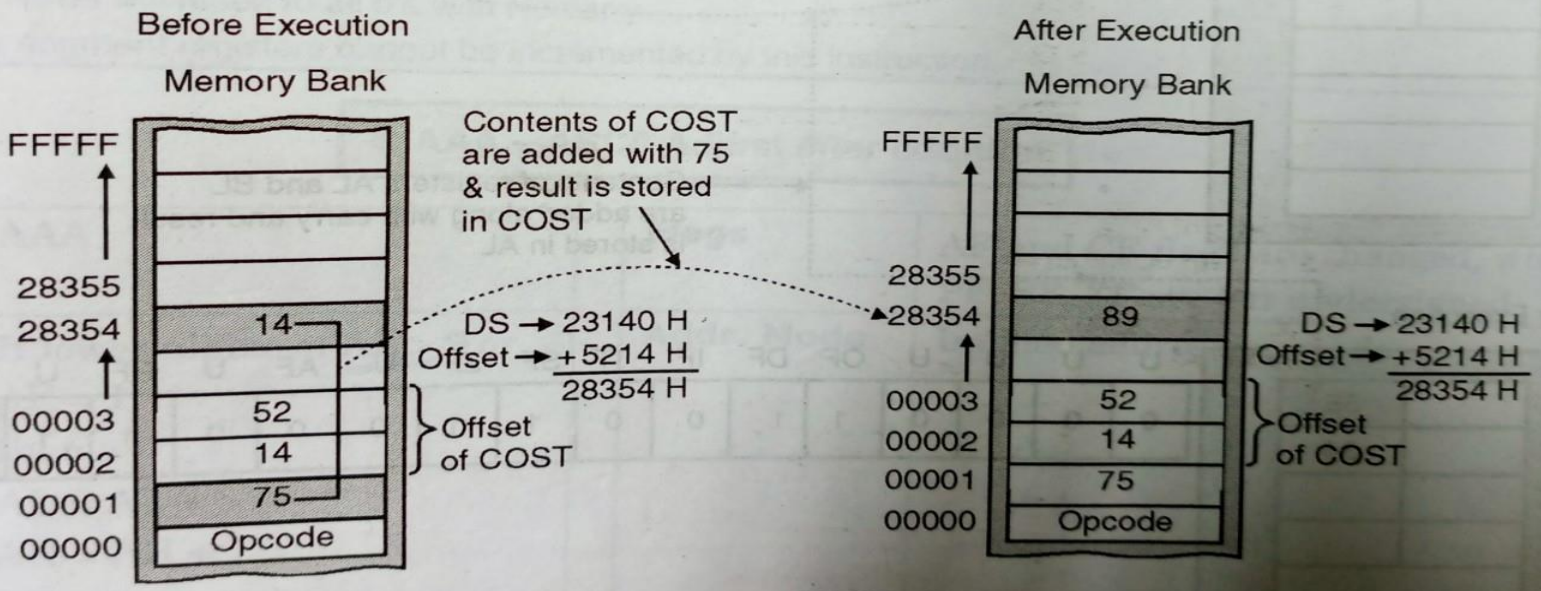


Fig. 5.7.5 : ADD COST, 75 H

Arithmetic group: Addition

2) **ADC: Add with carry**

ADC destination,source e.g.ADC AL,BL (ie.AL<-AL+BL+CY)

- This instruction adds destination operand contents,source operand contents and carry flag and answer is stored back to destination operand.

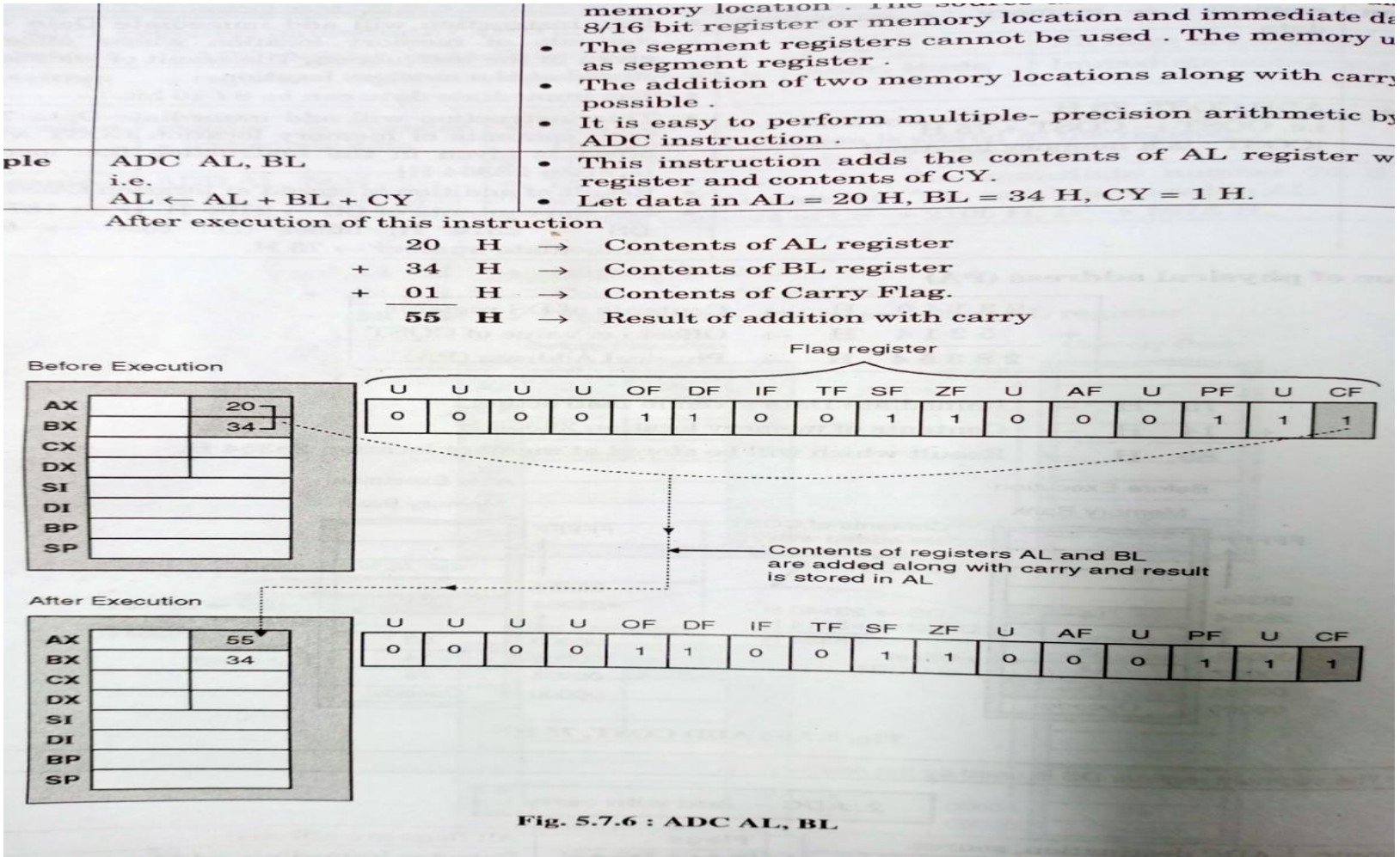


Fig. 5.7.6 : ADC AL, BL

Arithmetic group: Addition

3) INC: Increment byte or word by 1

INC destination e.g INC AL

thm	destination =destination + 1		affected.
		Addr. Mode	Implied addressing mode
tion	destination ← destination + 1	• This instruction adds 1 to the destination • The operand may be a byte or word and i an unsigned binary number. • The destination operand may be a register location.	
ple	INC CX INC AL	Add 1 to contents of CX Add 1 to contents of AL register.	

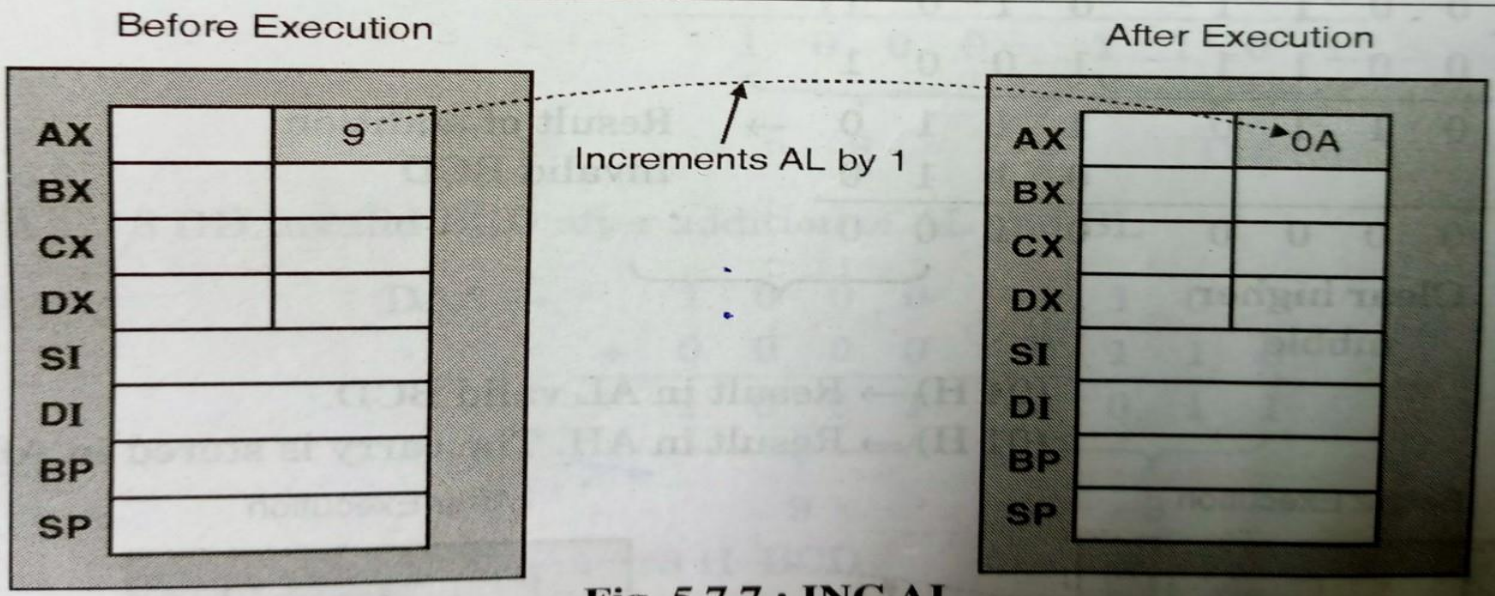


Fig. 5.7.7 : INC AL

Arithmetic group: Addition

4) **AAA: ASCII adjust after addition**

Algo: 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

- We enter data through keyboard is usually in ASCII code.The numbers 0 to 9 are represented by ASCII codes 30H to 39H.The 8086 allows to add the ASCII codes for two decimal digits without masking off “3” in the upper nibble of each.
- After addition, AAA instruction is used to make sure that the result is the correct unpacked BCD.
- AAA works only on AL register.

Arithmetic group: Addition

4) **AAA: ASCII adjust after addition**

ASCII code. The numbers ... allows to add the ASCII codes for two decimal digits ... nibble of each. After addition, AAA instruction is used to make sure that ... correct unpacked BCD.

- The AAA instruction works only on AL register.
- If the lower nibble of AL register after addition is greater than 9, a ... increment AH by 1. The auxiliary carry flag and carry flag are set.

Assume
AL = 0 0 1 1 0 1 0 1 ASCII 5
BL = 0 0 1 1 1 0 0 1 ASCII 9
Add AL, BL

```
  0 0 1 1   0 1 0 1
+ 0 0 1 1   1 0 0 1
-----------------------
  0 1 1 0   1 1 1 0   →  Result of addition
+                       Invalid BCD
          + 0 1 1 0
-----------------------
  0 0 0 0   0 1 0 0
y →  1      Clear higher
            nibble
```

⇓

(04 H) → Result in AL valid BCD.
(01 H) → Result in AH. The carry is stored in AH ...

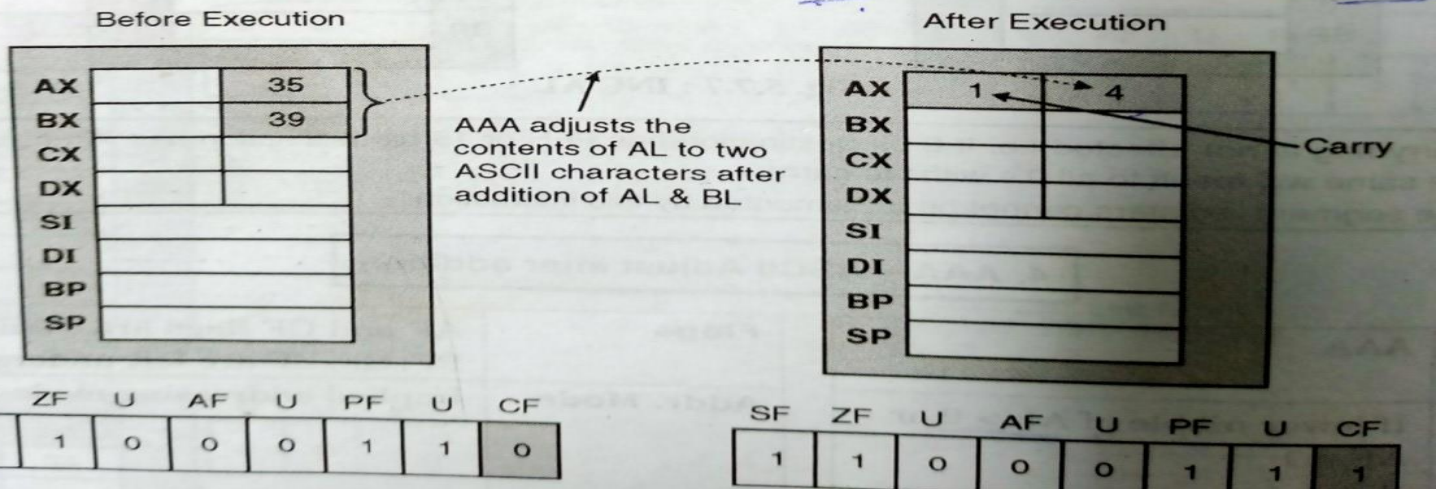


Fig. 5.7.8 : AAA

Arithmetic group: Addition

5) **DAA: Decimal adjustment for addition**

Algo: If lower nibble of AL>9 or AF=1

then:

AL=AL+06H

AF=1

If AL>9FH or CF=1 then,

AL=AL+60H

CF=1

- This instruction is used to make sure that the result of adding two packed BCD numbers is adjusted to be a valid BCD number.
- It operates only on AL register.

e.g. Add AL,BL

AL=59h

BL= 34h

Arithmetic group: Addition

5) **DAA: Decimal adjustment for addition**

on	AL ← Sum in adjusted to packed BCD format	• This instruction is used to make sure that the result two packed BCD numbers is adjusted to be a va number. • It operates only on AL register. • If number in the lower nibble of AL register after a greater than 9 or if the auxiliary carry flag is set add 6 • If the lower nibble of AL is greater than 9 or if the ca set then add 60 H.

If AL = 59 H valid BCD, BL = 34 H valid BCD

ADD AL, BL

```
  0 1 0 1     1 0 0 1
+ 0 0 1 1   + 0 1 0 0
-----------------------
  1 0 0 0     1 1 0 1
  ---------   ---------
      8           D
```

AL = 8 DH invalid BCD after addition of AL and BL

DAA →

```
  1 0 0 0     1 1 0 1
+ 0 0 0 0   + 0 1 1 0
-----------------------
  1 0 0 1     0 0 1 1
  ---------   ---------
      9           3
```

∴ AL = 93 H BCD

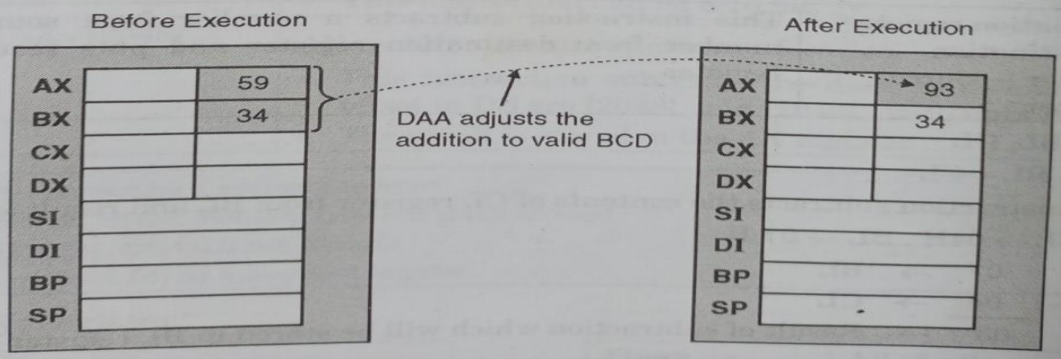


Fig. 5.7.9 : DAA

Arithmetic group: Subtraction

1. SUB
2. SBB
3. DEC
4. NEG
5. CMP
6. AAS
7. DAS

Arithmetic group: Subtraction

1. SUB: Subtract byte or word

   SUB destination,source

- This instruction subtracts a number from source with number from destination and puts result in the destination location.

Destination	Source
Register	Register
Register	memory
Memory	Register
Accumulator	immediate
Register	immediate
Memory	immediate

Arithmetic group: Subtraction(SUB reg,reg)

Example : SUB BL, CL

$BL \leftarrow BL - CL$

This instruction subtracts the contents of CL register from BL and result is s

Let $CL \rightarrow 04H$, $BL \rightarrow 07\ H$

$$\begin{array}{rcl} 07 & \rightarrow & BL \\ -\ 04 & \rightarrow & CL \\ \hline 03 & \rightarrow & \text{Result of subtraction which will be stored in BL register} \end{array}$$

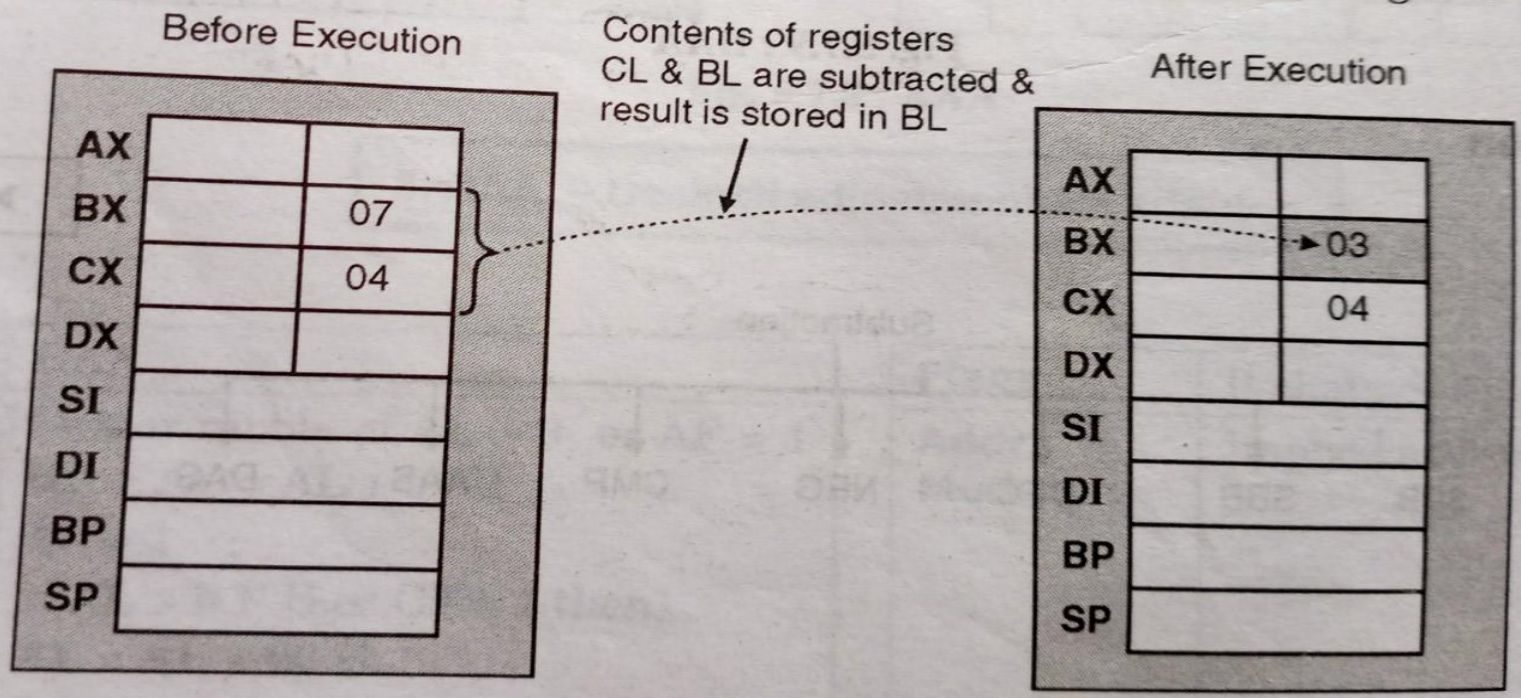


Fig. 5.7.10 : SUB BL, CL

Arithmetic group: Subtraction(SUB mem,reg) e.g.SUB [2048],AX

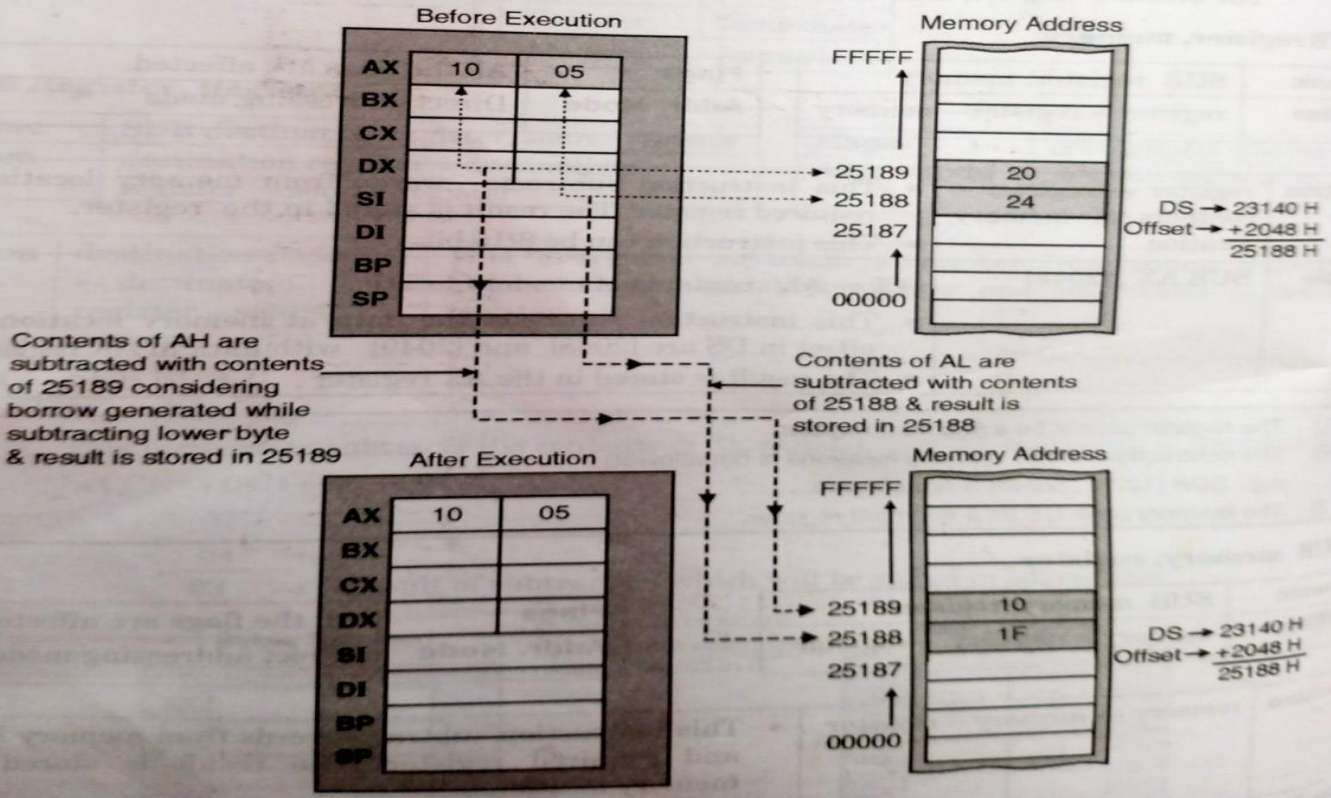
ote : 1) The register cannot be a segment register.
2) The subtraction of two memory locations is not allowed.
e.g. SUB [1024] , [2048] is not allowed.
3) The memory uses DS as a segment register.

Let the contents of CS → 2105 H, IP → 187 A , DS → 2314 H, offset → [2048] H

neration of physical address

	23140 H	→	Contents of DS register
+	2048 H	→	Offset given in instruction
	25188 H	→	Physical address (PA)

	2024 H H	→	Contents of memory location 25188 H and 25189 H
−	1005 H H	→	Contents of AX register
	101F H H	→	Result is stored at memory location 25188 H & 25189 H



Arithmetic group: Subtraction(SUB reg,immediate) e.g.SUB AL,24h

L, 24 H AL – 24	• This immediate data can be 8/16 bit. • This instruction subtracts the immediate number 24 H wit contents of AL register. The result is stored in AL.

74 H → Contents of AL register
– 24 H → Contents of Immediate Data
50 H → Result which will be stored in AL register

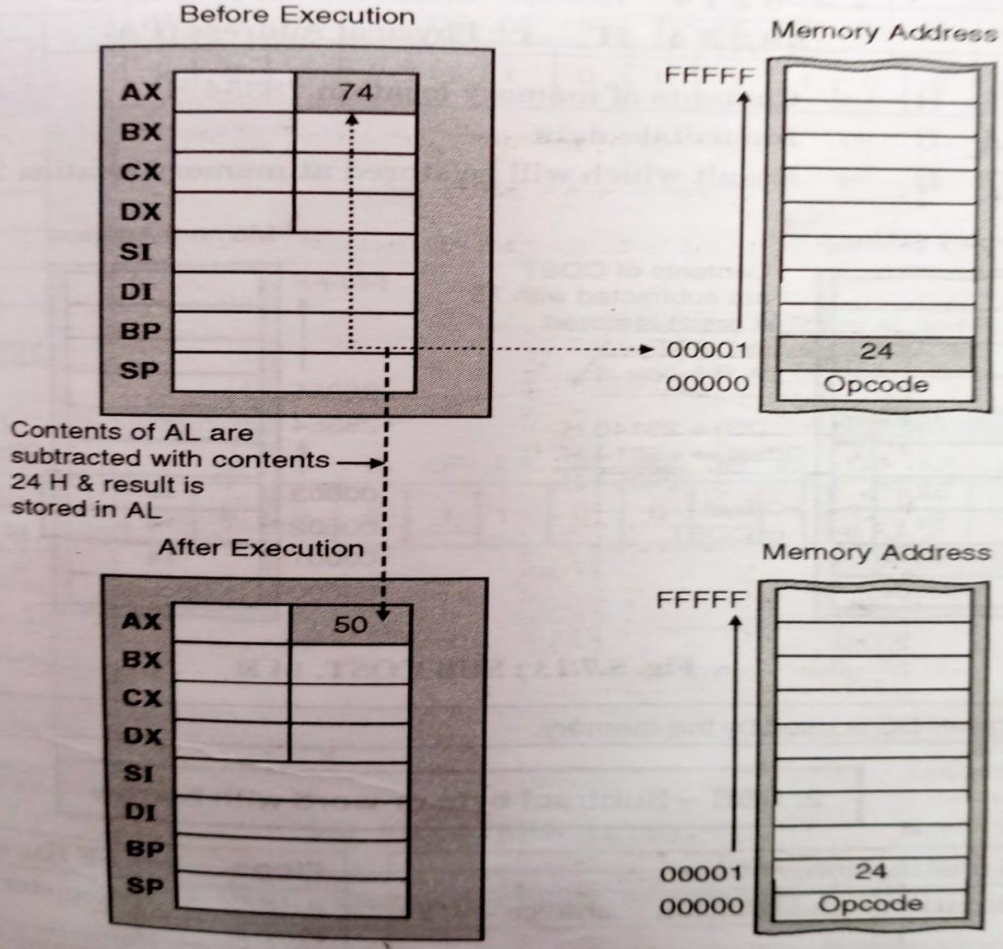


Fig. 5.7.12 : SUB AL, 24 H

Arithmetic group: Subtraction(SUB mem,immediate) e.g.SUB COST,14h

	COST ← COST –14 H [Cost : is a memory location]	contents of memory ... instruction [i.e. memory location 28354 H] • Result of subtraction is stored at COST.

Let the contents of CS → 2105 H, IP → 187 AH, DS → 2314 H, offset (i.e. cost) → 5214, immediate number → 75 H.

Generation of physical address (PA) :

	23140	H →	Contents of DS register
+	5214	H →	Offset i.e. value of COST
	28354	**H →**	Physical Address (PA)

	75	H →	Contents of memory location 28354 H
–	14	H →	Immediate data
	61	**H →**	Result which will be stored at memory location 28354 H.

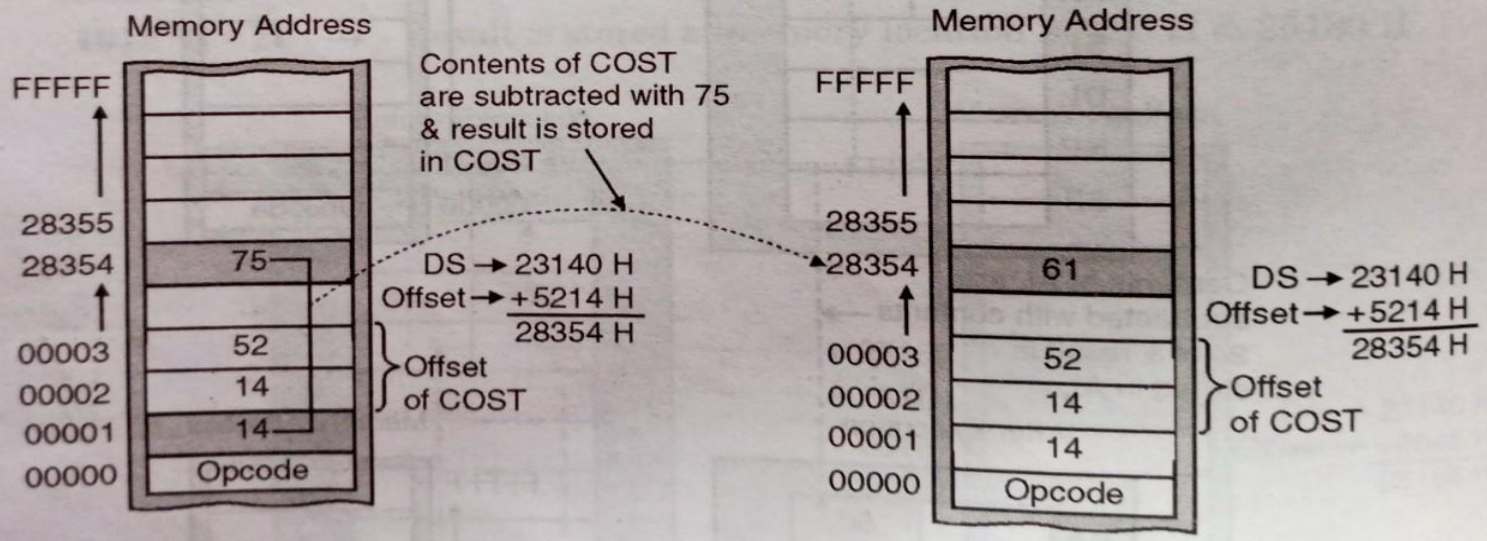


Fig. 5.7.13 : SUB COST, 14 H

Arithmetic group: Subtraction

2. SBB: Subtract byte or word with borrow

   SBB destination,source

   SBB AL,BL

- This instruction subtracts source and carry flag(i.e. Borrow) from destination.

Note:

1. The source and destination register cannot be a segment register.
2. The subtraction of two memory locations is not allowed.

   e.g. SUB [1024],[2018] not allowed
3. The memory uses DS as a segment register.

Arithmetic group: Subtraction

et AL → 34 H, BL → 24 H CY=1 H

	3 4 H	→	Contents of AL register
–	2 4 H	→	Contents of BL register
	1 0 H		
–	0 1 H	→	Contents of Carry Flag
	0 F H	→	Result

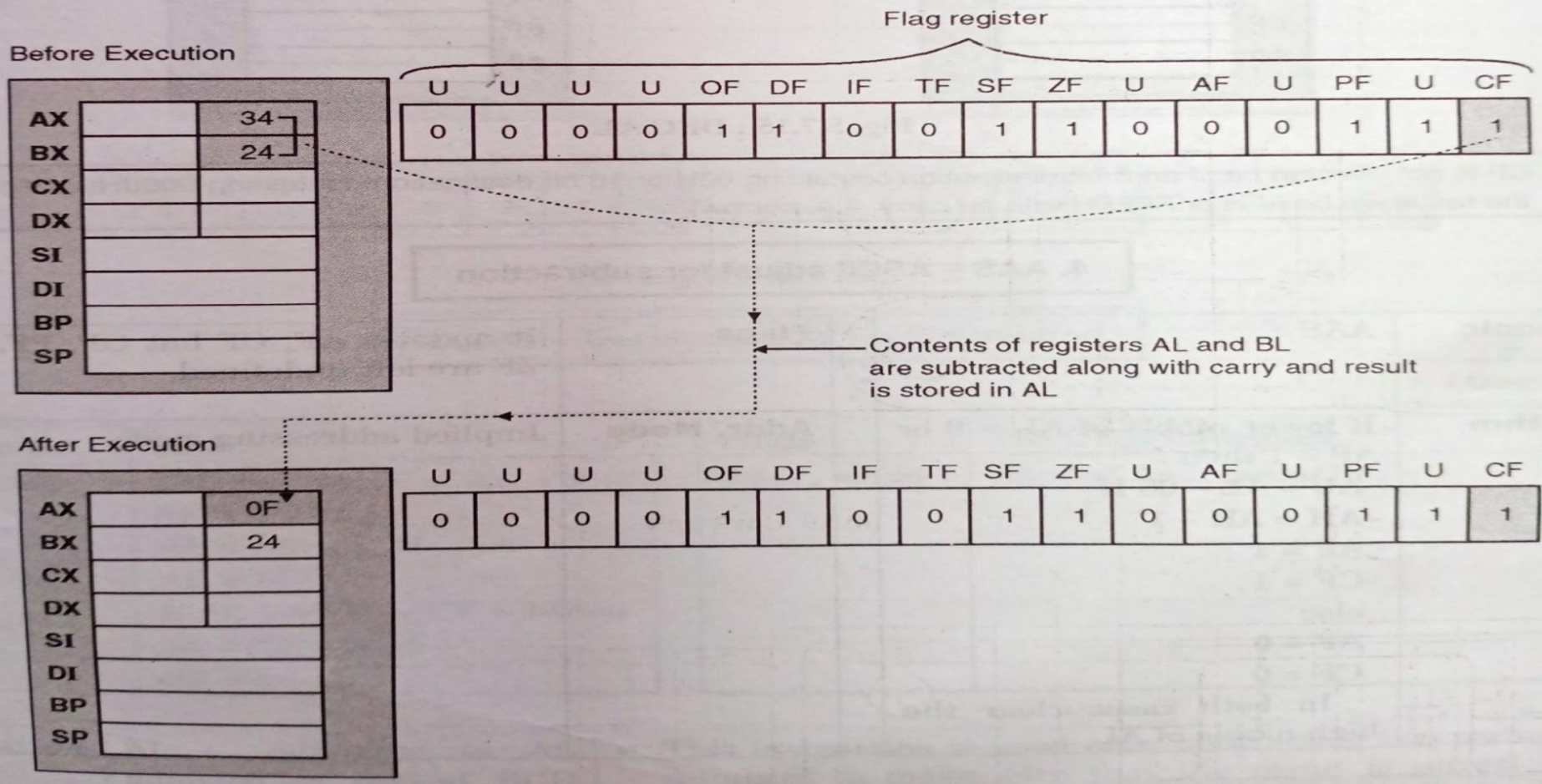


Fig. 5.7.14 : SBB AL, BL

Arithmetic group: Subtraction

3. DEC: Decrement byte or word by 1

   DEC destination

   DEC AL

- This instruction subtracts 1 from the destination word or byte .
- The destination can be a register or a memory location.
- This instruction cannot be used to decrement the contents of segment registers.

Arithmetic group: Subtraction

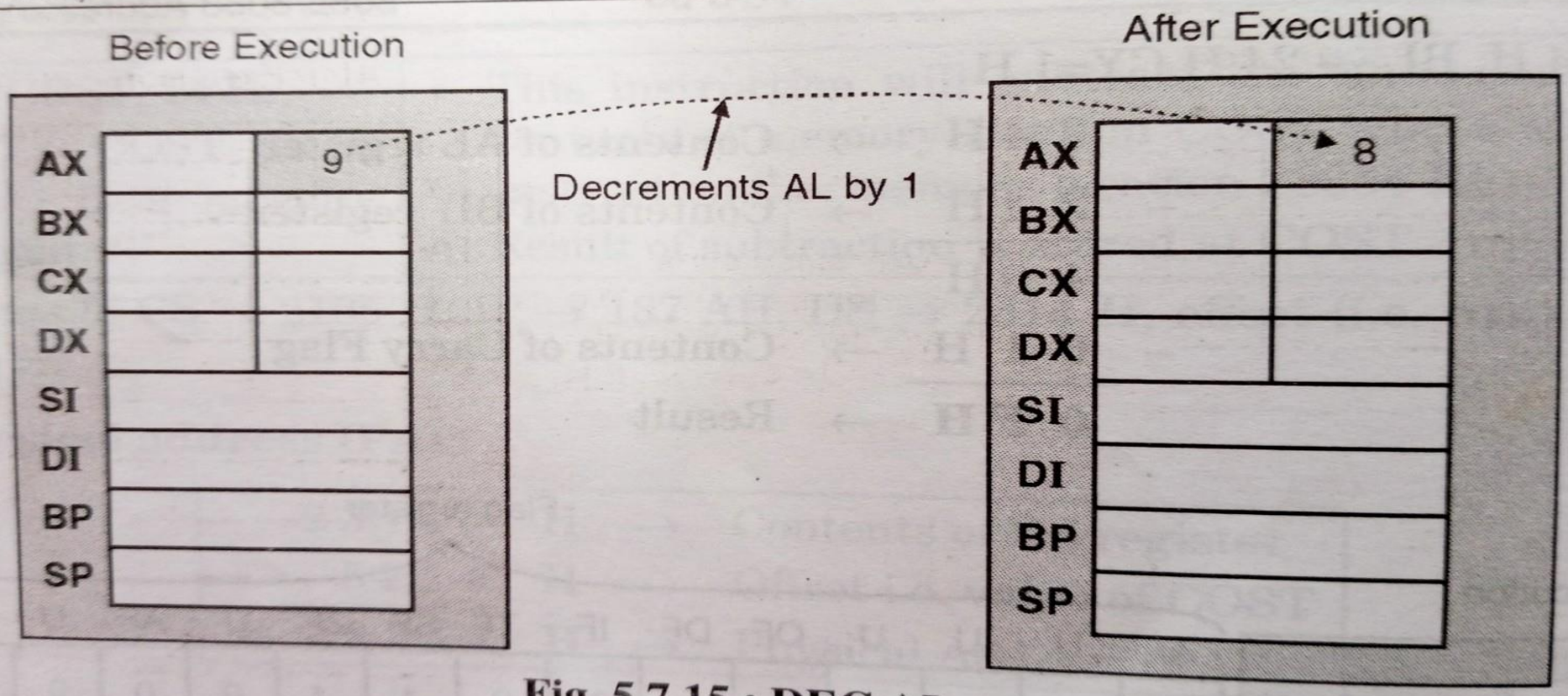


Fig. 5.7.15 : DEC AL

affected i.e. if an 8 bit destination containing 00H or 16 bit destination
will be FFH or FFFFH with

Arithmetic group: Subtraction

4. AAS: ASCII adjust for subtraction.

   AAS

Algo: If lower nibble of AL>9 or AF=1 then:

AL=AL-06H

AH=AH-1

AF=1

CF=1

else

AF=0

CF=0

Arithmetic group: Subtraction

high nibble of AL

- The number 0 to 9 are represented as 30 – 39 in ASCII code
- The 8086 allows us to subtract the ASCII codes for two decimal digits "3" in upper nibble of each.
- The AAS instruction is used to make sure that the result is in unpacke
- It works only on the AL register.

Let AL = 0011 1001 = 39 H = ASCII 9

Let BL = 0011 0101 = 35 H = ASCII 5

SUB AL, BH → Result AAS	=	0 0 1 1 1 0 0 1	
	–	0 0 1 1 0 1 0 1	
	0	0 0 0 0 0 1 0 0	= BCD = 04
CF = 0 no borrow required			

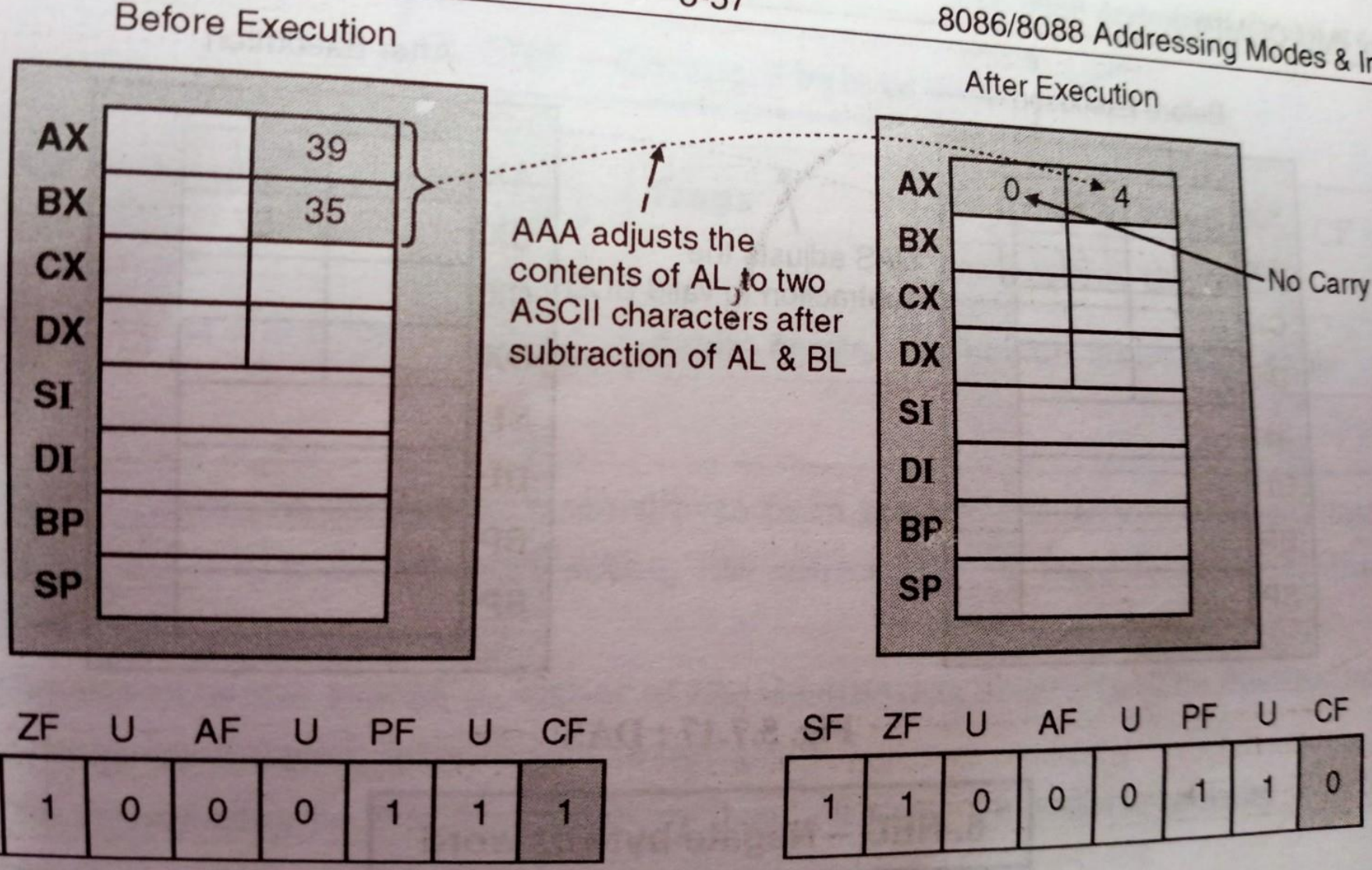

Before Execution
AX
BX
CX
DX
SI
DI
BP
SP
39
35
AAA adjusts the contents of AL to two ASCII characters after subtraction of AL & BL
After Execution
AX
BX
CX
DX
SI
DI
BP
SP
0
4
No Carry
SF ZF U AF U PF U CF
1 1 0 0 0 1 1 1
SF ZF U AF U PF U CF
1 1 0 0 0 1 1 0

Fig. 5.7.16 : AAS

Arithmetic group: Subtraction

5. DAS : Decimal adjust for subtraction.

DAS

Algo: If lower nibble of AL>9 or AF=1 then:

AL=AL-06H

AH=AH-1

AF=1

If AL > 9FH or CF=1 then

AL=AL-60H

CF=1

- This instruction is used after subtracting two packed BCD numbers to make sure that the result is correct packed BCD. The result of subtraction must be in AL for DAS to work correctly.
- This instruction works only on AL register.

e.g. SUB AL,BH

DAS

Arithmetic group: Subtraction

5. DAS : Decimal adjust for subtraction.

packed BCD format	instruction is used after subtracting two packe numbers to make sure that the result is correct BCD. The result of subtraction must be in AL for work correctly. • This instruction works only on AL register. • If the lower nibble in AL after a subtraction is greate 9 or if the auxiliary carry flag is set then th instruction will subtract 6 from the lower nibble of A • If the result in the upper nibble is greater than 9 carry flag is set the DAS instruction will subtrac from the AL register
AL = 1000 0110 = 86 BCD BH = 0101 0111 = 57 BCD SUB AL, BH DAS	AL = 0010 1111 = 2F H, CF = 0 Lower nibble of result is F H, i.e. greater than 9 so DAS subtracts 0000 0110 to give AL = 0010 1001 = 29 H BCD

Arithmetic group: Subtraction

5. DAS : Decimal adjust for subtraction.

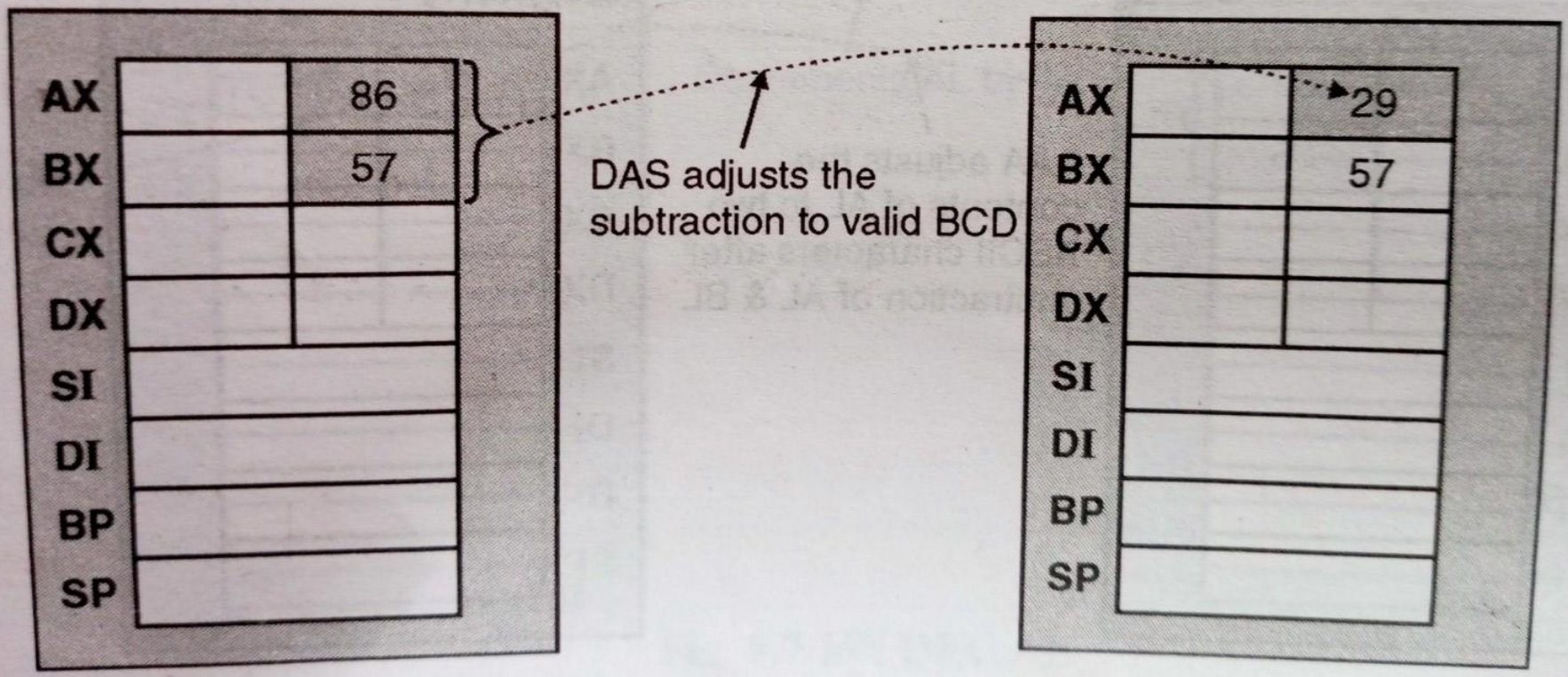


Fig. 5.7.17 : DAS

Arithmetic group: Subtraction

6. NEG: Negate byte or word

NEG destination

NEG AX

Algo: Invert all bits of operand and add 1 to inverted operand.

- This instruction replaces the number in a destination with 2's complement of that number.
- The destination can be a register or memory location.
- This instruction forms the 2's complement by subtracting the original word or byte in the indicated destination from zero.
- It is useful for changing the sign of a signed word or a byte.

Arithmetic group: Subtraction

6. NEG: Negate byte or word

- Attempt to negate a byte containing – 128 or a word ... change to the operand and sets Overflow flag.

Example : NEG AX

AX ← 2's complement of number in AX.

Suppose AX =00A3 H = 0000 0000 1010 0011

Then it's 2's complement will be,

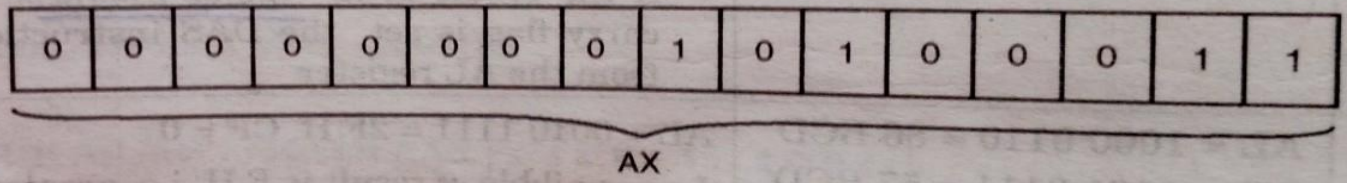
0000 0000 1010 0011 = 1111 1111 0101 1100 ← 1's complement of 00A3 H

\+ 1

1111 1111 0101 1101 ← 2's complement of number

= FF5D H ← contents of AX after execution

Before Execution



After Execution

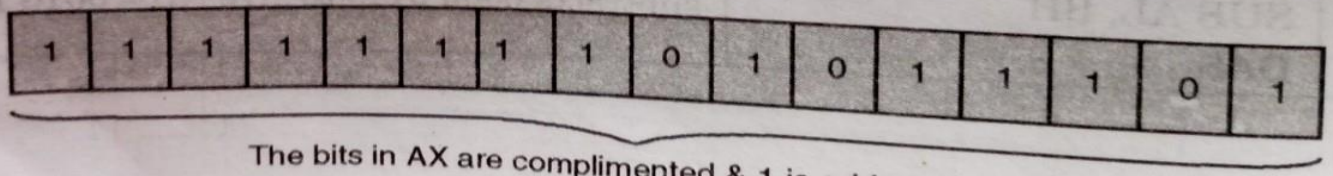


Fig. 5.7.18 : NEG AX

Arithmetic group: Subtraction

7. CMP: Compare byte or word

CMP destination,source

E.g. CMP BH,CL

Algo: Destination-source

- This instruction compares a word/byte from source with byte/word from destination. The comparison is done by subtracting the source byte or word from the destination byte or word.
- The result is not stored in either of the source or destination. The destination and source remain unchanged , only flags are updated.
- The source may be register, memory location or an immediate number.
- The destination may be register , memory location.

Compare	CF	ZF	SF
Source>destination	1	0	1
Source<destination	0	0	0
Source=destination	0	1	0

Arithmetic group: Subtraction

7. CMP: Compare byte or word

flags are updated.

- The source may be register, memory location or an immediate number.
- The destination may be a register or memory location.

Compare	CF	ZF	SF	
Source > destination	1	0	1	Subtraction required borrow, so C
Source < destination	0	0	0	No borrow required CF = 0
Source = destination	0	1	0	Result of subtraction is zero

nple : CMP BH, CL

Compares a byte in CL with byte in BH.

Let CL → 05 H and BH → 04 H, CF → 0, ZF → 0, SF → 0:

```
        =  0 0 0 0   0 1 0 0   → BH
        −  0 0 0 0   0 1 0 1   → CL
      ------------------------
Carry flag → [0]   0 0 0 0   0 1 0 0   → Result of subtraction
Carry flag → [1]   1 1 1 1   1 0 1 1   ← 2's complement of the result i.e. result
                                          comparison of BH and CL registers
```

: (i) The source and destination both cannot be memory locations.

(ii) The compare instructions are often used with conditional Jump instructions.

Arithmetic group: Multiplication

1.MUL

2. IMUL

3. AAM

1.MUL: Multiply byte or word unsigned

MUL source

MUL CX

Algo: When operand is a byte AX=AL*operand,

When operand is a word(DX:AX)=AX*operand

- This instruction multiplies an unsigned byte from source with an unsigned byte in the AL register or an unsigned word from source with an unsigned word in AX.
- Byte then result in AX
- Word then result in DX and AX.

Arithmetic group: Multiplication

1 MUL

	• The source can be a register or memory location. • This instruction cannot be used to multiply immediate data. If we want to multiply immediate data. Then that immediate data has to be stored into some valid register and then multiplied with byte or word in AL or AX	
Example	MUL CX	DX:AX =CX * AX : Result of multiplication MSB will be stored in DX register and the LSB will be stored in AX register.

Let AX ← 0009 H, CX ← 0054 H, DX ← FFFF H (initial data) CS ← 2105 H, IP ← 187 AH

MUL CX : 0009 H ← contents of AX register

× 0054 H ← contents of CX register

02F4 H ← Result of multiplication which will be stored as follows :

DX ← 0000 H (i.e. MSB)

AX ← 02F4 H (i.e. LSB)

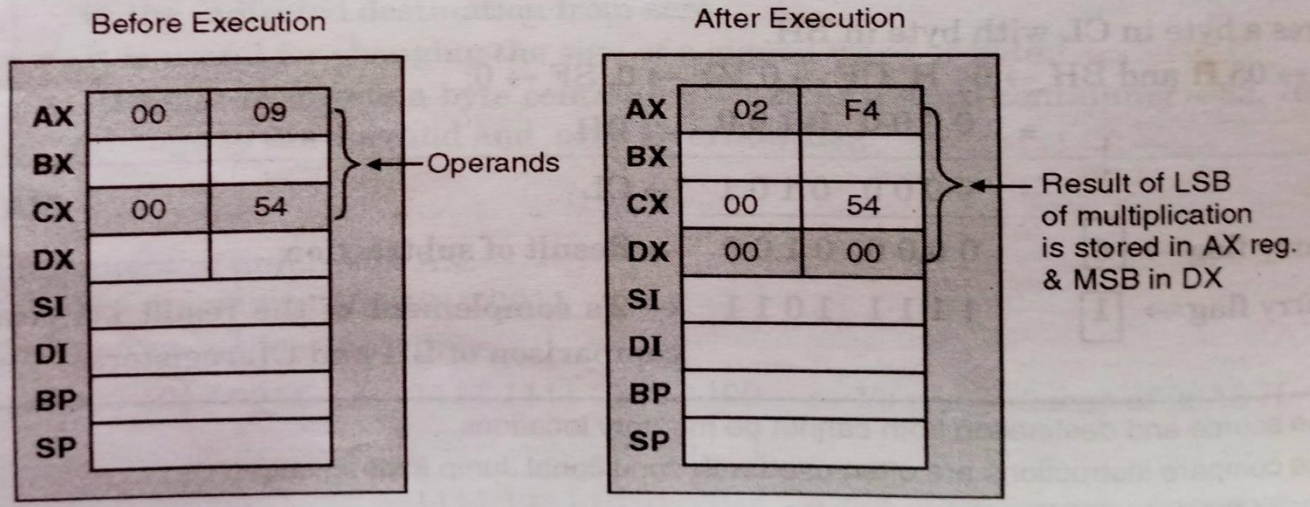


Fig. 5.7.19 : MUL AX, CX

Arithmetic group: Multiplication

2.IMUL: Multiply byte or word signed.

IMUL source

IMUL BL

Algo: When operand is a byte → AX=AL*operand

When operand is a word → (DX:AX)=AX*operand

- This instruction multiplies a signed byte from some source and a signed byte in AL, or a signed word from some source and a signed word in AX.
- The source can be register or memory location.
- When signed byte is multiplied by AL , a signed result will be put in AX.
- When a signed word is multiplied by AX,the MSB 16-bits are put in DX and LSB 16-bits are put in AX.
- If the magnitude of product does not require all bits of the destination, the unused bits are filled with the copies of the sign bit.
- To multiply a signed byte by a signed word it is necessary to move signed byte into lower byte of word and fill the upper byte of word with copies of sign bit. This can be done by CBW instruction.

Arithmetic group: Multiplication

2.IMUL: Multiply byte or word signed.

CBW instruction.

- If the upper byte of 16 bit result or upper word of 32 bit result contains only copies of sig bit (all 0's or all 1's) then the CF and 0F will both be zeros.
- If the upper byte of 16 bit result or upper word of 32 bit result contains part of th product then the CF and 0F will both be zeros.

Example : IMUL BL

Let AL = 69 decimal = 0100 0101 = 45H

BL = 14 decimal = 0000 1110 = 0EH.

```
IMUL BL      45 H    ← Contents of AL register
           ×  0E H    ← Contents of BL register
           ----------
           03 CE H   ← Result of multiplication  AX=03CE H
```

MSB = 0, Positive result magnitude in true form. ∴ SF = 0, CF = 0F = 1

Before Execution

AX		45
BX		0E
CX		
DX		
SI		
DI		
BP		
SP		

← Operands

After Execution

AX	03	CE
BX		0E
CX		
DX	00	00
SI		
DI		
BP		
SP		

← Result of LSB of signed multiplication is stored in AX reg.

SF	ZF	OF	AF	U	PF	U	CF
1	1	0	0	0	1	1	1

SF	ZF	OF	AF	U	PF	U	CF
0	1	0	0	0	1	1	0

Fig. 5.7.20 : IMUL BH

Arithmetic group: Multiplication

3. AAM – Integer multiply byte or word ASCII adjust for multiply

AAM

e.g. MUL BH

AAM AX

Algo: AH=Quotient of AL/10

AL=remainder of AL/10

- Numerical data coming into a computer from a terminal through keyboard is usually in ASCII code. The numbers 0 to 9 are represented by ASCII codes 30H to 39H
- Before multiplying two ASCII digits, the upper nibble bits of each need to be masked. This leaves unpacked BCD in each byte. After the two unpacked BCD digits are multiplied, the AAM instruction is used to adjust the product of two unpacked BCD digits in AX.
- It works only on register AL.
- It is used after multiplying the two unpacked BCD numbers.

Arithmetic group: Multiplication

3. AAM – Integer multiply byte or word ASCII adjust for multiply

digits in AX.

- It works only on register AL.
  It is used after multiplying the two unpacked BCD numbers

Example

Let AL → 0000 0101 = unpacked BCD 5
BH → 0000 1001 = unpacked BCD 9
MUL BH : AL * BH → Result in AX register
AX = 0000 0000 00101101 = 002D H
AAM AX : 0000 0100 0000 0101 = 0405 H
Which is unpacked BCD for 45.

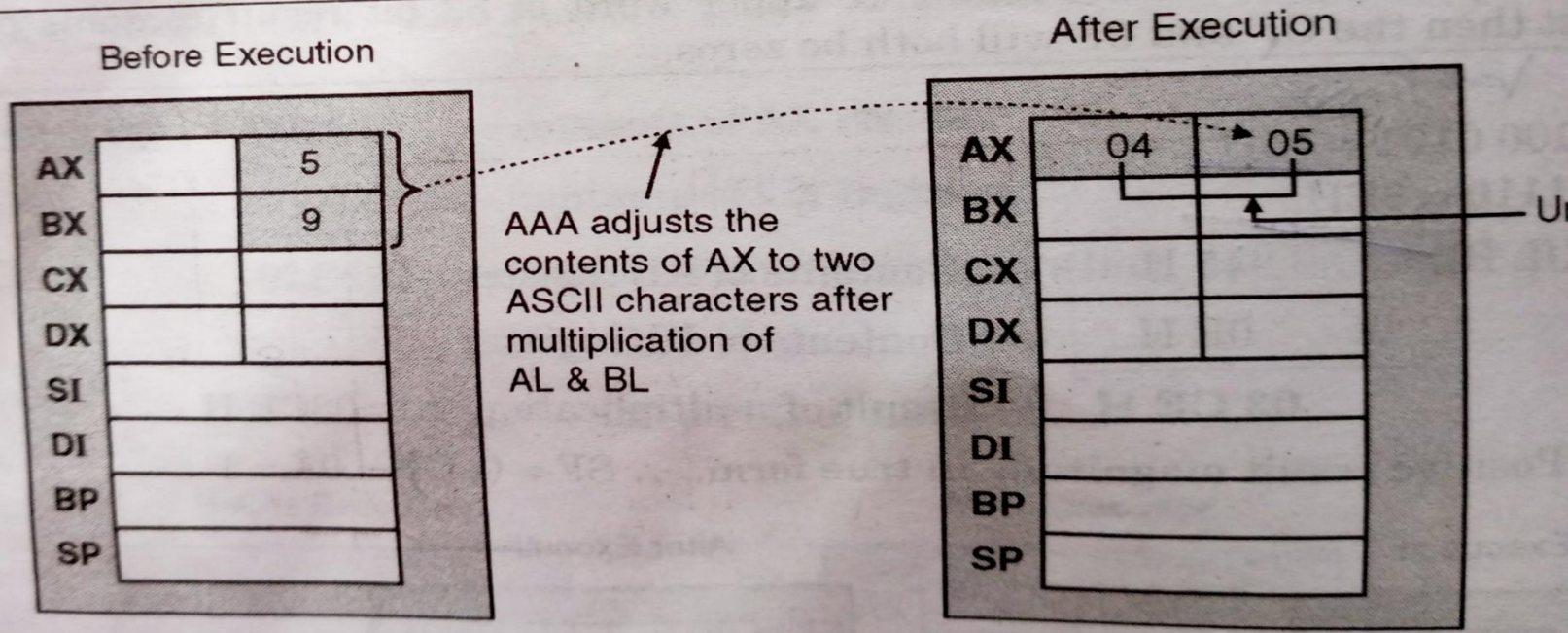


Fig. 5.7.21 : AAM

Arithmetic group: Division

1. DIV
2. IDIV
3. AAD
4. CBW
5. CWD

1. DIV- Divide byte or word unsigned

DIV source

DIV BH

Algo: When operand is a byte:

AL=AX/operand (Quotient)

AH=remainder (Modulus)

When operand is a word:

AX=(DS:AX)/operand (Quotient)

DX= remainder(modulus)

Arithmetic group: Division

1.**DIV- Divide byte or word unsigned:**

BYTE divisor

(AL) ← Quotient of (AX) / (source)
(AH) ← Remainder of (AX)/(source)

Word divisor

(AX) ← Quotient of (DX : AX)/ (source)
(DX) ← Remainder of (DX : AX)/(source)

Example : DIV BH

Divide word in AX by the byte in BH
AX / BH, AL → Quotient = 5 E H = 94 decimal.
AH → remainder = 65 H = 101 decimal.

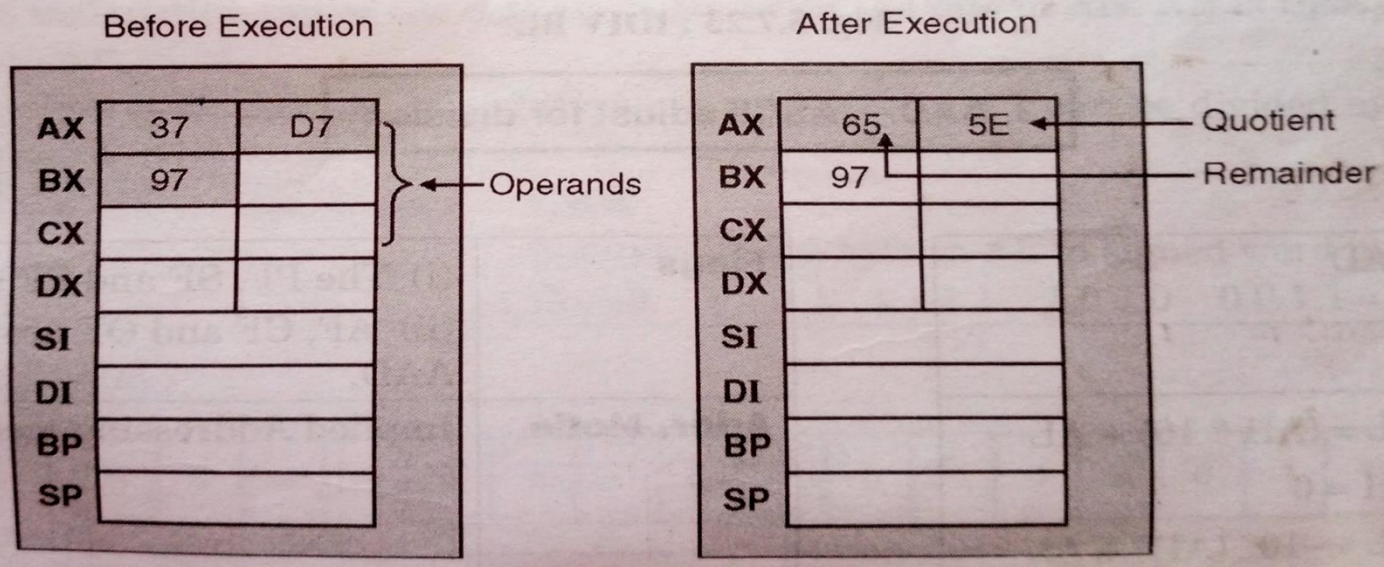


Fig. 5.7.22 : DIV BH

Arithmetic group: Division

2.**IDIV- Integer Divide byte or word**

IDIV source

IDIV BL

Algo: When operand is a byte:

AL=AX/operand (Quotient)

AH=remainder (Modulus)

When operand is a word:

AX=(DS:AX)/operand (Quotient)

DX= remainder(modulus)

e.g. Let AX=03 ABH BL=00 D3H

IDIV BL

Quotient in AL=ECH Remainder in AH=27H

Arithmetic group: Division

2.**IDIV- Integer Divide byte or word**

	AX ← quotient of (DS : AX) / source DX ← remainder of (DS : AX) / source This instruction performs two operation : 1. Divide a signed word by a signed byte 2. Divide a signed double word by a signed word.
mple	A signed word divided by a single byte Let AX = 03 AB H BL = 00 D3 H IDIV BL Quotient in AL = EC H Remainder in AH = 27 H

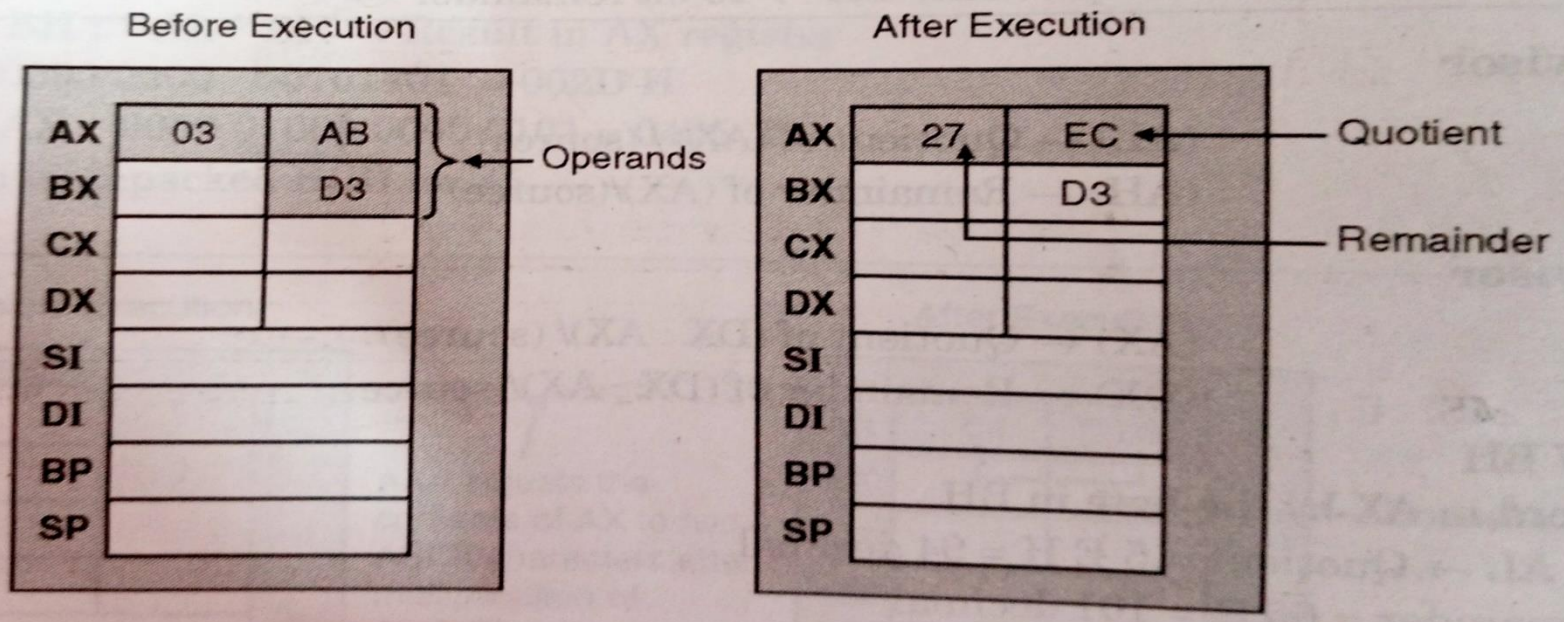


Fig. 5.7.23 : IDIV BL

Arithmetic group: Division

3. AAD: ASCII adjust for division

AAD

DIV CL

Algo: AL= (AH*10) + AL

AH=0

- It converts two unpacked BCD digits in AH and AL to the equivalent binary number in AL. This adjustment must be made before dividing the two unpacked BCD digits in AX, by an unpacked BCD byte .

  After the division

  AL→ unpacked BCD quotient

  AH→ unpacked BCD remainder

Arithmetic group: Division

3. AAD: ASCII adjust for division

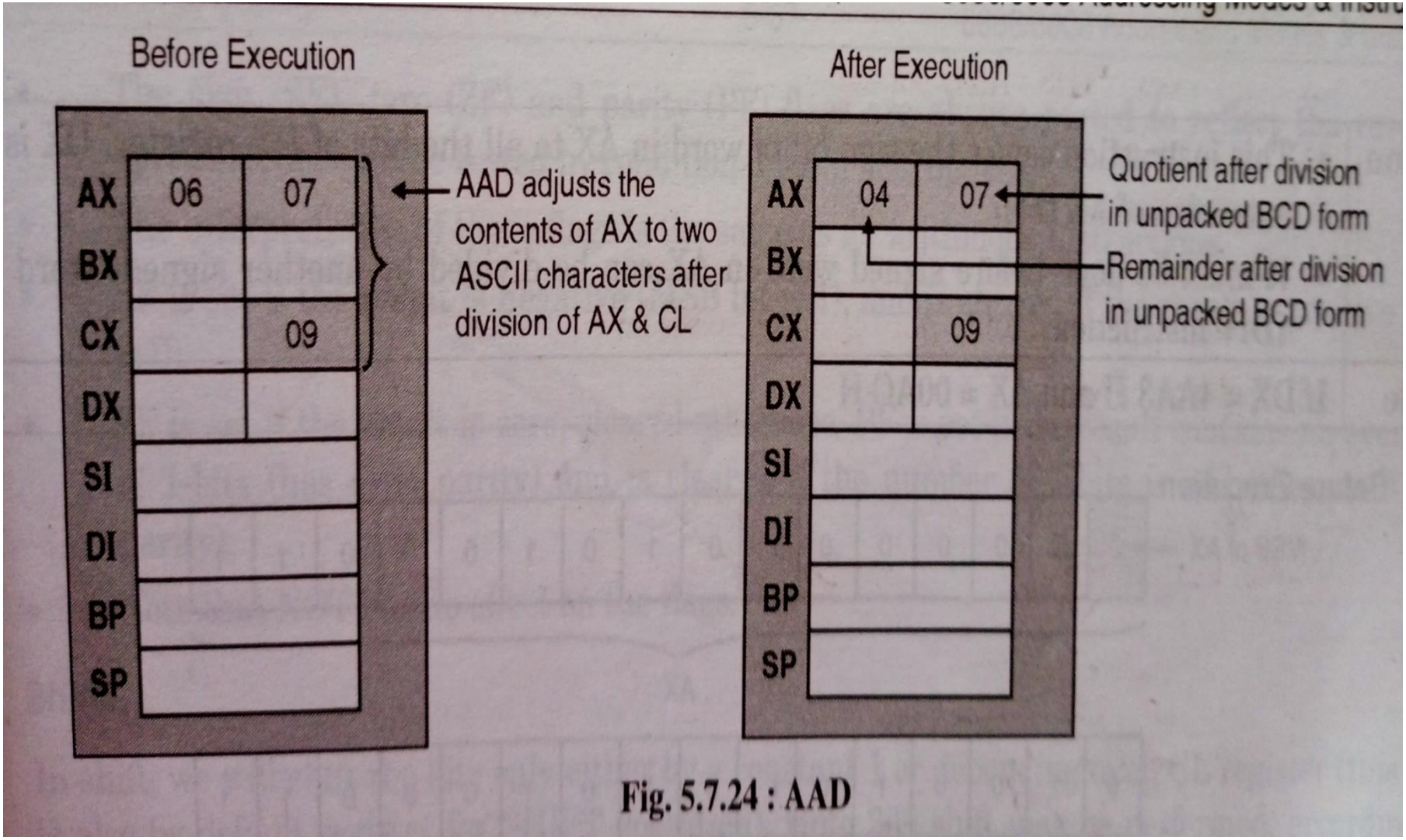


Fig. 5.7.24 : AAD

Arithmetic group: Division

4. **CBW: Convert byte to word**

CBW

Algo: If MSB bit of AL=1 then AH=255(FFH)

else AH=0

- This instruction copies the sign bit in AL to all the bits in AH. AH is then said to be a sign extension of AL.
- This operation must be done before a signed byte in AL can be divided by another signed byte with IDIV instruction.

Arithmetic group: Division

CBW – Convert byte to word

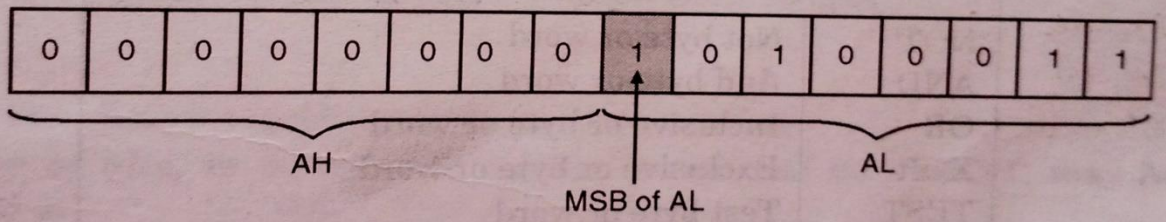
Mnemonic	CBW

Algorithm	If MSB bit of AL = 1 then
	AH = 255 (FF H)
	else
	AH = 0

Flags	No flags are affected.
Addr. Mode	Implied Addressing mode

Operation	• This instruction copies the sign bit in AL to all the bits in AH. AH is then said to be a sign extension of AL. • This operation must be done before a signed byte in AL can be divided by another signed byte with IDIV instruction.	
Example	AX = 00AC H = – 163 decimal	**CBW** Convert signed byte in AL to signed word in AX. Result : 1111 1111 1010 0011 = – 163 decimal

Before Execution



After Execution

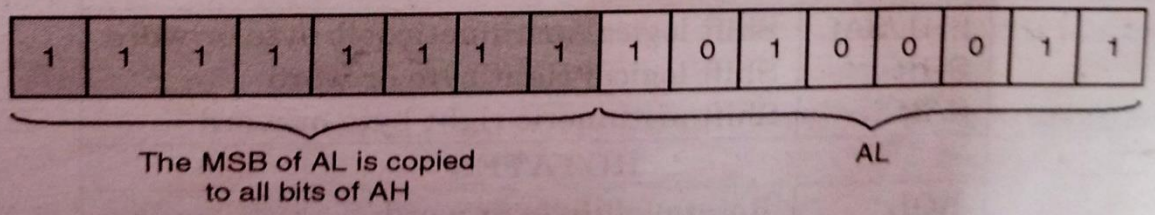


Fig. 5.7.25 : CBW

Arithmetic group: Division

5. **CWD: Convert word to double word.**

CWD

Algo: If MSB bit of AX=1 then, DX= 65535(FFFFH)

else DX=0

- This instruction copies the sign bit of word in AX to all the bits of DX register. DX is sign extension of AX then.
- It must be done before signed word in AX can be divided by another signed word with IDIV instruction.

Arithmetic group: Division

5.**CWD: Convert word to double word.**

ation	• This instruction copies the sign bit of word in AX to all the bits of DX register. I extension of AX then. • It must be done before signed word in AX can be divided by another signed IDIV instruction.
ple	If DX = 4AA3 H and AX = 00AC H

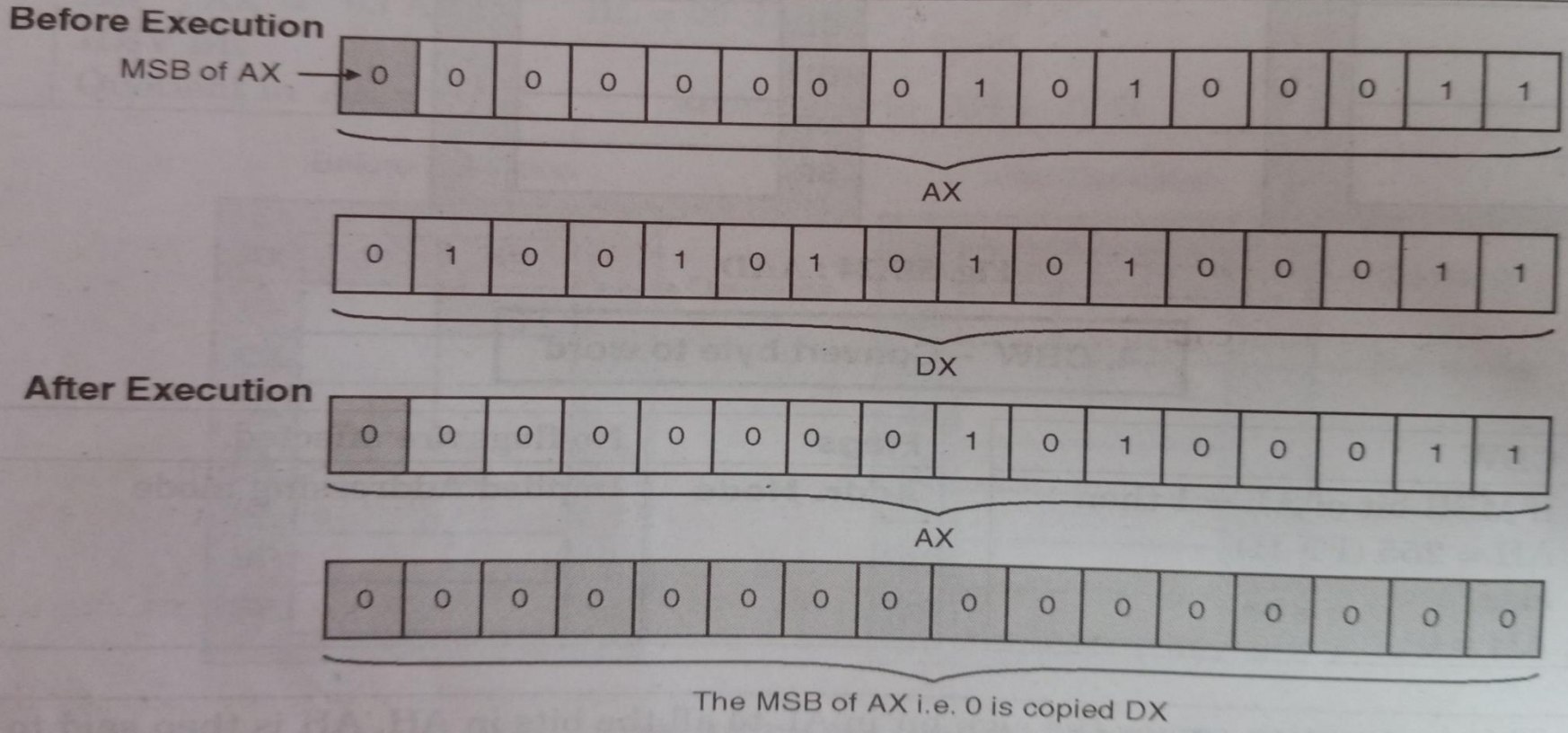


Fig. 5.7.26 : CWD

Bit manipulation instructions

Logicals:

1) NOT- Not byte or word

NOT destination

Algo: If bit is 1 turn it to 0

If bit is 0 turn it to 1

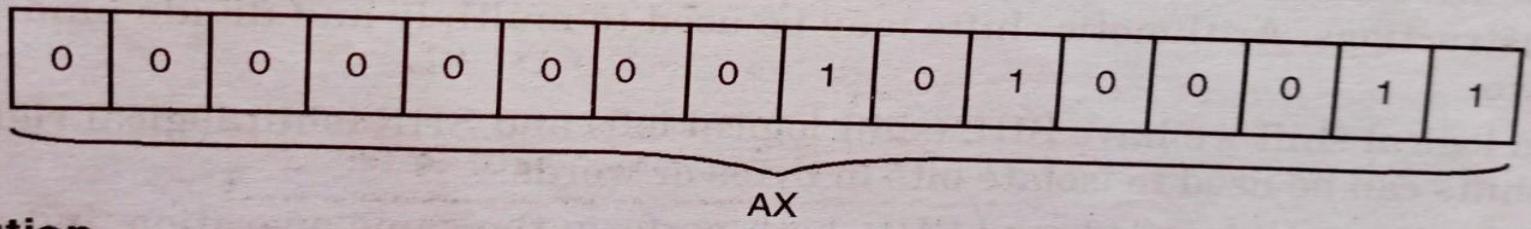
- This instruction inverts each bit of byte or word at the specified destination i.e. it finds 1's complement of the number.
- The destination can be a register or a memory location.
- The destination cannot be immediate data.
- The destination cannot be a segment register.

Bit manipulation instructions

1) NOT- Not byte or word

eration	Destination ← $\overline{\text{Destination}}$	• This instruction inverts each bit of byte specified destination i.e. it finds 1's comp number. • The destination can be a register or a memo • The destination cannot be immediate data. The destination cannot be a segment registe
mple	NOT AX AX ← $\overline{\text{AX}}$	This instruction complements the contents of A

Before Execution



After Execution

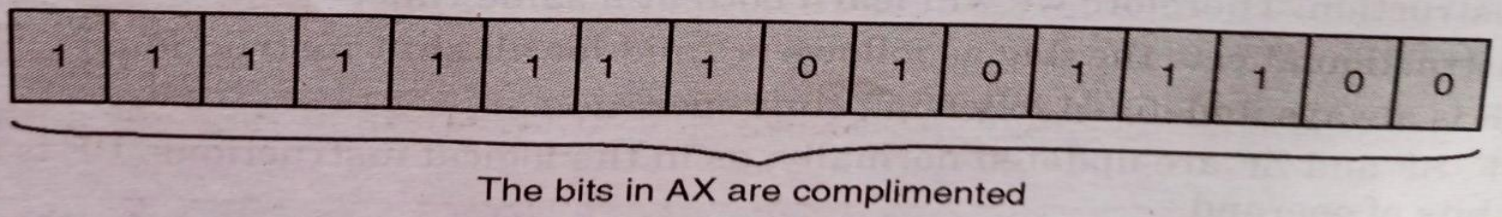


Fig. 5.8.1 : NOT AX

Bit manipulation instructions

Logicals:

2) AND - AND byte or word

AND destination ,source

AND AL,BL

Algo: destination= destination ^ source

- This instruction ANDs each bit in a source byte or word with the same number bit in a destination byte or word . The result is stored at specified location.
- The contents of specified source do not change.
- The source can be a register, memory location or an immediate number.
- The destination can be register or memory location.
- The source and destination both cannot be memory locations.
- The segment registers cannot be used as source or destination.

Bit manipulation instructions

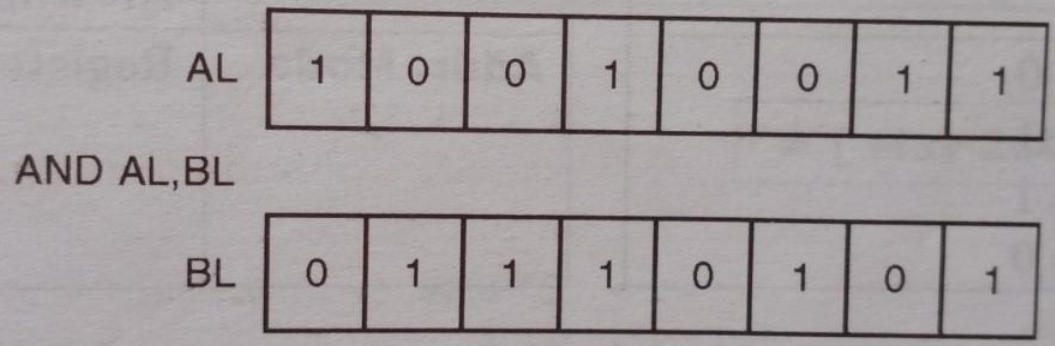
2) AND - AND byte or word

et

AL = 1 0 0 1 0 0 1 1 = 93 H
BL = 0 1 1 1 0 1 0 1 = 75 H
AL = 0 0 0 1 0 0 0 1 = 11 H

	Before execution	After execution
AL	93 H	11 H
BL	75 H	75 H

Before Execution



After Execution

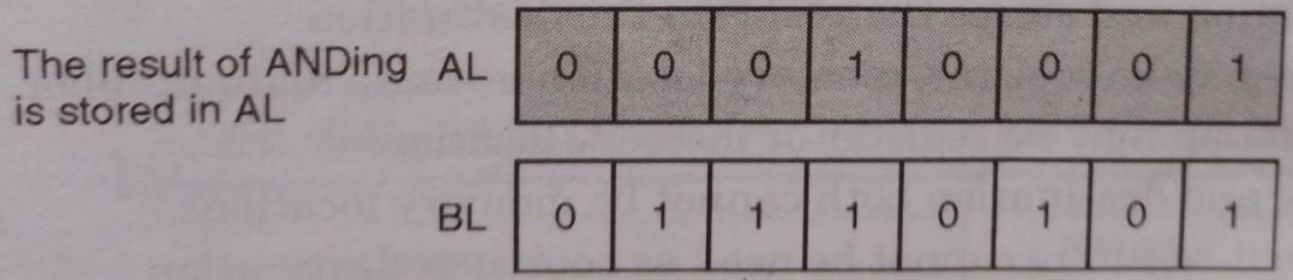


Fig. 5.8.2 : AND AL, BL

Bit manipulation instructions

Logicals:

3) OR – Inclusive or byte or word

OR destination,source

OR AL,BL

Algo: destination= destination \/ source

- This instruction ORs each bit in a source byte or word with the same number bit in a destination byte or word . The result is stored at specified location.
- The contents of specified source do not change.
- The source can be a register, memory location or an immediate number.
- The destination can be register or memory location.
- The source and destination both cannot be memory locations.
- The segment registers cannot be used as source or destination.

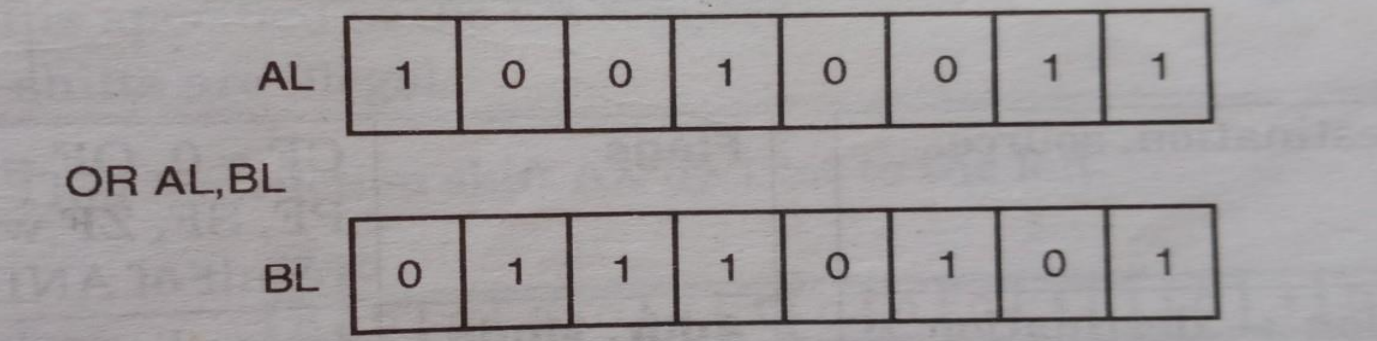
Bit manipulation instructions

3) OR – Inclusive or byte or word

	The segment registers cannot be used as source or des Destination can be a register or Memory location.
OR AL , BL	• This instruction ORs each bit in a AL register w register. • The result is stored in the AL register.

Let AL = 93 H and BL = 75 H

Before Execution



After Execution

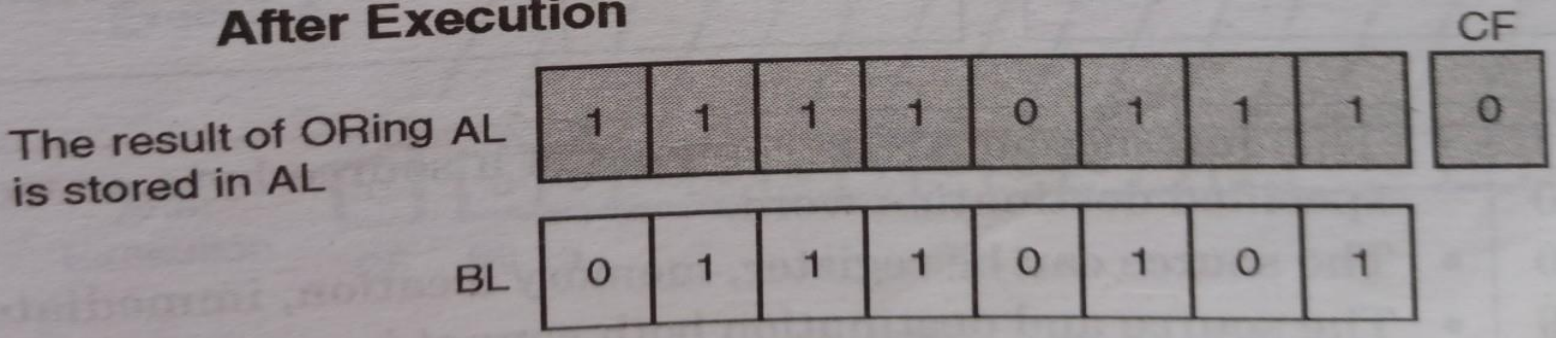


Fig. 5.8.3 : OR AL, BL

Bit manipulation instructions

Logicals:

4) XOR – Exclusive or byte or word

XOR destination,source

XOR AL,BL

Algo: destination= destination \/ source

- This instruction logically XORs each bit in a source byte or word the corresponding bit in the destination and stores the result in the destination.
- The contents of specified source do not change.
- The source can be a register, memory location or an immediate number.
- The destination can be register or memory location.
- The source and destination both cannot be memory locations.
- The segment registers cannot be used as source or destination.

Bit manipulation instructions

4) XOR – Exclusive or byte or word

	• The source and destination both cannot be memory locations. • The segment registers cannot be used as source or destination.	
mple	XOR AL , BL	• This instruction XORs each bit in a in BL register. • The result is stored in the AL register

L = 93 H and BL = 75H

Before Execution

AL	1	0	0	1	0	0	1	1
XOR AL,BL								
BL	0	1	1	1	0	1	0	1

After Execution

The result of XORing is stored in AL	AL	1	1	1	0	0	1	1	0
	BL	0	1	1	1	0	1	0	1

Fig. 5.8.4 : XOR AL, BL

Bit manipulation instructions

Logicals:

5) TEST – Test byte or word

TEST destination,source

TEST AL,75h

Algo: destination= destination ^ source

- This instruction ANDs contents of a source byte or word with contents of specified destination word.
- The source can be register , memory location,immediate data.
- The source and destination both cannot be memory locations.
- Segment registers are not allowed to be used as source or destination.
- The destination can be a register or memory location.
- Flags are affected but neither operand is changed
- It is used to set flags before conditional JUMP.

TEST AL,75h (AND immediate number 75h with AL)

PF=0,SF=0,ZF=0,CF=0,OF=0

Bit manipulation instructions

5) TEST – Test byte or word

		• The destination can be a register or memory location. • Flags are affected, but neither operand is changed. • It is used to set flags before conditional JUMP.
	TEST AL, 75 H	Let AL = 0 1 1 1 0 1 0 1 AND Immediate number 75H with AL. PF = 0, SF = 0, ZF = 0, CF = 0, OF = 0