2. Register addressing mode (Data in register)☒☒

☒ In register addressing mode data to be operand is in general purpose register

Example :

MOV CL, BL → Content (8 bit data) of reg. BL is transferred into reg. CL

MOV CX, BX → Content (16 bit data) of reg. BX is transferred into reg. CX

MOV CL, BX ✗ It is not possible to perform 16 bit operation with 8 bit reg or vice versa.

3. Direct addressing mode (Address in Instruction)☒☒☒

☒☒
☒ In direct addressing mode operand is given by a direct address where the data is present.

Example 1 :

Anything in [ ] refers address

MOV CL, [2000] → The Content which present at memory location 2000 transfer into reg. ~~CL~~
CL DS : [2000]

DS

0000	1A
0001	08
.	
.	
→ 2000	04 → CL = 04
2001	05
2002	06
.	
FFFF	

Note :

- Data is always refer from data segment (DS)
- DS has starting address
- From 0000 to FFFF these are the offset
- BIU section of 8086 generate 20 bit physical address using following formula :

$$\boxed{PA = \text{Seg address} * 10\,h + \text{offset}}$$

Direct addressing mode (Address in Instruction)☒☒☒

☒☒
☒ In direct addressing mode operand is given by a direct address where the data is present

Example 2 :

MOV [2000], CL → The Content of reg. CL transfer into memory location 2000
CL → DS : [2000]. assume CL =07

DS

0000	1A
0001	08
.	
.	
→ 2000	~~04~~ 07
2001	05
2002	06
.	
FFFF	

Note :

- Data is always refer from data segment (DS)
- DS has starting address
- From 0000 to FFFF these are the offset
- BIU section of 8086 generate 20 bit physical address using following formula :

$$\boxed{PA = \text{Seg address} * 10\,h + \text{offset}}$$

1. Immediate addressing mode (Data in Instruction)

In immediate addressing mode the data to be used is immediately given in the instruction.

Example :

MOV CL, 02 H → 02 (8 bit data) is transfer into reg. CL

MOV CX, 2005 H → 2005 (16 bit data) is transfer into reg. CX in following manner :

[redacted]

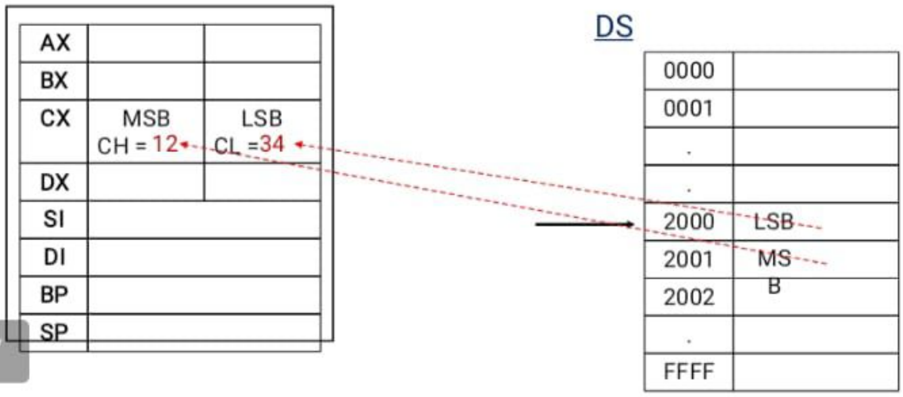
Direct addressing mode (Address in Instruction)

In direct addressing mode operand is given by a direct address where the data is present

Example 3 : MOV CX , [2000] ⟶ The Content of reg. CX transfer into memory location 2000
CX ⟵ DS : [2000]

16 bit data 1234 is stored into memory locations in following manner :

12	34
MSB	LSB



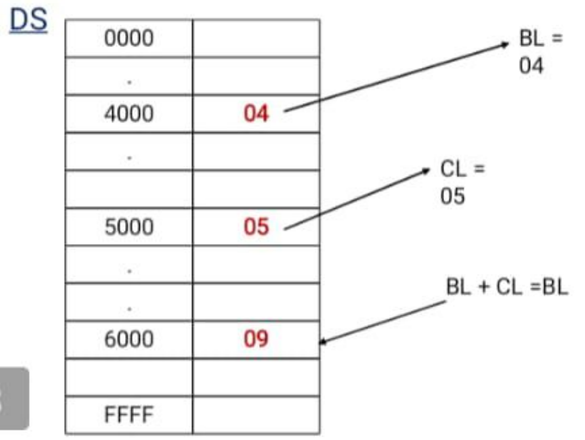
AX		
BX		
CX	MSB CH = 12	LSB CL =34
DX		
SI		
DI		
BP		
SP		

DS

0000	
0001	
.	
.	
2000	LSB
2001	MSB
2002	
.	
FFFF	

Direct addressing mode (Address in Instruction)

1. Move content of 4000 th location into BL register
2. Move content of 5000 th location into CL register
3. Add contents of BL and CL
4. Store the result on 6000 th memory location



DS

0000	
.	
4000	04
.	
5000	05
.	
.	
6000	09
FFFF	

1. Mov BL, [4000]
2. Mov CL, [5000]
3. Add BL, CL
4. Mov [6000], BL

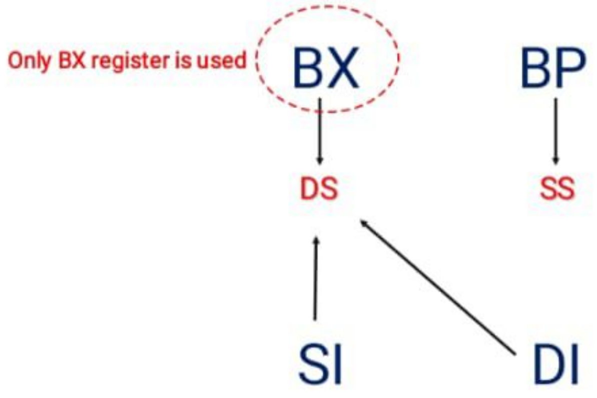
4. Indirect addressing mode (Address in register)

In indirect addressing mode the instruction dose not have the address of the data to be operated on. But the instruction points where the address is stored i.e. it is indirectly specifying the address of memory location where the data is stored or is to be stored.

There are four types of indirect addressing mode :

1. Register Indirect (address simply given by register)
2. Register relative (address in reg + relative)
3. Based indexed (address in base reg + index reg)
4. Based relative indexed (address in base reg + index reg + relative)

Rules related with register



1.Register Indirect addressing mode (Address in reg)

In direct addressing mode operand is given by a direct address where the data is present

Example : MOV CL,[BX] ⟶ The Content of memory location 4000 transfer into CL
CL ⟵ DS : [4000].

Note : • Only BX register is used

DS

0000	1A
0001	08
.	
.	
4000	04
4001	05
4002	06
.	
FFFF	

CL = 04

MOV BX, 4000H Load 4000 H immediate data into reg BX

MOV CL,[BX] Now 4000 H will be treated as a memory location and content of this location will be transfer into reg. CL

Difference between direct and indirect addressing :

Direct addressing :

Mov BX, [4000]
CL ⟵ DS : [4000]

DS

0000	1A
0001	08
.	
.	
4000	07
4001	05
4002	06
.	
FFFF	

Indirect addressing :

Mov BX, 4000H ⟶ Initialisation of BX with 4000
Mov CL, [BX]
CL ⟵ DS : [4000]

In C prog:

Arr[0] If x = a[5], x will get data at arr[5]
.... If x = a[9], x will get data at arr[9]
Arr[9] Direct addressing :

i = 5
x = a[i] Indirect addressing :

Advantage of Indirect addressing:

1. Suppose we want to transfer 100 location from 4000 then using direct addressing mode we have to write 100 times above instruction for e.g. MOV CL,[4001] , MOV CL,[4002] and so on.

2. By using indirect addressing we can implement following code

```
Mov BX, 4000H
Mov CL, [BX]
INC BX
```

2 Register relative addressing mode (address in ~~reg~~+ reg+ relative)

-> In this mode, the operand address is calculate using one of the base registers and an 8 bit or a 16 bit ~~distar~~ displacement.

ex :- MOV CL,[BX + ~~p~~displacement]

MOV CL, [BX + 02h]

Note :- Only BX register is used

3 Based indexed (address in base reg + index reg)

-> In this mode, operand address is calculated as base register ~~+~~ plus an index register and an 8 bit or a 16 bit displacement.

ex:- MOV CL,[BX +SI]

4.Based relative Index addressing mode (Address in base reg + Index + rel

In this mode, operand address is calculated as base register plus an index register and 8 or 16 bit displacement.

Example :

MOV CL,[BX + SI + displacement] → This instruction moves a byte from the address pointed by BX + SI + 02 in data segment to CL.

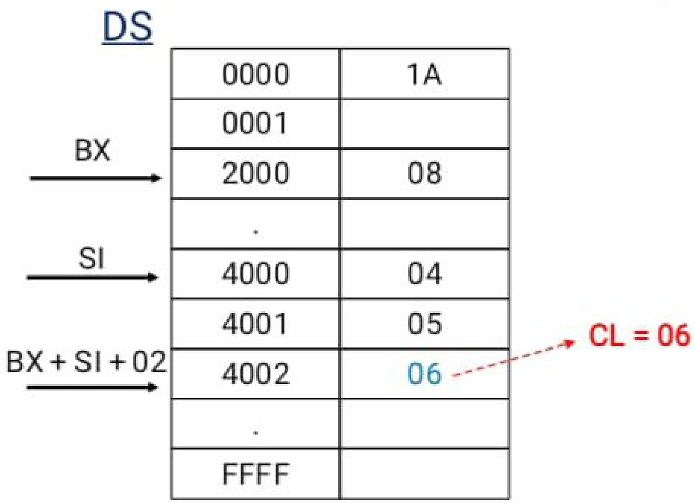
MOV CL,[BX + SI + 02h]

CL ← DS : [2000 + SI + 02].

Note : • Only BX register is used

MOV CL,[BX +SI + displacement] Increment memory locations by displacement

MOV CL,[BX + SI - displacement] Decrement memory locations by displacemer



DS

0000	1A
0001	
2000	08
.	
4000	04
4001	05
4002	06
.	
FFFF	

5. Implied addressing mode (Nothing is given in instruction)

In this mode, the operands are implied and are hence not specified in the instruction.

Example :

Operations are related with specific register

STC → Set carry flag

DAA → Decimal adjust after addition

1. Immediate addressing mode (Data in Instruction)

2. Register addressing mode (Data in register)

3. Direct addressing mode (Address in Instruction)

4. Indirect addressing mode (Address in Register)

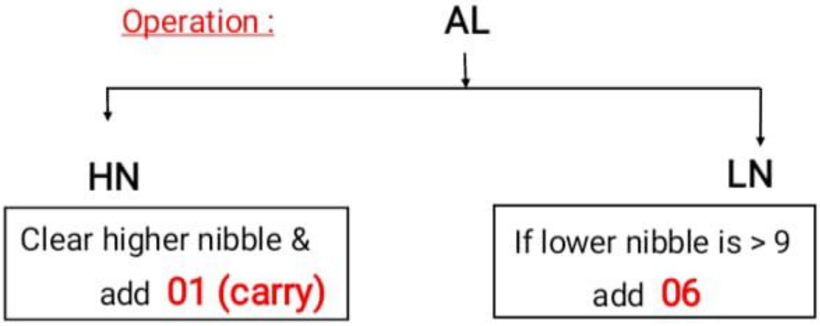
5. Implied addressing mode (Nothing is given in instruction)

AAA (ASCII Adjust after addition)

- 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 .
- The 8086 allows to add the ASCII codes for two decimal digits without masking off "3" in the upper nibble for each.
- After addition , AAA instruction is used to make sure that the result is the correct unpacked BCD
- The AAA instruction works only AL register.

Mnemonic: AAA

Operation :



Clear higher nibble & add 01 (carry)

If lower nibble is > 9 add 06

Flags :

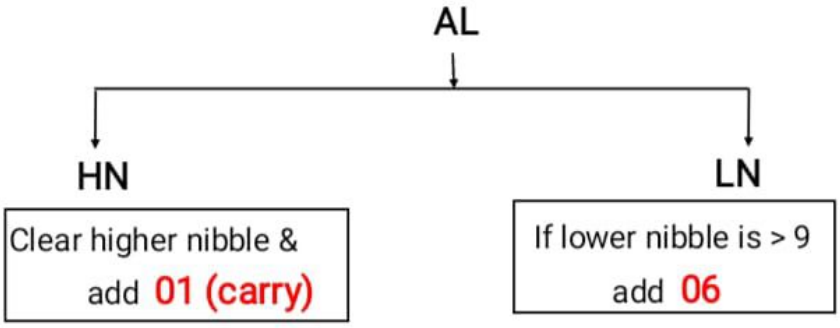
AF and CF flags are changed

Addressing mode : Implied

AL = 0011 0101 (ASCII 5)

BL = 0011 1001 (ASCII 9)

Rule :



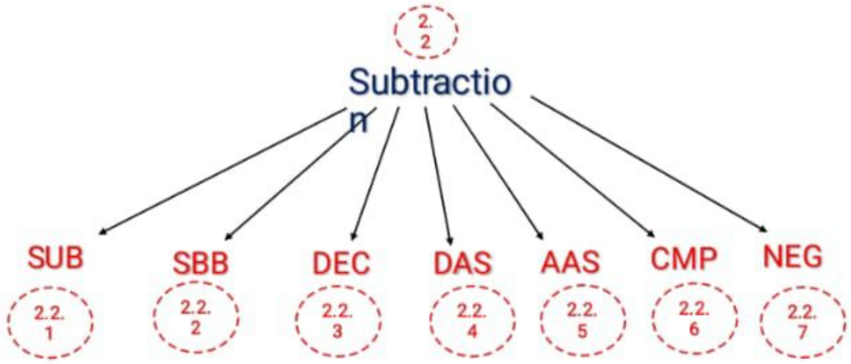
Clear higher nibble & add 01 (carry)

If lower nibble is > 9 add 06

```
   35 h
 + 39 h
 ------
   6E h

              0011  0101
            + 0011  1001
          ------------------
              0110  1110
        +           0110  Add 6
        carry  1     11
          ------------------
 Clear HN     0000  0100
           +     1
          ------------------
              0001  0100  14 h
```

2.2 Subtraction Group□


2.2

Subtraction

SUB 2.2.1

SBB 2.2.2

DEC 2.2.3

DAS 2.2.4

AAS 2.2.5

CMP 2.2.6

NEG 2.2.7

2.1.1 SUB – Subtract byte or word

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

Mnemonic: SUB destination , source

SUB Operand 1 , Operand 2

Operation :

Destination ← Destination - Source

Operand 1 ← Operand 1 - Operand 2

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

Flags :

All Flags affected

SUB register , register

- This instruction subtracts the data in registers.
- The result is stored in register.
- This instruction can be 8/16 bit.

Mnemonic: SUB register , register

Example: SUB BL, CL

BL ← BL - CL

Operation :

Register ← Register - Register

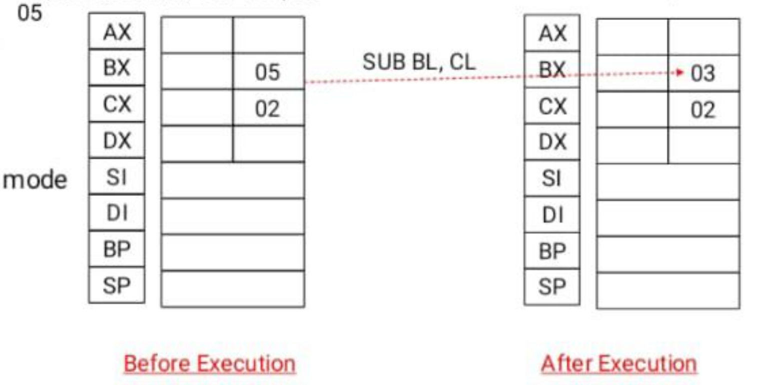
Addressing mode : Register addressing mode

Flags :

All Flags affected

Before execution CL = 02 , BL = 05

After execution CL = 04 , BL = 07


SUB BL, CL

AX		
BX		05
CX		02
DX		
SI		
DI		
BP		
SP		

Before Execution

AX		
BX		03
CX		02
DX		
SI		
DI		
BP		
SP		

After Execution

2.2.2 SBB – Subtract with borrow

- This instruction subtracts destination operand contents , source operand contents and Carry flag content.
- Result is stored back to destination operand.
- The source and destination can be 8/16 bit register or memory location. The source can also 8/16 bit register or memory location and immediate data.
- It is easy to perform multiple – precision arithmetic by using SBB instruction.

Mnemonic: SBB destination , source

Operation :

Destination ⟵ Destination - Source - CY

Example: SBB BL, CL

Flags :

All Flags affected

2.2.3 DEC – Decrement byte or word by 1

- This instruction subtracts 1 from the destination operand.
- The operand can be a register or memory location.
- The operand may be a byte or word and it is treated as an unsigned binary number.

Mnemonic: DEC destination

Operation :

Destination ⟵ Destination - 1

Flags : All Flags affected except carry flag.(carry flag not changed)

Example: DEC CX ⟶ IF CX = 1234 , after DEC CX will be 1233

DEC AL ⟶ IF AL = 08 , after DEC AL will be 07

DEC [2000] ⟶ This instruction decrements the content of memory location 2000 by 1.

If 2000 = 04 , after DEC it will be 04

2.2.6 CMP –Compare byte or word

- 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 destination or source.
- The destination and source remain unchanged, only flags are updated.

Mnemonic: CMP Destination , Source

Operation :

Destination - source

Flags : AF, OF, SF, ZF, PF are updated

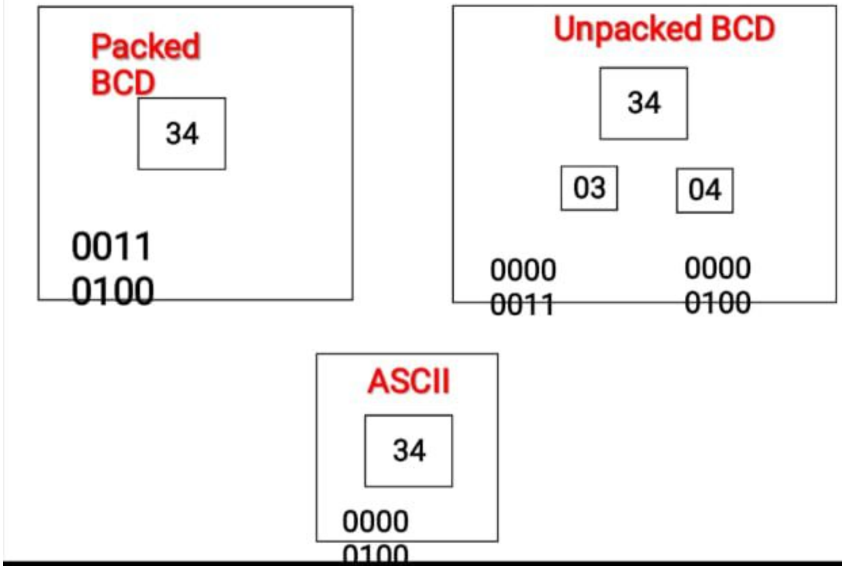
Compare	CF	ZF	SF	
Source > Destination	1	0	1	Subtraction required borrow, CF =1
Source < Destination	0	0	0	No borrow required, CF =0
Source = Destination	0	1	0	Result of subtraction is zero.

******* After each operation in this instruction carry flag changed......

if 0 then it will become 1

if 1 then it will become 0

AAA Instruction



Packed BCD

34

0011 0100

Unpacked BCD

34

03

04

0000 0011

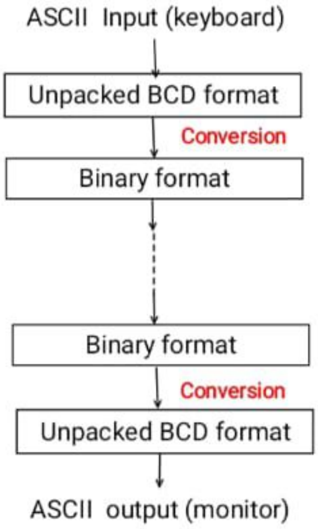
0000 0100

ASCII

34

0000 0100

- Computer follows ASCII input and output.
- We get from keyboard ASCII character and we have to send out ASCII character to monitor. Therefore we have to convert ASCII to binary



ASCII Input (keyboard)

Unpacked BCD format

Conversion

Binary format

Binary format

Conversion

Unpacked BCD format

ASCII output (monitor)

ASCII code for digit 0-9

Key	ASCII (hex)	BCD (unpacked)
0	30	0011 0000
1	31	0011 0001
2	32	0011 0010
3	33	0011 0011
4	34	0011 0100
5	35	0011 0101
6	36	0011 0110
7	37	0011 0111
8	38	0011 1000
9	39	0011 1001

2.2.7 NEG –Negate byte or word

- This instruction replace the number is a destination with 2's complement of that number.
- The destination can be register or memory.

Example: NEG BL

BL ← 2's complement of BL

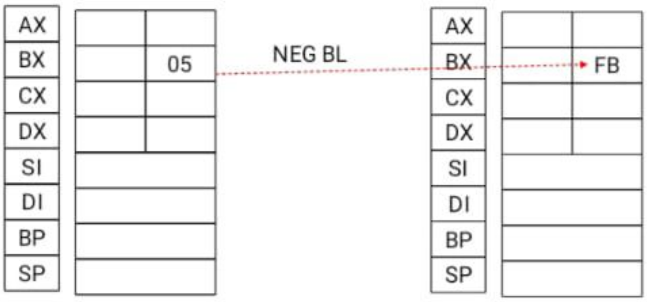
Mnemonic: NEG Destination

Operation :

Destination ← 2's complement of Destination

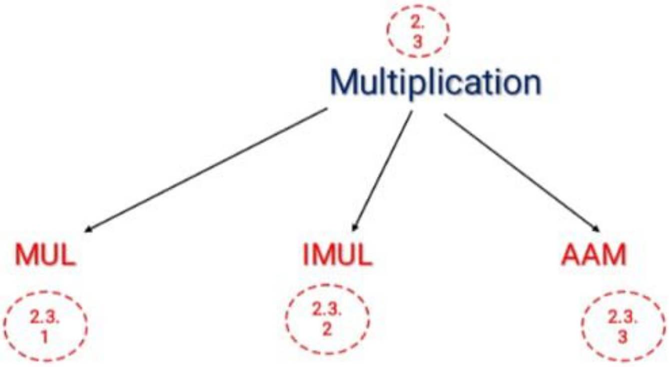
Flags : All flags are updated

Before execution BL = 05   After execution BL = FB



Before Execution   After Execution

2.3 Multiplication Group



2.3.1 MUL – Multiply byte or word unsigned

- This instruction multiplies an unsigned byte from source with an byte in the AL register or an unsigned word from source with an unsigned word in AX.
- When a byte is multiplied by contents of AL, the result is stored in AX.
- The MSB of result is stored in AH register and the LSB of result is stored in the AL register.
- When a word is multiplied by contents of AX, then MSB of result is stored in DX register and the LSB of result is stored in the DX register.

Example: MUL BL   Where AL = 09 H,
BL = 02 H

Operation : Mnemonic: MUL multiplier

AX ← Multiplication

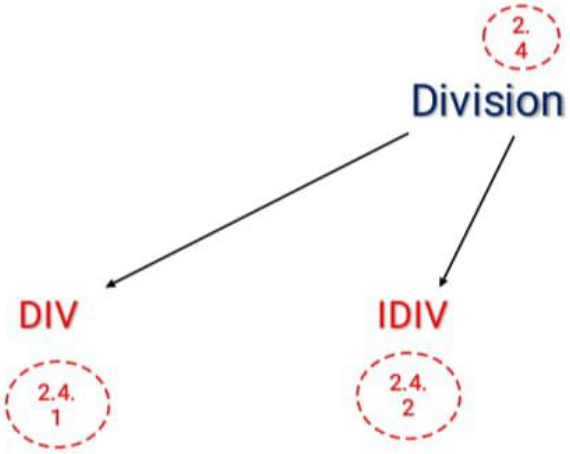
MSB AH   LSB AL

$$\begin{array}{r} 09\text{ h} \\ \times\ 02\text{ h} \\ \hline 0012\text{ h} \end{array}$$

AH = 00 H
AL = 12 H

Flags : AF,PF,SF,ZF undefined
CF and OF will both be 0

2.4 Division Group



2.4

Division

DIV

2.4.1

IDIV

2.4.2

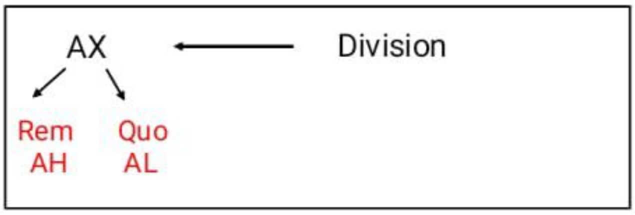
2.4.1 DIV – Multiply byte or word unsigned

- This instruction divides an unsigned byte from source with an byte in the AL register or an unsigned word from source with an unsigned word in AX.
- When a byte is divided , the result is stored in AX.
- The remainder of result is stored in AH register and quotient of result is stored in the AL register.
- For 16 bit operation DX register is used.

Example: DIV BL

Where AL = 08 H,
BL = 02 H

Operation : Mnemonic: DIV Divider



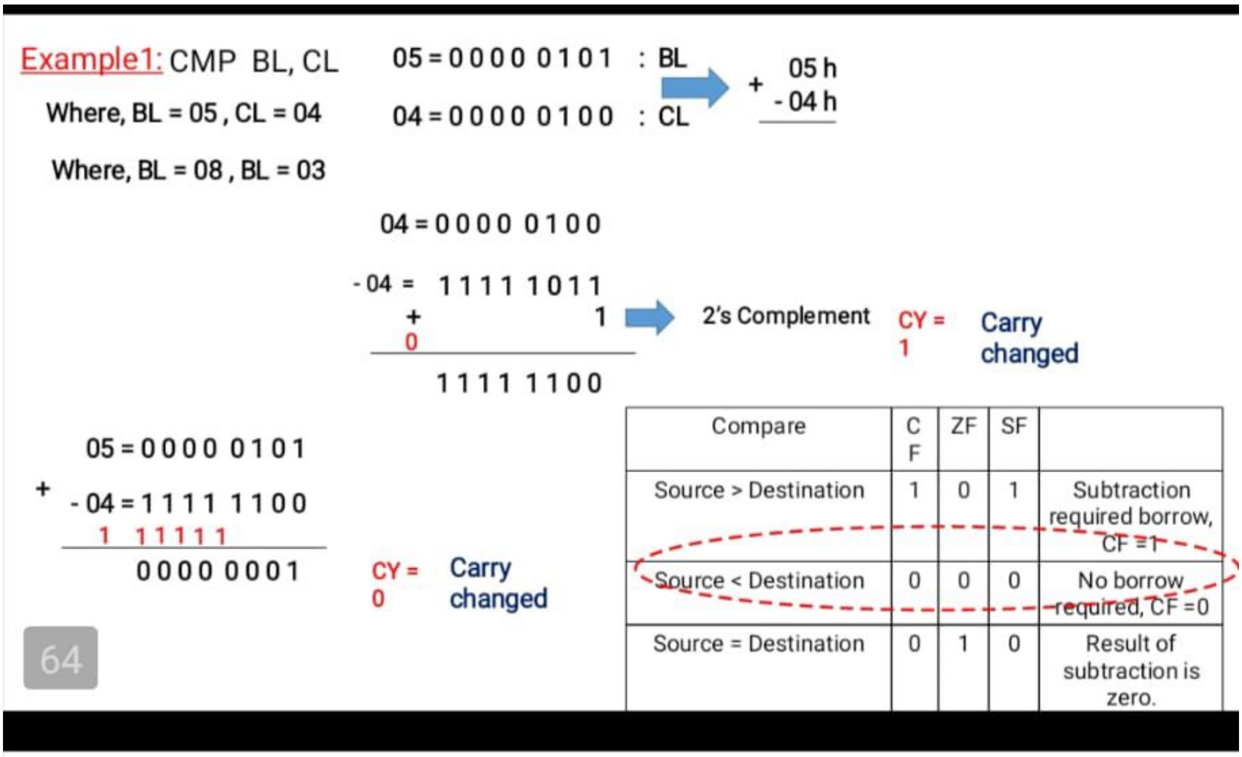
AX ← Division

Rem AH

Quo AL

$$\begin{array}{r} 08\text{ h} \\ /\ \ 02\text{ h} \\ \hline 04\text{ h} \end{array} \longrightarrow \text{Q}$$

AH = remainder
AL = quotient



Example1: CMP BL, CL

Where, BL = 05 , CL = 04

Where, BL = 08 , BL = 03

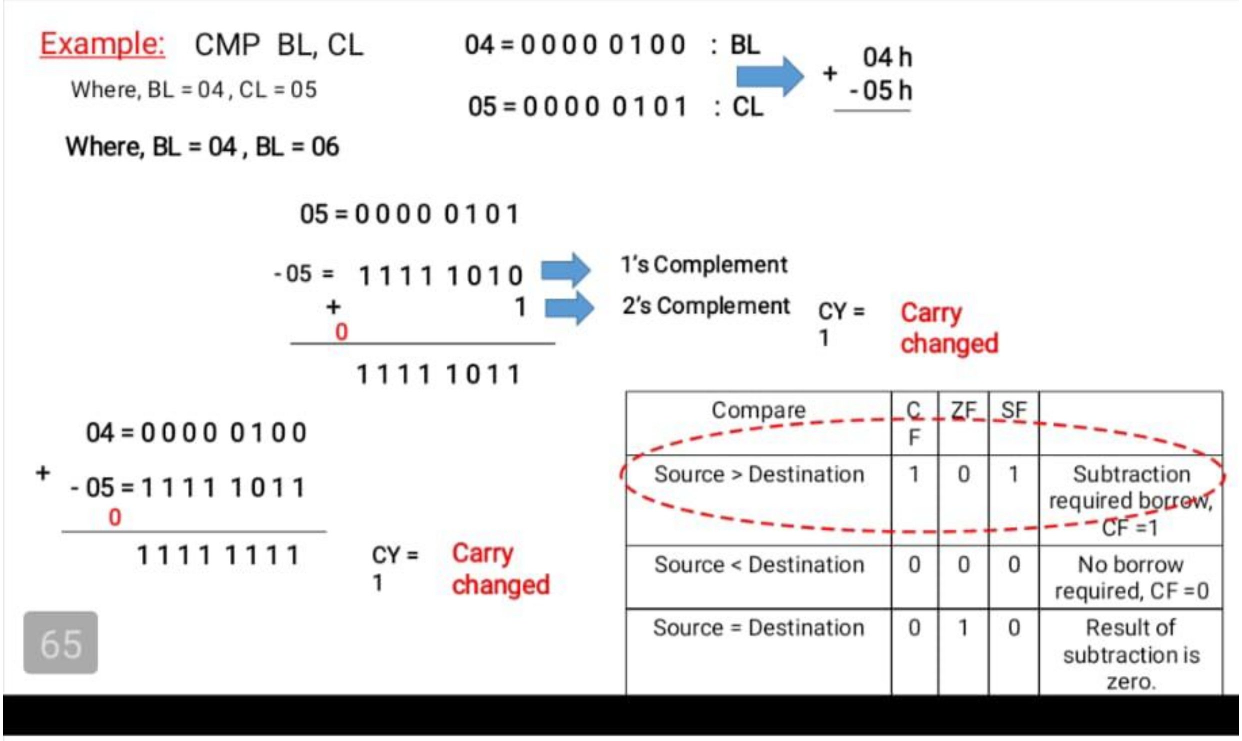
```
05 = 0000 0101  : BL        + 05 h
04 = 0000 0100  : CL        - 04 h
```

```
 04 = 0000 0100

- 04 =  1111 1011
   +            1    2's Complement   CY =   Carry
   0                                  1      changed
        1111 1100
```

```
    05 = 0000 0101
+ - 04 = 1111 1100
     1  11111
         0000 0001        CY =   Carry
                          0      changed
```

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



Example: CMP BL, CL

Where, BL = 04 , CL = 05

Where, BL = 04 , BL = 06

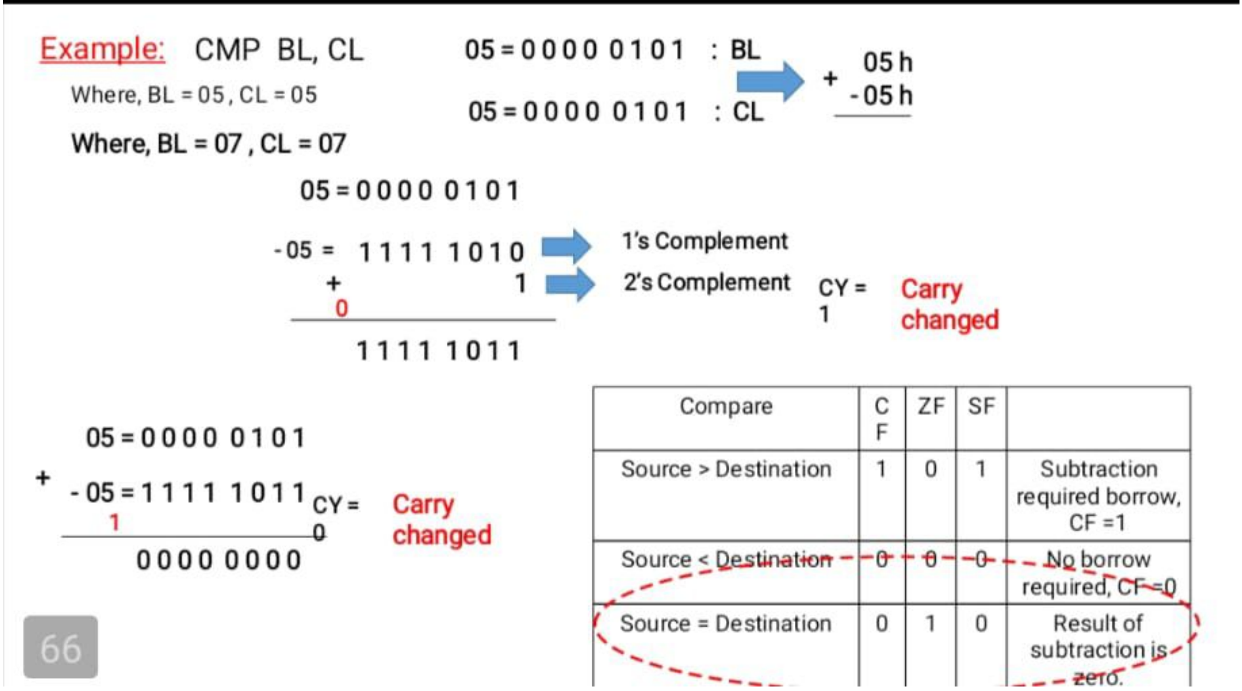
```
04 = 0000 0100  : BL        + 04 h
05 = 0000 0101  : CL        - 05 h
```

```
 05 = 0000 0101

- 05 =  1111 1010    1's Complement
   +            1    2's Complement   CY =   Carry
   0                                  1      changed
        1111 1011
```

```
    04 = 0000 0100
+ - 05 = 1111 1011
     0
         1111 1111        CY =   Carry
                          1      changed
```

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



Example: CMP BL, CL

Where, BL = 05 , CL = 05

Where, BL = 07 , CL = 07

```
05 = 0000 0101  : BL        + 05 h
05 = 0000 0101  : CL        - 05 h
```

```
 05 = 0000 0101

- 05 =  1111 1010    1's Complement
   +            1    2's Complement   CY =   Carry
   0                                  1      changed
        1111 1011
```

```
    05 = 0000 0101
+ - 05 = 1111 1011  CY =   Carry
     1              0      changed
         0000 0000
```

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

2.1.1 ADD – Add byte or word

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

Mnemonic: ADD destination , source

ADD Operand 1 , Operand 2

Operation :

Destination ← Source + Destination

Operand 1 ← Operand 1 + Operand 2

Flags :

All Flags affected

ADD register , register

Mnemonic: ADD register , register

Operation :

Register ← Register + Register

Addressing mode : Register addressing mode

Flags :

All Flags affected

ADD register , memory

Mnemonic: ADD register , memory

Operation :

Register ← Register + content of memory location

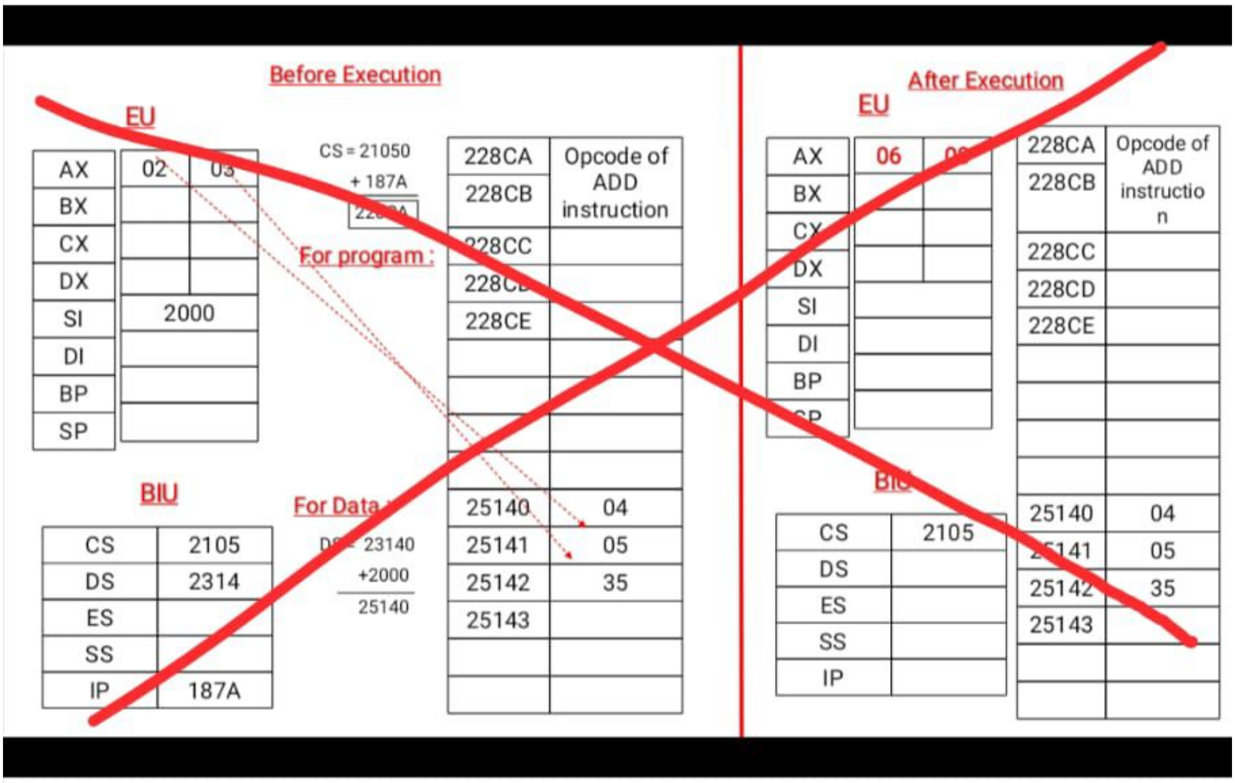
Addressing mode : Register addressing mode

Example: ADD AX, [2000]

AX ← AX + content of memory location

Flags :

All Flags affected



Before Execution

EU

AX	02	03
BX		
CX		
DX		
SI	2000	
DI		
BP		
SP		

CS = 21050
+ 187A
22[illegible]A

For program :

For Data :

D[illegible] 23140
+2000
25140

BIU

CS	2105
DS	2314
ES	
SS	
IP	187A

228CA	Opcode of ADD instruction
228CB	
228CC	
228C[illegible]	
228CE	
25140	04
25141	05
25142	35
25143	

After Execution

EU

AX	06	0[illegible]
BX		
CX		
DX		
SI		
DI		
BP		
SP		

BIU

CS	2105
DS	
ES	
SS	
IP	

228CA	Opcode of ADD instructio n
228CB	
228CC	
228CD	
228CE	
25140	04
25141	05
25142	35
25143	

2.1.2 ADC – Add with carry

- This instruction adds destination operand contents , source operand contents and Carry flag content.
- Result is stored back to destination operand.
- The source and destination can be 8/16 bit register or memory location. The source can also 8/16 bit register or memory location and immediate data.
- It is easy to perform multiple – precision arithmetic by using ADC instruction.

Mnemonic: ADD destination , source

Operation :

Destination ← Destination + Source + CY

Flags :

All Flags affected

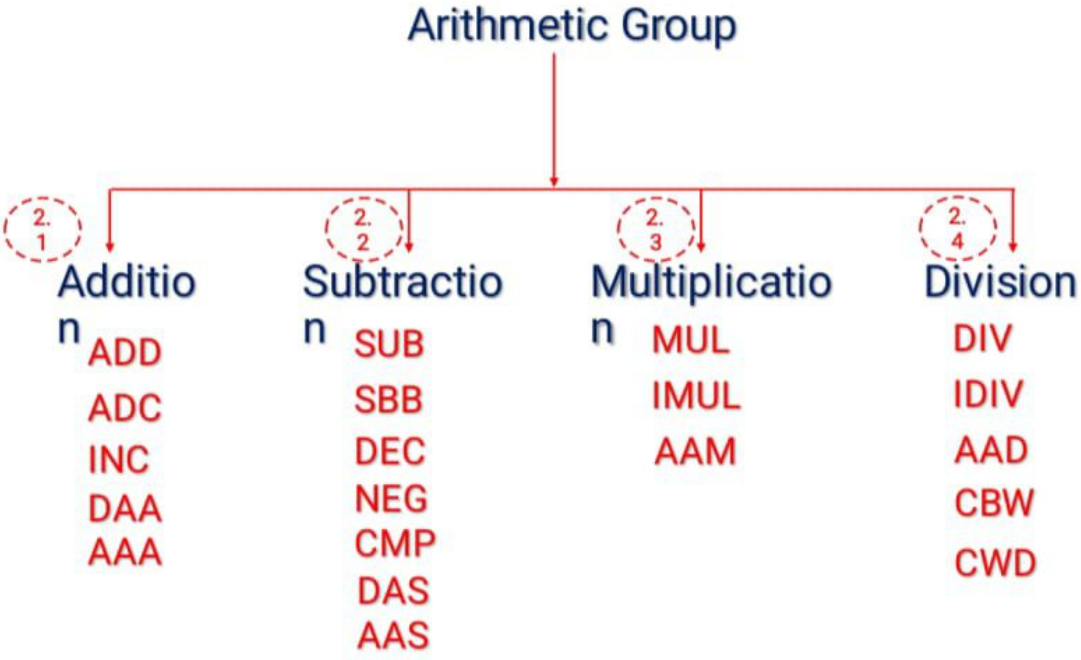
Two 16 bit number addition

```
ADD     1234
      + 1234
      ------
        2468
```

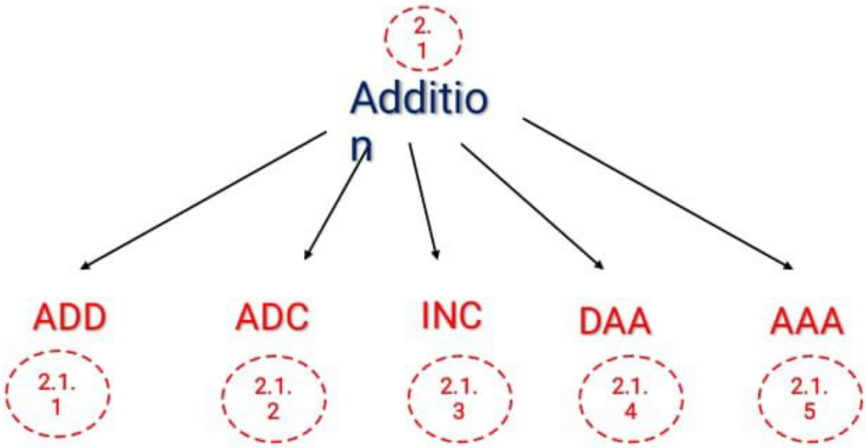
Two 32 bit number addition

```
ADD + CY   1234   8123
ADC      + 1234   8123  ADD
              1
          -------------
           2469   6246
```

2. Arithmetic group



2.1 Addition Group



2.1.4 DAA (Decimal Adjust After Addition)

This instruction is used to changed the contents of accumulator from a binary value to its equivalent BCD number.

Mnemonic: DAA

Addressing mode : Implied

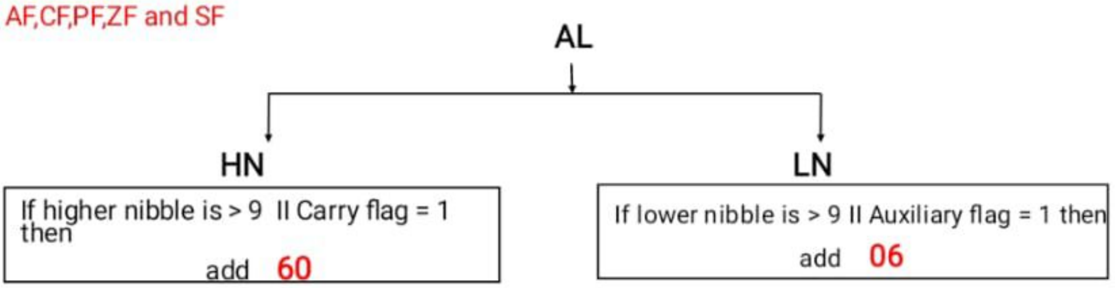
Flags : AF,CF,PF,ZF and SF

0 0 0 0 0 1 1 0

higher nibble lower nibble

60/06

Working of DAA instruction :



If higher nibble is > 9 II Carry flag = 1 then add 60

If lower nibble is > 9 II Auxiliary flag = 1 then add 06

Why 6 is added?

Why 6 is added?

Up to 9 numbers BCD and HEX numbers are same

From 10 to 15 in HEX , 6 Characters are used and after these 6 characters 10 onwards numbers are started.

Decimal	BCD	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	0001 0000	A
11	0001 0001	B
12	0001 0010	C
13	0001 0011	D
14	0001 0100	E
15	0001 0101	F
16	0001 0110	10
17	0001 0111	11

So there is a 6 characters gap between BCD and hex and to overcome this gap 6 is added

Addition of hexadecimal numbers :

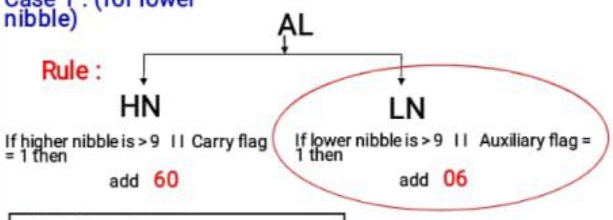
Case 1 :					
	30 h + 20 h	24 h + 24 h	25 h + 25 h	28 h + 27 h	28 h + 28 h
Expected answer →	50 h	48 h	50 h	55 h	56h
Actual answer →	50 h	48 h	4A h	4F h	50 h

Case 2 :			
	50 h + 50 h	64 h + 54h	84 h + 94h
Expected answer →	100 h	118 h	168 h
Actual answer →	A0 h	B8 h	108 h

Case 3 :	
	88 h + 88 h
Expected answer →	176 h
Actual answer →	110 h

There is a gap of 6 numbers

Case 1 : (for lower nibble)



AL

Rule :

HN

If higher nibble is > 9 | | Carry flag = 1 then

add 60

LN

If lower nibble is > 9 | | Auxiliary flag = 1 then

add 06

a) Both conditions not true

30 h
+ 20 h
50 h

LN is not > 9 , Auxiliary flag = 0

Therefore , output same

b) one conditions true i.e. LN > 9

25 h
+ 25 h
4A h

LN > 9 , Auxiliary flag = 0

Therefore add 06 into LN

0010 0101
+ 0010 0101
0100 1010 4A h
+ 0000 0110 06 h
0101 0000 50 h

c) one conditions are true i.e. Ln>! 9 and AF = 1

28 h
+ 28 h
50 h

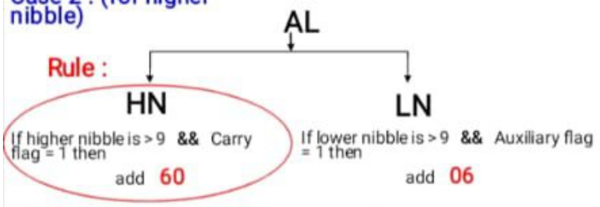
LN > 9 & Auxiliary flag = 1

Therefore add 06 into LN

0010 1000
+ 0010 1000
1
0101 0000 50 h
+ 0000 0110 06 h
0101 0110 56 h

Back

Case 2 : (for higher nibble)



AL

Rule :

HN

If higher nibble is > 9 && Carry flag = 1 then

add 60

LN

If lower nibble is > 9 && Auxiliary flag = 1 then

add 06

a) Both conditions not true

30 h
+ 20 h
50 h

HN is not > 9 , Carry flag = 0

Therefore , output same

b) one conditions true i.e. LN > 9

50 h
+ 50 h
A0 h

HN > 9 , Carry flag = 0

Therefore add 06 into LN

0101 0000
+ 0101 0000
1010 0000 A0 h
+ 0110 0000 60 h
1 0000 0000 100 h

c) both conditions are true i.e. HN> 9 and CF = 1

84 h
+ 94 h
108 h

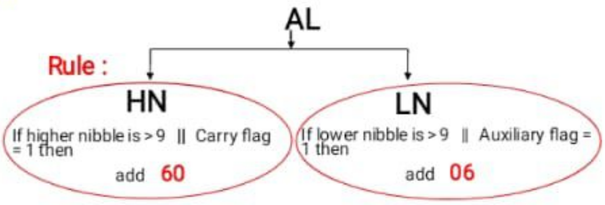
HN >9 & Carry flag = 1

Therefore add 06 into LN

1000 0100
+ 1001 0100
1 0001 1000 108 h
+ 0110 0000 60 h
Prev carry 1 0111 1000 178 h

Back

Case 3 : (for both nibble)



AL

Rule :

HN

If higher nibble is > 9 || Carry flag = 1 then

add 60

LN

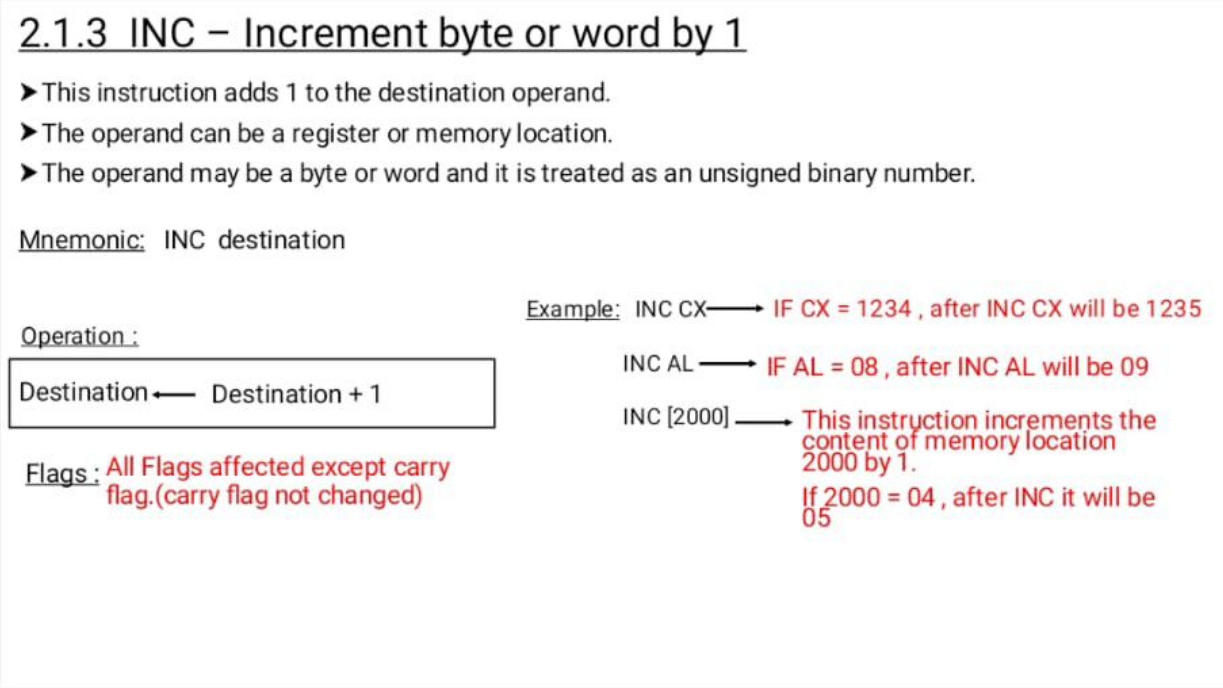
If lower nibble is > 9 || Auxiliary flag = 1 then

add 06

both conditions are true i.e. HN/LN > 9 and CF/AF = 1

88 h
+ 88 h
108 h

1000 1000
+ 1000 1000
1
1 0001 0000 100 h
+ 0110 0110 66 h
Prev carry 1 0111 0110 176 h



2.1.3 INC – Increment byte or word by 1

- This instruction adds 1 to the destination operand.
- The operand can be a register or memory location.
- The operand may be a byte or word and it is treated as an unsigned binary number.

Mnemonic: INC destination

Operation :

Destination ← Destination + 1

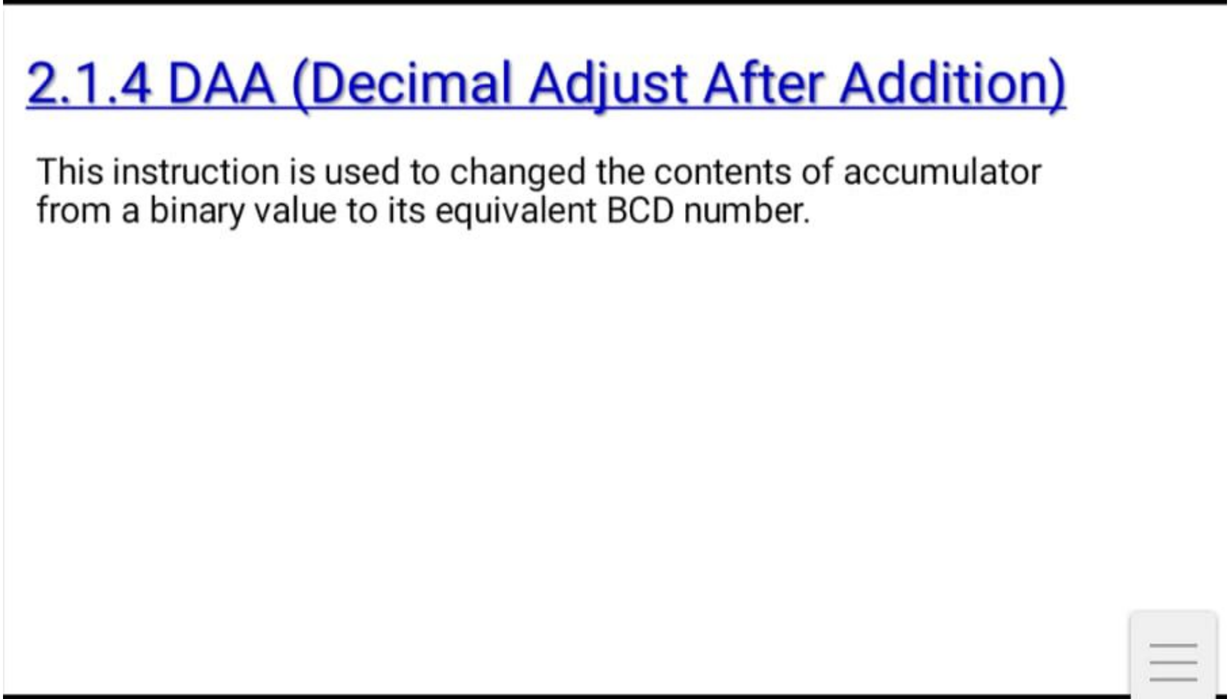
Flags : All Flags affected except carry flag.(carry flag not changed)

Example: INC CX → IF CX = 1234 , after INC CX will be 1235

INC AL → IF AL = 08 , after INC AL will be 09

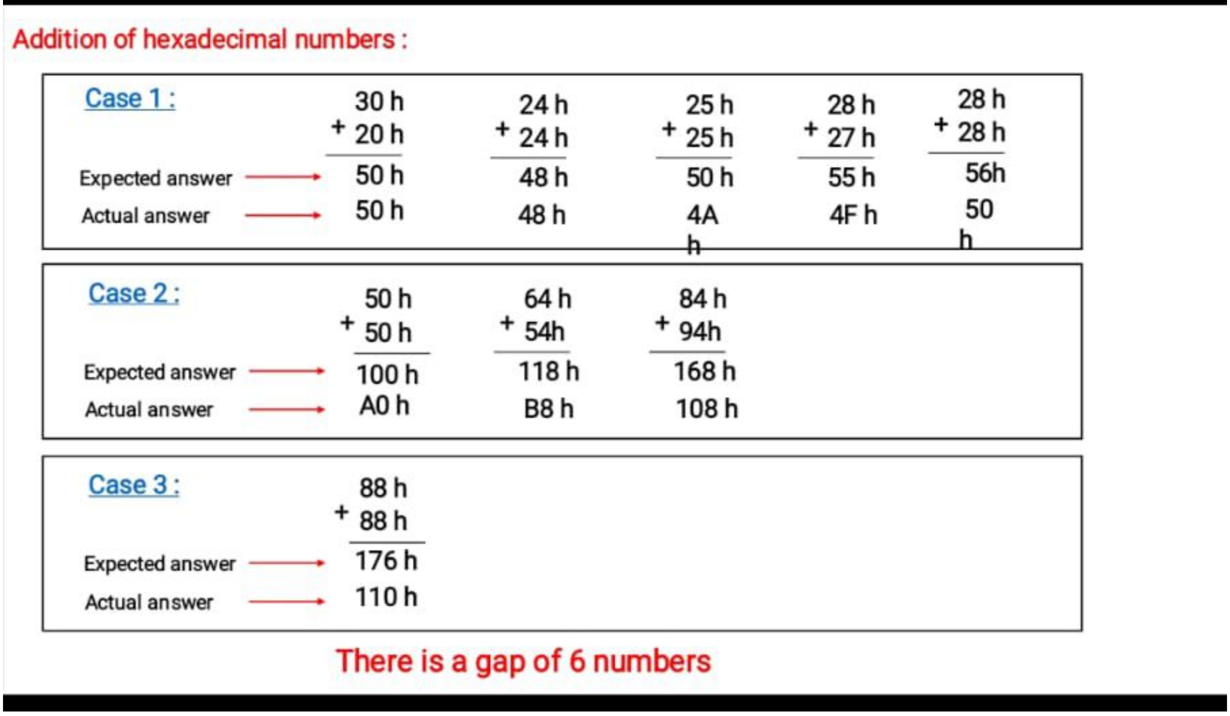
INC [2000] → This instruction increments the content of memory location 2000 by 1.

If 2000 = 04 , after INC it will be 05



2.1.4 DAA (Decimal Adjust After Addition)

This instruction is used to changed the contents of accumulator from a binary value to its equivalent BCD number.



Addition of hexadecimal numbers :

Case 1 :					
	30 h + 20 h	24 h + 24 h	25 h + 25 h	28 h + 27 h	28 h + 28 h
Expected answer →	50 h	48 h	50 h	55 h	56h
Actual answer →	50 h	48 h	4A h	4F h	50 h

Case 2 :			
	50 h + 50 h	64 h + 54h	84 h + 94h
Expected answer →	100 h	118 h	168 h
Actual answer →	A0 h	B8 h	108 h

Case 3 :	
	88 h + 88 h
Expected answer →	176 h
Actual answer →	110 h

There is a gap of 6 numbers