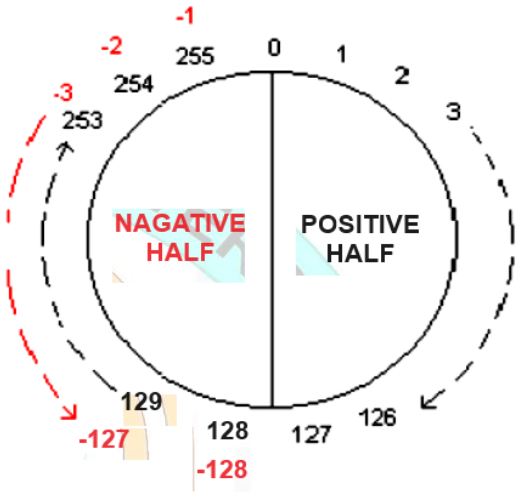
5.1 Signed and Unsigned Numbers:

An 8-bit number system can be used to create 256 combinations (from $0_D=0_H$ to $255_D=FF_H$), and the first 128 combinations ($0_D=0_H$ to $127_D=7F_H$) represent positive numbers and next 128 combinations ($128_D=80_H$ to $255_D=FF_H$) represent negative numbers.

Unsigned Number	Binary	Hexa.	Signed Number
0	0000 0000	00	0
1	0000 0001	01	+1
2	0000 0010	02	+2
⋮	⋮	⋮	⋮
127	0111 1111	7F	+127
128	1000 0000	80	-128
129	1000 0001	81	-127
⋮	⋮	⋮	⋮
254	1111 1110	FE	-2
255	1111 1111	FF	-1



In Decimal in order to get –2, we subtract 2 from the number of combinations (256), which gives, 256 – 2 = 254 (In Hexadecimal 100_H -2_H = FE_H). In Binary all the Signed Numbers have a '1' in the Most Significant Bit (MSB) position which represents a negative number and a '0' in the Most Significant Bit (MSB) position which represents a positive number.

Also, in Binary, the 2's Complement of a number is the negative equivalent of the positive number.

Equation	Binary	Hex	Signed
2 =	0000 0010	02	+2
1's Complement =	1111 1101	FD	
Add '1'	+0000 0001	+01	
2's Complement =	1111 1110	FE	-2

So, as above, +2 = 0000 0010 and the 2's Complement is 1111 1110 which represents - 2.

A 16-bit number system can be used to create 65536 combinations (from $0_D=0_H$ to $65535_D=FFFF_H$), and the first 32768 combinations ($0_D=0_H$ to $32767_D=7FFF_H$) represent positive numbers and next 32768 combinations ($32768_D=8000_H$ to $65535_D=FFFF_H$) represent negative numbers.

So, as above, +2 = 0000 0000 0000 0010 (0002_H) and the 1's Complement is 1111 1111 1111 1101 ($FFFD_H$) add 1, then the 2's Complement 1111 1111 1111 1110 ($FFFE_H$) which represents – 2.

In a 16-bit number system the Signed Numbers have a '1' in the Most Significant Bit (MSB) position 1xxxxxxxxxxxxxxx which represents a negative number. A '0' in the Most Significant Bit (MSB) position 0xxxxxxxxxxxxxxx which represents a positive number.

5.2 Arithmetic Instructions:

These instructions are those which are useful to perform Arithmetic calculations, such as addition, subtraction, multiplication division, comparison, negation, increment, and decrement. They are again classified into four groups. They are:

Addition Instructions	Subtraction Instructions	Multiplication Instructions	Division Instructions
ADD ADC INC	SUB SBB DEC NEG CMP	MUL IMUL	DIV IDIV CBW CWD

The state that results from the execution of an arithmetic instruction is recorded in the flags register. The flags that are affected by the arithmetic instructions are C, A, S, Z, P, O.

5.2.1 Addition Instructions:

Addition (ADD) appears in many forms in the microprocessor for 8-, and 16-bit binary addition. A second form of addition, called add-with-carry, is introduced with the ADC instruction. Finally, the increment instruction (INC) is a special type of addition that adds 1 to a number. The only types of addition not allowed are memory-to-memory and segment register. The segment registers can only be moved, pushed, or popped.

The general forms of these instructions are shown below:

Mnemonic	Meaning	Format	Operation	Flags Effected
ADD	Addition	ADD D, S	(S)+(D) → (D) carry → (CF)	O, S, Z, A, P, C
ADC	Add with carry	ADC D, S	(S)+(D) +(CF)→ (D) carry → (CF)	O, S, Z, A, P, C
INC	Increment by 1	INC D	(D)+1 → D	O, S, Z, A, P

Immediate Addition: Immediate addition is employed whenever constant or known data are added. An 8-bit immediate addition appears for example when; DL is first loaded with 12H by using an immediate move instruction. Next, 33H is added to the 12H in DL by an immediate addition instruction. After the addition, the sum (45H) moves into register DL and the flags change, as follows:

```
MOV DL,12H
ADD DL,33H
```

Z = 0 (result not zero)
C = 0 (no carry)
A = 0 (no half-carry)
S = 0 (result positive)
P = 0 (odd parity)
O = 0 (no overflow)

Register Addition: Example below shows a simple sequence of instructions that uses register addition to add the contents of several registers. In this example, the contents of AX, BX, CX, and DX are added to form a 16-bit result stored in the AX register.

Example:

```
MOV AX,1200H
MOV BX,1400H
MOV CX,2100H
MOV DX,2200H
ADD AX,BX
ADD AX,CX
ADD AX,DX
```

The Final Result will be AX=6900H and P=1 while other flags remains 0.

Whenever arithmetic and logic instructions execute, the contents of the flag register change. Note that the contents of the interrupt, trap, and other flags do not change due to arithmetic and logic instructions. Only the flags located in the rightmost 8 bits of the flag register and the overflow flag change. These rightmost flags denote the result of the arithmetic or a logic operation. Any ADD instruction modifies the contents of the sign, zero, carry, auxiliary carry, parity, and overflow flags.

Memory-to-Register Addition: Suppose that an application requires memory data to be added to the AL register. Example below shows an example that adds two consecutive bytes of data, stored at the data segment offset locations with offset $01A1_H$ and $01A2_H$, to the AL register.

Example:

```
MOV DI, 01A1H          ;Store the memory address in DI
MOV AL, 0
ADD AL, [DI]           ;Add the memory contents of address DI to content of AL
ADD AL, [DI+1]         ;Add the memory contents of address DI+1 to content of AL
```

The first instruction loads the destination index register (DI) with offset address NUMB. The

DI register, used in this example, addresses data in the data segment beginning at memory location $01A1_H$. After clearing the sum to zero, the ADD AL,[DI] instruction adds the contents of memory location $01A1_H$ to AL. Finally, the ADD AL,[DI+1] instruction adds the contents of memory location $01A1_H$ plus 1 byte to the AL register. After both ADD instructions execute, the result appears in the AL register as the sum of the contents of $01A1_H$ plus the contents of $01A2_H$.

Example:

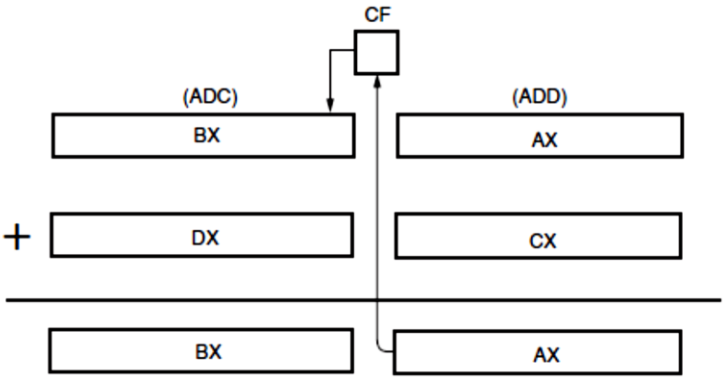
```
ADD AL,74H             ;Add immediate number 74H to content of AL
ADC CL,BL              ;Add contents of BL plus carry status to contents of CL Results in CL
ADD DX, [SI]           ;Add word from memory at offset [SI] in DS to contents of DX
```

Array Addition: Memory arrays are sequential lists of data. Suppose that an array of data (ARRAY) contains 10 bytes, numbered from element 0 through element 9. Example below shows how to add the contents of array elements 3, 5, and 7 together.

Example:

```
MOV AL,0               ;clear sum
MOV SI,3               ;address element 3
ADD AL,ARRAY[SI]       ;add element 3
ADD AL,ARRAY[SI+2] ;add element 5
ADD AL,ARRAY[SI+4] ;add element 7
ARRAY DB  1,2,3,4,5,6,7,8,9,10
```

Addition-with-Carry: An addition-with-carry instruction (ADC) adds the bit in the carry flag (C) to the operand data, ADC affects the flags after the addition.



Example below shows how the contents of registers AX and CX add to form the least significant 16 bits of the sum. This addition may or may not generate a carry. A carry appears in the carry flag if the sum is greater than FFFFH. Because it is impossible to predict a carry, the most significant 16 bits of this addition are added with the carry flag using the ADC instruction. The ADC instruction adds the 1 or the 0 in the carry flag to the most significant 16 bits of the result. This program adds BX–AX to DX–CX, with the sum appearing in BX–AX.

Example:

```
ADD AX,CX
ADC BX,DX
```

Addition of Un Signed numbers:

```
ADD CL, BL
Assume that CL = 01110011 =115 decimal ; BL = 01001111 = 79 decimal Result in CL =
11000010 = 194 decimal
```

Example:

```
MOV BX, 1200H          ; Store the memory address in BX
MOV CX, 3              ; Element 3
MOV AX,[BX+2*CX]       ; Get Element 3
MOV CX, 5              ; Element 5
ADD AX,[BX+2*CX]       ; ADD Element 5
MOV CX, 7              ; Element 7
ADD AX,[BX+2*CX]       ; ADD Element 7
```

Addition of Signed numbers

```
ADD CL, BL
Assume that CL = 01110011 = + 115 decimal ; BL = 01001111 = +79 decimal
Result in CL = 11000010 = - 62 decimal
                ; Incorrect because result is too large to fit in 7 bits.
```

INC Instruction - Increment - INC destination

INC instruction adds one to the operand (register or memory location except segment register) and sets the flag according to the result. INC instruction is treated as an unsigned binary number.

Example:

```
Assume AX = 7FFFh
INC AX              ;After this instruction AX = 8000h
```

Example:

```
MOV DI,01A1H        ;memory address with offset 01A1H
MOV AL,0            ;clear sum
ADD AL,[DI]         ;add the content of memory with offset 01A1H to register AL
INC DI              ;increment DI (memory address)
ADD AL,[DI]         ;add the content of memory with offset 01A2H to register AL
```

5.2.2 Subtraction Instructions:

Subtraction subgroup of instruction set is similar to the addition subgroup. For subtraction the carry flag CF acts as borrow flag. If borrow occur after subtraction then CF=1. If NO borrow occur after subtraction then CF=0. Subtraction subgroup content instruction shown in table below:

Mnemonic	Meaning	Format	Operation	Flags Effected
SUB	Subtraction	SUB D,S	(S)-(D) → (D) borrow → (CF)	O, S, Z, A, P, C
SBB	Subtract with borrow	SBB D,S	(S)-(D) -(CF)→ (D) borrow → (CF)	O, S, Z, A, P, C
DEC	Decrement by 1	DEC D	(D)-1 → D	O, S, Z, A, P
NEG	Negative	NEG	0 – (D)→ (D) 1 → (CF)	O, S, Z, A, P, C
CMP	Compare	CMP D,S	(S) - (D)	O, S, Z, A, P, C

Immediate Subtraction: The microprocessor also allows immediate operands for the subtraction of constant data. An Example presents a short sequence of instructions that subtract 44H from 22H. Here, we first load the 22H into CH using an immediate move instruction. Next, the SUB instruction, using immediate data 44H, subtracts 44H from the 22H. After the subtraction, the difference (0DEH) moves into the CH register. The flags change as follows for this subtraction:

(there is no overflow) → -128>overflow>+127

Example:

```
MOV CH,22H
SUB CH,44H
```

O = 0 (no overflow)
P = 1 (even parity)
S = 1 (result negative)
A = 1 (half-borrow)
C = 1 (borrow)
Z = 0 (result not zero)

22H = 0010 0010 B = +34D
44H = 0100 0100 B = +68D
22H = 0010 0010 B
BCH = 1011 1100 B
DEH = 1101 1110 B = -34D

Both carry flags (C and A) hold borrows after a subtraction instead of carries, as after an addition. Notice in this example that there is no overflow. This example subtracted 44H (+68) from 22H (+34), resulting in a 0DEH (–34). Because the correct 8-bit signed result is –34, there is no overflow in this example. An 8-bit overflow occurs only if the signed result is greater than +127 or less than –128.

Example:

```
SUB CX, BX        ; BX– CX; Result in CX
SBB CH, AL        ; AL – CH – CF ;Result in CH
SBB 3427H , AX    ;Subtract immediate number 3427H from AX
```

Register Subtraction: Example below shows a sequence of instructions that perform register subtraction. This example subtracts the 16-bit contents of registers CX and DX from the contents of register BX. After each subtraction, the microprocessor modifies the contents of the flag register. The flags change for most arithmetic and logic operations.

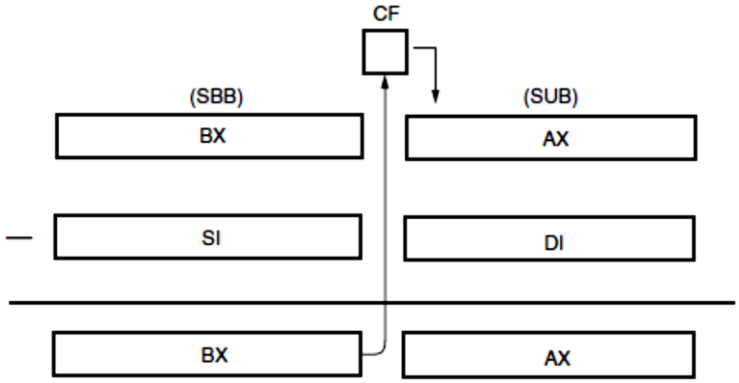
Example:

```
MOV BX,4400H
MOV CX,2100H
MOV DX,2200H
SUB BX,CX
SUB BX,DX
```

Subtraction-with-Borrow (SBB). A SBB instruction functions as a regular subtraction, except that the carry flag (C), which holds the borrow, also subtracts from the difference. The most common use for this instruction is for subtractions that are wider than 16 bits in the 8086–microprocessors. Figure below shows how the borrow propagates through the carry flag (C) for this task. Example 5–11 shows how this subtraction is performed by a program. The example uses the SUB instruction to subtract DI from AX, then uses SBB to subtract with-borrow SI from BX.

Example:

```
SUB AX,DI
SBB BX,SI
```



Subtracting unsigned number

Assume that CL = 10011100 = 156 decimal ;BH = 00110111 = 55 decimal
MOV CL, 156
MOV BH, 55
SUB CL, BH
CL = 01100101 = 101 decimal ; CF, AF, SF, ZF = 0, OF, PF = 1

Subtracting signed number

Assume that CL = 00101110 = + 46 decimal ;BH = 01001010= + 74 decimal
MOV CL, 46
MOV BH, 74
SUB CL, BH
CL = 11100100 = - 28 decimal ; CF = 1, AF, ZF =0,SF = 1 result negative

DEC Instruction: Decrement destination register or memory DEC destination.

DEC instruction subtracts one from the operand (register or the contents of a memory location) and sets the flag according to the result. DEC instruction is treated as an unsigned binary number.

Example:

```
MOV AX, 8000H        ; AX =8000H
DEC AX               ; After this instruction AX = 7FFFH
DEC BL               ; Subtract 1 from the contents of BL register
```

NEG Instruction – From 2's complement – NEG destination

NEG performs the two's complement subtraction of the operand from zero and sets the flags according to the result.

```
MOV AX , 2CBh
NEG AX                ;After executing NEG result AX =FD35h.
                        02CBH=0000 0010 1100 1011
                        FD35H=1111 1101 0011 0101
```

CMP Instruction - Compare byte or word -CMP destination, source.

The CMP instruction compares the destination and source i.e., it subtracts the source from destination, it changes only the flag bits. The result is not stored anywhere. It neglects the results, but sets the flags accordingly. A comparison is useful for checking the entire contents of a register or a memory location against another value. This instruction is usually used before a conditional jump instruction, which tests the condition of the flag bits.

```
CMP CL,BL          ;compare register CL with register BL
CMP AX,SP          ;compare register AX with register SP
CMP AX,2000H       ;compare register AX with 2000H
CMP [DI],CH        ;CH compared to the byte contents of the data segment memory location
                    addressed by DI
CMP CL,[BP]        ;The byte contents of the stack segment memory location addressed by BP
                    compared to CL
CMP AL,[DI+SI]     ;The byte contents of the data segment memory location addressed by DI
                    plus SI compared to AL
```

Example:

```
MOV AL, 5
MOV BL, 5
CMP AL, BL          ; AL = 5, ZF = 1 (so equal!)
```

Example:

```
MOV AL, 5
CMP AL, 10          ; AL = 5, ZF = 0 (not equal!)
```

5.2.3 Multiplication and Division Instructions:

The 8086 has instructions for multiplication and division of binary, BCD numbers, and signed or unsigned integers. Multiplication and division are performed on bytes or on words. The product after a multiplication is always a double-width product. If two 8-bit numbers are multiplied, they generate a 16-bit product; if two 16-bit numbers are multiplied, they generate a 32-bit product As shown below the form of these instructions.

Mnemonic	Meaning	Format	Operation
MUL	Multiply (Unsigned)	MUL S	(AL)*(S8) → (AX) ; (AX)*(S16) → (DX)(AX)
DIV	Division (Unsigned)	DIV S	Q((AX)/(S8)) → (AL) ; R((AX)/(S8)) → (AH) Q((DX,AX)/(S16)) → (AX) ; R((DX,AX)/(S16)) → (DX)
IMUL	Integer Multiply (Signed)	IMUL S	(AL)*(S8) → (AX) ; (AX)*(S16) → (DX)(AX)
IDIV	Integer Divide (Signed)	IDIV S	Q((AX)/(S8)) → (AL) ; R((AX)/(S8)) → (AH) Q((DX,AX)/(S16)) → (AX) ; R((DX,AX)/(S16)) → (DX)
CBW	Convert byte to word	CBW	(MSB of AL) → (All bits of AH)
CWD	Convert word to double word	CWD	(MSB of AX) → (All bits of DX)

Some flag bits (overflow and carry) change when the multiply instruction executes and produce predictable outcomes. The other flags also change, but their results are unpredictable and therefore are unused. In an 8-bit multiplication, if the most significant 8 bits of the result are zero, both C and O flag bits equal zero. These flag bits show that the result is 8 bits wide (C = 0) or 16 bits wide (C = 1). In a 16-bit multiplication, the DX register always contains the most significant 16 bits of the product, and AX contains the least significant 16 bits. if the most significant 16-bits part of the product is 0, both C and O clear to zero.

An 8-bit division uses the AX register to store the dividend that is divided by the contents of any 8-bit register or memory location. The quotient moves into AL after the division with AH containing a whole number remainder. For a signed division, the quotient is positive or negative; the remainder always assumes the sign of the dividend and is always an integer.

A 16-bit division is similar to 8-bit division, except that instead of dividing into AX, the 16-bit number is divided into DX–AX. The quotient appears in AX and the remainder appears in DX after a 16-bit division.

Example: what is the result of executing the following instruction?

(a) MUL CL
(b) IMUL CL

Assume that AL contains FFH (the 2'complement of the number 1), CL contain FEH (the 2'complement of the number 2).

Solution:

```
MOV AL, 0FFH
MOV CL, 0FEH
MUL CL
```

AL = FF, CL = FE (Before) → AX = FD02, CL = FE (After MUL)

FF(255)*FE(254) =FD02H(64770) CF, OF=1

```
MOV AL, 0FFH
MOV CL, 0FEH
IMUL CL
```

AL = FF, CL = FE (Before) → AX = 0002, CL = FE (After IMUL)

01(2's FF)*02(2's FE) =02H CF, OF=0

Example:

```
MOV AL, 69
MOV BL, 14
IMUL BL          ; 69 * 14
                 ; AL = 01000101 = 69 decimal, BL = 00001110 = 14 decimal
                 ;AX = 03C6H = + 966 decimal ,MSB = 0 because positive result
MOV AL, 228
MOV BL, 59
IMUL BL          ; - 28 * 59
                 ; AL = 11100100 = - 28 decimal, BL = 00111011 = 59 decimal
                 ;AX = F98Ch = - 1652 decimal, MSB = 1 because negative result
```

Example:

Assume that each instruction starts from these values: AL = 85H, BL = 35H, AH = 0H

MOV AL, 85H
MOV BL, 35H

(a) MUL BL =AL . BL = 85H * 35H = 1B89H →AX = 1B89H

(b) IMUL BL =AL . BL= 2'SAL * BL= 2'S(85H) * 35H =7BH * 35H =1977H→2's comp→E689H →AX.

(c) DIV BL =$\frac{AX}{BL} = \frac{0085H}{35H} =$

AH (remainder)	AL (quotient)
1BH	02H

$\frac{85H(133D)}{35H(53D)} = 2H$
Remainder 1BH(27D)

(d) IDIV BL =$\frac{AX}{BL} = \frac{0085H}{35H} =$

AH (remainder)	AL (quotient)
1BH	02H

Example:

Assume that each instruction starts from these values: AL = F3H, BL = 91H, AH = 00H

MOV AL, 0F3H
MOV BL, 91H

(a) MUL BL=AL . BL =F3H * 91H = 89A3H →AX = 89A3H

(b)IMUL BL =AL . BL= 2'SAL *2'SBL=2'S(F3H) *2'S(91H) =0DH * 6FH = 05A3H →AX.

(c) DIV BL =$\frac{AX}{BL} = \frac{00F3H}{91H} =$

AH (remainder)	AL (quotient)
62H	01H

$\frac{F3H(243D)}{91H(145D)} = 1H$
Remainder 62H(98D)

(d) IDIV BL =$\frac{AX}{BL} = \frac{00F3H}{2S(91H)} = \frac{00F3H}{6FH}$

AH (remainder)	AL (quotient)
15H	02H

$\frac{F3H(243D)}{6FH(111D)} = 2H$
Remainder 15H(21D)

But $\frac{\text{Positive}}{\text{Negative}} = \text{Negative}$, so

AH (remainder)	AL (quotient)
15H	2'S(02H)

⇒

AH (remainder)	AL (quotient)
15H	FEH

Example:

Assume that each instruction starts from these values: AX= F000H, BX= 9015H, DX= 0000H

MOV AX, 0F000H
MOV BX, 9015H

(a) MUL BX=AX . BX =F000H * 9015H =

DX	AX
8713	B000

(b)IMUL BX = 2'SAX *2'SBX=2'S(F000H) *2'S(9015H) =1000H * 6FEBH =

DX	AX
06FE	B000

(c) DIV BL =$\frac{AX}{BL} = \frac{F000H}{15H} = 0B6D \Rightarrow$ more than FFH ⇒ Divide Error ⇒ Correct DIV BX =$\frac{AX}{BX}$

(d) IDIV BL =$\frac{AX}{BL} = \frac{2'S(F000H)}{15H} = \frac{1000H}{15H} = C3H \Rightarrow$ more than 7FH ⇒ Divide Error ⇒ Correct IDIV BX =$\frac{AX}{BX}$

Example:

Assume that each instruction starts from these values: AX = 1250H, BL = 90H

MOV AX, 1250H
MOV BL, 90H

(a) DIV BL $= \frac{AX}{BL} = \frac{1250H}{90H} =$

AH (remainder)	AL (quotient)
50H	20H

$\frac{1250H(4688D)}{90H(144D)} = 20H$

Remainder 50H(80D)

(b) IDIV BL $= \frac{AX}{BL} = \frac{1250H}{90H} = \frac{Positive}{Negative} = \frac{Positive}{2'SNegative} = \frac{1250H}{2'S(90H)} = \frac{1250H}{70H} =$

AH (remainder)	AL (quotient)
60H	29H

$\frac{1250H(4688D)}{70H(112D)} = 29H$

Remainder 60H(96D)

But $\frac{Positive}{Negative} = Negative$, so

AH (remainder)	AL (quotient)		AH (remainder)	AL (quotient)
60H	2'S(29H)	⇒	60H	D7H

5.3 CBW Instruction-Convert signed Byte to signed word:

CBW converts the signed value in the AL register into an equivalent 16 bit signed value in the AX register by duplicating the sign bit to the left. This instruction copies the sign of a byte in AL to all the bits in AH. AH is then said to be the sign extension of AL.

Example:

```
MOV AX, 155  ; AX = 00000000 10011011 = - 155 decimal
CBW          ; Convert signed byte in AL to signed word in AX.
             ; Result in AX = 11111111 10011011 (FF9BH) and = - 155 decimal
```

5.4 CWD Instruction-Convert Signed Word to-Signed Double word:

CWD converts the 16 bit signed value in the AX register into an equivalent 32 bit signed value in DX: AX register pair by duplicating the sign bit to the left. The CWD instruction sets all the bits in the DX register to the same sign bit of the AX register. The effect is to create a 32- bit signed result that has same integer value as the original 16 bit operand.

Example:

Assume AX contains C435h. If the CWD instruction is executed, DX will contain FFFFh since bit 15 (MSB) of AX was 1. Both the original value of AX (C435h) and resulting value of DX : AX (FFFFC435h) represents the same signed number.

Example:

```
MOV AX, 0F0C7H    ; AX = 11110000 11000111 = - 3897 D
MOV DX, 0         ; DX = 00000000 00000000
CWD               ;Convert signed word in AX to signed double word in DX:AX (FFFF:F0C7)
                  ;Result DX = 11111111 11111111 and AX = 11110000 11000111 = -3897 D.
```

Example:

```
MOV AX, 0C435H    ; AX = 11000100 00110101 = - 15307 D
MOV DX, 0         ; DX = 00000000 00000000
CWD               ;Convert signed word in AX to signed double word in DX:AX (FFFF:C435)
                  ;Result DX = 11111111 11111111 and AX = 11000100 00110101 = -15307 D.
```