Q. what is meant by addressing modes? Describe address modes with one example. [W-08, W-09, S-10, W-12, S-13, W-09, W-12].

Answer:

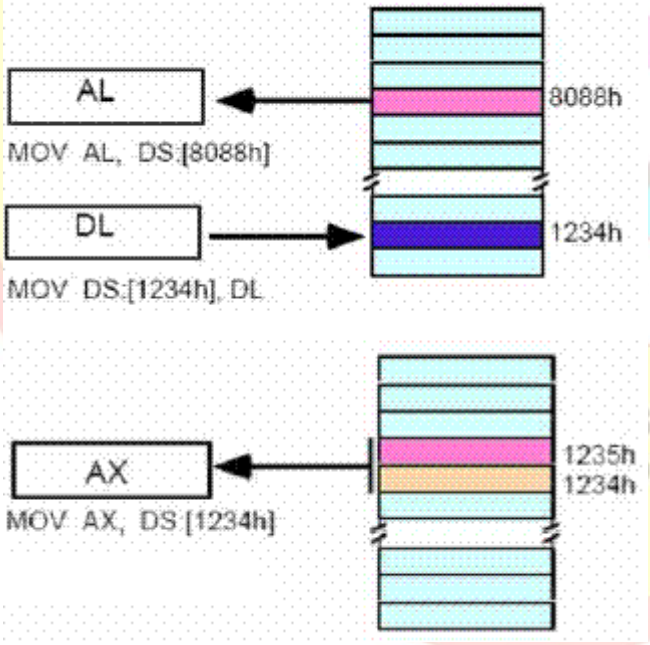
Definition: An instruction acts on any number of operands. The way an instruction accesses its operands is called its **Addressing modes**.

1. **Immediate addressing mode**
   In this addressing mode, the operand is stored as part of the instruction. The immediate operand, which is stored along with the instruction, resides in the code segment -- not in the data segment. This addressing mode is also faster to execute an instruction because the operand is read with the instruction from memory. Here are some examples:

Example: Immediate Operands

MOV AL, 20 ; move the constant 20 into register AL
ADD AX, 5 ; add constant 5 to register EAX
MOV DX, offset msg ; move the address of message to register DX

2. Direct Addressing Mode
   The instruction mov al,ds:[8088h] loads the AL register with a copy of the byte at memory location 8088h. Likewise, the instruction mov ds:[1234h],dl stores the value in the dl register to memory location 1234h.



3. Register Addressing Modes

   In this addressing mode, the operands may be:

- reg16: 16-bit general registers: AX, BX, CX, DX, SI, DI, SP or BP.
- reg8 : 8-bit general registers: AH, BH, CH, DH, AL, BL, CL, or DL.
- Sreg : segment registers: CS, DS, ES, or SS. There is an exception: CS cannot be a destination.

For register addressing modes, there is no need to compute the effective address. The operand is in a register and to get the operand there is no memory access involved.

Example: Register Operands

MOV AX, BX ; mov reg16, reg16
ADD AX, SI ; add reg16, reg16
MOV DS, AX ; mov Sreg, reg16

Some rules in register addressing modes:

1. You may not specify CS as the destination operand.

Example: mov CS, 02h –> wrong

2. Only one of the operands can be a segment register. You cannot move data from one segment register to another with a single mov instruction. To copy the value of cs to ds, you would have to use some sequence like:

mov ds,cs -> wrong
mov ax, cs
mov ds, ax -> the way we do it

You should never use the segment registers as data registers to hold arbitrary values. They should only contain segment addresses.

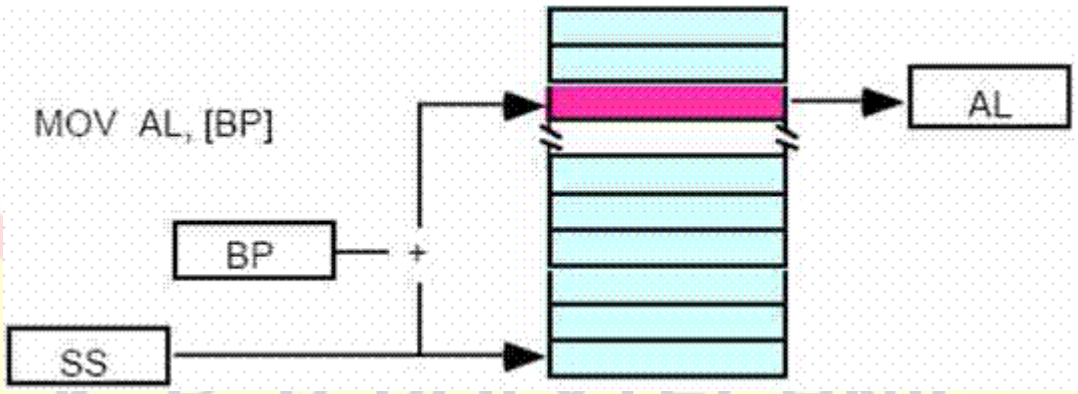
4. Register Indirect Addressing Mode

The 80x86 CPUs let you access memory indirectly through a register using the register indirect addressing modes. There are four forms of this addressing mode on the 8086, best demonstrated by the following instructions:

mov al, [bx]
mov al, [bp]
mov al, [si]
mov al, [di]

Code Example

MOV BX, 100H
MOV AL, [BX]



The [bx], [si], and [di] modes use the ds segment by default. The [bp] addressing mode uses the stack segment (ss) by default. You can use the segment override prefix symbols if you wish to access data in different segments. The following instructions demonstrate the use of these overrides:

5. Base Indexed Addressing Mode

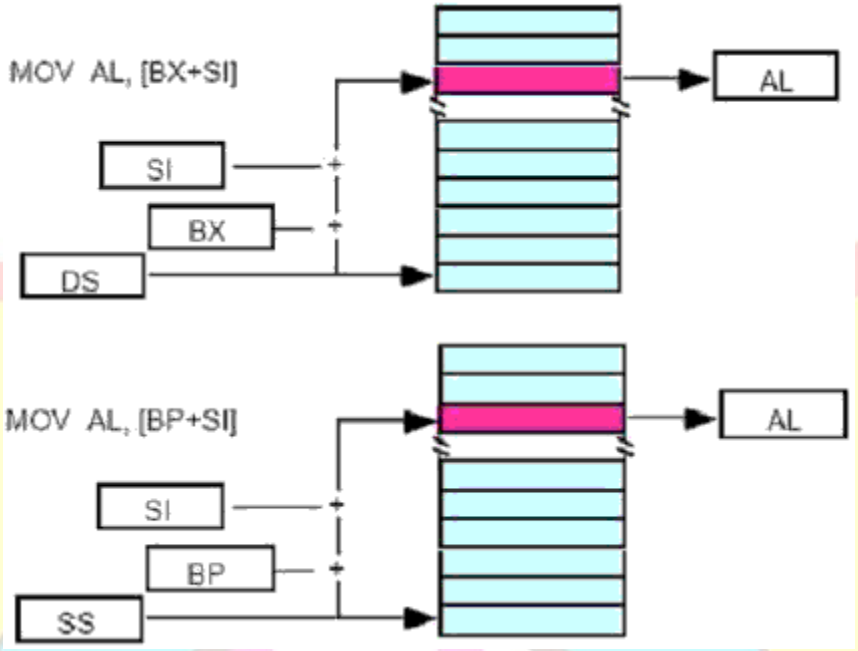
The based indexed addressing modes are simply combinations of the register indirect addressing modes. These addressing modes form the offset by adding together a base register (bx or bp) and an index register (si or di). The allowable forms for these addressing modes are:

```
mov al, [bx+si]
mov al, [bx+di]
mov al, [bp+si]
mov al, [bp+di]
```

Code Example

```
MOV BX, 100H
MOV SI, 200H
MOV AL, [BX + SI]
INC BX
INC SI
```

6.



Suppose that bx contains 1000h and si contains 880h. Then the instruction mov al,[bx][si] would load al from location DS:1880h. Likewise, if bp contains 1598h and di contains 1004, mov ax,[bp+di] will load the 16 bits in ax from locations SS:259C and SS:259D. The addressing modes that do not involve bp use the data segment by default. Those that have bp as an operand use the stack segment by default.

Q. Explain Following Instructions.

Answer: [Note- All the instructions are important. Concentrate on those instructions which is given in important questions]

1. **Data Transfer Instructions**

a. **MOV Instruction - MOV destination, source**

The MOV instruction copies a word or a byte of data from a specified source to a specified destination. MOV op1, op2

Example:

```
MOV CX,037AH          ; MOV 037AH into the CX.
MOV AX, BX            ;Copy the contents of register BX to AX
MOV DL,[BX]           ;Copy byte from memory at BX to DL , BX contains the offset of byte in DS.
```

b. **PUSH Instruction - PUSH source**

PUSH instruction decrements the stack pointer by 2 and copies a word from a specified source to the location in the stack segment where the stack pointer pointes.

Example:

```
PUSH BX                 ;Decrement SP by 2 and copy BX to stack
PUSH DS                 ;Decrement SP by 2 and copy DS to stack
PUSH TABLE[BX]      ;Decrement SP by 2 and copy word from memory in DS at EA =
TABLE + [BX] to stack .
```

c. **POP Instruction - POP destination**

POP instruction copies the word at the current top of the stack to the operand specified by op then increments the stack pointer to point to the next stack.

Example:

```
POP DX              ;Copy a word from top of the stack to DX and increments SP by 2.
POP DS              ; Copy a word from top of the stack to DS and increments SP by 2.
POP TABLE [BX]    ;Copy a word from top of stack to memory in DS with EA = TABLE +
[BX].
```

d. **XCHG Instruction - Exchange XCHG destination, source**

The Exchange instruction exchanges the contents of the register with the contents of another register (or) the contents of the register with the contents of the memory location. Direct memory to memory exchange is not supported.

Syntax: XCHG op1, op2 - The both operands must be the same size and one of the operand must always be a register.

Example:

```
XCHG AX, DX             ;Exchange word in AX with word in DX
XCHG BL, CH             ;Exchange byte in BL with byte in CH
XCHG AL, Money [BX]     ;Exchange byte in AL with byte in memory at EA.
```

e. **XLAT/XLATB Instruction - Translate a byte in AL**

XLAT exchanges the byte in AL register from the user table index to the table entry, addressed by BX. It transfers 16 bit information at a time. The no-operands form (XLATB) provides a "short form" of the XLAT instructions.

Example: MOV AL, [BX+AL]

SIMPLE INPUT AND OUTPUT PORT TRANSFER INSTRUCTION

f. IN Instruction - Copy data from a port IN accumulator, port

This IN instruction will copy data from a port to the AL or AX register. For the Fixed port IN instruction type the 8 – bit port address of a port is specified directly in the instruction.

Example:

```
IN AL,0C8H          ;Input a byte from port 0C8H to AL
IN AX, 34H          ;Input a word from port 34H to AX
A_TO_D EQU 4AH
IN AX, A_TO_D       ;Input a word from port 4AH to AX
```

For a variable port IN instruction, the port address is loaded in DX register before IN instruction. DX is 16 bit. Port address range from 0000H – FFFFH.

Example:

```
MOV DX, 0FF78H      ;Initialize DX point to port
IN AL, DX           ;Input a byte from a 8 bit port 0FF78H to AL
IN AX, DX           ;Input a word from 16 bit port to 0FF78H to AX.
```

g. OUT Instruction - Output a byte or word to a port – OUT port, accumulator AL or AX.

The OUT instruction copies a byte from AL or a word from AX or a double from the accumulator to I/O port specified by op. Two forms of OUT instruction are available : (a) Port number is specified by an immediate byte constant, (0 - 255).It is also called as fixed port form. (b) Port number is provided in the DX register (0 – 65535)

Example: (a)

```
OUT 3BH, AL         ;Copy the contents of the AL to port 3Bh
OUT 2CH,AX          ;Copy the contents of the AX to port 2Ch
```

Example: (b)

```
MOV DX, 0FFF8H      ;Load desired port address in DX
OUT DX, AL          ; Copy the contents of AL to FFF8h
OUT DX, AX          ;Copy content of AX to port FFF8H
```

SPECIAL ADDRESS TRANSFER INSTRUCTION

h. **LEA Instruction - Load Effective Address**

LEA Instruction - This instruction indicates the offset of the variable or memory location named as the source and put this offset in the indicated 16 – bit register.

Syntax – LEA register, source

Example:

```
LEA BX, PRICE          ;Load BX with offset of PRICE in DS
LEA BP, SS:STAK        ;Load BP with offset of STACK in SS
LEA CX, [BX][DI]       ;Load CX with EA=BX + DI
```

i. **LDS Instruction - Load register and Ds with words from memory**

LDS Instruction - This instruction loads a far pointer from the memory address specified by op2 into the DS segment register and the op1 to the register.

Syntax – LDS register, memory address of first word or LDS op1, op2

Example:

LDS BX, [4326] ; copy the contents of the memory at displacement 4326H in DS to BL, contents of the 4327H to BH. Copy contents of 4328H and 4329H in DS to DS register.

j. **LES Instruction - Load register and ES with words from memory**

This instruction loads a 32- bit pointer from the memory address specified to destination register and Extra Segment. The offset is placed in the destination register and the segment is placed in Extra Segment. Using this instruction, the loading of far pointers may be simplified.

Syntax – LES register, memory address of first word

- **LAHF Instruction - Load Register AH From Flags**

LAHF instruction copies the value of SF, ZF, AF, PF, and CF, into bits of 7, 6, 4, 2, 0 respectively of AH register. This LAHF instruction was provided to make conversion of assembly language programs written for 8080 and 8085 to 8086 easier.

- **SAHF instruction - Store AH Register into FLAGS**

SAHF instruction transfers the bits 0-7 of AH of SF, ZF, AF, PF, and CF, into the Flag register.

- **PUSHF Instruction - Push flag register on the stack**

This instruction decrements the SP by 2 and copies the word in flag register to the memory location pointed to by SP.

- **POPF Instruction - Pop word from top of stack to flag - register.**

This instruction copies a word from the two memory location at the top of the stack to flag register and increments the stack pointer by 2.

2. Arithmetic Instructions

- **ADD Instruction - ADD destination, source**

These instructions add a number from source to a number from some destination and put the result in the specified destination. The source and destination must be of same type , means they must be a byte location or a word location. If you want to add a byte to a word, you must copy the byte to a word location and fill the upper byte of the word with zeroes before adding.

- **ADC Instruction - Add with carry**

After performing the addition, the add with carry instruction ADC, adds the status of the carry flag into the result.

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, BX          ;Add contents of BX to contents of DX
ADD DX, [SI]        ;Add word from memory at offset [SI] in DS to contents of DX
                    ; Addition of Un Signed numbers
ADD CL, BL          ;CL = 01110011 =115 decimal + BL = 01001111 = 79 decimal Result in
CL = 11000010 = 194 decimal
                    ; Addition of Signed numbers
ADD CL, BL          ;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 and sets the flag according to the result. INC instruction is treated as an unsigned binary number.

Example:

```
                ; AX = 7FFFh
INC AX          ;After this instruction AX = 8000h
INC BL          ; Add 1 to the contents of BL register
INC CL          ; Add 1 to the contents of CX register.
```

- **AAA Instruction - ASCII Adjust after Addition**

AAA converts the result of the addition of two valid unpacked BCD digits to a valid 2-digit BCD number and takes the AL register as its implicit operand.

Two operands of the addition must have its lower 4 bits contain a number in the range from 0-9.The AAA instruction then adjust AL so that it contains a correct BCD digit. If the addition produce carry (AF=1), the AH register is incremented and the carry CF and auxiliary carry AF flags are set to 1. If the addition did not produce a decimal carry, CF and AF are cleared to 0 and AH is not altered. In both cases the higher 4 bits of AL are cleared to 0. AAA will adjust the result of the two ASCII characters that were in the range from 30h ("0") to 39h("9").This is because the lower 4 bits of those character fall in the range of 0-9.The result of addition is not a ASCII character but it is a BCD digit.

Example:

```
MOV AH,0        ;Clear AH for MSD
MOV AL,6        ;BCD 6 in AL
ADD AL,5        ;Add BCD 5 to digit in AL
AAA             ;AH=1, AL=1 representing BCD 11.
```

- **DAA Instruction - Decimal Adjust after Addition**

The contents after addition are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. S, Z, AC, P, CY flags are altered to reflect the results of the operation.

If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits.

If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits.

Example:

```
MOV AL, 0Fh      ; AL = 0Fh (15)
DAA              ; AL = 15h
RET
```

SUBTRACTION INSTRUCTIONS

SUB Instruction - Subtract two numbers

These instructions subtract the source number from destination number destination and put the result in the specified destination. The source and destination must be of same type , means they must be a byte location or a word location.

- **SBB Instruction - Subtract with borrow SBB destination, source**

SBB instruction subtracts source from destination, and then subtracts 1 from source if CF flag is set and result is stored destination and it is used to set the flag.

ie.,destination = destination -(source + CF)

Example:

```
SUB CX, BX                    ;CX – BX . Result in CX
SUBB CH, AL                   ; Subtract contents of AL and contents CF from contents of
CH .
;Result in CH
SUBB AX, 3427H                ;Subtract immediate number from AX
```

Example:

```
- Subtracting unsigned number
                              ; CL = 10011100 = 156 decimal BH = 00110111 = 55 decimal
SUB CL, BH                    ; CL = 01100101 = 101 decimal CF, AF, SF, ZF = 0, OF, PF =
1
- Subtracting signed number
                              ; CL = 00101110 = + 46 decimal BH = 01001010= + 74 decimal
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 and sets the flag according to the result. DEC instruction is treated as an unsigned binary number.

Example:

```
                              ; AX =8000h
DEC AX                        ;After this instruction AX = 7999h
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.

```
                    ;AX = 2CBh
NEG AX              ;after executing NEG result AX =FD35h.
```

Example:

```
NEG AL              ;Replace number in AL with its 2's complement

NEG BX              ;Replace word in BX with its 2's complement

NEG BYTE PTR[BX]    ; Replace byte at offset BX in DS with its 2's complement
```

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

The CMP instruction compares the destination and source ie.,it subtracts the source from destination. The result is not stored anywhere. It neglects the results, but sets the flags accordingly. This instruction is usually used before a conditional jump instruction.

Example:

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

- **AAS Instruction - ASCII Adjust for Subtraction**

AAS converts the result of the subtraction of two valid unpacked BCD digits to a single valid BCD number and takes the AL register as an implicit operand. The two operands of the subtraction must have its lower 4 bit contain number in the range from 0 to 9 .The AAS instruction then adjust AL so that it contain a correct BCD digit.

```
MOV AX,0901H        ;BCD 91
SUB AL, 9           ;Minus 9
AAS                 ; Give AX =0802 h (BCD 82)
```

Example:(a)

```
                    ;AL =0011 1001 =ASCII 9
                    ;BL=0011 0101 =ASCII 5
SUB AL, BL          ;(9 - 5) Result : ;AL = 00000100 = BCD 04,CF = 0
AAS                 ;Result : AL=00000100 =BCD 04 , CF = 0 NO Borrow required
```

Example:(b)

```
                    ;AL = 0011 0101 =ASCII 5
                    ;BL = 0011 1001 = ASCII 9
SUB AL, BL          ;( 5 - 9 ) Result : AL = 1111 1100 = – 4 in 2's complement CF = 1
AAS                 ;Results :AL = 0000 0100 =BCD 04, CF = 1 borrow needed
```

- **DAS Instruction - Decimal Adjust after Subtraction**

This instruction corrects the result (in AL) of subtraction of two packed BCD values. The flags which modify are AF, CF, PF, SF, ZF

if low nibble of AL > 9 or AF = 1 then:

- AL = AL – 6

- AF = 1

if AL > 9Fh or CF = 1 then:

- AL = AL - 60h

- CF = 1

Example:

```
MOV AL, 0FFh        ; AL = 0FFh (-1)
DAS                 ; AL = 99h, CF = 1
RET
```

MULTIPLICATION INSTRUCTIONS

- **MUL Instruction - Multiply unsigned bytes or words-MUL source**

MUL Instruction - This instruction multiplies an unsigned multiplication of the accumulator by the operand specified by op. The size of op may be a register or memory operand .

Syntax: MUL op

Example:

```
                    ;AL = 21h (33 decimal),BL = A1h(161 decimal )
MUL BL              ;AX =14C1h (5313 decimal) since AH≠0, CF and OF will set to
1.
MUL BH              ; AL times BH, result in AX
MUL CX              ;AX times CX, result high word in DX,low word in AX.
```

- **IMUL Instruction - Multiply signed number-IMUL source**

This instruction performs a signed multiplication.There are two types of syntax for this instruction. They are:

IMUL op ;In this form the accumulator is the multiplicand and op is the multiplier. op may be a register or a memory operand.
IMUL op1, op2 ;In this form op1 is always be a register operand and op2 may be a register or a memory operand.

Example:

IMUL BH ;Signed byte in AL times multiplied by signed byte in BH and result in AX .

Example:

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

- **AAM Instruction - ASCII adjust after Multiplication**

AAM Instruction - AAM converts the result of the multiplication of two valid unpacked BCD digits into a valid 2-digit unpacked BCD number and takes AX as an implicit operand. To give a valid result the digits that have been multiplied must be in the range of 0 – 9 and the result should have been placed in the AX register. Because both operands of multiply are required to be 9 or less, the result must be less than 81 and thus is completely contained in AL. AAM unpacks the result by dividing AX by 10, placing the quotient (MSD) in AH and the remainder (LSD) in AL.

Example:

```
MOV AL, 5
MOV BL, 7
MUL BL          ;Multiply AL by BL , result in AX
AAM             ;After AAM, AX =0305h (BCD 35)
```

DIVISION INSTRUCTIONS

- **DIV Instruction - Unsigned divide-Div source**

When a double word is divided by a word, the most significant word of the double word must be in DX and the least significant word of the double word must be in AX. After the division AX will contain the 16 –bit result (quotient) and DX will contain a 16 bit remainder. Again , if an attempt is made to divide by zero or quotient is too large to fit in AX (greater than FFFFH) the 8086 will do a type of 0 interrupt .

Example:

```
DIV CX                    ; (Quotient) AX= (DX:AX)/CX
                          ; (Reminder) DX=(DX:AX)%CX
```

For DIV the dividend must always be in AX or DX and AX, but the source of the divisor can be a register or a memory location specified by one of the 24 addressing modes. If you want to divide a byte by a byte, you must first put the dividend byte in AL and fill AH with all 0's . The SUB AH,AH instruction is a quick way to do. If you want to divide a word by a word, put the dividend word in AX and fill DX with all 0's. The SUB DX,DX instruction does this quickly.

Example:

```
                          ; AX = 37D7H = 14, 295 decimal and BH = 97H = 151 decimal
DIV BH                    ;AX / BH
                          ; AX = Quotient = 5EH = 94 decimal and AH = Remainder = 65H =
101 decimal.
```

- **IDIV Instruction - Divide by signed byte or word IDIV source**

This instruction is used to divide a signed word by a signed byte or to divide a signed double word by a signed word. If source is a byte value, AX is divided by register and the quotient is stored in AL and the remainder in AH. If source is a word value, DX:AX is divided by register, and the quotient is stored in AL and the remainder in DX.

Example:

```
IDIV BL                   ;Signed word in AX is divided by signed byte in BL
```

- **AAD Instruction - ASCII adjust before Division**

ADD converts unpacked BCD digits in the AH and AL register into a single binary number in the AX register in preparation for a division operation. Before executing AAD, place the Most significant BCD digit in the AH register and Last significant in the AL register. When AAD is

executed, the two BCD digits are combined into a single binary number by setting AL=(AH*10)+AL and clearing AH to 0.

Example:

```
MOV AX,0205h          ;The unpacked BCD number 25
AAD                   ;After AAD , AH=0 and AL=19h (25).
```

After the division AL will then contain the unpacked BCD quotient and AH will contain the unpacked BCD remainder.

Example:

```
                      ;AX=0607 unpacked BCD for 67 decimal CH=09H.
AAD                   ;Adjust to binary before division AX=0043 = 43H =67 decimal.
DIV CH                ;Divide AX by unpacked BCD in CH, AL = quotient = 07 unpacked
BCD, AH = remainder = 04 unpacked BCD
```

- **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:

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

- **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:

;DX = 00000000 00000000 and AX = 11110000 11000111 = - 3897 decimal
CWD ;Convert signed word in AX to signed double word in DX:AX
;Result DX = 11111111 11111111 and AX = 11110000 11000111 = -3897 decimal.

3. Logical Instructions

NOT Instruction - Invert each bit of operand

NOT perform the bitwise complement of operand and stores the result back into operand itself.

Syntax–NOT destination

Example :

NOT BX ;Complement contents of BX register. - DX =F038h
NOT DX ;after the instruction DX = 0FC7h

- **AND Instruction - AND corresponding bits of two operands**

This Performs a bitwise Logical AND of two operands. The result of the operation is stored in the op1 and used to set the flags. AND op1, op2 To perform a bitwise AND of the two operands, each bit of the result is set to 1 if and only if the corresponding bit in both of the operands is 1, otherwise the bit in the result I cleared to 0 .

Example :

AND BH, CL ;AND byte in CL with byte in BH ;result in BH
AND BX,00FFh ;AND word in BX with immediate 00FFH. Mask upper byte, leave lower unchanged
AND CX,[SI] ; AND word at offset [SI] in data segment with word in CX register . Result in CX register and BX = 10110011 01011110
AND BX,00FFh ;Mask out upper 8 bits of BX. ;Result BX = 00000000 01011110 and CF =0 , OF = 0, PF = 0, SF = 0 ,ZF = 0.

- **OR Instruction - Logically OR corresponding of two operands**

OR Instruction - OR instruction perform the bit wise logical OR of two operands .Each bit of the result is cleared to 0 if and only if both corresponding bits in each operand are 0, other wise the bit in the result is set to 1.

Syntax–- OR destination, source.

Examples :

```
OR AH, CL          ;CL is OR'ed with AH, result in AH.
                   ;CX = 00111110 10100101
OR CX,FF00h        ;OR CX with immediate FF00h result in CX = 11111111 10100101
                   ;Upper byte are all 1's lower bytes are unchanged.
```

- **XOR Instruction - Exclusive** XOR destination, source

XOR Instruction - XOR performs a bit wise logical XOR of the operands specified by op1 and op2. The result of the operand is stored in op1 and is used to set the flag.

Syntax–- XOR destination, source.

Example : (Numerical)

```
                   ; BX = 00111101 01101001 and CX = 00000000 11111111
XOR BX, CX         ;Exclusive OR CX with BX and Result BX = 00111101 10010110
```

- **TEST Instruction – AND operand to update flags**

TEST Instruction - This instruction ANDs the contents of a source byte or word with the contents of specified destination word. Flags are updated but neither operand is changed . TEST instruction is often used to set flags before a condition jump instruction

Examples:

```
TEST AL, BH        ;AND BH with AL. no result is stored . Update PF, SF, ZF TEST
CX, 0001H ;AND CX with immediate number
                   ;no result is stored, Update PF,SF
```

Example :

```
                   ;AL = 01010001
TEST Al, 80H       ;AND immediate 80H with AL to test f MSB of AL is 1 or 0
                   ;ZF = 1 if MSB of AL = 0 and AL = 01010001 (unchanged),PF = 0 , SF =
0,
                   ;ZF = 1 because ANDing produced is 00
```

SHIFT INSTRUCTIONS

SAL/SHL Instruction - Shift operand bits left, put zero in LSB(s)

SAL instruction shifts the bits in the operand specified by op1 to its left by the count specified in op2. As a bit is shifted out of LSB position a 0 is kept in LSB position. CF will contain MSB bit.

Syntax - SAL/AHL destination, count

Example:

```
                    ;CF = 0, BX = 11100101 11010011
SAL BX, 1           ;Shift BX register contents by 1 bit position towards left
                    ;CF = 1, BX = 11001011 1010011
```

- **SHR Instruction - Shift operand bits right, put zero in MSB**

SHR instruction shifts the bits in op1 to right by the number of times specified by op2 .

Example:(1)

```
SHR BP, 1           ; Shift word in BP by 1 bit position to right and 0 is kept to MSB
```

Example:(2)

```
MOV CL, 03H         ;Load desired number of shifts into CL
SHR BYTE PYR[BX]    ;Shift bytes in DS at offset BX and rotate 3 bits to right and keep
3 0's in MSB
```

Example:(3)

```
                    ;SI = 10010011 10101101 , CF = 0
SHR SI, 1           ; Result: SI = 01001001 11010110 and CF = 1, OF = 1, SF = 0, ZF =
0
```

- **SAR Instruction - Shift operand bits right, new MAB = old MSB**

SAR instruction shifts the bits in the operand specified by op1 towards right by count specified in op2.As bit is shifted out a copy of old MSB is taken in MSB. MSB position and LSB is shifted to CF.

Syntax - SAR destination, count.

Example: (1)

```
                    ; AL = 00011101 = +29 decimal, CF = 0
SAR AL, 1           ;Shift signed byte in AL towards right ( divide by 2 )
                    ;AL = 00001110 = + 14 decimal, CF = 1
```

Example: (2)

```
                    ;BH = 11110011 = - 13 decimal, CF = 1
SAR BH, 1           ;Shifted signed byte in BH to right and BH = 11111001 = - 7 decimal,
CF = 1.
```

ROTATE INSTRUCTIONS

ROL Instruction - Rotate all bits of operand left, MSB to LSB

ROL instruction rotates the bits in the operand specified by oper1 towards left by the count specified in oper2. ROL moves each bit in the operand to next higher bit position. The higher order bit is moved to lower order position. Last bit rotated is copied into carry flag.

Syntax - ROL destination, count.

Example: (1)

```
ROL AX, 1          ;Word in AX is moved to left by 1 bit and MSB bit is to LSB, and CF and
CF =0 ,BH =10101110
ROL BH, 1          ;Result: CF ,Of =1 , BH = 01011101
```

Example : (2)

```
                   ;BX = 01011100 11010011 and CL = 8 bits to rotate
ROL BH, CL         ;Rotate BX 8 bits towards left and CF =0, BX =11010011 01011100
```

- **ROR Instruction - Rotate all bits of operand right, LSB to MSB**

ROR instruction rotates the bits in the operand oper1 to wards right by count specified in op2. The last bit rotated is copied into CF.

Syntax - ROR destination, count

Example:(1)

```
ROR BL, 1          ;Rotate all bits in BL towards right by 1 bit position, LSB bit is moved to
MSB and CF has last rotated bit.
```

Example:(2)

```
                   ;CF =0, BX = 00111011 01110101
ROR BX, 1          ;Rotate all bits of BX of 1 bit position towards right and CF =1, BX =
10011101 10111010
```

Example (3)

```
                   ;CF = 0, AL = 10110011,
MOVE CL, 04H       ; Load CL
ROR AL, CL         ;Rotate all bits of AL towards right by 4 bits, CF = 0 ,AL = 00111011
```

- **RCL Instruction - Rotate operand around to the left through CF**

RCL instruction rotates the bits in the operand specified by oper1 towards left by the count specified in oper2.The operation is circular, the MSB of operand is rotated into a carry flag and the bit in the CF is rotated around into the LSB of operand.

Syntax - RCL destination, source.

Example(1):

```
CLC                 ;put 0 in CF
RCL AX, 1           ;save higher-order bit of AX in CF
RCL DX, 1           ;save higher-order bit of DX in CF
ADC AX, 0           ; set lower order bit if needed.
```

Example (2):

RCL DX, 1 ;Word in DX of 1 bit is moved to left, and MSB of word is given to CF and CF to LSB CF=0, BH = 10110011
RCL BH, 1 ;Result : BH =01100110 CF = 1, OF = 1 because MSB changed CF =1,AX =00011111 10101001
MOV CL, 2 ;Load CL for rotating 2 bit position
RCL AX, CL ;Result: CF =0, OF undefined AX = 01111110 10100110

- **RCR Instruction - Rotate operand around to the right through CF**

RCR Instruction - RCR instruction rotates the bits in the operand specified by operand1 towards right by the count specified in operand2.

Syntax - RCR destination, count

Example:(1)

RCR BX, 1 ;Word in BX is rotated by 1 bit towards right and CF will contain MSB bit and LSB contain CF bit .

Example:(2)

```
                    ;CF = 1, BL = 00111000
RCR BL, 1           ;Result: BL = 10011100, CF =0 OF = 1 because MSB is changed to 1.
```

4. **Control Transfer Instructions**
   These instructions cause change in the sequence of the execution of instruction. This change can be through a condition or sometimes unconditional. The conditions are represented by flags.

CALL Des:
This instruction is used to call a subroutine or function or procedure. The address of next instruction after CALL is saved onto stack.
RET:
It returns the control from procedure to calling program. Every CALL instruction should have a RET.
JMP Des:

This instruction is used for unconditional jump from one place to another.

Jxx Des (Conditional Jump):

All the conditional jumps follow some conditional statements or any instruction that affects the flag.

Mnemonic	Meaning	Jump Condition
JA	Jump if Above	CF = 0 and ZF = 0
JAE	Jump if Above or Equal	CF = 0
JB	Jump if Below	CF = 1
JBE	Jump if Below or Equal	CF = 1 or ZF = 1
JC	Jump if Carry	CF = 1
JE	Jump if Equal	ZF = 1
JNC	Jump if Not Carry	CF = 0
JNE	Jump if Not Equal	ZF = 0
JNZ	Jump if Not Zero	ZF = 0
JPE	Jump if Parity Even	PF = 1
JPO	Jump if Parity Odd	PF = 0
JZ	Jump if Zero	ZF = 1

Loop Des:

This is a looping instruction. The number of times looping is required is placed in the CX register. With each iteration, the contents of CX are decremented. ZF is checked whether to loop again or not.

5. **String Instructiuons**

String in assembly language is just a sequentially stored bytes or words. There are very strong set of string instructions in 8086. By using these string instructions, the size of the program is considerably reduced.

CMPS Des, Src:

It compares the string bytes or words.

SCAS String:

It scans a string. It compares the String with byte in AL or with word in AX.

MOVS / MOVSB / MOVSW:

It causes moving of byte or word from one string to another. In this instruction, the source string is in Data Segment and destination string is in Extra Segment. SI and DI store the offset values for source and destination index.

REP (Repeat): This is an instruction prefix. It causes the repetition of the instruction until CX becomes zero.

E.g.: REP MOVSB STR1, STR2

It copies byte by byte contents. REP repeats the operation MOVSB until CX becomes zero.

6. Flag Instructions: please refer Tech-max