

# Instruction Description

- **AAA** Instruction - ASCII Adjust after Addition
- **AAD** Instruction - ASCII adjust before Division
- **AAM** Instruction - ASCII adjust after Multiplication
- **AAS** Instruction - ASCII Adjust for Subtraction
- **ADC** Instruction - Add with carry.
- **ADD** Instruction - ADD destination, source
- **AND** Instruction - AND corresponding bits of two operands

# Example

- **AAA** Instruction - 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.

## Example (cont..)

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.
```

## Example (cont..)

- **AAD** Instruction - 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)
```

## Example (cont..)

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

➤ **Example:**

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

## Example (cont..)

- **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 (cont..)

➤ **Example:**

**MOV AL, 5**

**MOV BL, 7**

**MUL BL ;Multiply AL by BL , result in AX**

**AAM ;After AAM, AX =0305h (BCD 35)**

## Example (cont..)

- **AAS** Instruction - 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 (cont..)

( 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 (cont..)

( 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 .
```

## Example (cont..)

➤ **ADD** Instruction - These instructions add a number from source to a number from some destination and put the result in the specified destination. The add with carry instruction ADC, also add the status of the carry flag into the result. 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.

➤ **EXAMPLE:**

```
ADD      AL,74H      ;Add immediate number 74H to  
                        ; content of AL
```

## Example (cont..)

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

## Example (cont..)

*; 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.**

## Example (cont..)

- **AND** Instruction - 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 is cleared to 0 .

**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**

## Example (cont..)

```
AND    CX,[SI]    ; AND word at offset [SI] in data
                    ;segment with word in CX
                    ;register . Result in CX register .
                    ;BX = 10110011 01011110

AND    BX,00FFh   ;Mask out upper 8 bits of BX
                    ;Result BX = 00000000 01011110
                    ;CF = 0 , OF = 0, PF = 0, SF = 0 ,
                    ;ZF = 0
```

## Example (cont..)

- **CALL** Instruction
  - Direct within-segment (near or intrasegment)
  - Indirect within-segment (near or intrasegment)
  - Direct to another segment (far or intersegment)
  - Indirect to another segment (far or intersegment)
- **CBW** Instruction - Convert signed Byte to signed word
- **CLC** Instruction - Clear the carry flag
- **CLD** Instruction - Clear direction flag



## Example (cont..)

- **CLI** Instruction - Clear interrupt flag
- **CMC** Instruction - Complement the carry flag
- **CMP** Instruction - Compare byte or word-  
CMP destination, source.
- **CMPS/CMPSB/  
CMPSW** Instruction - Compare string bytes or  
string words
- **CWD** Instruction - Convert Signed Word to -  
Signed Double word

# *Example*

- **CALL** Instruction - This Instruction is used to transfer execution to a subprogram or procedure. There are two basic types of CALL 's : Near and Far.

A Near CALL is a call to a procedure which is in the same code segment as the CALL instruction .

When 8086 executes the near CALL instruction it decrements the stack pointer by two and copies the offset of the next instruction after the CALL on the stack. This offset saved on the stack is referred as the return address, because this is the address that execution will return to after the procedure executes. A near CALL instruction will also load the instruction pointer with the offset of the first instruction in the procedure.

## Example (cont..)

- A RET instruction at the end of the procedure will return execution to the instruction after the CALL by copying the offset saved on the stack back to IP. A Far CALL is a call to a procedure which is in a different segment from that which contains the CALL instruction. When 8086 executes the Far CALL instruction it decrements the stack pointer by two again and copies the content of CS register to the stack. It then decrements the stack pointer by two again and copies the offset contents of the instruction after the CALL to the stack. Finally it loads CS with segment base of the segment which contains the procedure and IP with the offset of the first instruction of the procedure in segment. A RET instruction at end of procedure will return to the next instruction after the CALL by restoring the saved CS and IP from the stack.

## Example (cont..)

**;Direct within-segment ( near or intrasegment )**

**CALL          MULTO          ;MULTO is the name of the procedure. The assembler determines displacement of MULTO from the instruction after the CALL and codes this displacement in as part of the instruction .**

**;Indirect within-segment ( near or intrasegment )**

**CALL          BX          ; BX contains the offset of the first instruction of the procedure .Replaces contents of word of IP with contents o register BX.**

## Example (cont..)

**CALL      WORD PTR[BX]      ;Offset of first instruction  
of procedure is in two memory addresses in DS .Replaces  
contents of IP with contents of word memory location in  
DS pointed to by BX.**

**;Direct to another segment- far or intersegment.**

**CALL      SMART      ;SMART is the name of the  
                                 ;Procedure**

**SMART    PROC   FAR ; Procedure must be declare as  
                                 ;an far**

## Example (cont..)

- **CBW** Instruction - 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:

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

## Example (cont..)

- **CLC** Instruction - CLC clear the carry flag ( CF ) to 0  
This instruction has no affect on the processor, registers, or other flags. It is often used to clear the CF before returning from a procedure to indicate a successful termination. It is also use to clear the CF during rotate operation involving the CF such as ADC, RCL, RCR .

**Example:**

**CLC ;Clear carry flag.**

## Example (cont..)

- **CLD** Instruction - This instruction reset the designation flag to zero. This instruction has no effect on the registers or other flags. When the direction flag is cleared / reset SI and DI will automatically be incremented when one of the string instruction such as MOVS, CMPS, SCAS,MOVSB and STOSB executes.

**Example :**

**CLD           ;Clear direction flag so that string pointers  
                  ;auto increment**



## Example (cont..)

- **CLI** Instruction - This instruction resets the interrupt flag to zero. No other flags are affected. If the interrupt flag is reset, the 8086 will not respond to an interrupt signal on its INTR input. This CLI instruction has no effect on the nonmaskable interrupt input, NMI
- **CMC** Instruction - If the carry flag CF is a zero before this instruction, it will be set to a one after the instruction. If the carry flag is one before this instruction, it will be reset to a zero after the instruction executes. CMC has no effect on other flags.

### **Example:**

**CMC ;Invert the carry flag.**

## Example (cont..)

- **CWD** Instruction - 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 (cont..)

Example:

CWD

;DX = 00000000 00000000

;AX = 11110000 11000111 = - 3897 decimal

;Convert signed word in AX to signed double

;word in DX:AX

;Result DX = 11111111 11111111

;AX = 11110000 11000111 = -3897 decimal .

## Example (cont..)

- **DAA** Instruction - Decimal Adjust Accumulator
- **DAS** Instruction - Decimal Adjust after Subtraction
- **DEC** Instruction - Decrement destination register or memory DEC destination.
- **DIV** Instruction - Unsigned divide-Div source
- **ESC** Instruction

## Example (cont..)

- **DIV** Instruction - This instruction is used to divide an Unsigned word by a byte or to divide an unsigned double word by a word.

When dividing a word by a byte , the word must be in the **AX** register. After the division **AL** will contains an 8-bit result (quotient) and **AH** will contain an 8- bit remainder. If an attempt is made to divide by 0 or the quotient is too large to fit in **AL** ( greater than **FFH** ), the 8086 will automatically do a type 0 interrupt .

### **Example:**

**DIV            BL            ;Word in AX / byte in BL**  
**;Quotient in AL . Remainder in AH.**

## Example (cont..)

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**

## Example (cont..)

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:**

	<b>; AX = 37D7H = 14, 295 decimal</b>
	<b>; BH = 97H = 151 decimal</b>
<b>DIV BH</b>	<b>; AX / BH</b>
	<b>; AX = Quotient = 5EH = 94 decimal</b>
	<b>; AH = Remainder = 65H = 101 decimal</b>

## Example (cont..)

- **ESC Instruction** - Escape instruction is used to pass instruction to a coprocessor such as the 8087 math coprocessor which shares the address and data bus with an 8086. Instruction for the coprocessor are represented by a 6 bit code embedded in the escape instruction. As the 8086 fetches instruction byte, the coprocessor also catches these bytes from data bus and puts them in its queue. The coprocessor treats all of the 8086 instruction as an NOP. When 8086 fetches an ESC instruction , the coprocessor decodes the instruction and carries out the action specified by the 6 bit code. In most of the case 8086 treats ESC instruction as an NOP.



## Example (cont..)

- **HLT** Instruction - HALT processing
- **IDIV** Instruction - Divide by signed byte or word  
IDIV source
- **IMUL** Instruction - Multiply signed number-IMUL  
source
- **IN** Instruction - Copy data from a port  
IN accumulator, port
- **INC** Instruction - Increment - INC destination

## Example (cont..)

- **HALT** Instruction - The HLT instruction will cause the 8086 to stop fetching and executing instructions. The 8086 will enter a halt state. The only way to get the processor out of the halt state are with an interrupt signal on the INTR pin or an interrupt signal on NMI pin or a reset signal on the RESET input .
- **IDIV** Instruction - This instruction is used to divide a signed word by a signed byte or to divide a signed double word by a signed word.
- **Example:**  
**IDIV BL ;Signed word in AX is divided by signed ;byte in BL**

## Example (cont..)

### ➤ Example:

**IDIV BP ;divide a Signed double word in DX and  
;AX by signed word in BP**

**IDIV BYTE PTR[BX] ; divide AX by a byte at  
;offset [BX] in DS**

- **A signed word divided by a signed byte**

**;AX = 0000011 10101011 = 03ABH=39 decimal**

**;BL = 11010011 = D3H = - 2DH = - 45 decimal**

**IDIV BL;Quotient AL= ECH = - 14H = -20 decimal**

**;Remainder AH = 27H = + 39 decimal**

## Example (cont..)

- **IMUL** Instruction - This instruction performs a signed multiplication.

**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 (cont..)

### ➤ Example:

**IMUL      BL**

**; 69 \* 14**

**; AL = 01000101 = 69 decimal**

**; BL = 00001110 = 14 decimal**

**; AX = 03C6H = + 966 decimal**

**; MSB = 0 because positive result**

**IMUL      BL**

**; - 28 \* 59**

**; AL = 11100100 = - 28 decimal**

**; BL = 00001110 = 14 decimal**

**; AX = F98Ch = - 1652 decimal**

**; MSB = 1 because negative result**

## Example (cont..)

- **IN** Instruction - 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**

## Example (cont..)

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.**

## Example (cont..)

- **INC** Instruction - 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:**

```
INC AX ; AX = 7FFFh
      ;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.
```



## Example (cont..)

- **INT** Instruction - Interrupt program
- **INTO** Instruction - Interrupt on overflow.
- **IRET** Instruction - Interrupt return
- **JA/JNBE** Instruction - Jump if above/Jump if not below nor equal.
- **JAE/JNB/  
JNC** Instructions - Jump if above or equal/  
Jump if not below/  
Jump if no carry.

## Example (cont..)

➤ **JA / JNBE** - This instruction performs the Jump if above (or) Jump if not below or equal operations according to the condition, if CF and ZF = 0 .

➤ **Example: ( 1 )**

**CMP AX, 4371H ;Compare by subtracting 4371H  
;from AX**

**JA RUN\_PRESS ;Jump to label RUN\_PRESS if  
;AX above 4371H**

**( 2 )**

**CMP AX, 4371H ;Compare ( AX – 4371H)**

**JNBE RUN\_PRESS ;Jump to label RUN\_PRESS if  
;AX not below or equal to 4371H**

## Example (cont..)

➤ **JAE / JNB / JNC** - This instructions performs the Jump if above or equal, Jump if not below, Jump if no carry operations according to the condition, if  $CF = 0$ .

➤ **Examples:**

- CMP**      **AX, 4371H**      ;Compare ( **AX – 4371H**)  
**JAE**      **RUN**      ;Jump to the label **RUN** if **AX** is  
                                 ;above or equal to **4371H** .
- CMP**      **AX, 4371H**      ;Compare ( **AX – 4371H**)  
**JNB**      **RUN\_1**      ;Jump to the label **RUN\_1** if **AX**  
                                 ;is not below than **4371H**
- ADD**      **AL, BL**      ; Add **AL, BL**. If result is with in  
**JNC**      **OK**      ;acceptable range, continue

## Example (cont..)

- **JB/JC/JNAE** Instruction - Jump if below/Jump if carry/  
Jump if not above nor equal
- **JBE/JNA** Instructions - Jump if below or equal /  
Jump if not above
- **JCXZ** Instruction - Jump if the CX register is  
zero
- **JE/JZ** Instruction - Jump if equal/Jump if zero
- **JG/JNLE** Instruction - Jump if greater/Jump if not  
less than nor equal

## Example (cont..)

- **JB/JC/JNAE** Instruction - This instruction performs the Jump if below (or) Jump if carry (or) Jump if not below/ equal operations according to the condition, if  $CF = 1$

- **Example:**

- CMP AX, 4371H ;Compare ( AX – 4371H )**  
**JB RUN\_P ;Jump to label RUN\_P if AX is**  
**;below 4371H**
  
- ADD BX, CX ;Add two words and Jump to**  
**JC ERROR ; label ERROR if CF = 1**

## Example (cont..)

➤ **JBE/JNA** Instruction - This instruction performs the Jump if below or equal (or) Jump if not above operations according to the condition, if CF and ZF = 1

➤ **Example:**

**CMP          AX, 4371H      ;Compare ( AX – 4371H )**

**JBA          RUN            ;Jump to label RUN if AX is  
;below or equal to 4371H**

**CMP          AX, 4371H      ;Compare ( AX – 4371H )**

**JNA          RUN\_R            ;Jump to label RUN\_R if AX is  
;not above than 4371H**

## Example (cont..)

➤ **JCXZ** Instruction - This instruction performs the Jump if CX register is zero. If CX does not contain all zeros, execution will simply proceed to the next instruction.

➤ **Example:**

```
JCXZ  SKIP_LOOP;If CX = 0, skip the process  
NXT: SUB  [BX], 07H    ;Subtract 7 from data value  
INC   BX              ; BX point to next value  
LOOP  NXT             ; Loop until CX = 0  
SKIP_LOOP            ;Next instruction
```

## Example (cont..)

- **JE/JZ** Instruction Instruction - This instruction performs the Jump if equal (or) Jump if zero operations according to the condition if  $ZF = 1$

- **Example:**

```
NXT: CMP    BX, DX ;Compare ( BX – DX )  
    JE     DONE  ;Jump to DONE if BX = DX,  
    SUB    BX, AX ;Else subtract Ax  
    INC    CX     ;Increment counter  
    JUMP   NXT    ;Check again  
DONE: MOV   AX, CX ;Copy count to AX
```



## Example (cont..)

### ➤ Example:

```
IN    AL, 8FH    ;read data from port 8FH  
SUB   AL, 30H    ;Subtract minimum value  
JZ    STATR     ; Jump to label if result of  
                ;subtraction was 0
```

## Example (cont..)

➤ **JG/JNLE** Instruction - This instruction performs the Jump if greater (or) Jump if not less than or equal operations according to the condition if ZF = 0 and SF = OF

➤ **Example:**

<b>CMP</b>	<b>BL, 39H</b>	<b>;Compare by subtracting ;39H from BL</b>
------------	----------------	---

<b>JG</b>	<b>NEXT1</b>	<b>;Jump to label if BL is ;more positive than 39H</b>
-----------	--------------	--

<b>CMP</b>	<b>BL, 39H</b>	<b>;Compare by subtracting ;39H from BL</b>
------------	----------------	---

<b>JNLE</b>	<b>NEXT2</b>	<b>;Jump to label if BL is not less than or equal 39H</b>
-------------	--------------	---

## Example (cont..)

- **JGE/JNL** Instruction - Jump if greater than or equal/  
Jump if not less than
- **JL/JNGE** Instruction - Jump if less than/Jump if not  
greater than or equal
- **JLE/JNG** Instruction - Jump if less than or equal/  
Jump if not greater
- **JMP** Instruction - Unconditional jump to -  
specified destination

## Example (cont..)

➤ **JGE/JNL** Instruction - This instruction performs the Jump if greater than or equal / Jump if not less than operation according to the condition if SF = OF

➤ **Example:**

<b>CMP</b>	<b>BL, 39H</b>	<b>;Compare by the ;subtracting 39H from BL</b>
<b>JGE</b>	<b>NEXT11</b>	<b>;Jump to label if BL is ;more positive than 39H ; or equal to 39H</b>
<b>CMP</b>	<b>BL, 39H</b>	<b>;Compare by subtracting ;39H from BL</b>
<b>JNL</b>	<b>NEXT22</b>	<b>;Jump to label if BL is not ;less than 39H</b>

## Example (cont..)

➤ **JL/JNGE** Instruction - This instruction performs the Jump if less than / Jump if not greater than or equal operation according to the condition, if SF  $\neq$  OF

➤ **Example:**

**CMP BL, 39H ;Compare by subtracting 39H  
;from BL**

**JL AGAIN ;Jump to the label if BL is more  
;negative than 39H**

**CMP BL, 39H ;Compare by subtracting 39H  
;from BL**

**JNGE AGAIN1 ; Jump to the label if BL is not  
;more positive than 39H or  
;not equal to 39H**

## Example (cont..)

➤ **JLE/JNG** Instruction - This instruction performs the Jump if less than or equal / Jump if not greater operation according to the condition, if ZF=1 and SF ≠ OF

➤ **Example:**

**CMP**      **BL, 39h**      ; Compare by subtracting 39h  
   ;from BL

**JLE**      **NXT1**      ;Jump to the label if BL is more  
   ;negative than 39h or equal to 39h

**CMP**      **BL, 39h**      ;Compare by subtracting 39h  
   ;from BL

**JNG**      **AGAIN2**      ; Jump to the label if BL is not  
   ;more positive than 39h

## Example (cont..)

- **JNA/JBE** Instruction - Jump if not above/Jump if below or equal
- **JNAE/JB** Instruction - Jump if not above or equal/  
Jump if below
- **JNB/JNC/JAE** Instruction - Jump if not below/Jump if no carry/Jump if above or equal
- **JNE/JNZ** Instruction - Jump if not equal/Jump if not zero

## Example (cont..)

- **JNE/JNZ** Instruction - This instruction performs the Jump if not equal / Jump if not zero operation according to the condition, if ZF=0

- **Example:**

```
NXT: IN     AL, 0F8H    ;Read data value from port  
      CMP   AL, 72     ;Compare ( AL – 72 )  
      JNE   NXT        ;Jump to NXT if AL ≠ 72  
      IN   AL, 0F9H    ;Read next port when AL = 72
```

```
      MOV   BX, 2734H  ; Load BX as counter  
NXT_1:ADD  AX, 0002H  ;Add count factor to AX  
      DEC  BX         ;Decrement BX  
      JNZ  NXT_1      Repeat until BX = 0
```



## Example (cont..)

- **JNG/JLE** Instruction - Jump if not greater/ Jump if less than or equal
- **JNGE/JL** Instruction - Jump if not greater than nor equal/Jump if less than
- **JNL/JGE** Instruction - Jump if not less than/ Jump if greater than or equal
- **JNLE/JG** Instruction - Jump if not less than nor equal to /Jump if greater than

## Example (cont..)

- **JNO** Instruction – Jump if no overflow
- **JNP/JPO** Instruction – Jump if no parity/ Jump if parity odd
- **JNS** Instruction - Jump if not signed (Jump if positive)
- **JNZ/JNE** Instruction - Jump if not zero / jump if not equal
- **JO** Instruction - Jump if overflow

## Example (cont..)

➤ **JNO** Instruction – This instruction performs the Jump if no overflow operation according to the condition, if OF=0

➤ **Example:**

```
ADD AL, BL ; Add signed bytes in AL and BL  
JNO DONE ;Process done if no overflow -  
MOV AL, 00H ;Else load error code in AL  
DONE: OUT 24H, AL ; Send result to display
```

## Example (cont..)

➤ **JNP/JPO** Instruction – This instruction performs the Jump if not parity / Jump if parity odd operation according to the condition, if PF=0

➤ **Example:**

```
IN    AL, 0F8H    ;Read ASCII char from UART  
OR    AL, AL      ;Set flags  
JPO   ERROR1     ;If even parity executed, if not  
                ;send error message
```

## Example (cont..)

- **JNS** Instruction - This instruction performs the Jump if not signed (Jump if positive) operation according to the condition, if SF=0

- Example:

**DEC AL ;Decrement counter**

**JNS REDO ; Jump to label REDO if counter has not ;decremented to FFH**

- **JO** Instruction - This instruction performs Jump if overflow operation according to the condition OF = 0

- Example:

**ADD AL, BL ;Add signed bits in AL and BL**

**JO ERROR ; Jump to label if overflow occur ;in addition**

**MOV SUM, AL ; else put the result in memory ;location named SUM**

## Example (cont..)

- **JPE/JP** Instruction - Jump if parity even/ Jump if parity
- **JPO/JNP** Instruction - Jump if parity odd/ Jump if no parity
- **JS** Instruction - Jump if signed (Jump if negative)
- **JZ/JE** Instruction - Jump if zero/Jump if equal

## Example (cont..)

- **JPE/JP** Instruction - This instruction performs the Jump if parity even / Jump if parity operation according to the condition, if PF=1

### Example:

```
IN    AL, 0F8H    ;Read ASCII char from UART  
OR    AL, AL      ;Set flags  
JPE   ERROR2     ;odd parity is expected, if not  
                ;send error message
```

- **JS** Instruction - This instruction performs the Jump if sign operation according to the condition, if SF=1

### ➤ Example:

```
ADD   BL, DH      ;Add signed bytes DH to BL  
JS    JJS_S1      ;Jump to label if result is  
                ;negative
```

## Example (cont..)

- **LAHF** Instruction - Copy low byte of flag register to AH
- **LDS** Instruction - Load register and DS with words from memory – LDS register, memory address of first word
- **LEA** Instruction - Load effective address-LEA register, source
- **LES** Instruction - Load register and ES with words from memory –LES register, memory address of first word.



## Example (cont..)

- **LAHF** Instruction - LAHF instruction copies the value of SF, ZF, AF, PF, 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.
- **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. **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.**

## Example (cont..)

➤ **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.

➤ **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**

## Example (cont..)

- **LOCK** Instruction - Assert bus lock signal
- **LODS/LODSB/LODSW** Instruction - Load string byte into AL or Load string word into AX.
- **LOOP** Instruction - Loop to specified label until  $CX = 0$
- **LOOPE / LOOPZ** Instruction - loop while  $CX \neq 0$  and  $ZF = 1$

## Example (cont..)

- **LODS/LODSB/LODSW** Instruction - This instruction copies a byte from a string location pointed to by SI to AL or a word from a string location pointed to by SI to AX. If DF is cleared to 0, SI will automatically incremented to point to the next element of string.

- **Example:**

**CLD** ;Clear direction flag so SI is auto incremented

**MOV SI, OFFSET SOURCE\_STRING**  
;point SI at start of the string

**LODS SOURCE\_STRING** ;Copy byte or word from  
;string to AL or AX

## Example (cont..)

➤ **LOOP** Instruction - This instruction is used to repeat a series of instruction some number of times

➤ **Example:**

**MOV BX, OFFSET PRICE**

**;Point BX at first element in array**

**MOV CX, 40**

**;Load CX with number of  
;elements in array**

**NEXT: MOV AL, [BX]**

**; Get elements from array**

**ADD AL, 07H**

**;Ad correction factor**

**DAA**

**; decimal adjust result**

**MOV [BX], AL**

**; Put result back in array**

**LOOP NEXT**

**; Repeat until all elements  
;adjusted.**

## Example (cont..)

➤ **LOOPE / LOOPZ** Instruction - This instruction is used to repeat a group of instruction some number of times until  $CX = 0$  and  $ZF = 0$

➤ **Example:**

```
MOV BX, OFFSET ARRAY
```

```
;point BX at start of the array
```

```
DEC BX
```

```
MOV CX, 100
```

```
;put number of array elements in  
;CX
```

```
NEXT:INC BX
```

```
;point to next element in array
```

```
CMP [BX], 0FFH
```

```
;Compare array elements FFH
```

```
LOOP NEXT
```

## Example (cont..)

➤ **LOOPNE/LOOPNZ** Instruction - This instruction is used to repeat a group of instruction some number of times until  $CX = 0$  and  $ZF = 1$

➤ **Example:**

```
MOV BX, OFFSET ARRAY1
```

```
;point BX at start of the array
```

```
DEC BX
```

```
MOV CX, 100
```

```
;put number of array elements in  
;CX
```

```
NEXT:INC BX
```

```
;point to next elements in array
```

```
CMP [BX], 0FFH
```

```
;Compare array elements 0DH
```

```
LOOPNE NEXT
```

## Example (cont..)

- **MOV** Instruction - MOV destination, source
- **MOVS/MOVS**/  
**MOVSW** Instruction - Move string byte or string  
word-MOVS destination,  
source
- **MUL** Instruction - Multiply unsigned bytes or  
words-MUL source
- **NEG** Instruction - From 2's complement –  
NEG destination
- **NOP** Instruction - Performs no operation.



## Example (cont..)

- **MOV** Instruction - 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.**

## Example (cont..)

- **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 . **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.**

## Example (cont..)

- **NEG** Instruction - 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**

## Example (cont..)

- **NOP** Instruction - This instruction simply uses up the three clock cycles and increments the instruction pointer to point to the next instruction. NOP does not change the status of any flag. The NOP instruction is used to increase the delay of a delay loop.

## Example (cont..)

- **NOT** Instruction - Invert each bit of operand – NOT destination.
- **OR** Instruction - Logically OR corresponding of two operands- OR destination, source.
- **OUT** Instruction - Output a byte or word to a port – OUT port, accumulator AL or AX.
- **POP** Instruction - POP destination

## Example (cont..)

- **NOT** Instruction - NOT perform the bitwise complement of op and stores the result back into op.

**NOT op**

**Example :**

**NOT BX ;Complement contents of BX register.**

**;DX =F038h**

**NOT DX ;after the instruction DX = 0FC7h**

## Example (cont..)

- **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.

**OR op1, op2**

**Examples :**

**OR AH, CL ;CL ORed 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.**

## Example (cont..)

- **OUT Instruction** - 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 : (1) Port number is specified by an immediate byte constant, ( 0 - 255 ).It is also called as fixed port form. (2) Port number is provided in the DX register ( 0 – 65535 )

- **Example:** (1)

**OUT 3BH, AL ;Copy the contents of the AL to port 3Bh**

**OUT 2CH,AX ;Copy the contents of the AX to port 2Ch**

(2)

**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**



## Example (cont..)

➤ **POP** Instruction - 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].**

## Example (cont..)

- **POPF** Instruction - Pop word from top of stack to flag - register.
- **PUSH** Instruction - PUSH source
- **PUSHF** Instruction- Push flag register on the stack
- **RCL** Instruction - Rotate operand around to the left through CF –RCL destination, source.
- **RCR** Instruction - Rotate operand around to the right through CF- RCR destination, count

## Example (cont..)

- **POPF** Instruction - 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.
- **PUSH** Instruction - 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 points.

- **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 .
```

## Example (cont..)

- **PUSHF** Instruction - This instruction decrements the SP by 2 and copies the word in flag register to the memory location pointed to by SP.
- **RCL** Instruction - RCL instruction rotates the bits in the operand specified by op1 towards left by the count specified in op2. 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. **RCR op1, op2**
- **Example:**

**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 (cont..)

### ➤ Example :

**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**

## Example (cont..)

➤ **RCR** Instruction - RCR instruction rotates the bits in the operand specified by op1 towards right by the count specified in op2. **RCR op1, op2**

➤ **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 .**

**( 2) ;CF = 1, BL = 00111000**

**RCR BL, 1 ;Result: BL = 10011100, CF =0**

**;OF = 1 because MSB is changed to 1.**

## Example (cont..)

- **REP/REPE/REPZ/REPNE/REPNZ** - (Prefix) Repeat String instruction until specified condition exist
- **RET** Instruction - Return execution from procedure to calling program.
- **ROL** Instruction - Rotate all bits of operand left, MSB to LSB  
ROL destination, count.

## Example (cont..)

- **ROL Instruction** - ROL instruction rotates the bits in the operand specified by op1 towards left by the count specified in op2. 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.

**ROL op1, op2**

- **Example: ( 1 )**

**ROL AX, 1 ;Word in AX is moved to left by 1 bit  
;and MSB bit is to LSB, and CF**

**;CF =0 ,BH =10101110**

**ROL BH, 1 ;Result: CF ,Of =1 , BH = 01011101**



## Example (cont..)

➤ **Example : ( 2 )**

**ROL      BH, CL**

**;BX = 01011100 11010011**

**;CL = 8 bits to rotate**

**;Rotate BX 8 bits towards left**

**;CF =0, BX =11010011 01011100**

## Example (cont..)

- **ROR** Instruction - Rotate all bits of operand right, LSB to MSB – ROR destination, count
- **SAHF** Instruction – Copy AH register to low byte of flag register

## Example (cont..)

➤ **ROR** Instruction - ROR instruction rotates the bits in the operand op1 towards right by count specified in op2. The last bit rotated is copied into CF. `ROR op1, op2`

➤ **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.**

**( 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 (cont..)

### ➤ 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
```

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

## Example (cont..)

- **SAL/SHL** Instruction - Shift operand bits left, put zero in LSB(s)  
SAL/AHL destination, count
- **SAR** Instruction - Shift operand bits right, new MAB = old MSB  
SAR destination, count.
- **SBB** Instruction - Subtract with borrow  
SBB destination, source

## Example (cont..)

- **SAL / SHL** Instruction - 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.

**SAL op1,op2**

- **Example:**

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

## Example (cont..)

- **SAR** Instruction - 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. **SAR op1, op2**
- **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**
- ( 2 )**  
**;BH = 11110011 = - 13 decimal, CF = 1**  
**SAR BH, 1 ;Shifted signed byte in BH to right**  
**;BH = 11111001 = - 7 decimal, CF = 1**

## Example (cont..)

➤ **SBB** Instruction - SUBB instruction subtracts op2 from op1, then subtracts 1 from op1 if CF flag is set and result is stored in op1 and it is used to set the flag.

➤ **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 (cont..)

- **STD** Instruction - Set the direction flag to 1
- **STI** Instruction - Set interrupt flag ( IF)
- **STOS/STOSB/  
STOSW** Instruction - Store byte or word in string.
- **SCAS/SCASB/  
SCASW** Instruction - Scan string byte or a  
string word.
- **SHR** Instruction - Shift operand bits right, put  
zero in MSB
- **STC** Instruction - Set the carry flag to 1

## Example (cont..)

➤ **SHR** Instruction - 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**

( 2 )

**MOV CL, 03H ;Load desired number of shifts into ;CL**

**SHR BYTE PTR[BX] ;Shift bytes in DS at offset BX ;and rotate 3 bits to right and ;keep 3 0's in MSB**

( 3 ) ;SI = 10010011 10101101 , CF = 0

**SHR SI, 1 ; Result: SI = 01001001 11010110 ; CF = 1, OF = 1, SF = 0, ZF = 0**

## Example (cont..)

- **TEST** Instruction – AND operand to update flags
- **WAIT** Instruction - Wait for test signal or interrupt signal
- **XCHG** Instruction - Exchange XCHG destination, source
- **XLAT/**  
**XLATB** Instruction - Translate a byte in AL
- **XOR** Instruction - Exclusive OR corresponding bits of two operands –  
XOR destination, source

## Example (cont..)

➤ **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 (cont..)

### ➤ Example :

**TEST      AI, 80H**

**;AL = 01010001**

**;AND immediate 80H with AL to  
;test if MSB of AL is 1 or 0**

**;ZF = 1 if MSB of AL = 0**

**;AL = 01010001 (unchanged)**

**;PF = 0 , SF = 0**

**;ZF = 1 because ANDing produced  
; is 00**

## Example (cont..)

- **WAIT** Instruction - When this **WAIT** instruction executes, the 8086 enters an idle condition. This will stay in this state until a signal is asserted on **TEST** input pin or a valid interrupt signal is received on the **INTR** or **NMI** pin.

**FSTSW STATUS** ;copy 8087 status word to memory

**FWAIT** ;wait for 8087 to finish before-  
; doing next 8086 instruction

**MOV AX,STATUS** ;copy status word to **AX** to  
;check bits

- In this code we are adding up of **FWAIT** instruction so that it will stop the execution of the command until the above instruction is finishes it's work .so that you are not loosing data and after that you will allow to continue the execution of instructions.

## Example (cont..)

- **XCHG** Instruction - 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 are not supported.

**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.



# Example

- **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.

**XOR** op1, op2

**Example : ( Numerical )**

**; BX = 00111101 01101001**

**;CX = 00000000 11111111**

**XOR BX, CX ;Exclusive OR CX with BX**

**;Result BX = 00111101 10010110**