UNIT - II

8086 Programming :- Instruction set, Addressing modes, assembler directives, programming with assembler, stack and stack structure, interrupt and interrupt service routines, interrupt cycle of 8086.

Instruction Set :-

The instruction set of 8086/8088 are categorised into following main types.

(a) **Data Copy/Transfer Instructions :-** These types of instructions are used to transfer data from source operand to destination operand.

(i) **MOV** : (move) This transfer instruction transfers data from one register/memory to another register/memory location

Eg:- MOV AX, 5000H
MOV DS, AX

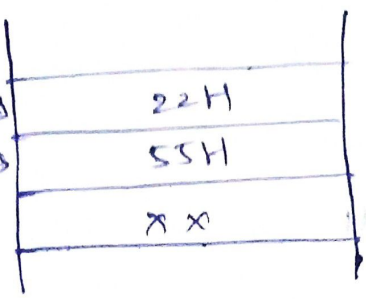
(ii) **PUSH** : (Push to stack) This instruction is used to push the content of register/memory onto the stack.

As the data is pushed the stack pointer is decremented by 2.

The higher byte is pushed first and then the lower byte.

Eg:- AL = 22H

AH = 55H

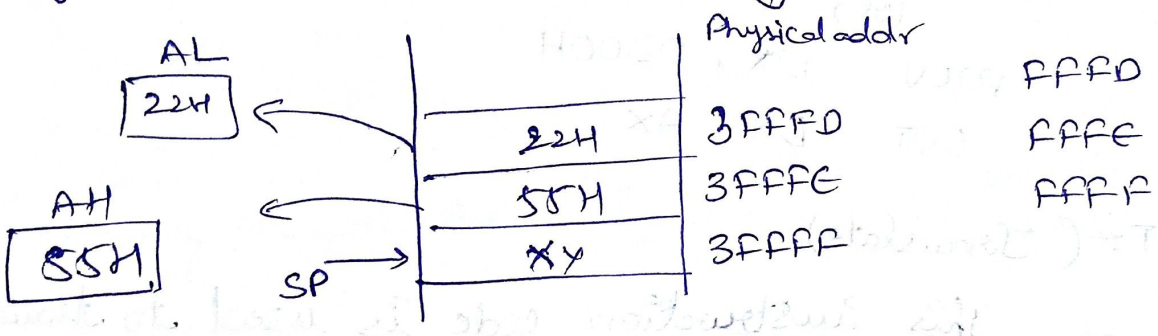


PUSH AX

(iii) POP :- (Pop from stack) Whenever this instruction is executed, it loads the register with the content of stack register and SP location specified.

SP pointer is incremented by 2.

Eg:- POP AX



(iv) XCHG (Exchange) :- Used to exchange the contents of specified source & destination operands.

Note:- Exchange b/w registers, b/w register and memory locations are allowed, But not b/w memory to memory.

Eg:- XCHG [5000H], AX

XCHG BX, AX.

(v) IN :- (Input the port) : Used for reading the input port, whose address value is specified directly or indirectly.

Note:- AX (or) AL, AH are used as destination for 16 bit & 8-bit operations.

And DX is used to carry the port address.

Eg:- IN AL, 03H

IN AX, DX

(or)

MOV DX, 0800H

IN AX, DX

(vi) OUT (out to the port):- This instruction is used for writing to an output port directly or indirectly as specified.

Eg:- OUT 03H, AL

OUT DX, AX

(or)

MOV DX, 0300H

OUT DX, AX

(v) XLAT:- (Translate)

This instruction code is used to translate one code to other using look up table technique.

Eg:- A hexadecimal pad having 16 keys (0-F) is interfaced with 8086 via 8255.

whenever a key is pressed, such key is to be translated to equivalent 7-segment display depending on the reference of its look up table stored in a location.

Program:- MOV AL, CODE # Code word is sent to AL

MOV BX, OFFSET TABLE # Look up table address loaded to BX

XLAT # Finding the equivalent code and store in AL

(vi) LEA (Load Effective Address) :- It load the effective address from ~~source~~ destination operand to source register as specified in the command.

Eg:- LEA BX, ADR . # offset of ADR to BX
LEA SI, ADR[BX] # offset of ADR added to BX for EA and stored to SI.

(vii) LDS/LES :- (Load pointer to ES or DS) It is used to load DS or ES register and the specified register in the instruction with the content of memory location specified as source.

Eg:- LDS BX, 5000H # The data in 5000H & 5001H is loaded to BH, BL, & DS
LES BX, 5000H or ES.

viii) LAHF :- (Load AH from lower byte of flag) It loads the AH register with the lower byte of flag register to observe its status.

ix) SAHF :- Store AH to flag register :- It stores the corresponding bits of AH to lower bytes of flag register positions.

x) PUSHF :- The push flag instruction pushes the flag register on to the stack. First the upper byte is pushed and then the lower byte is pushed. SP is decremented by '2'.

xi) POPF :- (Pop flag from stack)
It pops the flag status from the memory location addressed by SP & SS on to the flag register.
SP is incremented by '2'.

II Arithmetic Instructions:-

(i) ADD : (Addition) It adds an immediate data or the content of memory location specified in the instruction or register or memory location and stores it in destination operand.

Syntax:- ADD Destination, Source

Eg:- ADD AX, 0100H
ADD AX, BX

(ii) ADC : (Add with Carry) : This instruction is used to add an immediate data or content of memory location and also carry flag to the result.

Eg:- ADC 0100H.
ADC AX, BX.

(iii) INC (Increment) :- This instruction add 1 to the operand.

Eg:- INC AX

(iv) DEC (Decrement):- This instruction subtracts 1 from the content of the specified register or memory location.

Eg:- DEC AX.

(v) SUB (Subtract):- This instruction subtracts the source operand from the destination operand and leaves the result in destination operand.

Eg:- SUB AX, 0100H
SUB AX, BX

SBB (Subtract with borrow):- It subtracts source operand and CF bit from the destination operand.

Eg:- SBB AX, 0100H
SBB AX, BX.

CMP (compare):- It compares source operand with destination operand value.

For comparison, it subtracts source from destination if the answer is zero both are equal & zero flag is set. If source > destination carry flag is set otherwise it is reset.

Eg:- CMP [5000H], 0100H
CMP BX, CX.

AAA (ASCII adjust after addition):- AAA is executed after addition operation that adds two ASCII coded operands.

AAS (ASCII adjust after subtraction):- It corrects the result in AL register after subtracting two ASCII operands.

AAM: (ASCII adjust after multiplication):- It converts the product available in AL into unpacked BCD format.

Eg:-
```
MOV AL, 04 ;
MOV BL, 09 ;
MUL BL     ; AL - AL ← 24H
AAM        ; AH ← 03   AL ← 06.
```

AAD (ASCII adjust before division):- In this instruction first division is performed and then converted to ASCII

DAA (Decimal adjust accumulator):- This instruction is used to convert the result of addition to packed BCD

number to a valid BCD number.

Eg:- AL = 53 CL = 29

```
ADD  AL, CL  ;  AL ← AL + CL
                AL ← 7C
DAA             AL ← 7C + 06  (as C > 9)
                AL ← 82.
```

DAS (Decimal adjust after subtraction) :- The result of subtraction of two packed BCD numbers is converted to valid BCD number.

Eg:- AL = 75 BH = 46

```
SUB  AL, BH  ;  AL ← 2F
DAS          ;  AL ← 29 (as F > 9, F - 6 = 9)
```

NEG (Negate) :- It performs 2's complement of the specified distination in the instruction.

MUL :- This instruction multiplies an unsigned byte or word by the content of AL and stores the result in AX and DX as well.

Eg:- MUL BH ; AL ← AL × BH.

IMUL (Signed Multiplication) :- This instruction multiplies a signed byte in source operand by a signed byte in AL and stores the result in AX or in DX

Eg:-

```
IMUL  BH    ;  AL ← BH × AL.
IMUL  [SI]  ;  AL ← [SI] × AL
```

CBW (Convert signed byte to word) :- It converts signed byte to signed word by copying signed bit of AL to AH and the result is in AX.

CWD (convert word to double word) :-
It converts signed word to double word by copying signed bit of AX to DX register

DIV (unsigned division) :- It divides unsigned word or double word with the divisor as specified by the operand of location.
The result will be in AL (quotient) and remainder will be in AH.

IDIV (signed division) :- It just perform signed numbers and memory locations and results are same as that of DIV operation.

(ii) Logical Instructions :-

AND :(Logical AND) It executes bit by bit AND operation b/w operands, register or memory location.

Eg:- AND AX, 0008H
AND AX, BX
AND AX, [5000H]

OR:(Logical OR): Carries out OR operation bit-by-bit same as AND

Eg:- OR AX, 0098H
OR AX, BX
OR BX, [5000H]

NOT (logical Invert):- The NOT instruction inverts the content of operand register or memory location bit by bit.

Eg:
```
NOT   AX
NOT   [5000H]
```

XOR (logical Exclusive OR) :- XOR operation to be carried out like same way as OR & AND operation.

Eg:
```
XOR   AX, 0089H
XOR   AX, BX
XOR   AX, [5000H].
```

TEST (logical compare operation):- If all the bits of operands are equal then the result is 1 otherwise it is '0'.

Eg:-
```
TEST   AX, BX
TEST   [0500], 06H
```

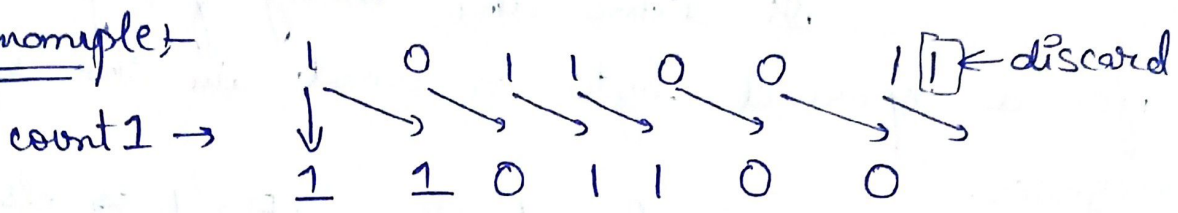
SHL/SAL (shift logical / Arithmetic left) :- This instructions shifts the operand word or byte bit by bit to the left and insert zeros at the LSB either by 1 or described by CL.

SHR/ Shift logical right :- Shifting the bit by 1 position in the register or memory. Or the no of counts specified by CL

SAR (Shift arithmetic Right):- Shifting the bit by 1 position towards right or the count as specified by CL But here the MSB position filled with previous

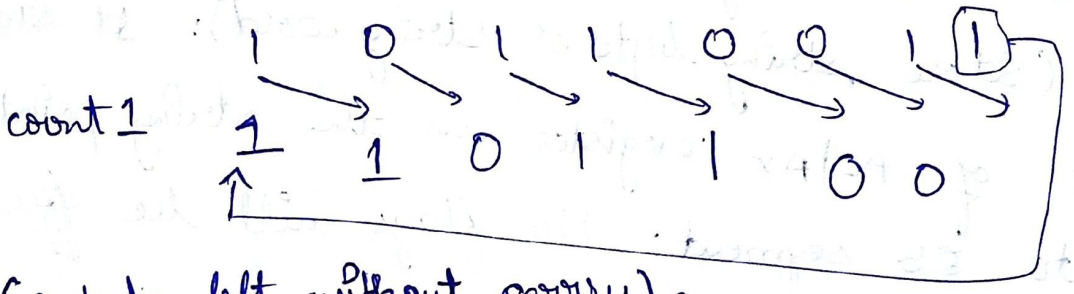
MSB value only to preserve the sign of the binary value.

For example:-



ROR (Rotate right without carry):- This instruction rotates the content of destination operand to the right by a bit of the count specified by CL. The LSB bit is pushed the MSB position.

Eg:-



ROL (Rotate left without carry):-

ROR (Rotate right without carry).

RCR (Rotate right with carry)

RCL (Rotate left with carry)

String manipulation Instructions:-

REP (Repeat instruction prefix) :- It is used to repeat the instruction no of times provided by CX register.

MOVSB/MOVSW:- (move string byte or string word)

used to move the string byte or word from source to destination.

Eg: Rep MOVSB

CMPS (Compare string byte or string word) :- Compare two strings of byte length or word length.

SCAS:- (Scan string byte or string word)
It scans the string of byte or word for a operand byte or word in the specified register.
If it is found then ZF = 1 & else ZF = 0.

LODS:- (Load string byte or string word):- It loads the AL/AX register by the content of string pointed by SI to the segment.

STOS:- (Store string byte or string word):- It stores the content of AL/AX register in the string pointed by DI to ES segment. No flags will be effected.

Branch Instructions:-

I Unconditional Branch Instructions

(i) Call: (Unconditional Call):- This is used to call a subroutine from a main program.

(ii) RET (Return from the procedure):-
(i) Return from a segment
(ii) Return from a procedure.

(iii) INT N:- Interrupt type N:-
There are 256 interrupts and here 'N' indicates one of the interrupts among 256

Eg:- INT 20H
INT 21H.

INTO (Interrupt on overflow) :- This command is executed when the overflow flag (OF) is set.

JMP (Unconditional jump) :- It transfer the control of execution from present to the specified address.

IRET (Return from ISR) :- Returning back to the previous execution once the interrupt service routine has been executed.

LOOP (Loop unconditionally) :- It executes the part of a program from the label or the address specified in the instruction

Eg:-
```
Label : MOV  AX, 1000H
        OR.  BX, AX
        AND  DX, AX
        Loop Label.
```

11) Conditional branch instructions :-

		Despription
(i)	JZ/JE	– Transfer execution control to address if ZF=1
(ii)	JNZ/JNE	– " " " " " if ZF=0
(iii)	JS	– " " " " " if SF=1
(iv)	JNS	– Jump if SF=0
(v)	JO	– Jump if OF=1
(vi)	JNO	– Jump if OF=0.
(vii)	JP/JPE	– Jump if PF=1
viii)	JNP	– Jump if PF=0.
ix)	JB/JNAE/JC	– Jump if CF=1
x)	JNB/JNC	– Jump if CF=0.

xi) JBE / JNA → Jump if ~~not~~ below or equal or Jump if not above. ie; if CF = 1 or ZF = 1

xii) JNBE / JA → Jump if not below or equal / Jump if above ; if CF = 0 or ZF = 0.

xiii) JL / JNGE → Jump if less / Jump if not greater or equal ; if neither SF = 1 nor OF = 1

xiv) JNL / JGE → Jump if not less / Jump if greater or equal ; if neither SF = 0 or OF = 0

xv) JLE / JNC → Jump if less or equal / Jump if no carry.

xvi) JNLE / JE → Jump if not less or equal / Jump if equal.

Flag manipulation & processor control instructions

(i) Flag manipulation Instructions :-

a) CLC — Clear carry flag.
b) CMC — Complement carry flag.
c) STC — Set carry flag.
d) CLD — Clear direction flag.
e) STD — Set direction flag.
f) CLI — Clear interrupt flag.
g) STI — Set interrupt flag.

(ii) Machine control instructions

a) Wait — wait for test pin to go low
b) HLT — Halt the processor
c) NOP — No operation
d) ESC — Escape
e) LOCK — Bus lock.