Chapter 6.1: Flags-control instructions: Monitors/controls state of instruction execution.

- **LAHF Load AH from flags (AH) ← (Flags)**
- **SAHF Store AH into flags (Flags) ← (AH)**
  Flags affected: SF, ZF, AF, PF, CF
- **CLC Clear Carry Flag (CF) ← 0**
- **STC Set Carry Flag (CF) ← 1**
- **CLI Clear Interrupt Flag (IF) ← 0**
- **STI Set interrupt flag (IF)** ← 1

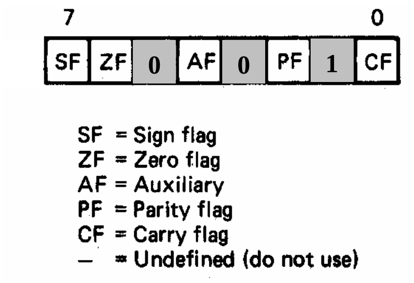


Figure 1

Figure 1 above shows the format of flag digits in AH register.

So when all flags are **set** ('1') ➔AH=D7$_H$ which is equivalent to having AH=FF$_H$.

But when all flags are **reset** ('1') ➔AH=02$_H$ which is equivalent to having AH=00$_H$.

Example 1: Write a program to complement the status of flags bits: SF, ZF, AF, PF, CF.

Solution 1:
LAHF ; *this will load the flag bits into AH register* (*Note: no operand needed*)
NOT AH ; *this will invert the status of flag bits*
SAHF ; *this will store back the complemented status of flag bits into Flag reg.*

Example 2: Write a program to compliment only the carry flag. ➔ **CMC**

6.1: Compare (CMP) Instruction: compares data and sets FLAGS-bits accordingly

Mnemonic	Meaning	Format	Operation	Flags Affected
CMP	Compare	CMP D,S	(D) – (S) is used in setting or resetting the flags	CF, AF, OF, PF, SF, ZF

CMP Ins subtracts (S) from (D) operand, but is only interested in how the result is affecting the flag-bits.

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

As 2's-complement affects the CF, so use it with caution in 'CMP' instruction

For ➔ CMP AL,BL => Al or BL value do not change after instruction is executed

Al = 99$_H$ = 10011001$_B$
(-) Bl = 1B$_H$ = 00011011$_B$
01111110$_B$

where final-result is not important but how Flags are affected is important ➔ such as, ZF=NZ as AL≠BL and **CF=NC** as AL>BL and also AF=AC and PF=PE

For ➔ CMP BL,AL

Bl = 1B$_H$ = 00011011$_B$
(-) Al = 99$_H$ = 10011001$_B$
10000010$_B$

where final-result is not important but how Flags are affected is important ➔ such as, ZF=NZ as AL≠BL and **CF=CY** as AL>BL and also AF=NA and PF=PE

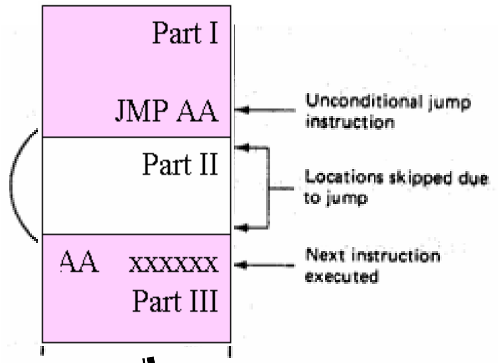
Write a program to compare AL and BL register contents and if they are not equal decrements the contents of AL and compares them again.

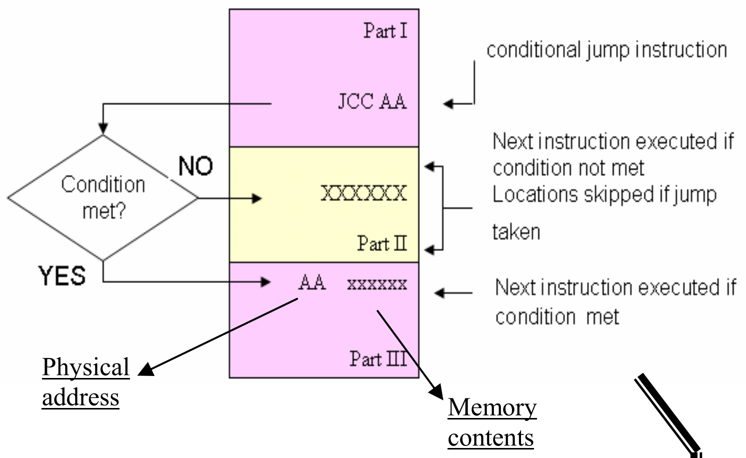
Chapter 6.3: Control flow and jump instructions:

- Since CS:IP points to the instruction to be executed next, JUMP instruction changes the contents of these registers to point to another instruction (*location we need to jump*)

- For *Unconditional* Jump, if only the IP is changed ➔ Intrasegment jump (or jump within same segment) **BUT** if CS:IP is changed ➔ Intersegment jump

- 2 Jump operations allowed by 8088; (a) **Unconditional** *and* (b) **Conditional** Jumps:





Mnemonic	Meaning	Format	Operation	Flags Affected
JMP	Unconditional jump	JMP Operand	Jump is initiated to the address specified by the operand	None

Example: JMP BX
and JMP [BX]

(a)

Operand
Short-label
Near label
Far-label
Memptr16
Regptr16
Memptr32

For intersegment jump operation { Short-label, Near label

- Short-label ➔ can jump -126_D to $+129_D$ bytes from location
- Near label ➔ can jump -32766_D to $+32769_D$ bytes from location
- Far-label ➔ For Inter-segment Jump operation

Conditional Jump:

Mnemonic	Description	Flag/Registers
JZ	Jump if ZERO	ZF=1
JE	Jump if EQUAL	ZF=1
JNZ	Jump if NOT ZERO	ZF=0
JNE	Jump if NOT EQUAL	ZF=0
JC	Jump if CARRY	CF=1
JNC	Jump if NO CARRY	CF=0
JCXZ	Jump if CX=0	CX=1
JECXZ	Jump if ECX=0	ECX=0
JP	Jump if PARITY EVEN	PF=1
JNP	Jump if PARITY ODD	PF=0

Example:

CMP AX,CX

JNZ BX

or/and JNE BX

or/and JA BX

Physical address to jump is the content of BX

That means the content of BX is copied to IP and the program points to the new P.A.= CS:IP location

Flags are based on unsigned numbers comparison:

Mnemonic	Description	Flag/Registers
JA	Jump if above op 1>op2	CF=0 and ZF=0
JNBE	Jump if not below or equal op1 not <= op2	CF=0 and ZF=0
JAE	Jump if above or equal op1>=op2	CF=0
JNB	Jump if not below op1 not <op2	CF=0
JB	Jump if below op1<op2	CF=1
JNAE	Jump if not above nor equal op1<op2	CF=1
JBE	Jump if below or equal Op1<=op2	CX=1 or ZF=1
JNA	Jump if not above Op1<=op2	CF=1 or ZF=1

Flags are based on signed numbers comparison:

JG	Jump if GREATER op1>op2	SF=OF and ZF=0
JNLE	Jump if NOT LESS THAN or equal op1>op2	SF=OF and ZF=0
JGE	Jump if GREATER THAN or equal op1>=op2	SF=OF
JNL	Jump if not LESS THAN op1>=op2	SF=OF
JL	Jump if LESS THAN op1<op2	SF<>OF
JNGE	Jump if not GREATER THAN nor equal op1<op2	SF<>OF
JLE	Jump if LESS THAN or equal Op1<=op2	ZF=1 or SF<>OF
JNG	Jump if not GREATER THAN op1<=op2 Op1<=op2	ZF=1 or SF<>OF

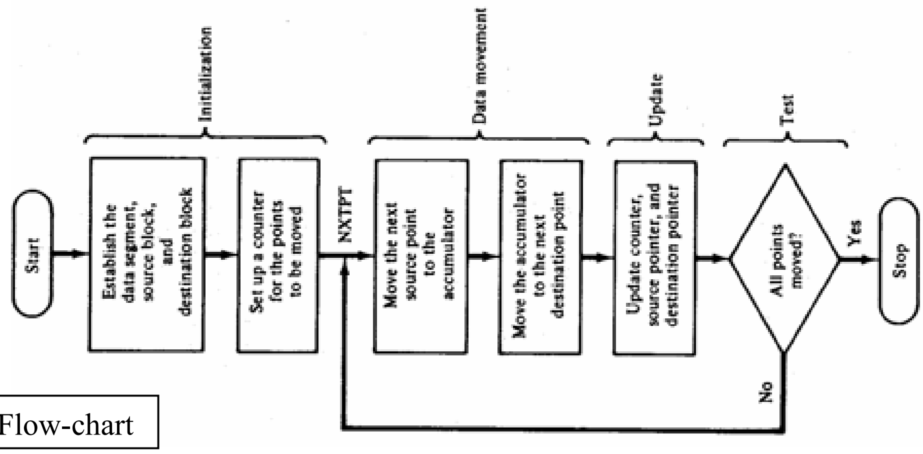
Examples of conditional jump commands:

```
        CMP   AX, BX
        JE    EQUAL
        ---   ---          ; Next instruction if (AX) ≠ (BX)
         .
         .
EQUAL:  ---   ---          ; Next instruction if (AX) = (BX)
         .
         .
        ---   ---
```

```
           AND   AL, 04H
           JNZ   BIT2_ONE
           ---   ---          ; Next instruction if B2 of AL = 0
            .
            .
           ---   ---
BIT2_ONE:  ---   ---          ; Next instruction if B2 of AL = 1
            .
            .
           ---   ---
```

```
           MOV   CL,03H
           SHR   AL, CL
           JC    BIT2_ONE
           ---   ---          ; Next instruction if B2 of AL = 0
            .
            .
           ---   ---
BIT2_ONE:  ---   ---          ; Next instruction if B2 of AL = 1
            .
            .
           ---   ---
```

```
        MOV   AX, DATASEGADDR
        MOV   DS, AX
        MOV   SI, BLK1ADDR
        MOV   DI, BLK2ADDR
        MOV   CX, N
NXTPT:  MOV   AH, [SI]
        MOV   [DI], AH
        INC   SI
        INC   DI
        DEC   CX
        JNZ   NXTPT
```

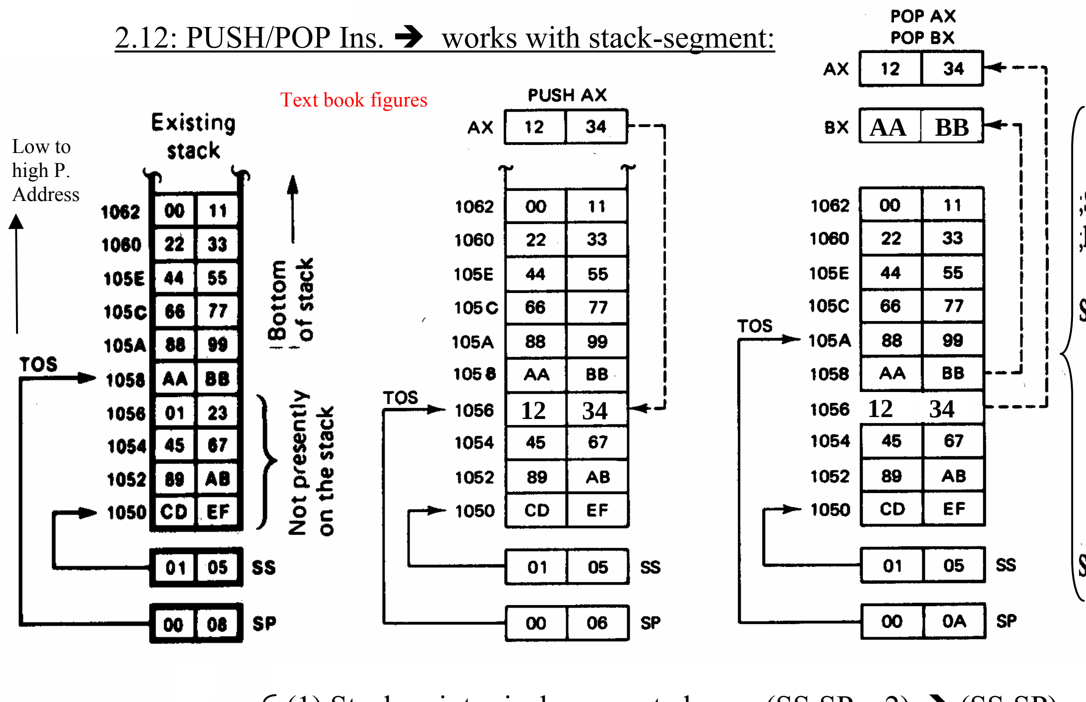


Flow-chart

DOS functions (20_H to $3F_H$): Commonly used DOS interrupts ➔ **INT 21_H**

- with AL=01_H ➔ data requested to be inputted from the keyboard with echo is stored in AL register
- with AL=07_H ➔ data requested to be inputted from the keyboard without echo is stored in AL register
- with AL=02_H ➔ ASCII code of the data stored in DL register is displayed in the monitor
- with AL=09_H ➔ Displays string of characters (*stored using* 'DB' *&terminated by* '$') in the monitor
- **WITH AX=$4C00_H$ ➔ Used to terminate program and return control to DOS or parent process**

2.12: PUSH/POP Ins. → works with stack-segment:



Example:

```
;Subroutine:   SQUARE
;Description:  (BX) = square of (BL)

SQUARE PROC   NEAR
       PUSH AX
       MOV  AL,BL
       IMUL BL
       MOV  BX,AX
       POP  AX
       RET
SQUARE ENDP
```

'PUSH S'
- (1) Stack pointer is decremented *or* (SS:SP – 2) → $(SS{:}SP)_{new}$
- (2) Source register contents are loaded in stack segment *or* (S) → [SS:SP]

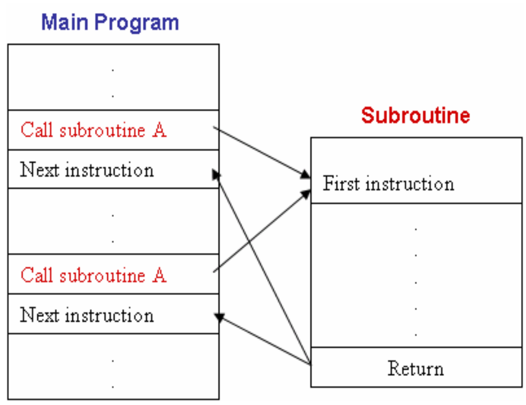
'POP D'
- (1) Stack seg. content is loaded into Destination register *or* [SS:SP] → (D)
- (2) Stack pointer is incremented *or* (SS:SP + 2) → $(SS{:}SP)_{new}$

HW: Solve and pass the problem in the WebCT regarding "Push-Pop and Jump"

Chapter 6.4: Subroutine-handling instructions:

Mnemonic	Meaning	Format	Operation	Flags
CALL	Subroutine call	CALL operand	Execution continuous from the address of the subroutine specified by operand. Information required to return back to the main program such as IP and CS are saved on the stack.	None

Mnemonic	Meaning	Format	Operation	Flags
RET	Return	RET	Return to the main program by restoring IP (and CS for far-proc).	None



Subroutines are special segment of program that can be called for execution from any point of the **main-program**. Once called and executed, the main program continues to execute from the point where the subroutine is called from. An Assembly Language subroutine is also called a **Procedure**.

Once executed, CALL Instruction; **1st** PUSH next IP of main-program; **2nd** Loads IP with operand address

Once executed, RET Instruction; Uses POP instruction to loads the $(\text{pushed-return address})_{\text{from stack}}$ into IP

Chapter 6.5: LOOP handling instruction: By default works with CX register

Mnemonic	Meaning	Format	Operation
LOOP	Loop	LOOP Short-label	(CX) (CX)-1 Jump is initiated to location definition by short-label if (CX)≠0; otherwise, execute next sequential instruction
LOOPE/LOOPZ	Loop while equal/loop while zero	LOOPE/LOOPZ short-label	(CX) (CX)-1 Jump to location definition by short-label if (CX)≠0 and ZF=1; otherwise, execute next sequential instruction
LOOPNE/LOOPNZ	Loop while not equal/loop while not zero	LOOPNE/LOOPNZ short-label	(CX) (CX)-1 Jump to location defined by short-label if (CX)≠0 and ZF=0; otherwise, execute next sequential instruction

Example: DEC CX
JNZ *** } LOOP ***

7.2 DB and DW ***directive*** statements ➔ Instructions to the Assembler & Not assembled

- 'DB' (or Define Byte) Instruction: Initialize byte size variables or locations.
- 'DW' (or Define Word) Instruction: Initialize word size variables or locations.

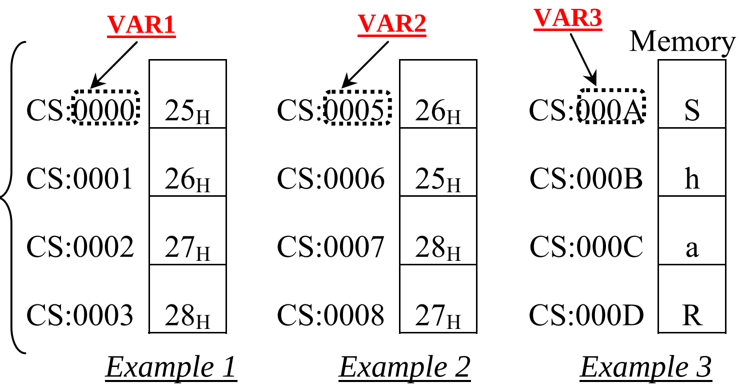
Examples for TASM program: (1) VAR1 DB 25_H, 26_H, 27_H, 28_H

(2) VAR2 DW 2526_H, 2728_H

(3) VAR3 DB "ShaR"

(4) VAR4 DB 10 DUP (0)
Initialize 10 location to 0_H

- 'EQU' Instruction
Assign permanent value
- '=' Instruction
Assigned value can be redefined



Chapter 7: Assembly Language Program Development

- To enter, assemble and execute the programs Using Turbo Assembler Program (TASM)

(a) EDIT Prog1.asm *{to write the program}*
(b) TASM Prog1 *{to assemble the program}*
(c) TLINK Prog1 *{to link the program}*
(d) TD Prog1 *{to execute the program}*

- *Remember another Assembler often used is called MASM (Microsoft assembler)*

….. SEE HAND-OUT for evolution of character-conversion-program….

```
TITLE "Use Subroutines to Store, Convert (small to capital) & restore Inputted letters"
.MODEL SMALL        ; Program fits with in 64 KB of memory
.STACK 032H         ; Program reserves 50 Bytes as stack segment
.DATA
        VAR1   DB   20  DUP(0)      } 'DB' is define byte, which allocates 20
                                      memory locations to VAR1 for data storage
.CODE
        ORG    00H                  } 'The main program area for codes starts

        MOV    AX, @DATA
        MOV    DS, AX
        LEA    DI,VAR1
        CALL   INPUT
        LEA    SI,VAR1
        CALL   CONVERT
        LEA    SI,VAR1     ≈ LEA SI,[VAR1]
        CALL   OUTPUT
        CALL   EXIT_TO_DOS
```

The main assembly language program area. Four subroutines are called from here;

(1) INPUT subroutine
(2) CONVERT subroutine
(3) OUTPUT subroutine
(4) EXIT_TO_DOS subroutine.

The advantage of using subroutines becomes clear when the statements with in the subroutines are to be called more than onces.

```
      INPUT  PROC    NEAR
     labelIN: MOV    AH,1
              INT    021H
              MOV    [DI],AL
              INC    DI
              CMP    AL,0DH
              JNZ    labelIN
              RET
      INPUT   ENDP
```

In this INPUT subroutine or procedure;

(1) Inputted characters from the keyboard are stored in the reserved memory locations of VAR1.
(2) The program requires the user to press 'ENTER key' after the last inputted character. That's why, 'OD_H' (equivalent to ASCII character for '*ENTER key*') is used to recognize the end of inputted characters.

```
   CONVERT PROC      NEAR
     labelC2: CMP    byte ptr [SI],061H
              JB     labelC1
              CMP    byte ptr [SI],07AH
              JA     labelC1
              SUB    byte ptr [SI],020H
     labelC1: INC    SI
              CMP    byte ptr [SI],0DH
              JNZ    labelC2
              RET
   CONVERT ENDP
```

In this CONVERT subroutine or procedure;

(1) Stored inputted characters are compared with the lower limit of '61_H' (ASCII 'a') and the upper limit of '$7A_H$' (ASCII 'z') of the small letters
(2) If any stored character satisfies above limit of small letters, then 20_H is subtracted from its equivalent hex value to convert it to capital letter.
(3) This process is repeated until 'OD' is found.

```
    OUTPUT  PROC     NEAR
    labelOUT: MOV    DL,byte ptr [SI]
              MOV    AH,2
              INT    021H
              INC    SI
              CMP    DL,0DH
              JNZ    labelOUT
              RET
    OUTPUT  ENDP
```

In this OUTPUT subroutine or procedure;

(1) The resulted capital letters, which are converted and stored in the same memory locations of VAR1, are then displayed in the monitor
(2) The DOS subroutine of 'INT 21_H' with AH=2_H is used for displaying individual characters. (For inputting characters, INT 21' with AH=1 is used.

```
 EXIT_TO_DOS  PROC   NEAR
              MOV    AX,4C00H
              INT    021H
 EXIT_TO_DOS  ENDP
END
```

In this EXIT_TO_DOS subroutine or procedure;

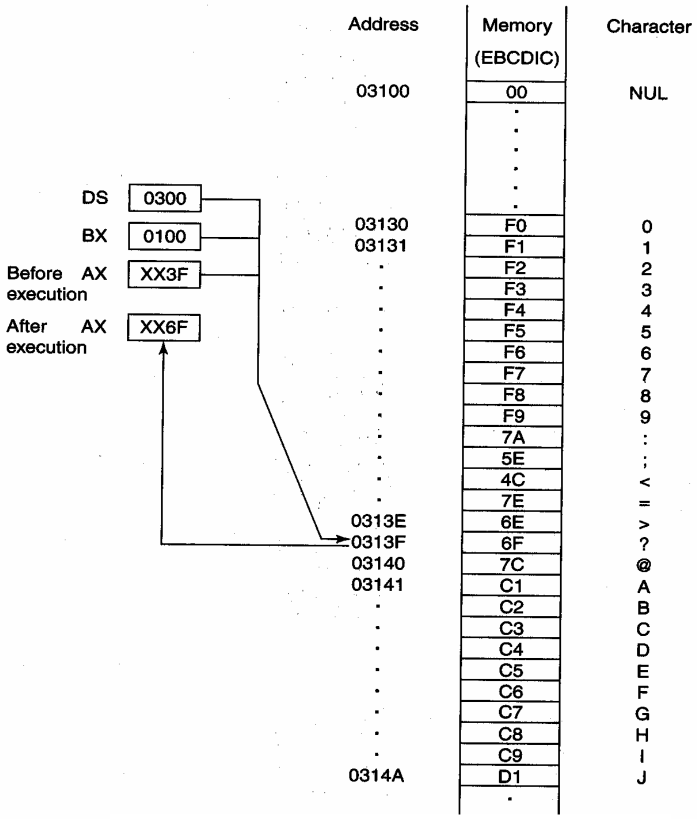
(1) MSDOS subroutine of 'INT 21_H' with AX=$4C00_H$ is also used for normal termination to DOS prompt after the program is executed.
(3) This is essential, if the assembled program is to be executed directly from MSDOS prompt; ***c:***\>

EE 390 : Digital System Engineering
Handout 14 by Dr Sheikh Sharif Iqbal

5.1: XLAT instruction: is used for Translation using predefined look-up tables.

- By default uses 'AL' and 'BX' registers of the CPU.
- If we want to access numbers stored using 'DB' in 'VAR' location, 'BX' is used to point to the 'VAR' and 'AL' points 'DATA' (*remember the count of **AL** always starts from **zero***)

```
TITLE "XLAT"
.MODEL SMALL
.STACK 32
.DATA
    VAR   DB  "1MISEIOHN
                 TO_IPAOTTAS"
    VAR1  DB  2H,3H,4H,5H,6H,7H,
              8H,9H,AH,BH,CH,DH,11H,23H
    VAR 3  EQU  10H
.CODE
      MOV   AX,@DATA
      MOV   DS,AX
      XOR   AH,AH
      MOV   AL,VAR3
      LEA   BX,VAR
      XLAT  ➔ AL= ----
      MOV   AX,4C00H
      INT   21H
END
```



6.6: String-handling instruction: STRING means series/block of data words (or bytes) that reside/sorted in consecutive memory locations.

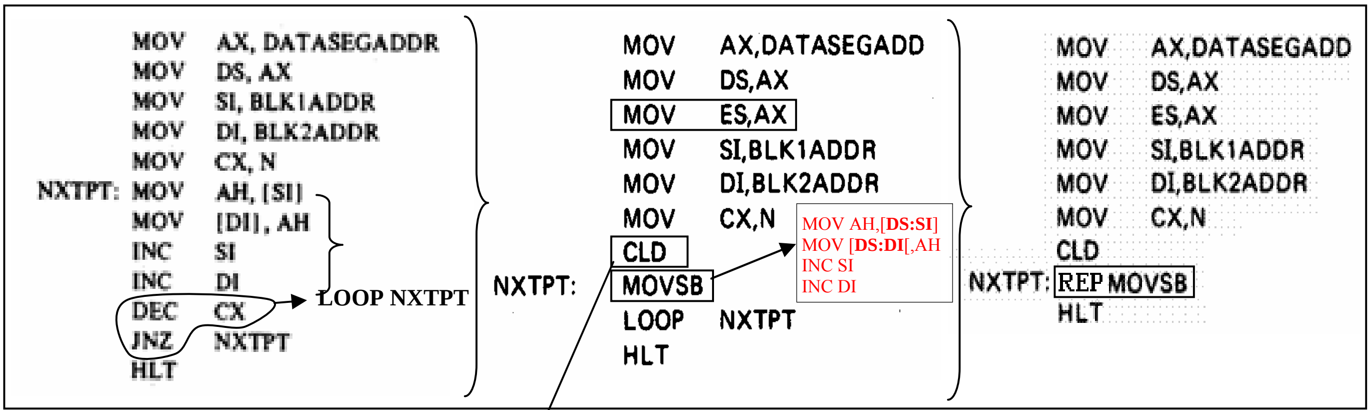
Mnemo..	Meaning	Format	Operation	Flags Affected
MOVS	Move string	MOVSB/ MOVSW	((ES))0+(DI) (DS)0+(SI) (SI) (SI)±1 or 2 (DI) (DI)±1 or 2	None
CMPS	Compare string	CMPSB/ CMPSW	Set flags as per ((DS))0+(SI) - (ES)0+(DI) [ES:DI] (SI) (SI)±1 or 2 (DI) (DI)±1 or 2	CF,PF,AF,ZF,SF,OF
SCAS	Scan string	SCASB/ SCASW	Set flags as per (AL or AX) - (ES)0+(DI) (DI) (DI)±1 or 2	CF,PF,AF,ZF,SF,OF
LODS	load string	LODSB/ LODSW	(AL or AX) (DS)0+(SI) (SI) (SI)±1 or 2	None
STOS	Store string	STOSB/ STOSB	(ES)0+(DI) (AL or AX)±1 or 2 (DI) (DI)±1 or 2	None

- See examples in **figures 6-33, 6-34 and 6-35** *in the book*. For **CLD** Ins. ➔ Figure 6-38

'REP prefixs' ➔ works with 'MOVS' and 'STOS' ➔ repeats while not end or string, CX ≠ 0

Prefix	Used with	Meaning
REP	MOVS STOS	Repeat while not end of string CX≠0
REPE/REPZ	CMPS SCAS	Repeat while not end of string and strings are equal CX≠0 and and ZF=1
REPNE/REPNZ	CMPS SCAS	Repeat while not end of string and strings are not equal CX≠0 and ZF=0

Modified example of Data block program using "REP" and "MOVSB" instruction:



CLD Ins. ➔ "clear DF" *or* DF='0' ➔ means auto-increment mode *or* 'SI' and/or 'DI' are auto-incremented by '1' for byte-data and '2' for word-data.

Example 2: write a program to copy a block of 32 consecutive bytes from the block Of memory locations starting at address MASTER in the current data segment (DS) to a block of locations starting at address COPY in the current extra Segment (ES)

Solution:

```
CLD
MOV AX, DATA_SEG
MOV DS, AX
MOV AX, EXTRA_SEG
MOV ES, AX
MOV CX, 20H
MOV SI, OFFSET MASTER
MOV DI, OFFSET COPY
REPZMOVSB
```

Exercise: Write a program, using "REPSTOSB" instruction, to store a data of '95_H' into memory locations starting from DS:$A000_H$ $A008_H$