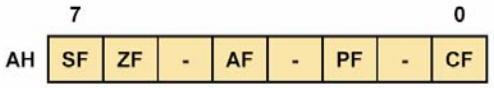
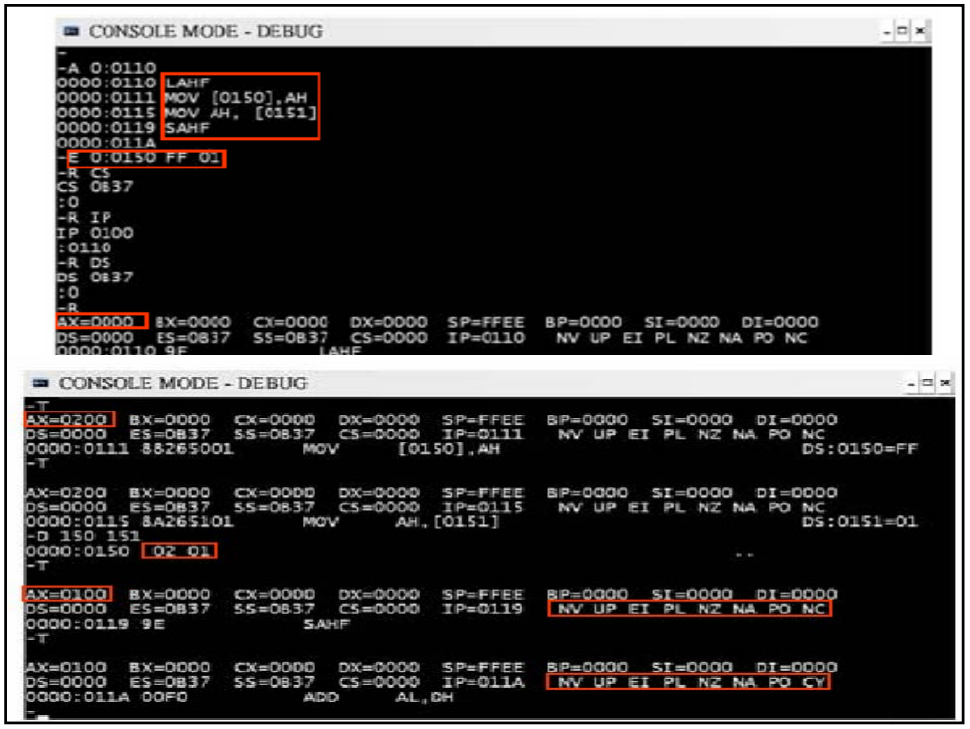
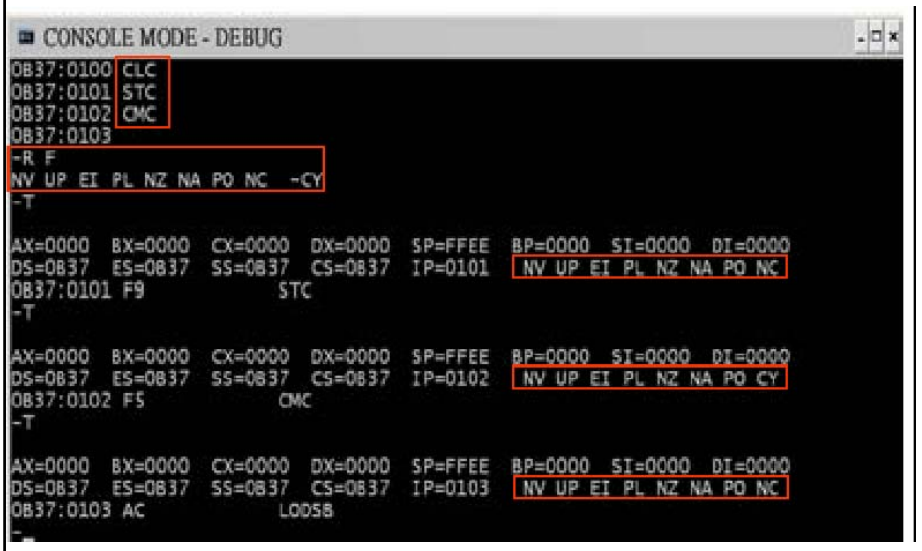
Flag-Control Instructions

Mnemonic	Meaning	Operation	Flags affected
LAHF	Load AH from flags	(AH)←(Flags)	None
SAHF	Store AH into flags	(Flags)←(AH)	SF,ZF,AF,PF,CF
CLC	Clear carry flag	(CF)←0	CF
STC	Set carry flag	(CF)←1	CF
CMC	Complement carry flag	(CF)←NOT (CF)	CF
CLI	Clear interrupt flag	(IF)←0	IF
STI	Set interrupt flag	(IF)←1	IF



SF = Sign flag
ZF = Zero flag
AF = Auxiliary
PF = Parity flag
CF = Carry flag
- = Undefined (do not use)





The Compare Instruction:

The compare operation enable us to determine the relationship between two numbers.

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

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

Compare operands	CF	ZF
Destination > source	0	0
Destination = source	0	1
Destination < source	1	0

EXAMPLE

Describe what happens to the status flags as the sequence of instructions that follows is executed.

```
MOV  AX, 1234H
MOV  BX, 0ABCDH
CMP  AX, BX
```

Solution:

$(AX) = 1234_{16} = 0001001000110100_2$

$(BX) = 1234_{16} = 0001001000110100_2$

$(AX) - (BX) = 0001001000110100_2 - 0001001000110100_2$

$= 0110011001100111_2$

Therefore, ZF = 0, SF = 0, OF = 0, PF = 0

CF = 1, AF = 1

EXAMPLE

Write an instruction sequence to save the current contents of the 8088's flags in the memory location at offset MEM1 of the current data segment and then reload the flags with the contents of the storage location at offset MEM2.

Solution:

```
LAHF              ; Load AH from flags
MOV  [MEM1], AH   ; Move content of AH to MEM1
MOV  AH, [MEM2]   ; Load AH from MEM2
SAHF              ; Store content of AH into flags
```

Write a program to find the highest among 5 grades and write it in DL

```
DATA    DB      51, 44, 99, 88, 80        ; 13h,2ch,63h,58h,50h
        MOV     CX,5                      ;set up loop counter
        MOV     BX, OFFSET DATA  ;BX points to GRADE data
        SUB     AL,AL                     ;AL holds highest grade found so far
AGAIN:  CMP     AL,[BX]                   ;compare next grade to highest
        JA      NEXT                      ;jump if AL still highest
        MOV     AL,[BX]                   ;else AL holds new highest
NEXT:   INC     BX                        ;point to next grade
        LOOP AGAIN                        ;continue search
        MOV DH, AL
```

- The PUSH and POP instructions

Mnemonic	Meaning	Format	Operation	Flags affected
PUSH	Push word onto stack	PUSH S	((SP))←(S) (SP) ←(SP)-2	None
POP	Pop word off stack	POP D	(D)←((SP)) (SP) ←(SP)+2	None

Operands (S or D)
Register
Seg-reg (CS illegal)
Memory

Allowed operands for PUSH and POP instruction

Control Flow and Jump Instructions

- Unconditional jump instruction
- Conditional jump instruction
- Branching structure – IF-THEN
- Loop program structure – REPEAT-UNTIL and WHILE-DO
- Applications using the loop and branch software structures

Control Flow and Jump Instructions

- Unconditional jump instruction

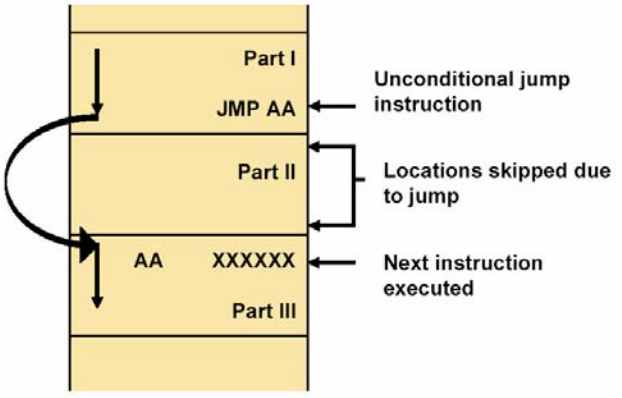
Mnemonic	Meaning	Format	Operation	Flags affected
JMP	Unconditional jump	JMP Operand	Jump is initiated to the address specified by the operand	None

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

Allowed operands for JMP instruction

Control Flow and Jump Instructions

- Unconditional and conditional jump



Unconditional jump program sequence

3 Control Flow and Jump Instructions

Conditional jump instruction

Mnemonic	Meaning	Condition
JNC	Not carry	CF=0
JNE	Not equal	ZF=0
JNG	Not greater	((SF xor OF) or ZF)=1
JNGE	Not greater nor equal	(SF xor OF)=1
JNL	Not less	SF=OF
JNLE	Not less or nor equal	ZF=0 and SF=OF
JNO	Not overflow	OF=0
JNP	Not parity	PF=0
JNS	Not sign	SF=0
JNZ	Not zero	ZF=0
JO	Overflow	OF=1
JP	Parity	PF=1
JPE	Parity even	PF=1
JPO	Parity odd	PF=0
JS	Sign	SF=1
JZ	Zero	ZF=1

Conditional jump instruction

Mnemonic	Meaning	Condition
JA	Above	CF=0 and ZF=0
JAE	Above or equal	CF=0
JB	Below	CF=1
JBE	Below or equal	CF=1 or ZF=1
JC	Carry	CF=1
JCXZ	CX register is zero	(CF or ZF)=0
JE	Equal	ZF=1
JG	Greater	ZF=0 and SF=OF
JGE	Greater or equal	SF=OF
JL	Less	(SF xor OF)=1
JLE	Less or equal	((SF xor OF) or ZF)=1
JNA	Not above	CF=1 or ZF=1
JNAE	Not above nor equal	CF=1
JNB	Not below	CF=0
JNBE	Not below nor equal	CF=0 and ZF=0

Type of conditional jump instructions

- Conditional jump instruction

Mnemonic	Meaning	Format	Operation	Flags affected
Jcc	Conditional jump	Jcc Operand	If the specified condition cc is true the jump to the address specified by the operand is initiated; otherwise the next instruction is executed	None

Branch program structure – IF-THEN

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

IF-THEN branch program structure using a flag-condition test

Branch program structure – IF-THEN

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

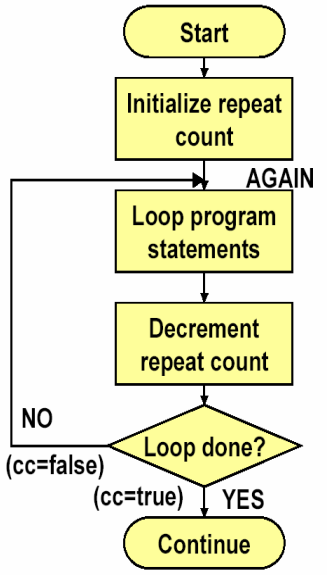
IF-THEN branch program structure using register-bit test

- Branch program structure – IF-THEN

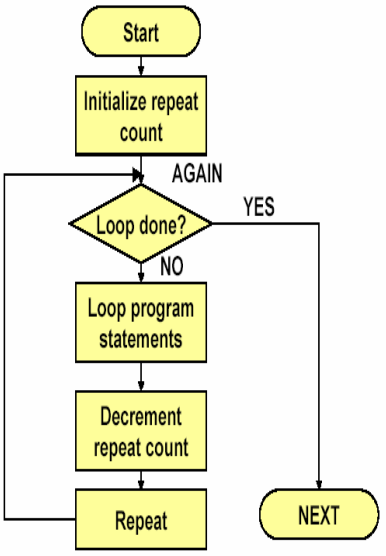
```
        MOV  CL, 03H
        SHR  AL, CL
        JC   BIT2_ONE
        ---  ---        ; Next instruction if B2 of AL=0
             .
             .
BIT2_ONE: ---  ---      ; Next instruction if B2 of AL=1
             .
             .
        ---  ---
```

IF-THEN branch program structure using an alternative register-bit test

Loop program structures – Loop program structures – WHILTE-DO



REPEAT-UNTIL program seq



WHILE-DO program sequence

- Loop program structures – WHILTE-DO

```
        MOV  CL, COUNT ; Set loop repeat count
AGAIN:  JZ   NEXT      ; Loop is complete is CL=0 (ZF=1)
        ---  ---       ; First instruction of loop
        ---  ---       ; Second instruction of loop
             .
             .
             .
        ---  ---       ; nth instruction of loop
        DEC  CL        ; Decrement repeat count by 1
        JMP  AGAIN     ; Repeat from AGAIN
NEXT:   ---  ---       ; First instruction executed after the
                       ; loop is complete
```

Typical WHILE-DO instruction sequence

Loop program structures – REPEAT-UNTIL

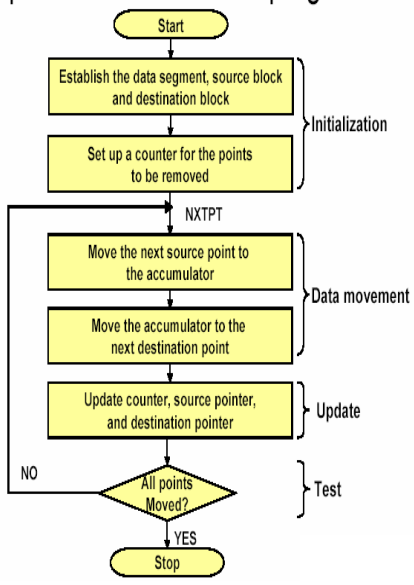
```
        MOV  CL, COUNT ; Set loop repeat count
AGAIN:  ---    ---     ; First instruction of loop
        ---    ---     ; Second instruction of loop .
                .
                .
                .
        ---    ---     ; nth instruction of loop
        DEC  CL        ; Decrement repeat count by 1
        JNZ  AGAIN     ; Repeat from AGAIN if (CL) ≠00H and (ZF)=0
        ---    ---     ; First instruction executed after the loop is
                       ;complete, (CL) =00H and (ZF)=1
```

Typical REPEAT-UNTIL instruction sequence

Example – The block-move program

```
        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
        HLT
```

Example – The block-move program



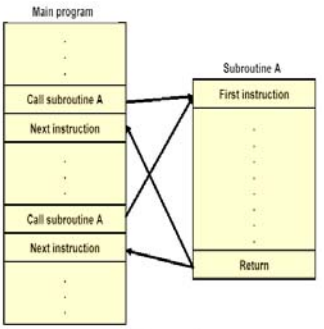
6.4 Subroutines and Subroutine-Handling Instructions

- A subroutine is a special program that can be called for execution from any point in a program.
- A subroutine is also known as a procedure.
- A return instruction must be included at the end of the subroutine to initiate the return sequence to the main program environment.
- CALL and RET instructions
- PUSH and POP instructions

- The CALL instruction

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

Operands
Near-proc
Far-proc
Memptr16
Regptr16
Memptr32



Subroutine concept

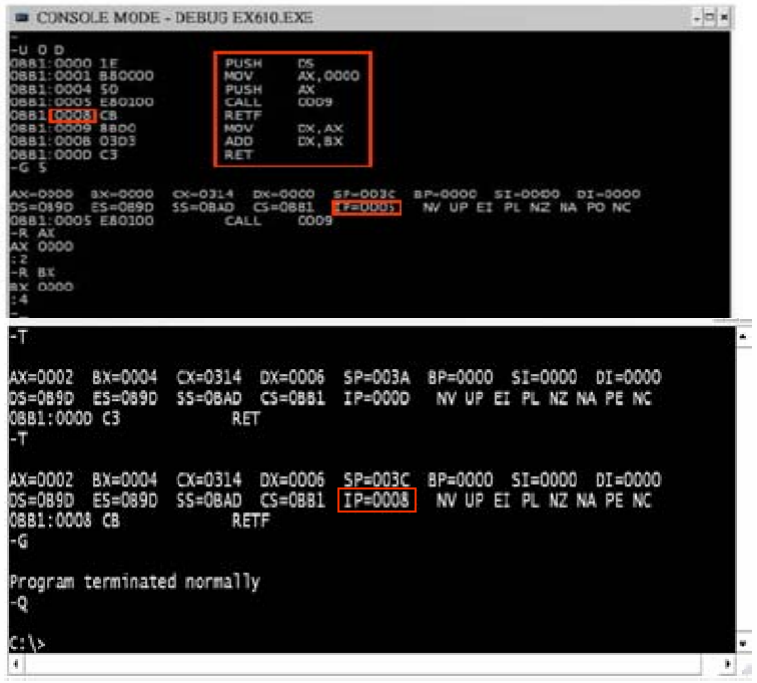
- The RET instruction

Mnemonic	Meaning	Format	Operation	Flags affected
RET	Return	RET or RET Operand	Return to the main program by restoring IP (and CS for far-proc). If Operand is present, it is added to the contents of SP	None

Operands
None
Disp16

Allowed operands for RET instruction

EXAMPLE



- The LOOP instructions

```
        MOV  CX, COUNT      ; Load count for the number of repeats
NEXT:     .
          .
          .                 ; Body of routine that is repeated
          .
          .
        LOOP NEXT           ; Loop back to label NEXT if count not zero
```

Typical loop routine structure

- Example – The block-move program

```
                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
                LOOP NXTPT
                HLT
```

- **Loop Instruction (LOOP)**
  - Used to loop a set of instructions **CX** times
  - As with the REP instruction, **CX** is decremented every time the LOOP instruction is executed and once **CX** = 0 the loop is exited and the program continues
  - If **CX** needs to be used in the set of instructions looped then it should be saved onto the stack first:

```
        MOV CX, 10
Here:   PUSH CX
        MOV CX, 2000h
        CALL Output_To_Screen
        POP CX
        LOOP Here                (Loop 10 times)
```

- The LOOP instructions

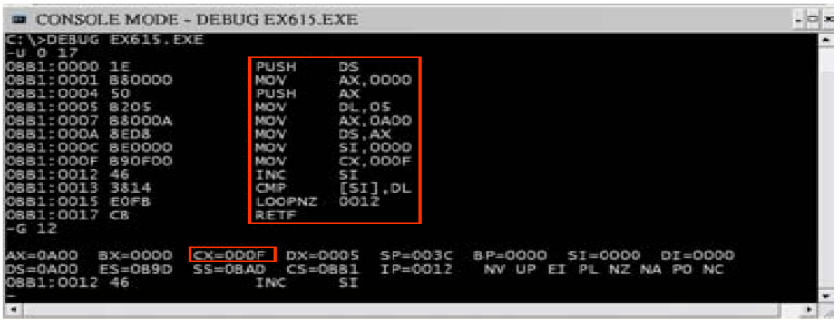
Mnemonic	Meaning	Format	Operation
LOOP	Loop	LOOP Short-label	(CX) ←(CX)-1 Jump is initiated to location defined by short-label if (CX)≠0; otherwise, execute next sequential instruction
LOOPE LOOPZ	Loop while equal Loop while zero	LOOPE/LOOPZ Short-label	(CX) ←(CX)-1 Jump to location defined by short-label 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

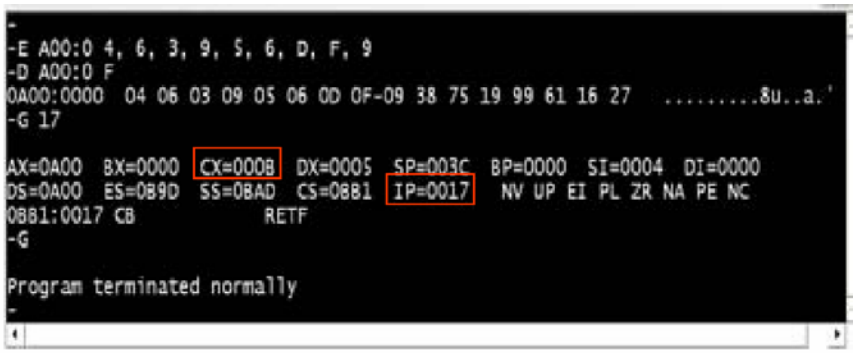
EXAMPLE

Given the following sequence of instructions, explain what happens as they are executed.

```
        MOV   DL, 05
        MOV   AX, 0A00H
        MOV   DS, AX
        MOV   SI, 0
        MOV   CX, 0FH
AGAIN:  INC   SI
        CMP   [SI], DL
        LOOPNE AGAIN
```

EXAMPLE





Subroutines and Subroutine-Handling Instructions

- The PUSH and POP instructions

To save registers and parameters on the stack	PUSH XX PUSH YY PUSH ZZ
Main body of the subroutine	. . . . .
To restore registers and Parameters from the stack	POP ZZ POP YY POP XX
Return to main program	RET

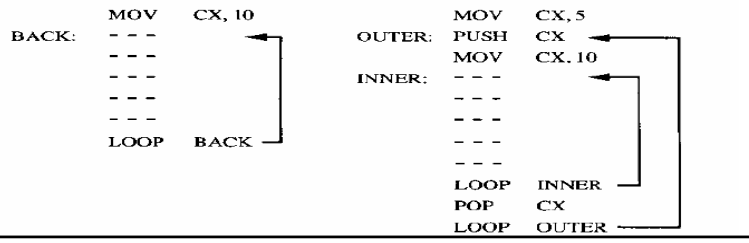
Structure of a subroutine

EXAMPLE

Write a procedure named SQUARE that squares the contents of BL and places the result in BX

Solution:

```
;Subroutine:   SQUARE
;Description:  (BX)=square of (BL)
SQUARE  PROC   NEAR
        PUSH  AX       ; Save the register to be used
        MOV   AX, BX ; Place the number in AL
        IMUL  BL       ; Multiply with itself
        MOV   BX, AX ; Save the result
        POP   AX       ; Restore the register used
        RET
SQUARE ENDP
```



String and String-Handling Instructions

- The basic string instructions

Mnemonic	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)) (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 STOSW	((ES)0+(DI))← (AL or AX) ±1 or 2 (DI) ←(DI)±1 or 2	None

String and String-Handling Instructions

Autoindexing for string instruction – CLD and STD instructions

- Move string – MOVSB, MOVSW

  Example – The block-move program using the move-string instruction

```
        MOV  AX, DATASEGADDR
        MOV  DS, AX
        MOV  ES, AX
        MOV  SI, BLK1ADDR
        MOV  DI, BLK2ADDR
        MOV  CX, N
        CLD
NXTPT:  MOVSB
        LOOP NXTPT
        HLT
```

Mnemonic	Meaning	Format	Operation	Flags affected
CLD	Clear DF	CLD	(DF)←0	DF
STD	Set DF	STD	(DF)←1	DF

- Compare string and scan string –

  CMPSB/CMPSW, SCASB/SCASW

  Example – Block scan operation using the SCASB instruction

```
        MOV  AX, DATASEGADDR
        MOV  DS, AX
        MOV  ES, AX
        MOV  AL, 05
        MOV  DI, 0A000H
        MOV  CX, 0FH
        CLD
AGAIN:  SCASB
        LOOPNE  AGAIN
NEXT:
```

- Load and store string –

LODSB/LODSW, STOSB/STOSW

Example – Initializing a block of memory with a store string instruction

```
        MOV  AX, 0
        MOV  DS, AX
        MOV  ES, AX
        MOV  AL, 05
        MOV  DI, 0A000H
        MOV  CX, 0FH
        CLD
AGAIN:  STOSB
        LOOP  AGAIN
```

- REP string – REP (repeat prefixes)

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 ZF=1
REPNE/REPNZ	CMPS SCAS	Repeat while not end of string and strings are not equal CX≠0 and ZF=0

- REP string – REP (repeat prefixes)

Example – Initializing a block of memory by repeating the STOSB instruction

```
        MOV  AX, 0
        MOV  DS, AX
        MOV  ES, AX
        MOV  AL, 05
        MOV  DI, 0A000H
        MOV  CX, 0FH
        CLD
        REPSTOSB
```

Describe what happen as the following sequence of instruction is executed.

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