8086 programming –Control Flow Instructions and Program Structures

Example: write a procedure named *Square* that squares the contents of BL and places the result in BX.

Solution:

```
Square: PUSH AX
        MOV AL, BL
        MUL BL
        MOV BX, AX
        POP AX
        RET
```

Example: write a program that computes $y = (AL)^2 + (AH)^2 + (DL)^2$ places the result in CX. Make use of the SQUARE subroutine defined in the previous example. (Assume result y doesn't exceed 16 bit)

Solution:

```
    MOV CX, 0000H
    MOV BL, AL
    CALL Square
    ADD CX, BX
    MOV BL, AH
    CALL Square
    ADD CX, BX
    MOV BL, DL
    CALL Square
    ADD CX, BX
  HLT
```

- Sometimes we want to save the content of the flag register, and if we save them, we will later have to restore them, these operations can be accomplished with push flags (PUSHF) and pop flags (POPF) instructions, respectively.

Mnemonic	Meaning	Operation	Flags affected
PUSHF	Push flags onto stack	((SP)) ←(flags) (SP) ←(SP)-2	None
POPF	Pop flagsfrom stack	(flags) ← ((SP)) (SP)← (SP)+2	OF, DF, IF TF, SF ZF, AF, PF , CF

Push flags and pop flags instructions

LOOPS AND LOOP-HANDLING INSTRUCIONS

The 8086 microprocessor has three instructions specifically designed for implementing loop operations. These instructions can be used in place of certain conditional jump instruction and give the programmer a simpler way of writing loop sequences. The loop instructions are listed in table below:

Mnemonic	Meaning	Format	Operation
LOOP	Loop	LOOP Short-label	(CX) ←(CX)-1 Jump is initiated to location definedby 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: Write a program to move a block of 100 consecutive bytes of data starting at offset address 400H in memory to another block of memory locations starting at offset address 600H. Assume both block at the same data segment F000H.

Solution:

```
         MOV AX, F000H
         MOV DS, AX
         MOV SI, 0400H
         MOV DI, 0600H
         MOV CX, 64H
NEXTPT:  MOV AH, [SI]
         MOV [DI], AH
         INC SI
         INC DI
         LOOP NEXTPT
         HLT
```

In this way we see that LOOP is a single instruction that functions the same as a decrement CX instruction followed by a JNZ instruction.

STRINGS AND STRING-HANDLING INSTRUCIONS

80x86 is equipped with special instructions to handle string operations. String: A series of data words (or bytes) that reside in consecutive memory locations Permits operations:

- Move data from one block of memory to a block elsewhere in memory,
- Scan a string of data elements stored in memory to look for a specific value,
- Compare two strings to determine if they are the same or different.

Five basic String Instructions define operations on one element of a string:

- Move byte or word string MOVSB/MOVSW
- Compare string CMPSB/CMPSW
- Scan string SCASB/SCASW
- Load string LODSB/LODSW
- Store string STOSB/STOSW

Repetition is needed to handle more than one element of a string.

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) (DI) ← (DI)±1 or 2	None

Basic string instructions

Auto-indexing of String Instructions

Execution of a string instruction causes the address indices in SI and DI to be either automatically incremented or decremented. The decision to increment or decrement is made based on the status of the direction flag. The direction Flag: Selects the auto increment (D=0) or the auto decrement (D=1) operation for the DI and SI registers during string operations.

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

Instruction for selecting auto incrementing and auto decrementing in string instruction

Example: Using string operation, implement the previous example to copy block of memory to another location.

Solution:

```
        MOV CX, 64H
        MOV AX, F000H
        MOV DS, AX
        MOV ES, AX
        MOV SI, 400H
        MOV DI, 600H
        CLD
NXTPT:  MOVSB
        LOOP NXTPT
        HTL
```

Example: Explain the function of the following sequence of instructions

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

```
        LOOPNE AGAIN
```

Solution: The first 5 instructions initialize internal registers and set up a data segment the loop in the program searches the 15 memory locations starting from Memory location A001Hfor the data stored in DL (05H). As long as the value In DL is not found the zero flag is reset, otherwise it is set. The LOOPNE Decrements CX and checks for CX=0 or ZF =1. If neither of these conditions is met the loop is repeated. If either condition is satisfied the loop is complete. Therefore, the loop is repeated until either 05 is found or all locations in the address range A001H through A00F have been checked and are found not to contain 5.

Example: Implement the previous example using SCAS instruction.

Solution:

```
        MOV AX, 0H
        MOV DS, AX
        MOV ES, AX
        MOV AL, 05
        MOV DI, A001H
        MOV CX, 0FH
        CLD
AGAIN:  SCASB
        LOOPNE AGAIN
```

Example: Write a program loads the block of memory locations from A000H through 0A00FH with number 5H.

Solution:

```
        MOV AX, 0H
        MOV DS, AX
```

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

In most applications, the basic string operations must be repeated in order to process arrays of data. Inserting a repeat prefix before the instruction that is to be repeated does this, the repeat prefixes of the 8086 are shown in table below

For example, the first prefix, **REP**, caused the basic string operation to be repeated until the contents of register CX become equal to 0. Each time the instruction is executed, it causes CX to be tested for 0. If CX is found not to be 0, it is decremented by 1 and the basic string operation is repeated. On the other hand, if it is 0, the repeat string operation is done and the next instruction in the program is s executed, the repeat count must be loaded into CX prior to executing the repeat string instruction.

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

Prefixes for use with the basic string operations

Example: write a program to copy a block of 32 consecutive bytes from the block of memory locations starting at address 2000H in the current Data Segment(DS) to a block of locations starting at address 3000H in the current Extra Segment (ES).

```
CLD
MOV AX, data_seg
MOV DS, AX
MOV AX, extra_seg
MOV ES, AX
MOV CX, 20H
MOV SI, 2000H
MOV DI, 3000H
REPZMOVSB
```

Example: Write a program that scans the 70 bytes start at location D0H in the current Data Segment for the value **45H** , if this value is found replace it with the value **29H**and exit scanning.

```
MOV ES, DS
CLD
MOV DI, 00D0H
MOV CX, 0046H
MOV AL, 45H
REPNE SCASB
DEC DI
MOV BYTE PTR [DI], 29H
HLT
```