4. STRING INSTRUCTIONS

The string instructions function easily on blocks of memory. They are user friendly instructions, which help for easy program writing and execution. They can speed up the manipulating code. They are useful in array handling, tables and records.By using these string instructions, the size of the program is considerably reduced.

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

The general forms of these instructions are as shown below:

Mnem.	Meaning	Format	Operation	Flags Effected
MOVS	Move string	**MOVSB/ MOVSW**	((DS)*10+(SI))→((ES)*10+ (DI)) (SI)± 1 → (SI); (DI) ± 1 → (DI) [byte] (SI) ± 2 → (SI); (DI) ± 2 → (DI) [word]	none
CMPS	Compare string	**CMPSB/ CMPSW**	((DS)*10+(SI))-((ES)*10+ (DI)) (SI) ± 1 → (SI); (DI) ± 1 → (DI) [byte] (SI) ± 2 → (SI); (DI) ± 2 → (DI) [word]	O, S, Z, A, P, C
SCAS	Scan string	**SCASB/ SCASW**	(AL) or (AX) -((ES)*10+ (DI)) (DI) ± 1 → (DI) [byte] (DI) ± 2 → (DI) [word]	O, S, Z, A, P, C
LODS	Load string	**LODSB/ LODSW**	((DS)*10+ (SI))→(AL) or (AX) (SI) ± 1 → (SI) [byte] (SI) ± 2 → (SI) [word]	none
STOS	Store string	**STOSB/ STOSW**	(AL) or (AX) →((ES)*10+ (DI)) (DI) ± 1 → (DI) [byte] (DI) ± 2 → (DI) [word]	none

Auto-indexing of String Instructions

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

Mnemonic	Meaning	Format	Operation	Flags Effected
CLD	Clear DF	**CLD**	0→(DF)	DF
STD	Set DF	**STD**	1→(DF)	DF

Prefixes and the String Instructions

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 executed, the repeat count must be loaded into CX prior to executing the repeat string instruction.

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

Example1: Writea program loads the block of memory locations from 0A000H through 0A00FH with number 5H.

Solution:

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

Example 2: write a program to copy a block of 32 consecutive bytes fromthe 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).

Solution:

```
        CLD
        MOV AX, data_seg
        MOV DS, AX
        MOV AX, extra_seg
        MOV ES, AX
```

```
MOV CX, 20H
MOV SI, 2000H
MOV DI, 3000H
MOVSB
REP
```

Example 3: Write a program that scans the 70 bytes start atlocation D0H in the current Data Segment for the value **45H**, if this value is found replace it with the value **29H**and exit scanning.

Solution:

```
MOV AX,data-seg
MOV DS, AX
CLD
MOV DI, 00D0H
MOV CX, 0046H
MOV AL, 45H
REPNE
SCASB
DEC DI
MOV  [DI], 29H
HLT
```

Example 4: 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. Use loop instructions.

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 5: 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 segmentthe loop in the program searches the 15 memory locations starting fromMemory location A001Hfor the data stored in DL (05H). As long as the valueIn DL is not found the zero flag is reset, otherwise it is set. The LOOPNEDecrements CX and checks for CX=0 or ZF =1. If neither of these conditions ismet the loop is repeated. If either condition is satisfied the loop is complete.Therefore, the loop is repeated until either 05 is found or alllocations in the address range A001H through A00F have been checked and are foundnot to contain 5.

Example 6: 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
```

5. CONTROL TRANSFER INSTRUCTIONS

These instructions transfer the program control from one address to other address. (Not in a sequence). They are again classified into four groups. They are:

Unconditional Transfer Instructions	Conditional Transfer Instructions		Iteration Control Instructions	Interrupt Instructions
JMP CALL RET	JA / JNBE JAE / JNB JB / JNAE JBE / JNA JC JE / JZ JG / JNLE JGE / JNL JL / JNGE	JLE / JNG JNC JNE / JNZ JNO JNP / JPO JNS JO JP / JPE JS	LOOP LOOPE / LOOPZ LOOPNE / LOOPNZ	INT INTO IRET

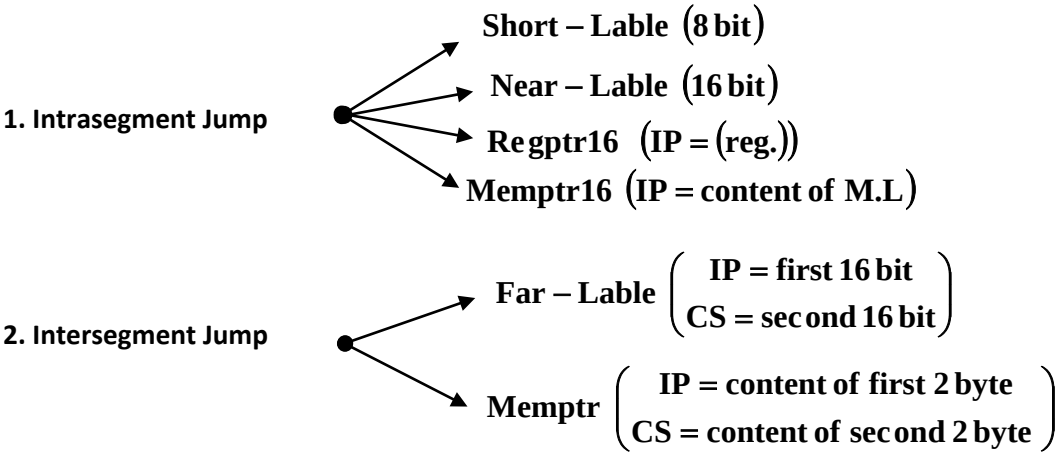
5.1 JUMP Instruction

8086 allowed two types of jump operation. They are the unconditional jump and the conditional jump.

5.1.1 Unconditional jump:

JMP (Jump) unconditionally transfers control from one code segment location to another. These locations can be within the same code segment (near control transfers) or in different code segments (far control transfers). There are two basic kinds of unconditional jumps:

1. **Intrasegment Jump:** is limited to addresses within the current code segment. This type of jump is achieved by just modifying the value in IP.
2. **Intersegment Jump:**permit jumps from one code segment to another. Implementation of this type of jump requires modification of the contents of both CS and IP.



Example 1: Assume the following state of 8086:(CS)=1075H, (IP)=0300H, (SI)=A00H, (DS)=400H, (DS:A00)=10H, (DS:A01)=B3H, (DS:A02)=22H, (DS:A03)=1AH. To what address is program control passed if each of the following JMP instruction is execute?

(a) JMP 85⇒ **1075:85**⇒Short jump

(b) JMB 1000H ⇒ **1075:1000** ⇒Near jump

(c) JMP [SI]⇒ **1075: B310**⇒Near jump

(d) JMP SI ⇒ **1075:0A00**⇒Near jump

(e) JMP FAR [SI] ⇒**1A22: B310**⇒ Far jump

(f) JMP 3000:1000 ⇒**3000:1000**⇒ Far jump

5.1.2Conditional Jump

The conditional jump instructions test the following flag bits: S, Z, C, P, and O. If the condition under test is true, a branch to the label associated with jump instruction occurs. If the condition is false, the next sequential step in the program executes. Tables below are a list of each of the conditional jump instructions.

Table1: Unsigned Conditional Transfers

Mnemonic	Meaning "Jump if….. "	Condition Tested
JA/JNBE	above/not below nor equal	(CF or ZF) = 0
JAE/JNB	above or equal/not below	CF = 0
JB/JNAE	below/not above nor equal	CF = 1
JBE/JNA	below or equal/not above	(CF or ZF) = 1
JC	Carry	CF = 1
JE/JZ	equal/zero	ZF = 1
JNC	not carry	CF = 0
JNE/JNZ	not equal/not zero	ZF = 0
JNP/JPO	not parity/parity odd	PF = 0
JP/JPE	parity/parity even	PF = 1
JCXZ	CX register is zero	CF or ZF = 0

Table2: Signed Conditional Transfers

Mnemonic	Meaning "Jump if….. "	Condition Tested
JG/JNLE	greater/not less nor equal	((SF xor OF) or ZF) = 0
JGE/JNL	greater or equal/not less	(SF xor OF) = 0
JL/JNGE	less/not greater nor equal	(SF xor OF) = 1
JLE/JNG	less or equal/not greater	((SF xor OF) or ZF) = 1
JNO	not overflow	OF = 0
JNS	not sign (positive, including 0)	SF = 0
JO	Overflow	OF = 1
JS	sign (negative)	SF = 1

Example 2: Write a program to move a block of 100 consecutive bytes of data string at offset address 8000H in memory to another block of memory location starting at offset address A000H. Assume that both blocks are in the same data segment value 3000H.

Solution:

```
        MOV AX, 3000H
        MOV DS, AX
        MOV SI, 8000H
        MOV DI, A000H
        MOV CX, 64H
NXT:    MOV AH, [SI]
        MOV [DI], AH
        INC  SI
        INC  DI
        DEC CX
        JNZ  NXT
        HLT
```

Example 3: Write a program to add (50)H numbers stored at memory locations start at4400:0100H , then store the result at address 200H in the same data segment.

Solution:

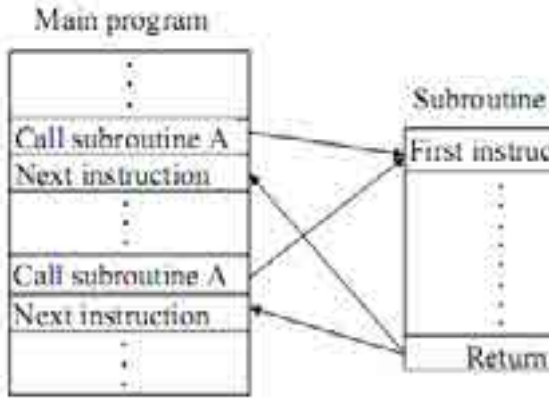
```
        MOV AX , 4400H
        MOV DS , AX
        MOV CX , 0050H counter
        MOV BX , 0100H  offset
Again: ADD AL, [BX]
        INC BX label
        DEC CX
        JNZ Again
        MOV [0200], AL
```

5.2CALL and RETInstructions

- A subroutine is a special segment of program that can be called for execution form any point in program.
- There two basic instructions for subroutine : **CALL** and **RET**
- **CALL** instruction is used to call the subroutine.
- **RET** instruction must be included at the end of the subroutine to initiate the return sequence to the main program environment.
- Just like the **JMP** instruction, **CALL** allows implementation of two types of operations: the *intrasegment call* and *intersegment call.*
- Every subroutine must end by executing an instruction that returns control to the main program. This is the return (**RET**).
- The operand of the call instruction initiates an intersegment or intrasegment call
- The intrasegment call causes contents of **IP** to be saved on Stack.
- The Operand specifies new value in the **IP** that is the first instruction in the subroutine.

- The Intersegment call causes contents of **IP** and **CS** to be saved in the stack and new values to be loaded in **IP** and **CS** that identifies the location of the first instruction of the subroutine.
- Execution of RET instruction at the end of the subroutine causes the original values of **IP** and **CS** to be POPed from stack.

Example 4:
CALL 1234h
CALL BX
CALL [BX]
CALL DWORD PTR [DI]



Example5: 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
```

Example6: 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
          MOVBL,AL
          CALL Square
          ADD CX, BX
          MOV BL,AH
          CALL Square
          ADD CX, BX
          MOV BL,DL
          CALL Square
          ADD CX, BX
          HLT
```

5.3 Iteration Control Instructions

The 8086 microprocessor has three instructions specifically designed for implementing loop operations. These instructions can be use 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 to location defined by short-label if (CX) ≠ 0; otherwise, execute next 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; an d (ZF)=1; otherwise, execute next instruction
LOOPNE/ LOOPNZ	Loop while not equal/ loop while not zero	LOOPNE/LOOPNZshort-label	(CX)←(CX)-1 Jump to location defined by short-label if (CX) ≠ 0; and (ZF)=0; otherwise, execute next 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. (Similar to the example viewed in lecture 6 at page 59). Use loop instructions.

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 LOOPNEXTPT
        HLT
```

6. PROCESS CONTROL INSTRUCTIONS

These instructions are used to change the process of the Microprocessor. They change the process with the stored information. They are again classified into two groups. They are:

1. Flage Control Instructions
2. External Hardware Synchronization Instructions

6.1 Flag Control Instructions:

These instructions directly affected the state of flags. Figure below shows these instructions.

Mnemonic	Meaning	Format	Operation	Flags Effected
STC	Set Carry Flag	**STC**	1→(CF)	CF
CLC	Clear Carry Flag	**CLC**	0→(CF)	CF
CMC	Complement Carry Flag	**CMC**	$(\overline{CF}) \rightarrow (CF)$	CF
STD	Set Direction Flag	**STD**	1→ (DF)	DF
CLD	Clear Direction Flag	**CLD**	0→ (DF)	DF
STI	Set Interrupt Flag	**STI**	1 → (IF)	IF
CLI	Clear Interrupt Flag	**CLI**	0 → (IF)	IF

Example: Write an instruction sequence to save the current contents of the 8086's flags in the memory location pointed to by SI and then reload the flags with the contents of memory location pointed toby DI

Solution:

```
LAHF
MOV [SI], AH
MOV AH, [DI]
SAHF
```

Example: Clear the carry flag without using CLC instruction.

Solution:

```
STC
CMC
```

6.2External Hardware Synchronization Instructions:

Mnemonic	Meaning
HLT	Halt processor
WAIT	Wait for TEST pin activity
ESC	Escape to external processor interface
LOCK	Lock bus during next instruction
NOP	No operation