String Instructions

A String is a sequence of characters, which is stored in consecutive memory locations. There are number of operations that can be performed with strings. 8086 processor supports following string instructions

1) MOVS / MOVSB / MOVSW
2) CMPS / CMPSB / CMPSW
3) SCAS / SCASB / SCASW
4) LODS / LODSB / LODSW
5) STOS / STOSB / STOSW
6) REP
7) REPE / REPZ
8) REPNE / REPNZ

'S' represents String, 'SB' represents String byte & 'SW' represents String word. All String instructions MOVS, CMPS, SCAS, LODS, STOS instructions require two operands. String instructions MOVS & CMPS assumes that the source operand is in the data segment pointed by

SI register & destination operand is in the extra segment pointed by DI register, so SI register to be loaded with effective address of source & DI register to be loaded with effective address of destination.

The instructions LODS, STOS and SCAS assumes that one of the operand in the accumulator.

The instructions REP, REPE & REPNE uses CX register as a counter.

SI and DI register are automatically incremented by 1 for byte and incremented by 2 for word string operations, if DF=0. SI & DI register are automatically decremented by 1 for byte and decremented by 2 for word string operation, if DF=1.

1) MOVS/MOVSB/MOVSW: Move String/Move String Byte/Move String word

MOVS instruction transfers a byte or word from the source operand pointed by SI register in Data segment to the destination string pointed by DI register in extra segment. On execution of MOVS

instruction, SI & DI are automatically incremented by 1 for byte move or incremented by 2 for word move if DF=0. SI & DI are automatically decremented by 1 for byte move or decremented by 2 for word move if DF=1.

Instruction MOVS moves either a byte or word from source string to destination string depending upon whether source & destination is declared either byte type or word type.

MOVSB instruction moves byte from source to destination. MOVSW instruction moves word from source to destination.

Eg:- Pgm to copy source string to destination string

```
.Data
    Src   DB   10H DUP(0)
    Len   DB   0
.   Dest  DB   10H DUP(0)
    msg1  DB   0DH, 0AH, "Enter Source String $"
    msg2  DB   0DH, 0AH, "Source String is $"
    msg3  DB   0DH, 0AH, "Destination String is $"
.   Code
        MOV  AX, @Data
        MOV  DS, AX
        MOV  ES, AX
```

```
msg6    DB  "Length of first String 8"
msg7    DB  "Length of Second String 8"
Str1    DB  10H  DUP(0)
Str2    DB  10H  DUP(0)
len1    DB  0
len2    DB  0

Code
        MOV  AX,@Data
        MOV  DS, AX
        MOV  ES, AX

        DISP  msg1
        MOV  SI, 00
BACK1:  MOV  AH, 01
        INT  21H
        CMP  AL, 0DH
        JE   NEXT1
        MOV  Str1[SI], AL
        INC  SI
        INC  Len1
        JMP  BACK1
NEXT1:  MOV  Str1[SI], '$'
        DISP  msg2
        MOV  SI, 00
BACK2:  MOV  AH, 01
        INT  21H
        CMP  AL, 0DH
        JE   NEXT2
        MOV  Str2[SI], AL
        INC  SI
        INC  Len2
        JMP  BACK2
NEXT2:  MOV  Str2[SI], '$'
        MOV  AL, Len1
        CMP  AL, Len2
        JNE  NOTEQUAL

        LEA  SI, Src
        LEA  DI, Dest
        MOV  CL, Len1
        CLD
BACK3:  CMPSB
        JNE  NOTEQUAL
        LOOP BACK3
        DISP  msg3
```

```
        DISP    msg5
        MOV     AL, LEN1
        aam
        Add     AX, 3030H
        MOV     BX, AX
        MOV     DL, BH
        MOV     AH, 02H
        INT     21H
        MOV     DL, BL
        MOV     AH, 02H
        INT     21H
        JMP     LAST

NOTEQUAL:   DISP    msg4
            DISP    msg6
            MOV     AL, LEN1
            AAM
            ADD     AX, 3030H
            MOV     BX, AX
            MOV     DL, BH
            MOV     AH, 02H
            INT     21H
            MOV     DL, BL
            MOV     AH, 02H
            INT     21H

        DISP    msg7
        MOV     AL, LEN2
        AAM
        ADD     AX, 3030H
        MOV     BX, AX
        MOV     DL, BH
        MOV     AH, 02H
        INT     21H
        MOV     DL, BL
        MOV     AH, 02H
        INT     21H

LAST:   MOV     AH, 4CH
        INT     21H
        END
```

3) SCAS / SCASB / SCASW : Scan String / Scan String Byte / Scan String word

SCAS instruction compares a byte or word pointed by DI register in extra segment with a byte or word in the accumulator. On execution of SCAS instruction, DI are automatically incremented by 1 for byte comparision or incremented by 2 for word comparision, if DF=0 DI are automatically decremented by 1 for byte comparision or decremented by 2 for word comparison if DF=1.

SCAS instruction compares a byte or word with a byte or word in accumulator depending upon whether string is declared as byte type or word type. SCASB instruction compares a byte with a byte in AL register. SCASW instruction compares a word with a word in AX register.

Eg: Program to search key element in an array using linear search technique.

```
· Data
    A   DB  10,20,15,18,25.
    len DB  ($-A)
    key DB  18
    msg1 DB "Element found at position $"
    msg2 DB "Element not found $"
· code
    MOV AX,@DATA
    MOV DS,AX
    MOV ES,AX

    LEA DI,A
    MOV CL,LEN
    MOV AL,KEY
    CLD
BACK: SCASB
    JE FOUND
    LOOP BACK

    DISP msg2
    JMP LAST

FOUND: DISP msg1
    MOV AL,LEN
    SUB AL,CL
    ADD AL,01
    AAM
    MOV BX,AX
    MOV DL,BH
    MOV AH,02H
    INT 21H
    MOV DL,BL
    MOV AH,02H
    INT 21H
LAST: MOV AH,4CH
    INT 21H
    END
```

4) LODS/LODSB/LODSW: Load String / Load String Byte / Load String Word

LODS instruction transfers a byte or word from the source string pointed by SI register in Data Segment to the accumulator. On execution of LODS instruction, SI are automatically incremented by 1 for byte transfer or automatically incremented by 2 for word transfer, if DF=0. SI are automatically decremented by 1 for byte transfer or decremented by 2 for word transfer, if DF=1

LODS instruction transfer either a byte or word from source string to accumulator depending upon source string is declared either of byte type or word type. LODSB instruction transfer byte from source string to AL register. LODSW instruction transfers word from source string to AX register.

Ex: Pgm to count number of vowels & consonents in a given string.


```
DATA
    STR DB "BMS COLLEGE OF ENGG &"
    LEN DB ($-STR)
    NOV DB 0
    NOC DB 0

CODE
    MOV AX, @DATA
    MOV DS, AX
    LEA SI, STR
    MOV CL, LEN
    CLD
BACK: LODSB
    CMP AL, 'A'
    JE NOWEL
    CMP AL, 'E'
    JE VOWEL
    CMP AL, 'I'
    JE VOWEL
    CMP AL, 'O'
    JE NOWEL
    CMP AL, 'U'
    JE VOWEL
    CMP AL, ' '
    JE NEXT
    INC NOC
NEXT: LOOP BACK
    JMP LAST
VOWEL: INC NOV
    LOOP BACK
LAST: MOV AH, 4CH
    INT 21H
END
```

5) STOS/STOSB/STOSW : Store String / Store String Byte / Store String word

STOS instruction transfers a byte or word from AL or AX register to the destination string pointed by DI register in extra segment. On execution of STOS instruction, DI are automatically incremented by 1 for byte transfer or incremented by 2 for word transfer, if DF=0. DI are automatically decremented by 1 for byte transfer or decremented by 2 for word transfer, if DF=1.

STOS instruction transfers a byte or word from accumulator to destination string depending upon destination string is declared either byte type or word type. STOSB instruction transfers a byte from AL register to destination string. STOSW instruction transfers a word from AX register to destination string.

Eg: Pgm to Store Character 'p' to all the locations

```
.DATA
    DEST  DB  10  DUP(?)

.CODE
    MOV  AX, @DATA
    MOV  DS, AX
    MOV  ES, AX

    LEA  DI, DEST
    MOV  AL, 'p'
    CLD
    MOV  CL, 10
BACK: STOSB
    LOOP  BACK
    MOV  AH, 4CH
    INT  21H
END
```

6) REP : Repeat

REP instruction is interpreted as "Repeat while CX ≠ 0" and always is used in confunction with the string instructions. TO execute the string instruction repeatedly, the number of bytes or words for string operation must be stored in CX register.

Eg:- Pgm to copy string from source to destination

```
.DATA
        SRC   DB   10H DUP(0)
        LEN   DB   0
        DEST  DB   10H DUP(?)
        MSG1  DB   "ENTER STRING$"
        MSG2  DB   "SOURCE STRING$"
        MSG3  DB   "DESTINATION STRING$"
.CODE
        MOV   AX,@DATA
        MOV   DS,AX
        MOV   ES,AX

        DISP  MSG1
        MOV   SI,00
BACK:   MOV   AH,01H
        INT   21H
        CMP   AL,0DH
        JE    NEXT
        MOV   SRC[SI],AL
        INC   SI
        INC   LEN
        JMP   BACK
NEXT:   MOV   SRC[SI],'$'

        LEA   SI,SRC
        LEA   DI,DEST
        CLD
        MOV   CX,LEN
        REP   MOVSB

        DISP  MSG2
        DISP  SRC
        DISP  MSG3
        DISP  DEST
        MOV   AH,4CH
        INT   21H
        END
```

=

7) REPE / REPZ : Repeat While Equal / Repeat while zero

REPE instruction is interpreted as "Repeat While (X ≠ 0 and ZF = 1" and always used in conjuction with string instructions. To execute the string instructions repeatedly, the number of bytes or words for string operations must be stored in CX register. REPE instruction are used with CMPS & SCAS instructions.

Eg: Program to Validate the Password

- Data

```
Password1   DB  10 DUP(?)
Password2   DB  10 DUP(?)
Len1        DB  0
Len2        DB  0
msg1        DB  "Enter password$"
msg2        DB  "Re enter password$"
msg3        DB  "Success $"
msg4        DB  "Reenter, Invalid password $"
```

- Code

```
        MOV  AX, @DATA
        MOV  DS, AX
        MOV  ES, AX
BACK3:  DISP MSG1
        MOV  SI, 00
```

```
BACK1: MOV AH, 08H ; Reads Character from keyboard without echo on CRT
       INT 21H
       CMP AL, 0DH
       JE NEXT1
       MOV PASSWORD1[SI], AL
       INC SI
       INC LEN1
       JMP BACK1

NEXT1: DISP MSG2
       MOV SI, 00
BACK2: MOV AH, 08H ; Reads Character from keyboard without echo on CRT
       INT 21H
       CMP AL, 0DH
       JE NEXT2
       MOV PASSWORD2[SI], AL
       INC SI
       INC LEN2
       JMP BACK2

       MOV AL, LEN1
       CMP AL, LEN2
       JNE NOTEQUAL

       LEA SI, PASSWORD1
       LEA DI PASSWORD2
       MOV CX LEN1
       CLD
       REPE CMPSB
       JNE NOTEQUAL
       DISP MSG3
       JMP LAST

NOTEQUAL: DISP MSG4
       JMP BACK3

LAST:  MOV AH, 4CH
       INT 21H

       END
```

8) REPNE / REPNZ : Repeat while not Equal / Repeat while not zero

REPNE instruction is interpreted as "Repeat while $CX \neq 0$ and $ZF=0$" and always used in conduction with string instructions. To execute the string instructions repeatedly, the number of bytes or words for string operations must be stored in CX register. REPNE instructions are used with CMPS & SCAS instructions.

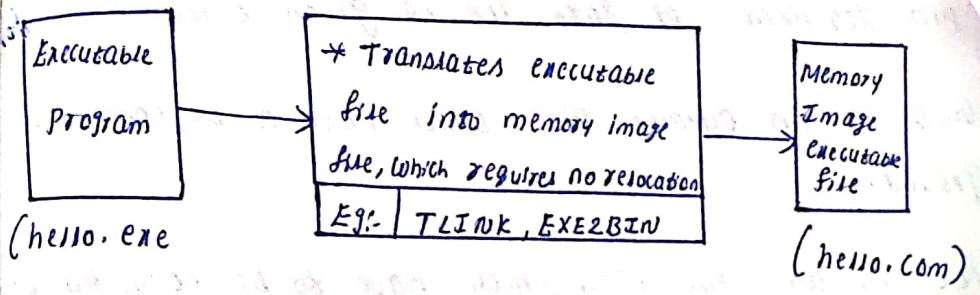
Eg:- Pgm to count number of characters in a given string.

```
.Data
    Str   DB   100H  DUP(?)

.Code
    MOV  AX,@DATA
    MOV  DS,AX
    MOV  ES,AX

    LEA  DX,STR
    MOV  AH,0AH      } Reads string & stores it into
    INT  21H         } location pointed by DX register
                       until enter key pressed by user &
    MOV  AL,'$'      } stores '$' at the end
    LEA  DI,STR      } of the string
    CLD
    REPNE SCASB
```

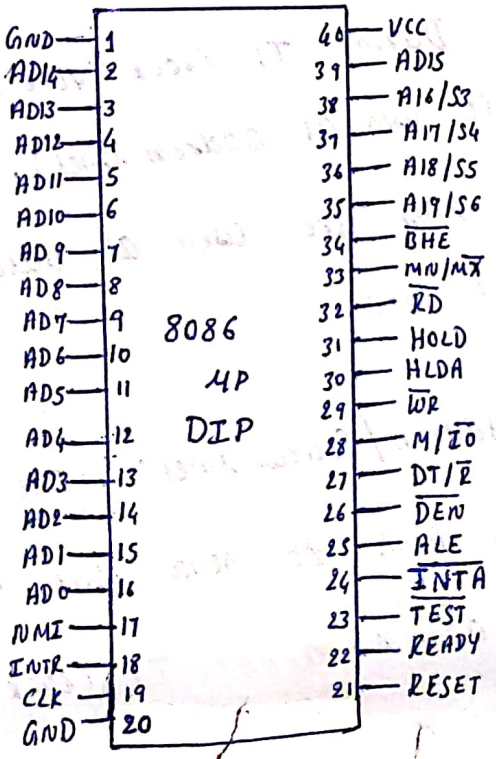
DI → | B | 2000 |
M	2001
S	2002
'$'	2003



) Emulator :- Emulator is a combination of software & hardware. Emulators are used to test & debug the hardware & software of an external system.

Pin Diagram of 8086 microprocessor (μP)

8086 μP is a 40 pin Dual In line package (DIP), requires +5V power supply. It has two GND pins, which have to be connected to ground. The pin diagram of 8086 μP is shown below



...ie pin description of 8086 µP is given below

GND:- GND is an output pins, which have to be connected to ground.

Vcc:- Vcc is an input pin., which have to be connected to +5V power supply.

CLK:- CLK is an input pin. Any µP needs a clock to perform operation. 8086 µP does not have internal clock generator circuit, so this pin have to be connected to external clock generator, such as 8284 clock generator

$AD_{15} - AD_0$:- Address/Data lines:- 8086 µP is a 16 bit µP. It requires 16 data lines to transfer data b/w µP & Peripheral devices. 8086 µP has 20 address lines. With 20 address lines it can address memory space of $2^{20} = 1$ MB memory. To save pins, 16 data lines are multiplexed with the least significant 16 address lines. During T1 clock cycle, µP sends out address on $AD_{15} - AD_0$ (ie uses as address lines). In the later clock cycle, the same pins are used as data lines to sends out data or to read data.

$A_{19} - A_{16} / S_6 - S_3$:- Address/Status lines:- The pins $A_{19} - A_{16}$ are used as address pins to sends out most significant 4 bits of memory address during T1 clock cycle. In the later clock cycles the same pins are used to send Stat...

information of the processor. The status pins S3 & S4 indicates segment register to be used for accessing data

S4	S3	Segment register
0	0	ES
0	1	SS
1	0	CS
1	1	DS

Status pin S5 indicates whether interrupt flag is enabled or not. S6 always 0 & is not used.

INTR:- This is an input pin. Whenever external device sends signal on this pin, MP will be interrupted only if IF is enabled, otherwise it ignores the signal on this pin.

NMI: Non maskable Interrupt:- This is also an input pin. Whenever external device sends a signal on this pin, MP will be interrupted without checking whether IF is enabled or no

$\overline{INTA}$:- Interrupt Acknowledgement:- This is an output pin. Whenever external device sends a signal on INTR pin, then MP will be interrupted, if the IF is enabled. MP finishes its current instruction execution & sends low signal on $\overline{INTA}$ pin to external device indicating that it has accepted the interrupt.

BHE: Bus High enable :- The main memory is divided into even bank & odd bank ... maximum of size 512 KB. This pin

is an output pin connected to odd bank. Whenever µp wa to read data from odd location or write data into odd location, it sends low signal on this pin to enable odd ba

MN/$\overline{MX}$:- µp can operate either in minimum mode or maxim mode. This is an input pin, If this pin is connected to + power supply, then µp operates in a minimum mode + generate control signals directly to all the external devices. If thi pin is grounded, then µp operates in a maximum mode wit the supporting chips 8087 coprocessor, 8089 I/o processor, 8288 bus controller etc.

$\overline{RD}$:- This is an output pin connected to memory + I/p device. Whenever µp wants to read data from memory or I/ device, it sends low signal on this pin

$\overline{WR}$:- This is an output pin connected to memory + o/p device. Whenever µp wants to write the data into memory or o/p device, it sends low signal on this pin.

HOLD & $\overline{HLDA}$:- This is an input pin connected to DMA controller. Whenever DMA controller wants to direct transfer of data b/w memory + I/o device, it sends request signal on this pin requesting addres bus + data bus. µp stops its execution + gives control of addres bus + data bus to DMA controller by sending acknowledge

Output Pin.

M/$\overline{IO}$:- This is an output pin connected to memory & I/O devices. Whenever MP wants to communicate with memory, it sends high signal on this pin. Whenever MP wants to communicate with $\overline{I/O}$ devices, it sends low signal on this pin.

DT/$\overline{R}$ & $\overline{DEN}$:- These pins are output pins connected to 8286 transreciver buffer. Whenever MP wants to read data or write data, it sends low signal on $\overline{DEN}$ pin, then 8286 buffer will be enabled for read/write operation, then it sends high signal on DT/$\overline{R}$ pin to transmit data from MP to memory/IO device & or low signal on DT/$\overline{R}$ pin to read data from memory or I/P device to MP.

ALE : Addrem latch enable :- This is an output pin connected to addrem latch 74LS373. During T1 clock cycle, when MP places addrem bits on Addrem lines, then it sends high signal on this pin to latch 74LS373 to latch the addrem bits sent by 8086 MP & make available that addrem to memory for the subsequent clock cycles.

$\overline{TEST}$:- $\overline{TEST}$ pin is an input pin used for synchronization b/w 8086 MP & 8087 coprocessor. Whenever MP encounters floating point instruction, it enters into wait state & floati...

point operation are performed by 8087 coprocesor, After completion of operation, 8087 processor sends signal on this pin to 8086 MP indicating it finishes floating point operation & 8086 processor continues its execution with next instruction.

READY: READY pin is an input pin connected to slower peripheral devices used to synchronize b/w MP & slower peripheral devices.

RESET: RESET pin is an input pin, when the signal on this pin, MP will be reset ie, it starts from initial state.