LECTURE # 13

STRING OPERATIONS

String Operation

- There is a group of instructions referred to as a *string instructions* in the x86 family of microprocessor.
- They are capable of performing operations on a series of operands located in consecutive memory locations.
- For example, while the CMP instruction can compare only two bytes or words of data, the CMPS (compare string) instruction is capable of comparing two arrays of data located in memory locations pointed at by SI and DI registers.

String Operation

Use of SI and DI, DS and ES in string instructions:

- For string operations to work, certain registers designed for specific functions.
- These registers must permanently provide the source and destination operands.
- In 8086, the SI and DI registers always point to the source and destination operands respectively.
- To generate the physical address, the 8086 always uses SI as the offset of the DS register and DI as the offset of ES.
- This is the default mode.
- It must be noted that the ES register must be initialized for the string operation to work.

String Operation

Byte and word operands in string instructions:

- In each of the string instructions, the operand can be a byte or a word.
- Operands are distinguished by the letters B (byte) and W (word) in the instruction mnemonic.
- Following table shows a summary of all the string instructions.

String Operation

Byte and word operands in string instructions:

Instruction	Mnemonic	Destination	Source	Prefix
Move string byte	MOVSB	ES:DI	DS:SI	REP
Move string word	MOVSW	ES:DI	DS:SI	REP
Store string byte	STOSB	ES:DI	AL	
Store string word	STOSW	ES:DI	AX	REP
Load string byte	LODSB	AL	DS:SI	None
Load string word	LODSW	AX	DS:SI	Non
Compare string byte	CMPSB	ES:DI	DS:SI	REPE/ REPNE
Compare string word	CMPSW	ES:DI	DS:SI	REPE/ REPNE
Scan string byte	SCASB	ES:DI	DS:SI	REPE/ REPNE
Scan string word	SCASW	ES:DI	DS:SI	REPE/ REPNE

DF, the direction flag

- To process operands located in consecutive memory locations requires that the pointer be incremented or decremented.
- In string operations this is achieved by the direction flag.
- It is the job of the string instruction to increment or decrement the SI and DI pointers, but it is the job of the programmer to specify the choice of increment or decrement by setting the direction flag to high or low.
- The instructions **CLD** (clear direction flag) and **STD** (set direction flag) are specifically designed for that.

DF, the direction flag

- **CLD** will reset DF, indicating that the string instruction should increment the pointers automatically.
- **STD** will set DF to 1, indicating to the string instruction that the pointers SI and DI should be decremented automatically.

REP Prefix

- The **REP** (repeat) prefix allows a string instruction to perform the operation repeatedly.
- REP assumes the CX holds the number of times that the instruction should be repeated.
- The REP prefix tells the CPU to perform the string operation and then decrements the CX register automatically. This process is repeated until CX becomes zero.

String Operation

Example:

Using string instructions, write a program that transfers a block of 20 bytes of data.

Solution:

```
DATA1   DB      'ABCDEFGHIJKLMNOPQRST'
        ORG     30H
DATA2   DB      20 DUP (?)
. . .           . . .       . . .
        MOV AX, @DATA
        MOV DS, AX
        MOV ES, AX
        CLD
        MOV SI, OFFSET DATA1
        MOV DI, OFFSET DATA2
        MOV CX, 20
        REP MOVSB       ; repeat until CX becomes zero
```

String Operation

STOS instruction:

- The **STOSB** instruction stores the byte in the AL register into memory locations pointed at by ES:DI and increments (if DF = 0) DI once. If DI = 1, then DI is decremented.
- The **STOSW** instruction stores the contents of AX in memory locations ES:DI and ES:DI+1 (AL into ES:DI and AH into ES:DI+1), the increments DI twice (if DF=0). If DF=1, DI is decremented twice.

String Operation

LODS instruction:

- The **LODSB** instruction loads the contents of memory locations pointed at by DS:SI into AL and increments or decrements SI once if DF = 0 or DF=1.
- T**he LODSW** loads the contents of memory locations pointed at by DS:SI into AL and DS:SI+1 into AH. The SI is incremented twice if DF=0. otherwise, it is decremented twice.
- LODS is never used with a REP prefix.

String Operation

Example:

Write a program that:

a) Uses STOSB to store byte AAH in 100 memory locations.

b) Uses LODS to test the contents of each location to see if AAH is thee. If the test fails, the system should display the message ‘bad memory’

```
            .DATA
MEM_AREA    DB      100 DUP(?)
MESSAGE     DB      ‘BAD MEMORY’,’$’
```

Solution:

```
            MOV AX, @DATA
            MOV DS, AX
            MOV ES, AX
            CLD
            MOV CX, 50
            MOV DI, OFFSET MEM_AREA
            MOV AX, AAAAH
            REP STOSW
            MOV SI, OFFSET MEM_AREA
            MOV CX, 100
AGAIN:      LODSB
            XOR AL, AH
            JNZ OVER
            LOOP AGAIN
            JMP EXIT
OVER:       MOV AH, 09  ; FOR DISPLAY STRING
            MOV DX, OFFSET MESSAGE
            INT 21H
EXIT:       MOV AH, 4CH
            INT 21H
```

Description of INT 21H Option 09:

- INT 21H option 09: outputting a string of data to the monitor.
- INT 21H can be used to send a set of ASCII data to the monitor.
- Register settings before INT 21H is invoked: AH=09 DX = the offset address of the ASCII data to be displayed.
- The address in DX register is an offset address. Data is assumed to be the data segment.
- INT 21H option 09 will display the ASCII data string pointed at by DX until it encounters the dollar sign '$'. Note that this option cannot display '$' character on the screen.

String Operation

The REPZ and REPNZ prefixes:

- These prefixes can be used with the CMPS and SCAS instructions for testing purposes.
- **REPZ** (repeat zero), which is the same as REPE (repeat equal), will repeat the string operation as long as the source and destination operands are equal (ZF=1) or until CX becomes zero.
- **REPNZ** (repeat not zero), which is the same as REPNE (repeat not equal) will repeat the string operation as long as the source and destination operands are not equal (ZF=0) or until CX becomes zero.
- These two prefixes will be used in the context of applications after the explanation of the CMPS and SCANS instructions.
- **CMPS** (compare string) allows the comparison of two arrays of data pointed at by the SI and DI registers.

String Operation

Example:

- Assuming that there is a spelling of ‘Europe’ in an electronic dictionary and a user types in ‘Euorope’, write a program that compares these two and displays the following message, depending on the result:
- If they are equal, display ‘The spelling is correct’.
- If they are not equal, display ‘Wrong spelling’.

String Operation

Example:

Solution:

```
DAT_DICT        DB      'Europe'
DAT_TYPED       DB      'Euorop'
MSG1            DB      'The spelling is correct','$'
MSG2            DB      'Wrong spelling','$'
. . .           . . .   . . .
                CLD                 ; DF=0 for increment
                MOV SI, OFFSET DAT_DICT
                MOV DI, OFFSET DAT_TYPED
                MOV CX, 06
                REPE CMPSB          ; repeat as long as equal or until CX=0
                JE OVER             ; if ZF=1 then display MSG1
                MOV DX, OFFSET MSG2
                JMP DISPLAY
OVER:           MOV DX, OFFSET MSG1
DISPLAY:        MOV AH, 09          ; For displaying
                INT 21H
```

String Operation

SCASB (scan string):

- The SCASB string instruction compares each byte of the array pointed at by ES:DI with the contents of the AL register, and depending on which prefix REPE or REPNE is used, a decision is made for equality or inequality.

String Operation

Example:

Write a program that scans the name ‘Mr. Gones’ and replaces the ‘G’ with the letter ‘J’, then displays the corrected name.

Solution:

```
DATA1   DB      ‘Mr. Gones’,’$’
. . .           . . .       . . .
        MOV AX, @DATA
        MOV DS, AX
        MOV ES, AX
        CLD
        MOV DI, OFFSET DATA1
        MOV CX, 09
        MOV AL, ‘G’
        REPNE SCASB

        JNE OVER
        DEC DI
        MOV BYTE PTR [DI], ‘J’
OVER:   MOV AH, 09
        MOV DX, OFFSET DATA1
        INT 21H
```

TABLE OPERATIONS

Table Operation

XLAT instruction and Look-up table:

- There is often a need in computer applications for a table that holds some important information.
- To access the elements of the table, 8088/86 microprocessors provide the XLAT (translate) instruction.
- To understand the XLAT instruction, one must first understand tables.
- The table is commonly referred to as a lookup table.

Table Operation

XLAT instruction and Look-up table:

- Assume that one needs a table for the values of x^2, where x is between 0 and 9.
- First the table is generated and stored in memory:

```
SQUR_TABLE          DB      0, 1, 4, 9, 16, 25, 36, 49, 64, 81
```

- Now one can access the square of any number from 0 to 9 by the use of XLAT.
- To do that, the register BX must have the offset address of the lookup table, and the number whose square is found must be in the AL register.
- Then after the execution of XLAT, the AL register will have the square of the number.

Table Operation

XLAT instruction and Look-up table:

- The following example shows how to get the square of 5 from the table:

```
MOV BX, OFFSET SQU_TABLE
MOV AL, 05
XLAT
```

- After execution of this program, the AL register will have 25, the square of 5.
- It must be noted that for XLAT to work, the entries of the lookup table must be in sequential order and must have one to one relation with the element itself.

Table Operation

Code conversion using XLAT:

- In many microprocessor based systems, the keyboard is not an ASCII type of keyboard.
- XLAT instruction is used to translate the hex keys of such keyboard to ASCII.
- Following program convert hex digits of 0 to F to their ASCII equivalents.

```
ASC_TABLE    DB      '0','1','2','3','4,'5','6','7','8',
                     '9', 'A','B','C','D','E','F'
HEX_VAL      DB      ?
ASC_VAL      DB      ?
. . .        . . .   . . .
             MOV BX, OFFSET ASC_TABLE
             MOV AL, HEX_VAL
             XLAT
             MOV ASC_VAL, AL
```