K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Batch: B1 **Roll No.: 1711072**

Experiment / assignment / tutorial No. 2

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: To exchange blocks of data using string instructions

Objective: To understand significance of string instructions

Expected Outcome of Experiment:

CO 1: Explain the process of Compilation from Assembly language to machine language

---

Books/ Journals/ Websites referred:

1) Microprocessor architecture and applications with 8085: By Ramesh Gaonkar (Penram International Publication).

2) 8086/8088 family: Design Programming and Interfacing: By John Uffenbeck (Pearson Education).

Pre Lab/ Prior Concepts: string instructions like MOVSB, CLD, REP have to be known. Assume no of blocks to be transferred and copy block from one location to another location using string instructions

Instructions used: LEA, CLD, REP, MOVSB**.**

Eg:

LEA SI, a ;loads effective address of 'a' in SI.

CLD ;used to clear the direction flag, so that instruction is read from left to right and sets counter in auto increment mode.

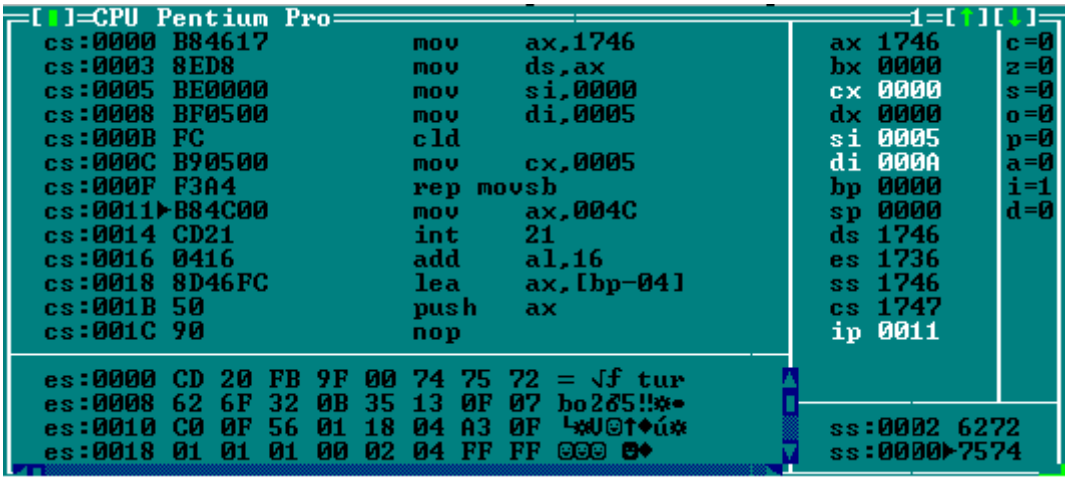
REP MOVSB ;copies the contents of the byte addressed by DS:SI to the byte addressed by ES:DI till counter is 0.

Algorithm/ Code:

```
DATA SEGMENT
a_str db 'turbo'
b_str db 5 dup(?)
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS: DATA

START:
MOV AX, DATA
MOV DS, AX
LEA SI, a_str
LEA DI, b_str
CLD
MOV CX, 05h
REP MOVSB
MOV AX, 4CH
INT 21H
```

```
CODE ENDS

END STARTS

END
```

Output:



Conclusion: The blocks of data were exchanged successfully by this program written in Assembly.

Post Lab Descriptive Questions (Add questions from examination point view)

Explain significance of various string instructions:

String Instruction Basics

- Source DS:SI, Destination ES:DI
  - You must ensure DS and ES are correct
  - You must ensure SI and DI are offsets into DS and ES respectively
- Direction Flag (0 = Up, 1 = Down)
  - CLD - Increment addresses (left to right)
  - STD - Decrement addresses (right to left)

The Direction Flag

- One of the control flags in the FLAGS register is the *direction flag* (DF)
- It determines the direction in which string operations will proceed
- The string operations are implemented by the two index registers SI and DI
- If DF = 0, SI and DI proceed in the direction of increasing memory addresses
- If DF = 1, they proceed in decreasing direction

CLD and STD

- **To make DF = 0, use the cld** instruction

```
cld     ;clear direction flag
```

- To make DF = 1, use the **std** instruction

```
std     ;set direction flag
```

- **cld** and **std** have no effect on the other flags

Moving a String

- Suppose we have defined two strings

```
DATASEG
string1    DB    "HELLO"
string2    DB    5 DUP (?)
```

- The **movsb** instruction

```
movsb      ;move string byte
```

- copies the contents of the byte addressed by DS:SI to the byte addressed by ES:DI
- after the byte is moved, both SI and DI are incremented if DF=0; if DF=1, they are decremented

MOVSB example

- To copy the first two bytes of **str1** to **str2**, we use the following instructions:

```
mov     ax,@data
mov     ds,ax      ;initialize ds
mov     es,ax      ;  and es
lea     si,[str1] ;si points to source string
lea     di,[str2] ;di points to dest string
cld            & nbsp;    ;set df=0 (increasing)
movsb             ;   ;move first byte
movsb             ;   ;move second byte
```

The REP Prefix

- **movsb** moves only a single byte from the source string to the destination
- To move the entire string, first initialize **cx** to the number *N* of bytes in the source string and execute **rep movsb**
- The **rep** prefix causes **movsb** to be executed *N* times
- After each **movsb**, **cx** is decremented until it becomes 0

REP Example

```
mov     ax,@data
mov     ds,ax      ;initialize ds
mov     es,ax      ; and es
lea     si,[str1] ;si points to source string
lea     di,[str2] ;di points to dest string
cld            & nbsp;    ;set df=0 (increasing)
mov     cx,5       ;# of chars in string1
rep movsb          ;copy the string
```

MOVSW

- The word form of **movsb** is **movsw**

```
movsw      ;move string word
```

- **movsw** moves words rather than bytes
- After the string word has been moved, both **SI** and **DI** are incremented (or decremented) by 2
- Neither **movsb** nor **movsw** have any effect on the flags

Example: Memory Shift

```
;shift bytes of A 3 bytes to right
mov     cx, 7              ;bytes to copy
mov     di, offset A+9     ;dest
```

```
mov     si, offset A+9-3 ;source
std           & nbsp;          ;copy from right
to left
rep movsb
```

Example: Replication

```
pattern db      "!@#*"          ;duplicate
        db      (100-4) dup (?) ;space
        mov     cx,100-4        ;96 bytes to
copy
        mov     si, offset pattern
        mov     di, offset pattern+4
        cld             &nb sp;
;destructive overlap
        rep movsb
```

The STOSB and STOSW Instructions

```
stosb     ;store string byte
```

- Moves the contents of the AL register to the byte addressed by ES:DI
- DI is incremented if DF=0 or decremented if DF=1
- Similarly,

```
stosw     ;store string word
```

- Moves the contents of AX register to the word addressed by ES:DI
- DI is incremented or decremented by 2
- Neither **stosb** nor **stosw** have any effect on the flags

Code using STOSB

```
mov    ax,@data
mov    es, ax     ;initialize es
lea    di,[str]   ;di points to str
cld           & nbsp;   ;process to the right
mov    al,'A'     ;al has char to store
stosb            ;  ;store an 'A'
stosb            ;  ;store another one
```

Example: Initializing Storage

```
arr    dw 200 dup (?) ;empty words
;to be initialized to A050A050...
mov    ax,50A0h
mov    di,offset arr
mov    cx,200         ;array size
```

```
cld
rep stosw
```

The LODSB Instruction

```
lodsb      ;load string byte
```

- Moves the byte addressed by DS:SI into the AL register
- SI is incremented if DF=0 or decremented if DF=1
- Similarly,

```
lodsw      ;store string word
```

- Moves the word addressed by DS:SI into the AX register
- SI is incremented or decremented by 2
- Neither **lodsb** nor **lodsw** have any effect on the flags

Code using LODSB

```
DATASEG
str DB     'ABC' ;define string
CODESEG
mov    ax,@data
mov    ds,ax      ;initialize ds
lea    si,[str]   ;si points to str
cld            & nbsp;  ;process left to
right
lodsb             ; ;load first byte in al
lodsb             ; ;load second byte in al
```

Example: Process Array

```
;array b = toUpper(array a)
    mov     di,offset b   ;dest
    mov     si,offset a   ;source
    mov     cx,30         ;array size
    cld            & nbsp;       ;left to right
    processing
lp:
    lodsb             ;      ;get next byte
    and     al,0DFh       ;to upper case
    stosb             ;      ;store at next
    location
    loop    lp
```

SCASW

- scasw is the word form of scan string

- The target word is in **ax**
- **di** is incremented or decremented by 2 depending on the value of **df**
- All the status flags are affected by **scasb** and **scasw**

SCASB Example

```
        DATASEG
    str DB     'ABC'      ;define string
        CODESEG
        mov    ax,@data
        mov    es,ax      ;initialize es
        cld             & nbsp; ;process left to
        right
        lea    di,[str]  ;di points to str
        mov    al,'B'    ;target character
        scasb            ;scan first byte
        scasb            ;scan second byte
```

REPNE, REPNZ, REPE, and REPZ

- In looking for a target byte, the string is traversed until a match is found or the string ends
- As with **rep**, **cx** is initialized to the length of the string
- **repne scansb** (*repeat while not equal*) will repeatedly subtract each string byte from **al**, update **di**, and decrement **cx** until either the target is found (**zf** = 1) or **cx** = 0
- **repnz** is a synonym for **repne**
- **repe(***repeat while equal*) repeats a string instruction until **zf** = 0 or **cx** = 0
- **repz** is a synonym for **repe**

Date: 28/01/2019 **Signature of faculty in-charge**