WARREN FERNANDES SE COMPS B 8940

Batch B

EXPERIMENT 6

OBJECTIVE: To perform transferring of source string from a particular location in source segment (data segment) to the desired location in the destination segment (extra Segment) and understand string instructions.

Aim: To perform a. Block transfer from source to destination
c. Check whether it is palindrome or not

Theory:

String is s series of data byte or word available in memory at consecutive locations. It is either referred as byte string or word string. Their memory is always allocated in a sequential order. Instructions used to manipulate strings are called string manipulation instructions.

Some of the string instructions are

- **REP** − Used to repeat the given instruction till CX ≠ 0.
- **REPE/REPZ** − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
- **REPNE/REPNZ** − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
- **MOVS/MOVSB/MOVSW** − Used to move the byte/word from one string to another.
- **COMS/COMPSB/COMPSW** − Used to compare two string bytes/words.
- **INS/INSB/INSW** − Used as an input string/byte/word from the I/O port to the provided memory location.
- **OUTS/OUTSB/OUTSW** − Used as an output string/byte/word from the provided memory location to the I/O port.
- **SCAS/SCASB/SCASW** − Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.
- **LODS/LODSB/LODSW** − Used to store the string byte into AL or string word into AX.

Algorithm:

a. Block Transfer

1. Initialize the data segment
2. Store source string in consecutive memory locations
3. Initialize extra segment
4. Allocate consecutive memory locations for transfer
5. Load the effective address of source string in SI register
6. Load the effective address of destination string in DI register
7. Initialize the direction flag for auto increment or auto decrement
8. Store number of bytes to be transferred in any of general purpose register

9. Transfer the source string using appropriate string instructions (MOVSB/MOVSW)
10. Decrement count
11. Check if count =0, if yes then stop else repeat steps 9-11
12. Stop

b. Check whether it is palindrome or not

1. Initialize Data segment (DS).
2. Initialize Extra segment (ES).
3. Make SI register point to first element of a string and DI register to point to last element of a string.
4. Initialize count register to number of comparisons required.
5. Move contents pointed by SI to AL.
6. Compare character in AL with character pointed by DI.
7. If there is a mismatch (ZF=0) display a message "Not a palindrome" and
8. stop.
9. If two characters are matching then increment SI, decrement DI and decrement count
10. register.
11. If count register becomes zero display a message " String is palindrome" and stop

Postlab:

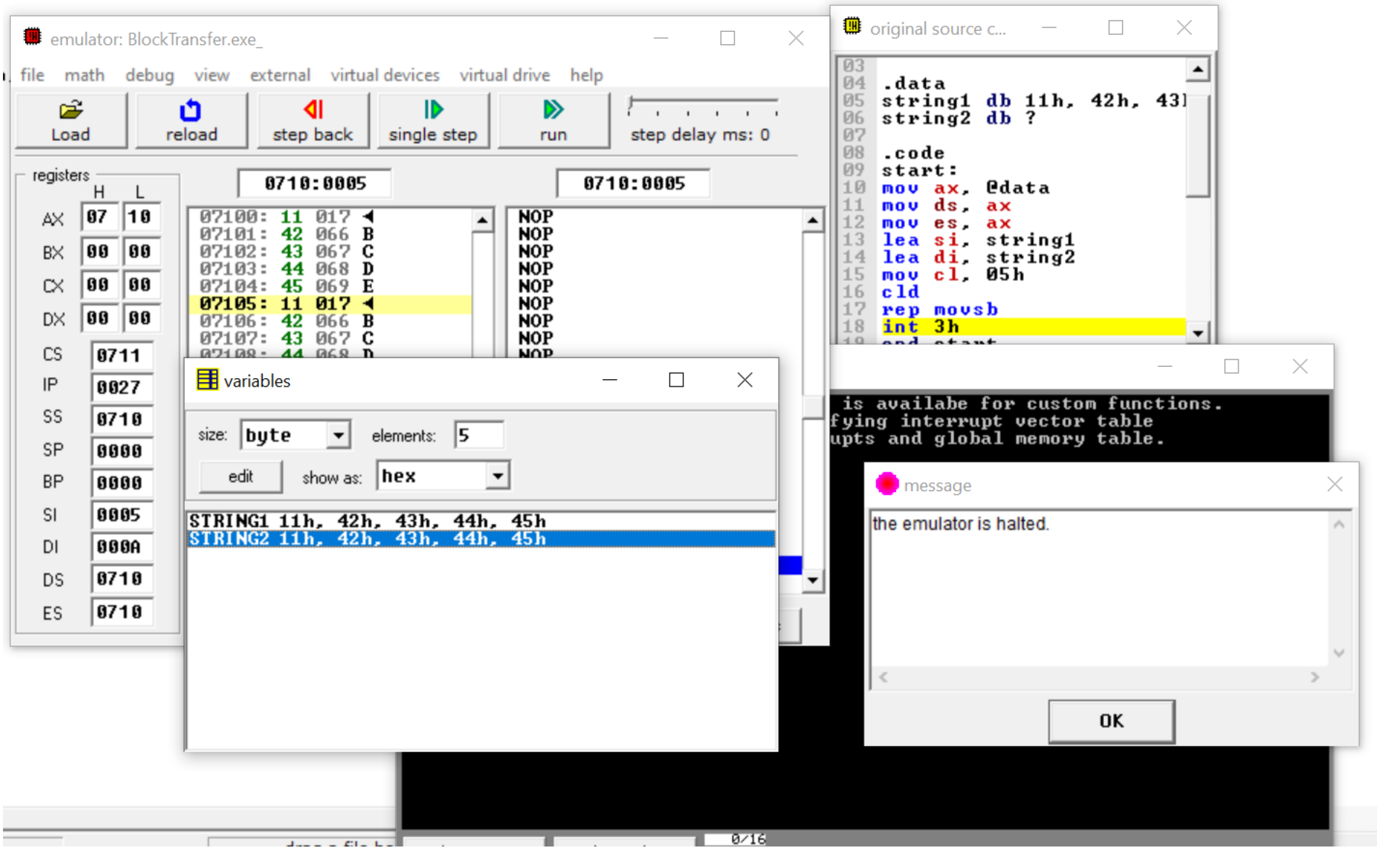
1. Explain any 5 string instructions with example.

a. Block transfer from source to destination

```
.8086
.model small

.data string1 db
41h,42h,43h,44h,45h string2 db
?

.code start:
    mov ax, @data
mov ds, ax
mov es,ax     lea
si,string1     lea
di,string2     mov
cl,05h     cld
    rep movsb
int 3h     end
start
```



	H	L
AX	07	10
BX	00	00
CX	00	00
DX	00	00

CS	0711
IP	0027
SS	0710
SP	0000
BP	0000
SI	0005
DI	000A
DS	0710
ES	0710

```
0710:0005

07100: 11 017 ◄
07101: 42 066 B
07102: 43 067 C
07103: 44 068 D
07104: 45 069 E
07105: 11 017 ◄
07106: 42 066 B
07107: 43 067 C
07108: 44 068 D
```

```
0710:0005

NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
```

```
.data
string1 db 11h, 42h, 43]
string2 db ?

.code
start:
mov ax, @data
mov ds, ax
mov es, ax
lea si, string1
lea di, string2
mov cl, 05h
cld
rep movsb
int 3h
end start
```

size: byte elements: 5
edit show as: hex

```
STRING1 11h, 42h, 43h, 44h, 45h
STRING2 11h, 42h, 43h, 44h, 45h
```

```
 is availabe for custom functions.
fying interrupt vector table
upts and global memory table.
```

the emulator is halted.

b. Check whether it is palindrome or not

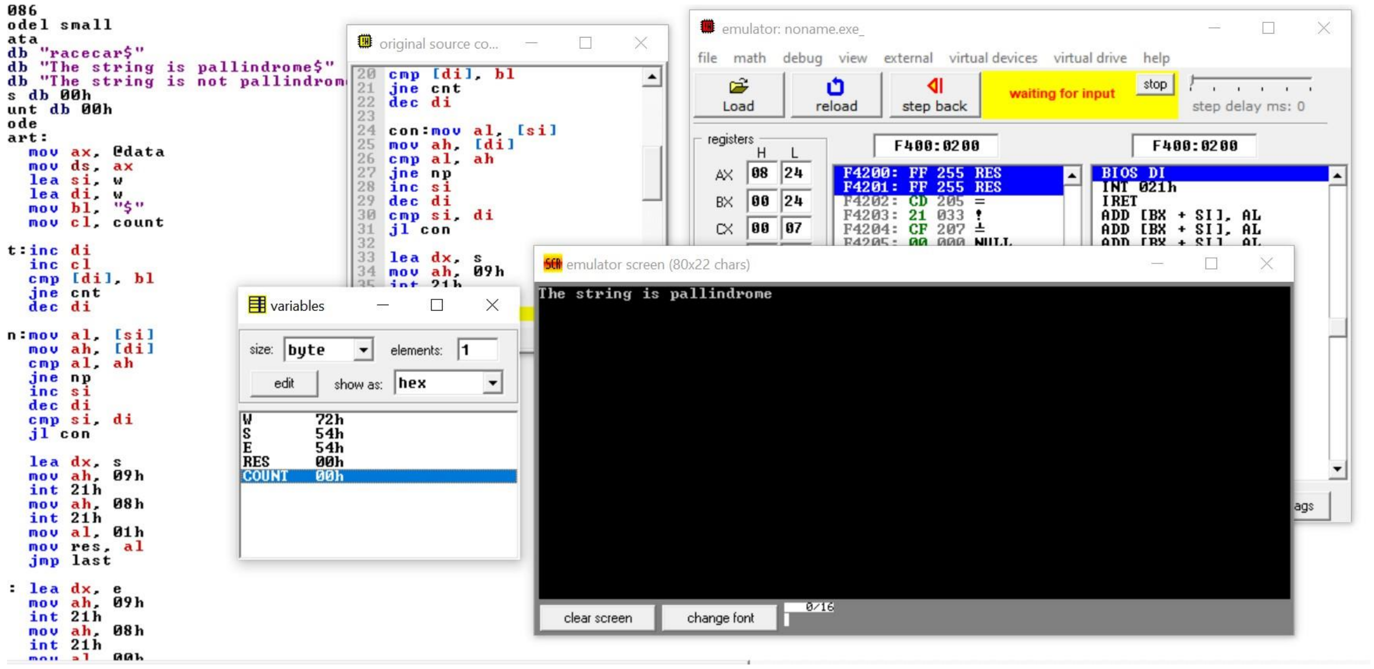
```
.8086
.model small .data w db "racecar$"
s db "The string is pallindrome$" e
db "The string is not pallindrome$"
res db 00h count db 00h .code start:
   mov ax, @data
mov ds, ax    lea
si, w    lea di, w
mov bl, "$"
mov cl, count

cnt:inc di
inc cl    cmp
[di], bl    jne
cnt
   dec di

con:mov al, [si]
mov ah, [di]
cmp al, ah
jne np    inc si
dec di    cmp
si, di    jl con

   lea dx, s
mov ah, 09h
int 21h    mov
ah, 08h    int
21h    mov al,
01h    mov
res, al
   jmp last

np: lea dx, e
mov ah, 09h
int 21h    mov
ah, 08h    int
21h    mov al,
00h    mov
res, al
last:mov ah, 4ch
int 21h
end start
```



```
086
odel small
ata
db "racecar$"
db "The string is pallindrome$"
db "The string is not pallindrom
s db 00h
unt db 00h
ode
art:
   mov ax, @data
   mov ds, ax
   lea si, w
   lea di, w
   mov bl, "$"
   mov cl, count

t:inc di
   inc cl
   cmp [di], bl
   jne cnt
   dec di

n:mov al, [si]
   mov ah, [di]
   cmp al, ah
   jne np
   inc si
   dec di
   cmp si, di
   jl con

   lea dx, s
   mov ah, 09h
   int 21h
   mov ah, 08h
   int 21h
   mov al, 01h
   mov res, al
   jmp last

: lea dx, e
   mov ah, 09h
   int 21h
   mov ah, 08h
   int 21h
   mov al, 00h
```

```
cmp [di], bl
jne cnt
dec di

con:mov al, [si]
mov ah, [di]
cmp al, ah
jne np
inc si
dec di
cmp si, di
jl con

lea dx, s
mov ah, 09h
int 21h
```

size: byte elements: 1
show as: hex

```
W       72h
S       54h
E       54h
RES     00h
COUNT   00h
```

```
    H  L
AX  08 24
BX  00 24
CX  00 07
```

```
F400:0200
F4200: FF 255 RES
F4201: FF 255 RES
F4202: CD 205 =
F4203: 21 033 !
F4204: CF 207 ╧
F4205: 00 000 NULL
```

```
F400:0200
BIOS DI
INT 021h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
```

```
The string is pallindrome
```

Postlab:

Q1.
String instructions were designed to operate on large data structures.

- LODS
- STOS
- MOVS
- CMPS
- SCAS

LODS: Loads the AL, AX or EAX registers with the content of the memory byte, word or double word pointed to by SI relative to DS. After the transfer is made, the SI register is automatically updated as follows: SI is incremented if DF=0. SI is decremented if DF=1.
Example:
LODS LIST AX=DS:[SI]; SI=SI ± 2 (if LIST is a word)
LODS MAX EAX=DS:[SI]; SI=SI ± 4 (if MAX is a double word)

STOS: Transfers the contents of the AL, AX or EAX registers to the memory byte, word or double word pointed to by DI relative to ES. After the transfer is made, the DI register is automatically updated as follows: DI is incremented if DF=0. DI is decremented if DF=1.
Example:
STOS LIST ES:[DI]=AX; DI=DI ± 2 (if LIST is a word)
STOS MAX ES:[DI]=EAX; DI=DI ± 4 (if MAX is a double word)

MOVS: Transfers the contents of the memory byte, word or double word pointed to by SI relative to DS to the memory byte, word or double word pointed to by DI relative to ES. After the transfer is made, the DI register is automatically updated as follows: DI is incremented if DF=0. DI is decremented if DF=1.
Example:
MOVS LIST ES:[DI]=DS:[SI]; DI=DI ± 2; SI=SI ± 2 (if LIST is a word)
MOVS MAX ES:[DI]=DS:[SI]; DI=DI ± 4; SI=SI ± 4 (if MAX is a double word)

CMPS: Compares the contents of the memory byte, word or double word pointed to by SI relative to DS to the memory byte, word or double word pointed to by DI relative to ES and changes the flags accordingly. After the comparison is made, the DI and SI registers are automatically updated as follows: DI and SI are incremented if DF=0. DI and SI are decremented if DF=1.

SCAS: Compares the contents of the AL, AX or EAX register with the memory byte, word or double word pointed to by DI relative to ES and changes the flags accordingly. After the comparison is made, the DI register is automatically updated as follows: DI is incremented if DF=0. DI is decremented if DF=1.