Assignment No 08

Name: Shubham Chemate

Roll Number: **21118**

PROBLEM STATEMENT /DEFINITION:

Write x86 ALP to perform non-overlapped and overlapped block transfer (with and without string specific instructions). Block containing data can be defined in the data segment

OBJECTIVE:

To learn

- Overlapped / Non – overlapped data transfer in segments
- Block transfer instruction of 8086
- Data storage in the memory and segments

OUTCOME:

Students will study different block transfer instructions and also understood block transfer within different segments

HARDWARE USED:

- Manufacturer and model: Acer Swift-3
- Processor: Intel core i5 – 8265U @1.60 GHz
- Memory: 8GB of DDR4 RAM and 512GB of ROM
- System Type: 64-bit OS, x-64 based PC

SOFTWARE USED:

- Operating system: Windows 10 and Ubuntu 20.04LTS. Using WSL 2 (Windows Subsystem for Linux) and Installed WSL plugin in VS Code.
- Text editor: VS Code (open source edition, version: 1.26.2)
- Assembler: NASM (version: 2.14.02)

THEORY:

Instructions used:

A] Non-overlapped block transfer:

1. MOVSB-This is a string instruction and it moves string byte from source to destination
2. REP- This is prefix that are applied to string operation. Each prefix cause the string instruction that follows to be repeated the number of times indicated in the count register.
3. CLD- Clear Direction flag. ESI and EDI will be incremented and DF = 0
4. STD- Set Direction flag. ESI and EDI will be incremented and DF = 1
5. ROL-Rotates bits of byte or word left.
6. AND-AND each bit in a byte or word with corresponding bit in another byte or word.
7. INC-Increments specified byte/word by1.
8. DEC-Decrements specified byte/word by1.
9. JNZ-Jumps if not equal to Zero.
10. JNC-Jumps if no carry is generated.
11. CMP-Compares to specified bytes or words.
12. JBE-Jumps if below or equal.
13. ADD-Adds specified byte to byte or word to word.
14. CALL-Transfers the control from calling program to procedure.
15. RET-Return from where call is made.

B] Overlapped block transfer:

1. MOVSB-This is a string instruction and it moves string byte from source to destination.
2. REP- This is prefix that are applied to string operation. Each prefix cause the string instruction that follows to be repeated the number of times indicated in the count register.
3. CLD- Clear Direction flag. ESI and EDI will be incremented and DF = 0
4. STD- Set Direction flag. ESI and EDI will be decremented and DF = 1 2
5. ROL-Rotates bits of byte or word left.
6. AND-AND each bit in a byte or word with corresponding bit in another byte or word.
7. INC-Increments specified byte/word by1.
8. DEC-Decrements specified byte/word by1.
9. JNZ-Jumps if not equal to Zero.
10. JNC-Jumps if no carry is generated.
11. CMP-Compares to specified bytes or words.
12. JBE-Jumps if below or equal.
13. ADD-Adds specified byte to byte or word to word.
14. CALL-Transfers the control from calling program to procedure.
15. RET-Return from where call is made.

ALGORITHMS:

1. Start
2. Initialize data section.
3. Initialize the count, source block and destination block.
4. Using Macro display the Menu for block transfer without string instruction, block transfer with string instruction and exit.
5. If choice = 1, call procedure for non-overlapping block transfer without string instruction.
6. If choice = 2, call procedure for non-overlapping block transfer with string instruction.
7. If choice = 3, call procedure for overlapping block transfer without string instruction.
8. If choice = 4, call procedure for non-overlapping block transfer with string instruction.
9. If choice = 5, terminate the program.

A] Non-overlapped block transfer:

- Without string instructions
    1. Initialize ESI and EDI with source and destination address.
    2. Move count in ECX register.
    3. Move contents at ESI to accumulator and from accumulator to memory location of EDI.
    4. Increment ESI and EDI to transfer next content. 5. Repeat procedure till count becomes zero.
- With string instructions
    1. Initialize ESI and EDI with source and destination address.
    2. Move count in ECX register.
    3. Clear the direction flag.
    4. Move the contents from source location to the destination location using string instruction.
    5. Repeat string instruction the number of times indicated in the count register.

B] Overlapped block transfer:

- Without string instructions
    1. Initialize ESI and EDI with source and destination address.
    2. Move count in ECX register.
    3. Move source block's and destination block's last content address in ESI and EDI.
    4. Move contents at ESI to accumulator and from accumulator to memory location of EDI.

5. Decrement ESI and EDI to transfer next content. 6. Repeat procedure till count becomes zero.

- With string instructions
    1. Initialize ESI and EDI with source and destination address.
    2. Move count in ECX register.
    3. Move source block's and destination block's last content address in ESI and EDI.
    4. Set the direction flag.
    5. Move the data from source location to the destination location using string instruction.
    6. Repeat string instruction the number of times indicated in the count register.

PROGRAM:

```
; 21118 Shubham
; MPL Assignment No: 04

%macro print 2
        mov rax,1
        mov rdi,1
        mov rsi,%1
        mov rdx,%2
        syscall
%endmacro

%macro read 2
        mov rax,0
        mov rdi,1
        mov rsi,%1
        mov rdx,%2
        syscall
%endmacro

%macro exit 0
        mov rax,60
        mov rdi,0
        syscall
%endmacro

section .data
        array : dq  01H, 02H, 03H, 04H, 05H, 00H, 00H, 00H, 00H, 00H
```

```
        ent : db 0x0A
        lent : equ $-ent
        colon : db " : "
        lencol : equ $-colon
        before : db "Before copy",10
        lenbe : equ $-before
        after : db "After copy",10
        lenaf : equ $-after

        menu :
        db "1 : Non-overlapping without string",10
        db "2 : Non-overlapping with string",10
        db "3 : Overlapping without string",10
        db "4 : Overlapping with string",10
        db "5 : Exit",10
        db "Enter your choice",10
        lenmenu : equ $-menu



section .bss
        data : resb 16
        cnt1 : resb 1
        cnt2 : resb 1
        choice : resb 2

global _start
section .text
_start:
        print menu,lenmenu
        read choice,2
        cmp byte[choice],31H
        JE NOWOS
        cmp byte[choice],32H
        JE NOWS
        cmp byte[choice],33H
        JE OWOS
        cmp byte[choice],34H
        JE OWS
        cmp byte[choice],35H
        JE exit
```

```
; ---------- Labels to call procedures ----------

NOWOS :
call CALL1
exit

OWOS :
call CALL2
exit

NOWS :
call CALL3
exit

OWS :
call CALL4
exit

; ---------- Hex to Ascii conversion ----------

h_to_a :
        mov rdi,data
        mov byte[cnt2],16

        UP1 :
                rol rbx,4
                mov dl,bl
                and dl,0FH
                cmp dl,09H
                jbe NEXT
                add dl,07H

        NEXT :
                add dl,30H
                mov byte[rdi],dl
                inc rdi
                dec byte[cnt2]
        JNZ UP1

        print data,16
```

```
        ret

        ; ---------- Printing array ----------

        print_array :
                mov rsi,array                   ;set source pointer to array
                mov byte[cnt1],10               ;count is having 10 because number
of numbers in array are 10

                UP :
                        mov rbx,rsi             ;first we print addresses so we move
rsi for htoa conversion
                        push rsi                ;push rsi because we'll require rsi
later
                        call h_to_a

                        print colon,lencol      ;after printing address we print
colon

                        pop rsi                 ;pop rsi pushed earlier and we
                        mov rbx,qword[rsi]      ;move byte[rdi] to rbx that is the
number to be printed
                        push rsi
                        call h_to_a
                                                ;after printing address colon and the
data at that address
                        print ent,lent          ;we print enter and loop for rest of
the array
                        pop rsi
                        add rsi,8
                        dec byte[cnt1]
                JNZ UP
        ret

        ; ---------- Non overlapping without string instructions ----------

        CALL1 :

                print ent,lent                  ;print array before copying data
                print before,lenbe
                call print_array
```

```
                mov rsi,array                   ;set source index to first element of array
                mov rdi,array+40                ;dest block starts after last element
                mov byte[cnt1],5

                UP2 :
                        mov rax,qword[rsi]      ;mov data from rsi to rdi using rax variable
                        mov qword[rdi],rax      ;rax is temp for copying
                        add rsi,8
                        add rdi,8
                        dec byte[cnt1]
                JNZ UP2

                print ent,lent                  ;print array after copying the contents
                print after,lenaf
                call print_array
        ret


        ; ---------- Overlapping without string ----------

        CALL2 :
                print ent,lent                  ;print array before copying data
                print before,lenbe
                call print_array

                mov rsi,array                   ;first array element
                add rsi,32                      ;third last array element
                mov rdi,rsi                     ;2 overlapping rdi begins from third last
element to array
                add rdi,24
                mov byte[cnt1],5

                UP3 :
                        mov rax,[rsi]           ;mov data from rsi to rdi using rax variable
                        mov [rdi],rax           ;rax is temp for copying
                        sub rsi,8
                        sub rdi,8
                        dec byte[cnt1]
                JNZ UP3
```

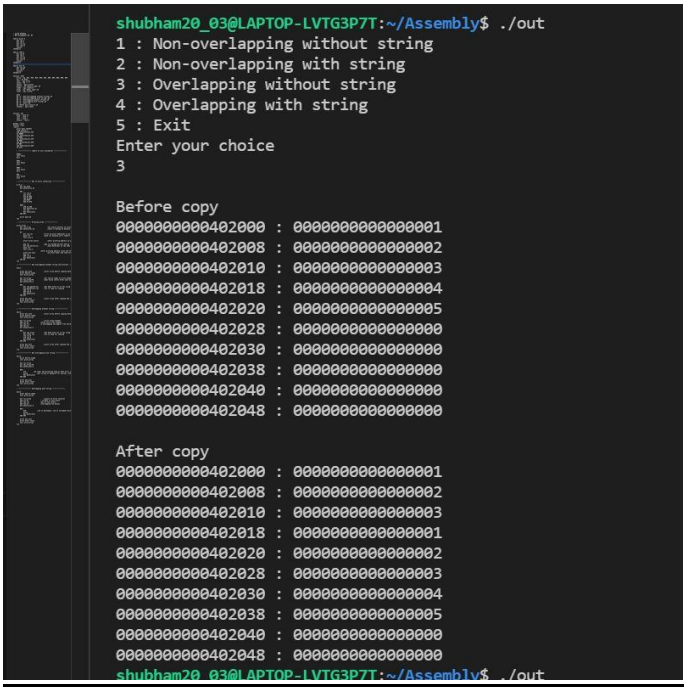
```
        print ent,lent                  ;print array after copying the contents
        print after,lenaf
        call print_array
    ret

    ; ---------- Non overlapping with string ----------

    CALL3 :
        print before,lenbe
        call print_array

        mov rsi,array
        mov rdi,array+40
        mov byte[cnt1],5

        UP4:
            CLD             ;we clear the direction flag so that we'll increment
array addresses automatically
            movsq           ;mov string of quadword for moving a complete
string of data from source to destination
            dec byte[cnt1]
        JNZ UP4

        print ent,lent
        print after,lenaf
        call print_array
    ret

    ; ---------- Overlapping with string ----------

    CALL4 :
        print before,lenbe
        call print_array

        mov rsi,array               ;points to first location
        add rsi,32                  ; +4 goes to last block
        mov rdi,rsi                 ;des to last block
        add rdi,24                  ;overlapping two blocks
        mov byte[cnt1],5

        UP5 :
```

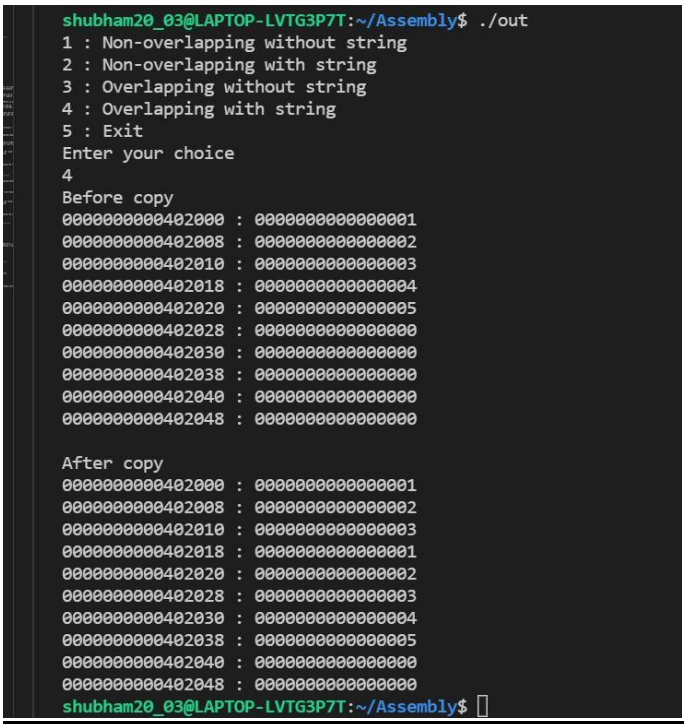
```
                STD                     ;std to decrement, cld to increment
direction flag
                movsq
                dec byte[cnt1]
            JNZ UP5

            print ent,lent
            print after,lenaf
            call print_array
        ret
```

OUTPUT:

```
shubham20_03@LAPTOP-LVTG3P7T:~/Assembly$ nasm -f elf64 block_transfer.a
sm
shubham20_03@LAPTOP-LVTG3P7T:~/Assembly$ ld -o out block_transfer.o
shubham20_03@LAPTOP-LVTG3P7T:~/Assembly$ ./out
1 : Non-overlapping without string
2 : Non-overlapping with string
3 : Overlapping without string
4 : Overlapping with string
5 : Exit
Enter your choice
1

Before copy
0000000000402000 : 0000000000000001
0000000000402008 : 0000000000000002
0000000000402010 : 0000000000000003
0000000000402018 : 0000000000000004
0000000000402020 : 0000000000000005
0000000000402028 : 0000000000000000
0000000000402030 : 0000000000000000
0000000000402038 : 0000000000000000
0000000000402040 : 0000000000000000
0000000000402048 : 0000000000000000

After copy
0000000000402000 : 0000000000000001
0000000000402008 : 0000000000000002
0000000000402010 : 0000000000000003
0000000000402018 : 0000000000000004
0000000000402020 : 0000000000000005
0000000000402028 : 0000000000000001
0000000000402030 : 0000000000000002
0000000000402038 : 0000000000000003
0000000000402040 : 0000000000000004
0000000000402048 : 0000000000000005
```

```
shubham20_03@LAPTOP-LVTG3P7T:~/Assembly$ ./out
1 : Non-overlapping without string
2 : Non-overlapping with string
3 : Overlapping without string
4 : Overlapping with string
5 : Exit
Enter your choice
2
Before copy
0000000000402000 : 0000000000000001
0000000000402008 : 0000000000000002
0000000000402010 : 0000000000000003
0000000000402018 : 0000000000000004
0000000000402020 : 0000000000000005
0000000000402028 : 0000000000000000
0000000000402030 : 0000000000000000
0000000000402038 : 0000000000000000
0000000000402040 : 0000000000000000
0000000000402048 : 0000000000000000

After copy
0000000000402000 : 0000000000000001
0000000000402008 : 0000000000000002
0000000000402010 : 0000000000000003
0000000000402018 : 0000000000000004
0000000000402020 : 0000000000000005
0000000000402028 : 0000000000000001
0000000000402030 : 0000000000000002
0000000000402038 : 0000000000000003
0000000000402040 : 0000000000000004
0000000000402048 : 0000000000000005
shubham20_03@LAPTOP-LVTG3P7T:~/Assembly$ ./out
```





CONCLUSION:

We have studied different block transfer instructions and also understood block transfer within different segments.