Pune Institute of Computer Technology, Pune

Department of Computer Engineering

Sub :- MPL Date:- 05/06/2021

Name: - Shrikrushna Santosh Zirape

Roll No: - 21286

Assignment No 5

Title :-

Non-overlapped block transfer without string inst.

Problem Statement :-

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

Sw/Hw requirements :-

Intel i5 8th generation 64 bit processor

OS - Ubuntu 16.0 LTS

Editor : - VS- Code , Gedit

Assembler: Nasm

Debugger : gdb

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

Theory :-

A] Overlapping

1. Study system call to read and display character on the screen.
2. Accept the Value of „N‟ i.e. how many numbers to add
3. initialize Sum =0
4. Read a number (two digit)
5. add it to sum
6. repeat the steps 4 and 5 to add all N numbers

7. Print the result /Sum

8. End.

B] Non Overlapping

1. Declare a source array.

2. Load the address of source array in one of the registers. (Index register)

3. Read the first byte from the source array.

4. Increment the pointer/SI by length of array which becomes the starting Address of destination array.

5. Move the source element to the destination address.

6. In case of overlapping mode based on the destination address either move the first Element or last element in beginning of transfer operation

Prerequisites: Instruction set of 80386

Concepts related Theory: One of the frequent operations used in programming is shifting/transferring the data form one memory location to another memory location. These operations can be with simple mov instructions which may result in more number of operations. We can make use of instructions like MOVSB to transfer the data.

The relevant instructions are

LOOP / MOVSB.

The data can be transferred either in overlapped fashion or non overlapped Fashion. In case of overlapped address the two possible situations are for the Source address to be greater than the destination address in which case the first element in the source is to be moved first or for the source address to be less than the destination address which requires the last element of the source
to be moved first

Algorithm:

A] **Overlapping**

1. Study system call to read and display character on the screen.
2. Accept the Value of „N‟ i.e. how many numbers to add
3. initialize Sum =0
4. Read a number (two digit)
5. add it to sum
6. repeat the steps 4 and 5 to add all N numbers
7. Print the result /Sum
8. End.

B] **Non Overlapping**

1. Declare a source array.

2. Load the address of source array in one of the registers. (Index register)

3. Read the first byte from the source array.4. Increment the pointer/SI by length of array which becomes the starting Address of destination array.

5. Move the source element to the destination address.

6. In case of overlapping mode based on the destination address either move the first Element or last element in beginning of transfer operation

System calls :

Write System Call :

```
Mov rax, 1
Mov rdi , 1
Mov rsi, msg
Mov rdx, 0BH
Syscall
```

Read System Call :

```
Mov rax, 1
Mov rdi , 1
Mov rsi, msg
Mov rdx, 0BH
Syscall
```

Exit System call

```
Mov rax, 60
Mov rsi, 00
syscall
```

Instructions Used :-

1. Add:
   Description: This instruction adds a number from source to number from destination and puts the result to specified destination.
   destination=destination+source
   Flags: CF, ZF, OF, PF
   e.g. add eax, ebx

2. jnz:

Description: This instruction is used to jump to next instruction in the program

when the zero flag is not equal to 0.

Flags: Only the ZF is affected.

Example: JNZ L1

3. DEC:

Description: This instruction subtracts 1 from the destination word, double word

or byte.

Flags: SF, ZF, OF, PF and AF are affected.

Example: DEC AL

4. INC:

Description: This instruction adds 1 to the destination operand.

Flags: SF, PF, OF, ZF, AF are affected.

Example: INC CX

Conclusion :-

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

Program : -

```
section .data
array db 10h,20h,30h,40h,50h
msg1: db 'Before overlapped :',0xa
len1: equ $-msg1
msg2: db 'After overlapped :',0xa

len2: equ $-msg2
msg3: db ' ',0xa
len3: equ $-msg3
msg4: db ' : '
len4: equ $-msg4
count db 0
count1 db 0
count2 db 0
count3 db 0
count4 db 0
section .bss
addr resb 16
num1 resb 2
```

```
section .text
global _start
_start:
mov rax,1
mov rdi,1
mov rsi,msg1
mov rdx,len1
syscall
xor rsi,rsi
mov rsi,array
mov byte[count],05
up:
mov rbx,rsi
push rsi
mov rdi,addr
call HtoA1
pop rsi
mov dl,[rsi]
push rsi
mov rdi,num1
call HtoA2
```

```
pop rsi
inc rsi
dec byte[count]
jnz up
mov rsi,array
mov rdi,array+5h
mov byte[count3],05h
loop10:
movsb
dec byte[count3]
jnz loop10
mov rax,1
mov rdi,1
mov rsi,msg2
mov rdx,len2
syscall
mov rsi,array
mov byte[count4],0Ah
up10:
mov rbx,rsi
push rsi
```

```
mov rdi,addr
call HtoA1
pop rsi
mov dl,[rsi]
push rsi
mov rdi,num1
call HtoA2
pop rsi
inc rsi
dec byte[count4]
jnz up10
mov rax,60
mov rdi,0
syscall
HtoA1:
mov byte[count1],16
dup1:
rol rbx,4
mov al,bl
and al,0fh
cmp al,09
```

```
jg p3
add al,30h
jmp p4
p3: add al,37h
p4:mov [rdi],al
inc rdi
dec byte[count1]
jnz dup1
mov rax,1
mov rdi,1
mov rsi,addr
mov rdx,16
syscall
mov rax,1
mov rdi,1
mov rsi,msg4
mov rdx,len4
syscall
ret
HtoA2:
mov byte[count2],02
```

```
dup2:
rol dl,04
mov al,dl
and al,0fh
cmp al,09h
jg p31
add al,30h
jmp p41
p31: add al,37h
p41:mov [rdi],al
inc rdi
dec byte[count2]
jnz dup2
mov rax,1
mov rdi,1
mov rsi,num1
mov rdx,02
syscall
mov rax,1
mov rdi,1
mov rsi,msg3
```

```
mov rdx,len3

syscall

ret
```

OUTPUT:-