# Name: Shrinivas Hatyalikar

# Div : CS-B

# Roll No: 24

**Title:**  Write 8086 ALP to perform block transfer operation

**Aim:-** Write 64 bit ALU program for non overlap block transfer without string specific instruction

**Apparatus:**

* Core 2 duo/i3/i5/i7 - 64bit processor
* OS – ubuntu 32bit/64bit OS
* Assembler used –nasm (the netwide assembler)
* Editor Used – gedit

**Theory:**

**dispBlock:**

This procedure takes the memory address of a block of data and displays the data byte by byte on the console as hexadecimal values. It uses a loop to iterate over the block of data and calls the disp procedure to convert each byte into its hexadecimal representation before displaying it

**disp:**

This procedure takes a byte of data and converts it into its hexadecimal representation. It does this by using the rol instruction to rotate the byte left by four bits, then masking the resulting value to extract only the least significant nibble. If the resulting value is less than or equal to 9, it adds 30h to convert it to the corresponding ASCII code for the digit. Otherwise, it adds 37h to convert it to the corresponding ASCII code for the letter. This process is repeated twice to generate the two hexadecimal digits, and the resulting string is stored in a memory location pointed to by the rdi register. The procedure then calls the write procedure to display the hexadecimal value on the console

**write:**

This procedure takes the memory address of a message string and its length and displays the message on the console. It does this by using the system call syscall to write the message to standard output. The write procedure is used throughout the program to display various messages and data values

**Algorithm:-**

1. Declare and initialize the variable in .data Section
2. Declare uninitialized variable in .bss section
3. Declare micros for read, print, Exit
4. Define source block in data section
5. Point this source block rsi to source block
6. Call Display\_block procedure
   * + 1. Initialize count
       2. Move contains of rsi in accumalater
       3. Push rsi
       4. Call display procedure
       5. Print space
       6. Pop rsi and increment rsi and decrement\_block
       7. Repeat step b to f until count equal to 0
7. Repeat step no 6 for destination\_block
8. Call block\_tansfer procedure
9. Repeat step 3 for display source block and destination block
10. Using micro terminte the process
11. Stop
12. Call Display\_block procedure

a) move contains of accumulator into accumulator increase count

* 1. Move contains of accumulator into rdi and increment rdi
  2. Repeat a to c until count become zero

**Code:**

section .data

sourceBlock db 12h,45h,87h,24h,97h

count equ 05

msg db "ALP for non overlapped block transfer using string instructions : ",10

msg\_len equ $ - msg

msgSource db 10,"The source block contains the elements : ",10

msgSource\_len equ $ - msgSource

msgDest db 10,10,"The destination block contains the elements : ",10

msgDest\_len equ $ - msgDest

bef db 10, "Before Block Transfer : ",10

beflen equ $ - bef

aft db 10,10 ,"After Block Transfer : ",10

aftlen equ $ - aft

section .bss

destBlock resb 5

result resb 4

%macro write 2

mov rax,1

mov rdi,1

mov rsi,%1

mov rdx,%2

syscall

%endmacro

section .text

global \_start

\_start:

write msg , msg\_len

write bef , beflen

write msgSource , msgSource\_len

mov rsi,sourceBlock

call dispBlock

write msgDest , msgDest\_len

mov rsi,destBlock

call dispBlock

mov rsi,sourceBlock

mov rdi,destBlock

mov rcx, count

cld

rep movsb

write aft , aftlen

write msgSource , msgSource\_len

mov rsi,sourceBlock

call dispBlock

write msgDest , msgDest\_len

mov rsi,destBlock

call dispBlock

mov rax,60

mov rdi,0

syscall

dispBlock:

mov rbp,count

next:mov al,[rsi]

push rsi

call disp

pop rsi

inc rsi

dec rbp

jnz next

ret

disp:

mov bl,al ;store number in bl

mov rdi, result ;point rdi to result variable

mov cx,02 ;load count of rotation in cl

up1:

rol bl,04 ;rotate number left by four bits

mov al,bl ;move lower byte in dl

and al,0fh ; get only LSB

cmp al,09h ;compare with 39h

jg add\_37 ;if grater than 39h skip add 37

add al,30h

jmp skip1 ;else add 30

add\_37: add al,37h

skip1: mov [rdi],al ;store ascii code in result variable

inc rdi ;point to next byte

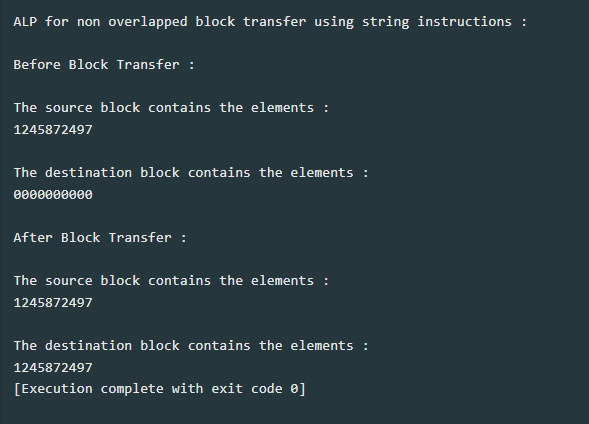
dec cx ;decrement the count of digits to display

jnz up1 ;if not zero jump to repeat

write result , 4

ret

**Output:**

****

**Conclusion:**

This program uses a loop to transfer quadwords (8-byte blocks) from the source buffer to the destination buffer, and then transfers any remaining bytes (less than 8) using a separate loop. The loop counter rcx is divided by 8 at the beginning of the program to get the number of quadwords to transfer, and any remaining bytes are stored in rdx for later use. The and rdx, 7 instruction is used to get the remaining byte count (less than 8) at the end of the loop.

In conclusion, this program demonstrates a simple and efficient way to transfer blocks of data between memory locations without using string specific instructions. It could be used in a variety of applications, such as copying large files or transferring data between processes**.**