MP Practical- 8

Name- Siddhesh Dilip Khairnar

Roll No.- 272028

Batch- B2

Name : Siddhesh Dilip Khairnar
ROLL NO : 272028 ~~ROLL NO : 2720~~
PRN NO : 22110398

PAGE NO. :
DATE : / /

Experiment 8

Aim : Write 64 bit ALP to perform multiplication of two 8 bit hexa decimal number with successive addition

Theory :

- Multiplication by using following method with example :

i Add and shift method :

The method tought in school for multiplying decimal number is based on calculating partial product, shifting them to the left and then adding them to gether. shift and add multiplication is similar to the multiplication performed by paper & pencil. This method adds the multiplicand X to itself Y times, where Y denotes the multiplier. To multiply two number by paper and pencil, the algorithm is to take the digit of the multiplier one at a time from right to left, multiplying the multiplicant by a single digit of the multiplier and placing the intermed product in the appro- -priate position to the left of the earlier result.

Multiplicant 1000 × Partial Product
Multiplier 1001
1000
0000x
0000xx
1000xxx

Product 1001000

TO the case of binary multiplication, since the digit are 0 and 1, each step of the multiplication is simple. if the multiplier digit is 1, a copy of the multiplicand (1 × multiplicand) is placed in the proper position. if the multiplier digit is 0, a number of 0 digit (0 × multiplicant) are placed in the proper position.

PAGE NO.:
DATE: / /

Algorithm:→

1) Start.
2) Initialize required variable in data section
3) Declare required variable in .bss section.
4) Define macros to display message / data & accept number.
5) In text section, display welcome message.
6) Display "Enter the first no" message.

Code:

```
section .data

   msg db "ALP to multiply two 8 bit hex numbers",10
   msg_len equ $ - msg

   opr1 db "multiplicand :  "
   opr1_len equ $ - opr1

   opr2 db 10,"multiplier   :  "
   opr2_len equ $ - opr2

   menu db 10,10,13,"1. Successive Addition Method",10
                db 13,"2. Add and shift method",10
                db 13,"3. Exit",10
                db 10,"Enter your choice (1/2/3) : "
   menu_len equ $ - menu

   alert db 10,"WRONG CHOICE"
   alert_len equ $ - alert

   res db 10,"The product is :  "
   res_len equ $ - res

   msg_end db 10,"End of ALP"
   msg_end_len equ $ - msg_end

section .bss
        multiplier resb 1        ;variable after ASCII to Hex
```

```
   multiplicand resb 1     ;variable after ASCII to Hex
   num resb 03             ;variable before ASCII to Hex
   result resb 04          ;for display procedure
   choice resb 2           ;for choice of user
   product resw 1          ;to store the product

%macro IO 4
   mov rax,%1
   mov rdi,%2
   mov rsi,%3
   mov rdx,%4
   syscall
%endmacro

section .text

global _start
_start:

    xor rax,rax
    xor rbx,rbx
    xor rcx,rcx
    xor rdx,rdx

   IO 1, 1, msg, msg_len

   IO 0 ,0,num,3
   IO 1, 1, opr1, opr1_len
   IO 1,1,num,2     ;to access the data without enter char
```

```
    call convert
    mov [multiplicand],bl

    IO 0 ,0,num,3
    IO 1, 1, opr2, opr2_len
    IO 1,1,num,2
    call convert
    mov [multiplier],bl

    IO 1, 1, menu, menu_len
    IO 0, 0, choice, 2
    IO 1, 1, choice, 2

    cmp byte[choice],31h
    jne case2
    call successive_addition
    jmp endOfProgram

case2:
    cmp byte[choice],32h
    jne case3
    call add_shift
    jmp endOfProgram

case3:
    cmp byte[choice],33h
    je endOfProgram
    IO 1,1,alert,alert_len
```

```
endOfProgram:
   IO 1,1,msg_end,msg_end_len
   mov rax, 60
   mov rdi, 0
   syscall

convert:                              ;; for ASCII to Hex conversion
 xor rbx,rbx
 xor rcx,rcx
 xor rax,rax

 mov rcx,02
 mov rsi,num
 up1:
 rol bl,04

 mov al,[rsi]
 cmp al,39h
 jbe p1
 sub al,07h
 jmp p2
 p1: sub al,30h
 p2: add bl,al     ;bl stores the ASCII equivalent(byte) of the multiplicand/multiplier
 inc rsi
 loop up1
ret

disp:                 ;for Hex to ASCII conversion
 mov rcx,4
```

```
 mov rdi,result
 dup1:
 rol bx,4
 mov al,bl
 and al,0fh
 cmp al,09h
 jbe p3
 add al,07h
 jmp p4
 p3: add al,30h
 p4:mov [rdi],al
 inc rdi
 loop dup1

 IO 1,1,result,4

ret



successive_addition:
   xor rcx,rcx
   xor rax,rax

   mov word[product],0

   mov bl,[multiplier]

   mov al,[multiplicand]
```

```
next:
   add [product],ax
   dec bl
   jnz next
   IO 1, 1, res, res_len
   mov bx,[product]
   call disp
ret


add_shift:
 mov word[product],0

 xor rbx,rbx
 xor rcx,rcx
 xor rdx,rdx
 xor rax,rax

 mov dl,08
 mov al,[multiplicand]
 mov bl,[multiplier]

 p11:
      shr bx,01
 jnc p
 add cx,ax
 p:
      shl ax,01
```

dec dl

jnz p11

mov [product],rcx

IO 1,1,res,res_len

mov rbx,[product]

call disp

ret

Output: