MICROPROCESSORS

Naveen Jhajhriya

Roll No.3233

SE-COMP-A

EXPERIMENT 01

AIM: Write an X86/64 ALP to accept a string and to display its length.

ALGORITHM:

1 Start

2. Declare & initialize the variables in .data section.

3. Declare uninitialized variables in .bss section.

4. Declare Macros for print and exit operation.

5. Initialize pointer to get input string from user.

6. Initialize counter for calculating no. of chatters in string.

7. Display string entered by user.

8. Put value of count in rax register

9. max size of display, for convenience set to 16 and rsi points to output

10. setting rdx to null without setting a null byte (a tip i saw on reddit) needed to clean dl for use

11. Declare the Procedurefor ascii conversion.

12. Stop.

PROGRAM:

```
section .data

    msg db "ALP to display the length of a string
enterd by the user",10

    msg_len equ $ - msg
    msg1 db 10, "String entered by the user is : ",10

    msg1_len equ $ - msg1


    msgop db 10, 10,"Length of the string is : " , 10

    msgop_len equ $ - msgop


section .bss

    string resb 50

    strl equ $-string


    result resb 50


%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 msg1, msg1_len

    mov rax, 0
    mov rdi, 0
    mov rsi, string
    mov rdx, 200
    syscall

    call disp

    mov rax, 60
    mov rdi, 0

    syscall


disp:


    mov rbx,rax ;store number in rbx

    mov rdi, result ;point rdi to result variable

    mov cx,16 ;load count of rotation in cl

up1:

    rol rbx,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 skip ;else add 30

add_37 : add al,37h

skip: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 string, 50

    write msgop,msgop_len



    write result,16 ;call to macro

    ret
```

EXPERIMENT 02

AIM: Write a switch case driven X86/64 ALP to perform 64-bit hexadecimal arithmetic operations (+, -, *, /) using suitable macros. Define procedure for each operation.

ALGORITHM:

1. Start

2. Take input for 2 hexadecimal numbers

3. Take input for arithmetic operation

4. Perform arithmetic using respective instructions (add, sub, mul, div) and registers (rax, rdx)

5. Print output

6. Stop

PROGRAM:

```
%macro IO 4

mov rax,%1

mov rdi,%2

mov rsi,%3

mov rdx,%4

syscall

%endmacro

section .data

    m1 db "enter choice (+,-,*, /)" ,10 ; 10d -> line
feed

    l1 equ $-m1

    m2 db "Write a switch case driven X86/64 ALP to
perform 64-bit hexadecimal arithmetic operations
```

```
(+,-,*, /) using suitable macros. Define procedure
for each operation." ,10

    l2 equ $-m2

    m3 db "rahul ghosh 3236" ,10

    l3 equ $-m3

    madd db "addition here" ,10

    l4 equ $-madd

    msub db "subtraction here" ,10

    l5 equ $-msub

    mmul db "multiplication here" ,10

    l6 equ $-mmul

    mdiv db "division here" ,10

    l7 equ $-mdiv

    mspace db 10

    m_result db "result is "

    m_result_l equ $-m_result

    m_qou db "qoutient is "

    m_qou_l equ $-m_qou

    m_rem db "remainder is "

    m_rem_l equ $-m_rem

    m_default db "enter correct choice",10

    m_default_l equ $-m_default

section .bss

    choice resb 2

    _output resq 1

    _n1 resq 1

    _n2 resq 1
    temp_1 resq 1
    temp_2 resq 1
section .text
    global _start
_start:
    IO 1,1,m2,l2
    IO 1,1,m3,l3
    IO 1,1,m1,l1
    IO 0,0,choice,2
    cmp byte [choice],'+'
    jne case2
    call add_fun
    jmp exit
case2:
    cmp byte [choice],'-'
    jne case3
    call sub_fun
    jmp exit
case3:
    cmp byte [choice],'*'
    jne case4
    call mul_fun
    jmp exit
case4:
    cmp byte [choice],'/'
    jne case5
    call div_fun
    jmp exit
case5:
    cmp byte [choice],'a'
    jne error
    call add_fun
    call sub_fun
    call mul_fun
    call div_fun
    jmp exit
error:
    IO 1,1,m_default,m_default_l
    jmp exit
exit:
    mov rax, 60
    mov rdi, 0
    syscall
add_fun:
    IO 1,1,madd,l4
    mov qword[_output],0
    IO 0,0,_n1,17
    IO 1,1,_n1,17
    call ascii_to_hex
    add qword[_output],rbx
    IO 0,0,_n1,17
    IO 1,1,_n1,17
    call ascii_to_hex
    add qword[_output],rbx
    mov rbx,[_output]
    IO 1,1,mspace,1
    IO 1,1,m_result,m_result_l
    call hex_to_ascii
    ret
sub_fun:
    IO 1,1,msub,l5
    mov qword[_output],0
    IO 0,0,_n1,17
    IO 1,1,_n1,17
    ;IO 1,1,mspace,1
    call ascii_to_hex
    add qword[_output],rbx
    IO 0,0,_n1,17
    IO 1,1,_n1,17
    ;IO 1,1,mspace,1
    call ascii_to_hex
    sub qword[_output],rbx
    mov rbx,[_output]
    IO 1,1,mspace,1
    IO 1,1,m_result,m_result_l
    call hex_to_ascii

    ret
```

```
mul_fun:
    IO 1,1,mmul,16 ; message
    IO 0,0,_n1,17    ; n1 input
    IO 1,1,_n1,17
    call ascii_to_hex; conversion returns hex value in rbx
    mov [temp_1],rbx ; storing hex in temp_1
    IO 0,0,_n1,17    ;n2 input
    IO 1,1,_n1,17
    call ascii_to_hex
    mov [temp_2],rbx ; putting hex of n2 in temp_2
    mov rax,[temp_1] ; temp_1->rax
    mov rbx,[temp_2] ;temp_2->rbx
    mul rbx          ; multiplication
    push rax
    push rdx
    IO 1,1,mspace,1
    IO 1,1,m_result,m_result_l
    pop rdx
    mov rbx,rdx; setting rbx value for conversion
    call hex_to_ascii
    pop rax
    mov rbx,rax; setting rbx value for conversion
    call hex_to_ascii  ; final output
ret
div_fun:
    IO 1,1,mdiv,17
    IO 0,0,_n1,17    ; n1 input
    IO 1,1,_n1,17
    call ascii_to_hex; conversion returns hex value in rbx
    mov [temp_1],rbx ; storing hex in temp_1
    IO 0,0,_n1,17    ;n2 input
    IO 1,1,_n1,17
    call ascii_to_hex
    mov [temp_2],rbx ; putting hex of n2 in temp_2
    mov rax,[temp_1] ; temp_1->rax
    mov rbx,[temp_2] ;temp_2->rbx
    xor rdx,rdx
    mov rax,[temp_1] ; temp_1->rax
    mov rbx,[temp_2] ; temp_2->rbx
    div rbx ; div
    push rax
    push rdx
    IO 1,1,mspace,1
    IO 1,1,m_rem,m_rem_l
    pop rdx
    mov rbx,rdx
    call hex_to_ascii; remainder output
    IO 1,1,mspace,1
    IO 1,1,m_qou,m_qou_l
    pop rax
    mov rbx,rax
    call hex_to_ascii; quotient output
    ret
ascii_to_hex:
    mov rsi, _n1
    mov rcx, 16
    xor rbx, rbx
    next1:
        rol rbx, 4
        mov al, [rsi]
        cmp al,47h
    jge error
        cmp al, 39h
        jbe sub30h
        sub al, 7
        sub30h:
            sub al, 30h
        add bl, al
        inc rsi
        loop next1
ret
hex_to_ascii:
    mov rcx, 16
    mov rsi,_output
    next2:
        rol rbx, 4
        mov al, bl
        and al, 0Fh
```

```
         cmp al, 9
         jbe add30h
         add al, 7
        add30h:
         add al, 30h
         mov [rsi], al
         inc rsi
         loop next2
         IO 1,1,_output,16
         IO 1,1,mspace,1
ret
```

EXPERIMENT 03

AIM: Write X86/64 ALP to convert 4-digit Hex number into its equivalent BCD number and 5- digit BCD number into its equivalent HEX number. Make your program user friendly to accept the choice from user for: (a) HEX to BCD b) BCD to HEX (c) EXIT.

ALGORITHM:

Algorithm for HEX to BCD conversion procedure

i. Start

ii. Display ‘Input 4 digit hex number’ message using Display macro

iii. Accept 4 digit HEX number from user using accept macro and store it

in num variable.

iv. Call Ascii_to_Hex procedure to convert accepted ascii value of num

digit into hexadecimal number.

v. Load result of step iv in RAX

vi. Initialise RCX=0005 i.e. number of times to divide the number by 0Ah

vii. Load RDX=0000

viii. Load RBX=000Ah,

ix. Divide the number using DIV RBX instruction, which produces

Quotient in RAX and remainder in RDX

x. Push DX on stack

xi. Decrement RCX, If not zero jump to step vii, else continue

xii. Display the result message using Display macro

xiii. Load RCX=0005, number of digits to display

xiv. Pop the last pushed remainder in DX for display

xv. Add 30H in DL to produce ASCII code of the digit, and display digit display macro.

xvi. Decrement RCX, If not zero jump to step xiv

xvii. Return

<u>Algorithm for BCD to HEX conversion procedure</u>

i. Start

ii. Display ‘Input 5 digit BCD number’ message using Display macro

iii. Accept 5 digit BCD number from user using accept macro and store it in num varialble.

iv. Initialize RAX=0

v. Load RBX=000Ah

vi. Make ESI point to num

vii. Initialise RCX=0005 i.e. number of times to multiply the previous number number by 0Ah and add new digit of BCD.

viii. Multiply RAX by RBX using MUL RBX instruction.

ix. Load RDX=0000

x. Move value pointed by ESI to dl register

xi. Sub 30H in DL to produce numeric code of the ascii digit stored in num variable

xii. Add RAX and RDX using ADD instruction

xiii. Decrement RCX, If not zero jump to step viii, else continue

xiv. Display the result message using Display macro

xv. Load value of RAX (HEX result) in RBX

xvi. Call Procedure Hex_to_Ascii to convert result of step xv. into ascii form to display it on standard output.

xvii. Return

PROGRAM:

```
%macro display 2

    mov rax,01

    mov rdi,01

    mov rsi,%1

    mov rdx,%2

    syscall

%endmacro



%macro accept 2

    mov rax,00

    mov rdi,00

    mov rsi,%1

    mov rdx,%2

    syscall

%endmacro



section .data

        menu db 10d,13d,"MENU"
```

```
             db 10d,"1. Hex to BCD"
             db 10d,"2. BCD to Hex"
             db 10d,"3. Exit"
             db 10d,"Enter your choice: "
        menulen equ $-menu
        m1 db 10d,13d,"Enter Hex Number: "
        l1 equ $-m1
        m2 db 10d,13d,"Enter BCD Number: "
        l2 equ $-m2

        m3 db 10d,13d,"Equivalent BCD Number: "
        l3 equ $-m3
        m4 db 10d,13d,"Equivalent Hex Number: "
        l4 equ $-m4
section .bss
        choice resb 1
        num resb 16
        output resq 1
        factor resq 1
```

```
section .text
    global _start
_start:
    display menu,menulen
    accept choice,2
    cmp byte[choice],'3'
    jae exit
    cmp byte[choice],'1'
    je hex2bcd
    cmp byte[choice],'2'
    je bcd2hex
exit:
    mov rax,60
    mov rdx,0
    syscall


hex2bcd:
    display m1,l1
    accept num,17
    call asciihex_to_hex

    mov rax,rbx
   mov rbx,10
   mov rdi,num+15
loop3:
   mov rdx,0
   div rbx
   add dl,30h
   mov [rdi],dl
   dec rdi
   cmp rax,0
   jne loop3

    display m3,l3
    display num,16
jmp _start
```

```
bcd2hex:
    display m2,l2
    accept num,17

    mov rcx,16
   mov rsi,num+15
   mov rbx,0
   mov qword[factor],1

loop4:
   mov rax,0
   mov al,[rsi]
        sub al,30h
   mul qword[factor]
   add rbx,rax
   mov rax,10
   mul qword[factor]
   mov qword[factor],rax
   dec rsi

   loop loop4



   display m4,l4

   mov rax,rbx

   call hex_to_ascii

jmp _start

asciihex_to_hex:

   mov rsi,num

   mov rcx,16

   mov rbx,0

   mov rax,0



loop1:

    rol rbx,04

   mov al,[rsi]

   cmp al,39h

   jbe skip1

   sub al,07h
```

```
skip1:
    sub al,30h
   add rbx,rax
   inc rsi
   dec rcx
   jnz loop1
ret


hex_to_ascii:
   mov rsi, output
   mov rcx, 16

   next2:
       rol rbx, 4
       mov al, bl
       and al, 0Fh
       cmp al, 9
       jbe add30h
       add al, 7

       add30h:
           add al, 30h

       mov [rsi], al
       inc rsi
       loop next2
   display output,16
ret
```