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 6

Title :-

Switch case driven ALP to perform 64-bit hexadecimal arithmetic operations (+,- ,*,%) using macros.

Problem Statement :-

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

Sw/Hw requirements :-

Intel i5 8th generation 64 bit processor

OS - Ubuntu 16.0 LTS

Editor : - VS- Code , Gedit

Assembler: Nasm

Debugger : gdb

Objective :-

1. To learn how to use macros

2. To learn about switch case

3. To perform operations on number

Theory :-

Consider that five numbers are stored in the array. The procedure is written for all operations. The format is .procname--- lines of code --- ret The macro is written for display interrupt. The format is macroname macro %arg1 %arg2 end The switch case format is also written for scanning the choice of user

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. sub:
   Description: This is going to subtract the number in source from number in destination. The result is stored in destination.

   Flag: CF, ZF, OF, PF

   e.g. sub eax, ebx

3. mul:
   Description: This is going to multiply the number in accumulator to the number in destination. The result is stored in accumulator

   Flags: CF, ZF, OF, PF

   e.g. mul ebx

4. div:

   Description: his is going to divide the number in accumulator by the number in destination. The quotient is in accumulator and reminder in edx/dx/dl

   Flags: CF, ZF, OF, PF

   e.g. div 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

Algorithm:

1) Initialize array of five 64 bit numbers.
2) Put counter for the array
3) Ask user for its choice.
4) If choice is 1, call addition procedure.
5) If choice is 2, call subtraction procedure.
6) If choice is 3, call multiplication procedure.
7) If choice is 4, call division procedure.8) If choice is different the exit from the program.
9) Display the result as per user choice

Conclusion :-

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

Program : -

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


section .data
        arr dq 0000000000000003h,0000000000000002h
        n equ 2


        menu db 10d,13d,"**********MENU**********"
            db 10d,13d,"1. Addition"
            db 10d,13d,"2. Subtraction"
            db 10d,13d,"3. Multiplication"
            db 10d,13d,"4. Division"
```

```
        db 10d,13d,"5. Exit"
        db 10d,13d,"Enter your Choice: "
    menu_len equ $-menu

    m1 db 10d,13d,"Addition: "
    l1 equ $-m1
    m2 db 10d,13d,"Substraction: "
    l2 equ $-m2
    m3 db 10d,13d,"Multiplication: "
    l3 equ $-m3
    m4 db 10d,13d,"Division: "
    l4 equ $-m4

section .bss
    answer resb 16          ;to store the result of operation
    choice resb 2
```

```
section .text
        global _start:
        _start:
        up:       scall 1,1,menu,menu_len
                  scall 0,0,choice,2
        cmp byte[choice],'1'
        je case1
        cmp byte[choice],'2'
        je case2
        cmp byte[choice],'3'
        je case3
        cmp byte[choice],'4'
        je case4
        cmp byte[choice],'5'
        je case5


        case1:  scall 1,1,m1,l1
                call addition
```

```
        jmp up
    case2:  scall 1,1,m2,l2
            call substraction
            jmp up
    case3:  scall 1,1,m3,l3
            call multiplication
            jmp up
    case4:  scall 1,1,m4,l4
            call division
            jmp up


    case5:  mov rax,60
            mov rdi,0
           syscall
;procedures for arithmetic and logical operations
addition:
    mov rcx,n
    dec rcx
```

```
        mov rsi,arr
        mov rax,[rsi]
up1:    add rsi,8
        mov rbx,[rsi]
        add rax,rbx
        loop up1
        call display
ret



substraction:
        mov rcx,n
        dec rcx
        mov rsi,arr
        mov rax,[rsi]
up2:    add rsi,8
        mov rbx,[rsi]
        sub rax,rbx
```

```
        loop up2
        call display
ret
multiplication:
        mov rcx,n
        dec rcx
        mov rsi,arr
        mov rax,[rsi]
up3:    add rsi,8
        mov rbx,[rsi]
        mul rbx
        loop up3
        call display
ret
division:
        mov rcx,n
        dec rcx
        mov rsi,arr
```

```
        mov rax,[rsi]
up4:    add rsi,8
        mov rbx,[rsi]
        mov rdx,0
        div rbx
        loop up4
        call display
ret
or:
        mov rcx,n
        dec rcx
        mov rsi,arr
        mov rax,[rsi]
up6:    add rsi,8
        mov rbx,[rsi]
        or rax,rbx
        loop up6
        call display
```

```
ret
xor:
        mov rcx,n
        dec rcx
        mov rsi,arr
        mov rax,[rsi]
up7:    add rsi,8
        mov rbx,[rsi]
        xor rax,rbx
        loop up7
        call display
ret
and:
        mov rcx,n
        dec rcx
        mov rsi,arr
        mov rax,[rsi]
up8:    add rsi,8
```

```
        mov rbx,[rsi]
        and rax,rbx
        loop up8
        call display
rer
display:
        mov rsi,answer+15
        mov rcx,16
cnt:    mov rdx,0
        mov rbx,16
        div rbx
        cmp dl,09h
        jbe add30
        add dl,07h
add30:  add dl,30h
        mov [rsi],dl
        dec rsi
        dec rcx
```

```
        jnz cnt

        scall 1,1,answer,16

ret
```

OUTPUT:-