**Assignment No:** 1

**DATE :** 25/ 05/ 2021

**ROLL NO.:** 21129

**Problem Statement:**

Write an x86/x64 ALP to accept five 64 bit hexadecimal no. from the user and store them in an array and display the accepted numbers.

**Learning objective:**

To understand how to accept a 64 bit number from the user

**Learning Outcome:**Students will be able to:

Write and execute assembly language program using concept of instruction set, read write system calls

**S/w and H/w Requirements**:

64 bit UNIX based OS eg. Linux Ubuntu 20.04.2 LTS 64-bit Netwide Assembler, 8 GB RAM, Intel i5-8300H, 4 Core, 8 logical processors

**Theory:**

1. **Template for NASM:**
   a. Section .data (Data Segment): The data section is only for initialized data.
   b. Section .bss (Data Segment): The .bss section for uninitialized data.
   c. Section .text (Text Segment): Here the body of code is

written as follows:
global _start
section .text
-start_

2. **Define directives:**
   5 basic define directives are:
   a. db - define a byte
   b. dw - define a word
   c. dd - define a double word
   d. dq - define a quad word
   e. dt - define a ten byte

   Syntax- <variable name> <directive> <value>

3. **Reserve Directives:**
   5 basic reserve directives are:
   a. resb - reserve a byte
   b. resw - reserve a word
   c. resd - reserve a double word
   d. resq - reserve a quad word
   e. rest - reserve a ten byte

   Syntax- <variable name> <directive> <no. of values>

4. **Instruction:**
   a. mov - This instruction is used to move the content of the 2nd parameter to the first one.
      Eg.: mov ax, 35h
   b. add - This instruction is used to increase the value of the 1st parameter by adding the 2nd parameter to it.
      eg.: add rax, 12

c. Dec - This instruction is used to decrement the value to directive or register by 1.
   eg.: dec byte[count]

5. **Macros:**

   A macros is a sequence of instructions assigned by a name and could be used anywhere in the program. In NASM, macros are defined with %macro and %end macro.
   Syntax:
   %macro <macro-name> <no. of parameters>
   <macro body>
   %end macro

6. System Calls:
   System calls are made to access the kernel to execute parameter snippets of codes.
   a. read syscall:
      e.g.:
      ```
      num db 2
      mov rax, 00
      mov rdi, 00
      mov rsi, num
      mov rdx, 01h
      syscall
      ```
   b. write syscall:
      e.g:
      ```
      str db “Hello World”
      mov rax, 00
      mov rdi, 01
      mov rsi, str
      ```

```
        mov rdx, 08h
```

**Algorithm:**

1. Start.
2. Set rbx to 00.
3. Move 5 to count.
4. Use rsi to point to arr.
5. Write read write syscall.
6. Add 17 to rbx.
7. Decrement count.
8. Use for Jump if not zero.
9. Use instructions above with write syscall.
10. Use Exit syscall.
11. End.

| **Char** | **Hex** | | **Memory Address** |
|---|---|---|---|
| newline | Ah | 0000 1010 | 4202605 |
| R | 52h | 0101 0010 | 4202604 |
| | | | |
| newline | Ah | 0000 1010 | 4202589 |
| 1 | 31h | 0011 0001 | 4202588 |
| 1 | 31h | 0011 0001 | 4202587 |
| | | | |
| newline | Ah | 0000 1010 | 4202573 |
| 5 | 35h | 0011 0101 | 4202572 |
| 0 | 30h | 0011 0000 | 4202571 |

|  |  |  |  |
|---|---|---|---|
| 1 | 31h | 0011 0001 | 4202570 |
|  |  |  |  |
| newline | Ah | 0000 1010 | 4202557 |
| d | 64h | 0110 0100 | 4202556 |
| c | 63h | 0110 0011 | 4202555 |
| b | 62h | 0110 0010 | 4202554 |
| a | 61h | 0110 0001 | 4202553 |
|  |  |  |  |
| newline | Ah | 0000 1010 | 4202541 |
| 3 | 33h | 0011 0011 | 4202540 |
| b | 42h | 0100 0010 | 4202539 |
| 2 | 32h | 0011 0010 | 4202538 |
| a | 41h | 0100 0001 | 4202537 |
| 1 | 31h | 0011 0001 | 4202536 |

**Code:**

```
%macro msgmarco 1
        mov rax, 01
        mov rdi, 01
        mov rsi, %1
        mov rdx, 20
        syscall
%endmacro

%macro rwmarco 1
        mov rax, %1
        mov rdi, %1
        add rsi, rbx
        mov rdx, 17
        syscall
        add rbx, 17
        dec byte[count]
%endmacro
section .data
        msg1 db "Enter the numbers: ",10
        msg2 db 10,"The numbers are : ",10
section .bss
        arr resb 85
        count resb 1
global _start
section .text
        _start:
```

```
        msgmarco msg1
        call setarr
        l1:
                rwmarco 0
                jnz l1
        msgmarco msg2
        call setarr
        l2:
                rwmarco 1
                jnz l2
        mov rax, 60
        mov rdi, 00
        syscall

        setarr:
                mov rbx, 00
                mov byte[count], 05
                mov rsi, arr
                ret
;nasm -f elf64 hello.asm && ld -s -o hello hello.o && ./hello
```

**Output:**

```
Enter the numbers:

1234567891234567

26854

6987

565486

561433665


 The numbers are :

1234567891234567

26854

6987

565486

561433665
```

**Conclusion:**

Hence, we have successfully accepted five 64 bit hexadecimal no. from the user and stored them in an array and displayed the accepted numbers.