PICT, PUNE

Subject: Microprocessor Lab

Roll No 21118

Assignment 01

Date of performance: 25-Feb-2021

Problem Statement:

Write an X86/64 ALP to accept five 64-bit Hexadecimal numbers from user & store them in array and display accepted numbers.

Software & hardware Requirements:

Operating System: Virtual Box is installed on windows 10. In VM ubuntu 20.04 OS is used.
for VM base memory allocated is 4096 MB with 3-processors.

Text Editor: gedit is used as text editor.
Version: 3.36.2

Assembler: NASM : Version: 2.14.02

Hardware of PC:

1) Manufacturer & model: Acer Swift 3 (SF314-55G)
2) Processor: Intel Core i5-8265U @ 1.60GH 1801GHz
3) Installed Memory (RAM): 800 GB (7.85 GB ~~Available~~ usable)
4) System Type: 64-bit OS, x64 based processor

Theory:

A) System Calls:

- System calls is a way for a program to interact with operating system.
  eg. usage: for display on console screen, to exit from the program etc.
- syntax of system call is
  1) mov rax {system call no}
  2) mov rdi, {file descriptions}

iv) mov rsi, {source/buffer/variable name}
v) mov rdx, {length in bytes}
vi) syscall

3 main system calls:

① for taking input from user / read syscall

```
mov rax, 00      ; 00 is system call number
mov rdi, 01
mov rsi, num     ; num is a variable
mov rdx, 01      ; size of num is 01 bytes
syscall.
```

② for printing / write syscall

```
mov rax, 01      ; 01 is system call no
mov rdi, 01
mov rsi, num     ; num is variable
mov rdx, 01      ; 01 bytes of num will be
syscall            displayed.
```

③ Exit syscall

```
mov rax, 60      ; system call no.
mov rdi, 00      ; 00 is file descriptor
syscall          ; call to interrupt.
```

B) Instructions used:

① MOV:

- it is a data movement instruction
- transfers byte, word or double word from source operand to destination operand.
- syntax:

```
mov [dest], [src]
```

- cases & examples

Roll No 21118

PICT, PUNE

register to memory: mov [sum], al
memory to register: mov bl, [sum]
between general registers: mov al, bl.
immediate data to register: mov al, 08h
immediate data to memory: mov [sum], 05h

② ADD instruction
- ~~syntax~~: It is a binary arithematic instruction
- syntax
  ADD [dest], [src]
- it replaces destination operand with the sum of source & destination operand.
- It sets CF if overflow
  eg let al contains 10h & bl contains 0B then instruction ADD al, bl, will change data of al to 1B & bl to 0B (no change).

③ DEC instruction:
- It is arithematic instruction
- It substract 1 from destination operand
- syntax:
  DEC [desb]
  eg let al contain 1B, then instruction DEC al will change data of al to 1A.

④ JNZ instruction:
- It is a conditional transfer instruction
- In jnz, ZF (zero flag) is tested
- The controls transfers if ZF is not zero.
- syntax:

PICT, PUNE

JNZ [l] ; l is the label

eg. consider the following code snippet:

```
mov [cnt], 03h
l:
    ; system call to print 'msg'
    dec [cnt]
    jnz l
```

each time before reaching jnz value of cnt will dec by 1. first two times value of cnt will be 2 & 1 hence jnz will move control to label l. third time, when value of cnt becomes 0, due to which zf will be set. This time jnz will not affect the execution.

~~Algorithm:~~

The template of assembly program is as follows:

```
section .data   ; defining data
  —
  —
section .bss    ; declaring data
  —
  —
global _start   ; program accesibility
section .text   ; code section
_start:
  —
  —
  .
  .
```

Roll No. 21118

PICT, PUNE

3 sections are there:
1) data section : for defining data
2) bss section: for declaring data
3) text section: for writing instructions.

- files are saved with .asm format.
- comments start with ; (semicolon)

Directives:

syntax:
<var_name> <directive> <value>
var_name: any allowed variable name
directive: one of the: db, dw, dd, dq, dt
value: value to be stored in variable.

Variable declaration:

syntax:
<var_name> <directive> <count_of_variable>
var_name: any allowed variable name.
directive: one among resb, resw, resd, resq, rest
count_of_variable: count of variable.
eg 1) num resb 1
2) num resq 1
3) num resb 10 ; arrays.

Algorithm:

① Delete Byte array of 85 bytes
for ~~4 5 64 bit~~ 5 64-bit number 80 bytes will be required. & 5 bytes will be required for enter key.
∴ Total 85 bytes of memory should be reserved.

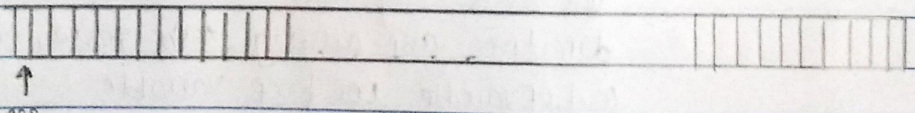
PICT, PUNE

② Set pointer to array
③ Set counter to 5
④ Read 64-bit number
⑤ Increment the pointer
⑥ Decrement counter
⑦ Jump to step ④ if zero flag is not set
⑧ Set pointer to same array
⑨ Set counter to 5.
⑩ print 64-bit number
⑪ Increment the pointer
⑫ Decrement counter
⑬ Jump to step ⑩ if zero flag is not set.
⑭ Exit system call

Memory Illustration

Reding The Numbers:

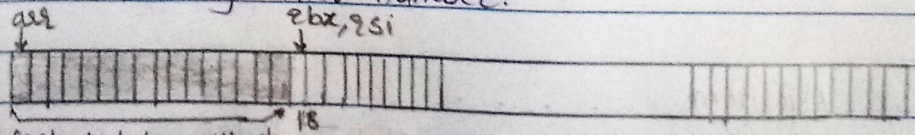
initially:



esi, ebx, esi

fig. block of 85-bytes in memory

After Reading first number:

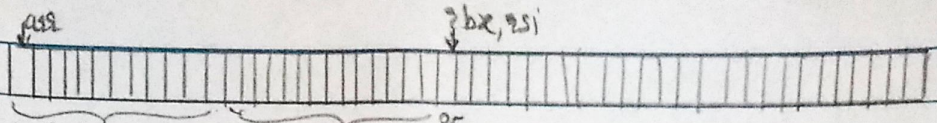


first 16-bytes will store number + 17-byte for enter key

After Reading second number:

Roll No 21118

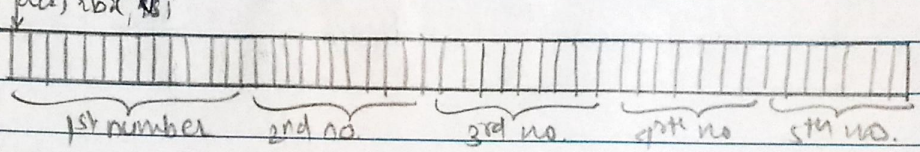
PICT, PUNE



Same will continue for remaining 3 numbers

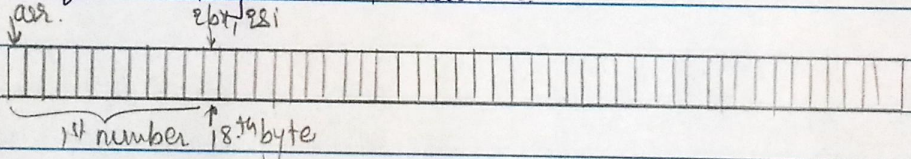
Writing the numbers:

Initially:



first 17 bytes will be printed.

After writing second number.



same will continue for remaining 4 numbers

Note that here we are accepting numbers from the user through keyboard, hence ASCII values will be accepted. Each keystroke will require 8-bit / 1byte of memory. Hence to store quadword we require 16bytes in memory.

Conclusion:

In this programming assignment, we learned to take input from the user through keyboard, use of registers, and about conditional transfer instructions