Page :
Date : / /

Assignment 1

Date :

Title : Write X86/64 ALP to count number of positive and negative numbers from the array.

Objective : To understand how to identify a positive number or negative number.

Outcome : Students will be able to use different index registers and find positive and negative numbers from array.

Theory :

A system call is essentially a function call which changes the CPU into kernel mode and executes a function which is part of the kernel. When you run a process on Linux it runs in user mode which means that it is limited to executing only "safe" instructions. It can move data within the program, do arithmetic, do branching, call functions etc, but there are instructions which your program can't do directly. For example it would be unsafe to allow only pro~~g~~gram to read or write directly to the disk device, so this is prevented by preventing user programs from executing input or output instructions.
System calls which are generally used are :

Page :
Date : / /

i) Write ii) read iii) exit

i) Sys_Write :-
The system service to output characters to the console is the system write (sys_write) The arguments for the write system service are as follows:

Register	Sys_write.
rax	~~Call~~ System call no. (1)
rdi	File descriptor no. (1)
rsi	Starting address.
rdx	size/length.

Example:
```
section. data
msg : db "hello"
len : equ $-msg
section .text
global main
main:
  mov rax, 1
  mov rdi, 1
  mov rsi, msg
  mov rdx, len
   syscall
```

Output : hello

Page :
Date : / /

ii) Sys_read :

The system service to read characters from the console is the system read (sys_read)
The arguments for the read system service are as follows:

Register	sys_read
rax	system call no. (0)
rdi	File descriptor no. (0/1)
rsi	Address of where to store characters
rdx	Size /length.

Example :

```
Section .bss
m : resb 3
section.text
mov rax,0
mov rdi,0
mov rsi,m
mov rdx,3
 Syscall
```

iii) sys_exit:

The system service to exit a running process is the system exit.

Register	sys_exit
rax	system call no. (60)
rdi	File descriptor no. (0)

Scanned with CamScanner

Commands to execute 64 bit assembly language program (NASM)

1) To assemble the program:

```
nasm -f elf64 filename.asm
```

where

nasm = Netwide assembler
-f = specifies the output file format
elf64 = Generates executable and linkable format object files.

4

2) To link

```
ld -o fullname filename.o
```

where -o is name of output file.
filename.o is the Object file.

3) To execute

```
./filename.
```

Datatypes used in assembly language program

1) db (define byte) (1 byte)
2) dw (define word) (2 bytes)
3) dd (define double word) (4 bytes)
4) dq (define quad word) (8 bytes)

Bit Test Instruction (BT)

It takes several instructions to extract or insert a bit field. Sometimes you need to extract or insert a single bit. However it can be simple and quicker to use the bit test instruction (bt) and either the bit Test and set instruction (bts) or the bit test and reset instruction (btr).

The bit instruction has 2 operands. The first operand is a 16, 32 or 64 bit word in memory or a register which contains the bit bit to test The second operand is the bit number from 0 to number of bits minus 1 for the word size which is either an immediate value or a value in a register. The bt instruction set the carry flag to the value of the bit being tested.

In computer system, the negative number is represented by different ways:

1) Sign-Magnitude representation:

It is simply an ordinary binary number with one digit placed in front to represent the sign. If this extra digit is a '1', it means that the rest of the digit represent a negative number. However if the same set of digits are used but the extra digit is a '0'. it means that the number is a positive one.

Example:

0100 = +4
1100 = -4

Page :
Date : / /

2) 2's complement representation:
The 2's complement is the binary number that results when we add 1 to the 1's complement. Example
0110 = +6
1010 = -6

Algorithm :

```
1) Start
2) Initialize an array of 10 numbers or accept 10 numbers from user and store them in one array.
3) Initialize pos_counter = 0 and neg_counter = 0, index_reg = array, counter = 10.
4) Read the number from index_reg into a register
5) Perform addition with 00H and check sign bit
6) If sign bit == 1 then.
    increment neg_counter = neg_counter + 1
    else
      pos_counter = pos_counter + 1
    end if.
7) Increment index_reg = index_reg + 1
8) Decrement counter = counter - 1
9) If counter != 0 then goto step number 4
```

Page :
Date : / /

Test cases:

Testcase	Output	Expected	Success / Fail
1) arr: 0X1234, 0X 5678, 0x BC23 0x F123	positive no.s 2 negative no.s 2	Same as output	Success
2) arr: dw -0X D2E4, -0xFBCA, -0X3F69, -0XF543, -0XABCD	positive no. 3 negative no. 2	positive no. 2 negative no. 3	Fail.

Conclusion : Using the different instructions of 8086 we ~~implemented~~ wrote a program to count the number of positive and negative numbers.

[illegible]