A Simple Operating System

Group-01

Group Members:

Tasbiraha Athaya-201414014

Sanjida Akter Sharna-201414021

Namrata Saha-201414025

Introduction:

- We are writing and building our first operating system in x86 assembly language.
- Our OS will do simple Addition and Subtraction operation.

Requirements:

- Installing Ubuntu.
- The fundamentals of the PC boot process
- Assembly language

Installing Ubuntu and Getting Tools:

We are using Linux. So we needed to install Ubuntu. Then to get all the tools, we entered following command in a terminal window:

```
sudo apt-get install build-essential qemu nasm
```

This gets us the development toolchain (compiler etc.), QEMU PC emulator and the NASM assembler, which converts assembly language into raw machine code executable files.

PC Boot Process:

- Power on: The PC starts up and begins executing the BIOS code.
- The BIOS looks for various media such as a floppy disk or hard drive.
- The BIOS loads a 512 byte boot sector from the specified media and begins executing it.
- Those 512 bytes then go on to load the OS itself, or a more complex boot loader.

Assembly Language:

- Our code of addition and subtraction is written in assembly language. The code is given here:

myfirst.asm

```
BITS 16

start:
        mov ax, 07C0h           ; Set up 4K stack space after this bootloader
        add ax, 288             ; (4096 + 512) / 16 bytes per paragraph
        mov ss, ax
        mov sp, 4096

        mov ax, 07C0h           ; Set data segment to where we're loaded
        mov ds, ax


        mov si, text_enter      ; Put string position into SI
        call print_string       ; Call our string-printing routine

        mov ah,00h      ;newline
        int 16h

        mov al,0Ah
        mov ah,0eh
        int 10h

        mov al,0dh
        mov ah,0eh
        int 10h



        mov ah,00h
        int 16h         ;it takes input from the keystroke and (1st input)
        mov ah,0eh
        mov dl,al
        mov cl,al
        sub cl,48
        int 10h
```

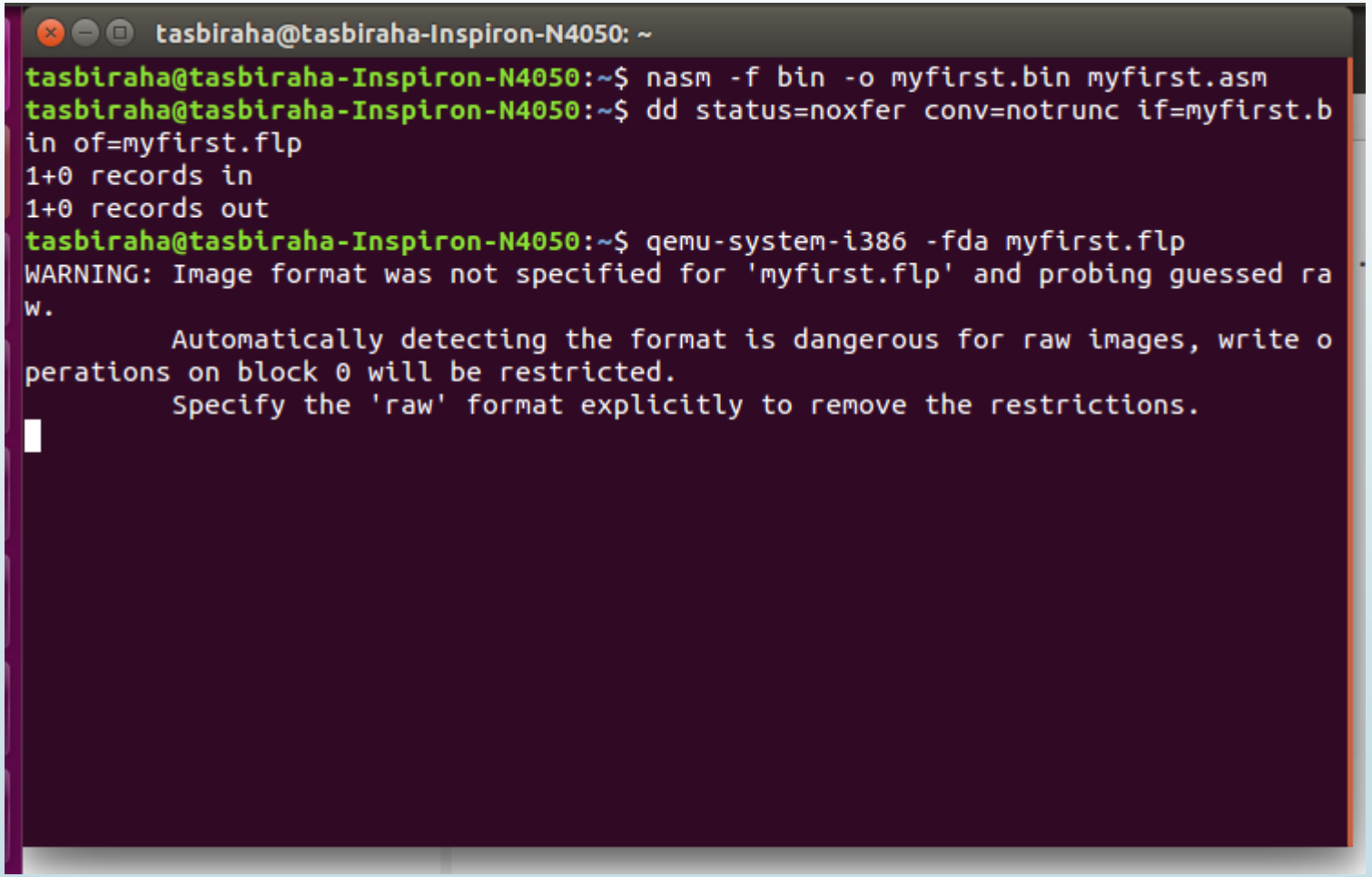
```
mov ah,00h          ;newline
 int 16h

 mov al,0Ah
 mov ah,0eh
 int 10h

 mov al,0dh
 mov ah,0eh
 int 10h




 mov ah,00h
 int 16h            ;it takes input from the keystroke and (2nd input)
 mov ah,0eh
 mov dl,al
 mov bl,al
 sub bl,48
 int 10h

 mov ah,00h         ;newline
 int 16h

 mov al,0Ah
 mov ah,0eh
 int 10h

 mov al,0dh
 mov ah,0eh
 int 10h
```

```
mov si, text_add ; Put string position into SI
 call print_string

 mov al,cl      ; add
 add al,bl
 add al,48
  mov dl,al
 mov ah,0eh
 int 10h

 mov ah,00h       ;newline
 int 16h

 mov al,0Ah
 mov ah,0eh
 int 10h

 mov al,0dh
 mov ah,0eh
 int 10h



 mov si, text_sub         ; Put string position into SI
 call print_string

 mov al,cl    ;sub
 sub al,bl
 add al,48
 mov dl,al
 mov ah,0eh
 int 10h
```

```
        jmp $                       ; Jump here - infinite loop!


        text_string db 'This is my cool new OS!', 0
        text_enter db 'enter two number:1st one must be greater', 0
        text_add db 'Addition operation:', 0
         text_sub db 'subtract operation:', 0




        ;in1 db 0
        ;in2 db 0


print_string:                       ; Routine: output string in SI to screen
        mov ah, 0Eh                 ; int 10h 'print char' function

.repeat:
        lodsb                       ; Get character from string
        cmp al, 0
        je .done                    ; If char is zero, end of string
        int 10h                     ; Otherwise, print it
        jmp .repeat

.done:
        ret


        times 510-($-$$) db 0       ; Pad remainder of boot sector with 0s
        dw 0xAA55                   ; The standard PC boot signature
```

Next Step:

In a terminal window, in home directory, we entered below command:



Command Explanation:

The first command is:

```
nasm -f bin -o myfirst.bin myfirst.asm
```

Here we assemble the code from our text file into a raw binary file of machine-code instructions. With the `-f bin` flag, we tell NASM that we want a plain binary file. The `-o myfirst.bin` part tells NASM to generate the resulting binary in a file called `myfirst.bin` .

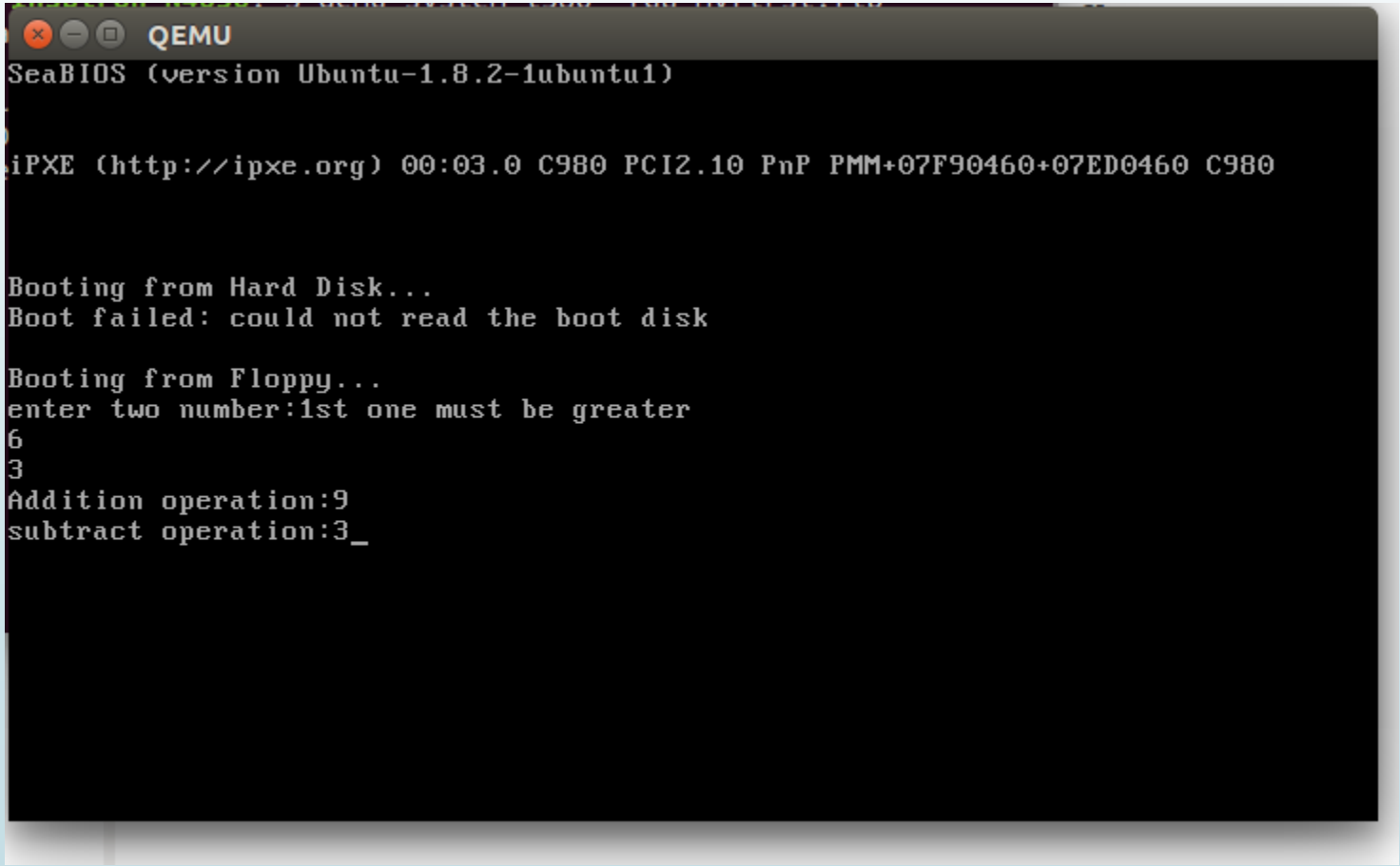
We needed a virtual floppy disk image to which we can write our bootloader-sized kernel. We copied mikeos.flp from the disk_images/ directory of the MikeOS bundle into your home directory, and rename it myfirst.flp. Then entered the second command:

```
dd status=noxfer conv=notrunc if=myfirst.bin of=myfirst.flp
```

This uses the 'dd' utility to directly copy our kernel to the first sector of the floppy disk image.

The last command is for booting our new OS using the QEMU PC emulator.

And there we are! Our OS will boot up in a virtual PC like this:



Conclusion


THANK YOU

Any Query