CS 310 BIOS Lab
Spring 2020

January 21, 2020

1 Introduction

In this lab, we will be learning how to write assembly language programs on a bare metal (no operating system) x86 emulator. We will be using an 16-bit 8086 emulator called emu8086, which is available from the course website with Mac and Windows versions. The 8086 came out in the mid 1970s, and it was the processor used in the original IBM PC, which is the predescessor of all modern laptops, desktops, and servers. Its instruction set lives on in the modern Intel processors, and we will be learning how to write x86 assembly language in this class because it is a ubiquitous platform. We will begin the class working with a simple 16-bit platform.

The BIOS is a piece of software that comes with every computer and provides some basic services before the operating system is loaded. It provides and API that you can call to print characters to the terminal, read and write from the hard disks, etc.

To print to the terminal in Java, we call `System.out.println()`, which calls the operating system, instructing it to print the string. But before the computer has booted up, there is no operating system. The BIOS provides basic services before the operating system boots. The complete list of APIs supported by the BIOS is documented in Ralf Brown's Interrupt List (RBIL), which you can google around for.

2 Calling the BIOS

Calling APIs in the BIOS is kind of like calling a function in Java. Just like a function, we pass parameters that tell the BIOS API what to do. In this section, we will learn about how to call the print character function in the BIOS. To do so, we will pass parameters in the CPU registers that the BIOS will interpret as a command to print a character:

Register	**Meaning**
AH	`0x0E`: Command code to write to terminal
AL	Character to write
BH	Page Number
BL	Foreground color

The following program sets up the registers with the correct values to print the character 'N' to the screen and then loops forever.. Type the program into emu8086 and verify that it works. Be sure to surround the N in *single quotes*—that's the one between the semicolon and enter keys. You should see the character print out to the virtual console.

```
main:
    ; Set up the registers for a BIOS call to print
    mov ah, 0x0e ; Write to terminal command
    xor bh,bh    ; Page 0
    mov bl,7     ; Foreground black
    mov al,'N'   ; Write an 'N' to the screen
    int 16       ; Call the BIOS!
loop:
    jmp loop     ; Loop forever
```

Once you've gotten this working, write a program that prints your name to the terminal.

3 Functions

In assembly language we can have functions just like in Java and other higher level languages. For now, we'll pass all of the parameters in the registers. We will also use the registers to store variables. The following is an example of a function that adds two numbers. The inputs are passed in AX and BX, and the result is returned in AX. Try typing this program in to emu8086 and running it.

```
main:
    mov ax,1
    mov bx,2
    call add2nums
    mov bx,3
    call add2nums
loop:
    jmp loop

add2nums:
    add ax,bx
    ret
```

This program starts in `main`, which loads the value 1 into AX and 2 into BX. `main` then calls `add2nums`, which adds the contents of BX to AX and returns. The `ret` instruction causes the program to jump back to main just after the function call.

4 Converting your BIOS Call to a Function

Now we're going to encapsulate the BIOS call from before into a function. This way, when we want to print a character, all we have to do is call our function. This function will be called `printChar`. It should take one parameter—the character to print—in AL. A skeleton program should look like this:

```
main:
    mov al,'N'
    call printChar
    mov al,'E'
    call printChar
    mov al,'I'
    call printChar
    mov al,'L'
    call printChar
loop:
    jmp loop

printChar:
    ; YOUR CODE FOR printChar HERE
    ret
```

This program is basically just loading AL with the letters in my name and calling `printChar` over and over.