Homework 2

Academic Honesty

Aside from the narrow exception for collaboration on homework, all work submitted in this course must be your own. Cheating and plagiarism will not be tolerated. If you have any questions about a specific case, please ask us. We will be checking for this!

NYU Tandon's Policy on Academic Misconduct:
https://engineering.nyu.edu/campus-and-community/student-life/office-student-affairs/policies/student-code-conduct#chapter-id-34265

Accepting the Assignment and Setting Up a Repo (Mandatory Step)

1. Head to **https://anubis.osiris.services/**
2. Press "LogIn" on the side bar, and log in with your NYU ID
3. Enter GitHub username (If not done yet)
4. Go to "Courses", and press "Join Class" with code **iloveos** to be registered to the class
5. Go to "Assignments", and accept the assignment
    a. Press "Create Assignment Repo" (This will create a repository on Github)

Working on the Assignment

For this assignment we will offer you **two options** for a platform to work on, or you can use your own platform. It's up to you what you use. It's **your responsibility** to turn it on time no matter what platform you choose. **To make this 100% clear**. Whether you choose **Anubis or Vagrant you are the only one responsible for turning it on time.** If Anubis or Vagrant doesn't work for you 5 minutes, 5 hours or 5 days before the deadline, it's not the Professor's responsibility or the TAs responsibility, it's **your responsibility.**

Using Anubis IDE (Option 1)

1. Select “Anubis Cloud IDE” from the Assignment
2. On the top menu bar, select “Terminal” and “New Terminal” to bring up a terminal
3. Complete your assignment in the project folder, and commit your changes for submission/grading

Notes:

- With auto save on, the IDE automatically commits your changes to GitHub every couple of minutes. Or type “autosave” in the shell to save your changes
    - Even with this option on, you are still **responsible** for saving your changes
- If you have any questions, don’t hesitate to post on Forum or email any of the TAs

Using Vagrant (Option 2)

1. Follow the instructions in the repo: **https://github.com/wabscale/cs3224-vm**
2. Pull the assignment repository from Github
3. Complete your assignment in the project folder, and commit your changes for submission/grading

Using your own environment (Option 3)

1. As always you are free to use your own linux environment
    a. Assignments are graded based on the Anubis IDE environment
2. Pull the assignment repository from Github
3. Complete your assignment in the project folder, and commit your changes for submission/grading

You must accept the assignment using Anubis for any of the options listed above.

Background

When the computer first boots, the BIOS initializes the hardware, and then passes control to the *bootloader*. An x86 processor starts in 16-bit “real” mode, a compatibility mode that mirrors the operation of the original 8086 microprocessor. A modern bootloader will transition out of this real mode immediately, but for this assignment we will write a program for this 16-bit environment.

Once basic initialization of hardware is complete, the BIOS will try to read a *boot sector* off of the boot medium. The boot medium could be a floppy disk, a hard drive, or whatever else the BIOS can figure out how to access (most modern BIOSes can even boot from the network). The BIOS reads the first 512 bytes from this device medium (the boot sector, containing the bootloader), checks to make sure the sector ends with the two bytes `0x55 0xAA`, loads the sector into memory at address `0x7C00`, and then sets the instruction pointer to `0x7C00` to execute the bootloader

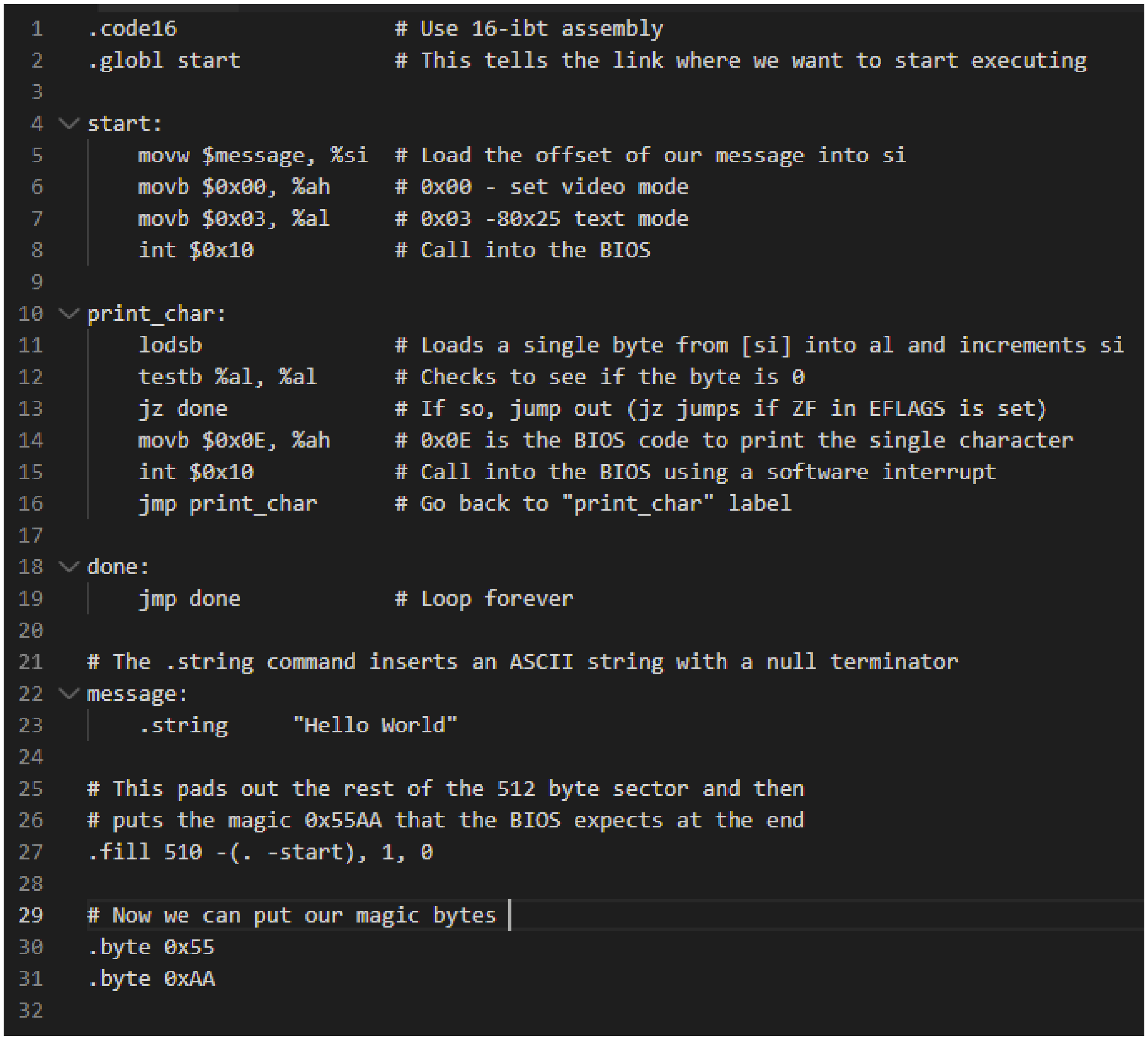
In order to write a real mode program, we will have to use 16-bit assembly code and then assemble that code into a binary boot sector. We can do this with the GNU assembler called `gas` (it is also used by xv6).

For this assignment, we will need to use QEMU, GDB, and i386 GCC Toolchain, but we will not need to use XV6 or modify the Makefile. We will use **AT&T Syntax** for this assignment and the course.

Part 1 Hello World

Goal: Make the BIOS load the following program from the boot sector and print “Hello world”

1. Find the file. “hello.S”, in the project folder and add the following code.
2. Run “make hello” to compile the code and generate “hello.o”
3. Run “make qemu-hello” to let QEMU use “hello.o” as the bootloader
   a. Press “Ctrl+a” then “x” to exit QEMU
   b. “make clean” To remove compiled files
4. To verify the 512 bytes in “hello.o”, run “ls -l hello”



```
.code16                 # Use 16-ibt assembly
.globl start            # This tells the link where we want to start executing

start:
    movw $message, %si  # Load the offset of our message into si
    movb $0x00, %ah     # 0x00 - set video mode
    movb $0x03, %al     # 0x03 -80x25 text mode
    int $0x10           # Call into the BIOS

print_char:
    lodsb               # Loads a single byte from [si] into al and increments si
    testb %al, %al      # Checks to see if the byte is 0
    jz done             # If so, jump out (jz jumps if ZF in EFLAGS is set)
    movb $0x0E, %ah     # 0x0E is the BIOS code to print the single character
    int $0x10           # Call into the BIOS using a software interrupt
    jmp print_char      # Go back to "print_char" label

done:
    jmp done            # Loop forever

# The .string command inserts an ASCII string with a null terminator
message:
    .string     "Hello World"

# This pads out the rest of the 512 byte sector and then
# puts the magic 0x55AA that the BIOS expects at the end
.fill 510 -(. -start), 1, 0

# Now we can put our magic bytes
.byte 0x55
.byte 0xAA
```

Part 2 The Guessing Game

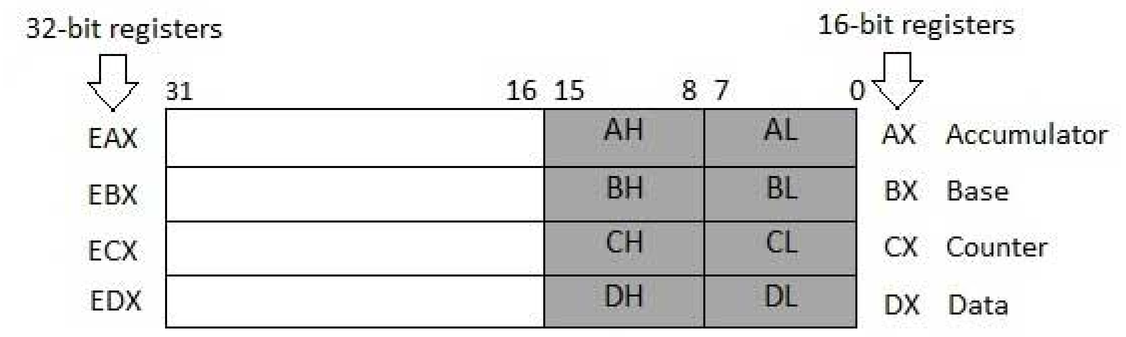
Goal: Write an x86 boot sector that implements a simple guessing game. You should ask the user for a number from 0 to 9, read their response, and then tell them if they got it right or wrong. If they got it right, then print a success message, and the program jumps into an infinite loop forever to simulate stop. To generate a random number, see Notes below.

1. Open the file “guess.S”, which contains the template for the guessing game
2. Run “make guess” to compile, or “make qemu-guess” to compile and run the program

Format and examples:

```
What number am I thinking of (0-9)? 4

Wrong!

What number am I thinking of (0-9)? 9

Wrong!

What number am I thinking of (0-9)? 2

Wrong!

What number am I thinking of (0-9)? 5

Right! Congratulations.
```

Notes:



- The messages and order print out should be similar to the ones above.
- BIOS interrupt (“int 0x16”) will read a single character from the user when “AH” is set to 0 and store the ASCII value of the result in “AL”.
    - More at https://en.wikipedia.org/wiki/BIOS_interrupt_call

- Writing a carriage return, ‘\r’, (ASCII 0x0d) will move the cursor to the beginning of the line.
- Writing a line feed, ‘\n’, (ASCII 0x0a) will move the cursor down one row.
- To generate a random number, use the seconds portion of the current time. Use the standard PC peripheral called the CMOS RTC (Real Time Clock) to get the current time. Access to the CMOS is done through *Port I/O* (specifically, ports \`0x70\` and \`0x71\`); recall from class that this uses the x86 \`inb\` and \`outb\` instructions. The seconds range from 0-59 and you want a number from 0-9, so you'll have to find some way to scale the result into something in the right range (division).
    - More at https://wiki.osdev.org/CMOS
- Anubis test does not check whether or not your program works, please test carefully on your own.

Debugging with GDB:

1. There are two targets in the makefile for debugging. To start guess in debug mode, run: “make qemu-gdb-guess”
2. Qemu will then wait for you to connect to it. Connect to it by running the gdb target in a new terminal window, run “make gdb-guess”
3. Assuming all goes well, you should be dropped into a gdb shell at the beginning of your start function.
4. Check on Campuswire under Files tab for Cheatsheet
5. Check out the Makefile for all available commands

Part 3 Short Answer Questions

Put your responses to these questions on Anubis. The questions can be accessed by clicking the assignment in the “Assignments” tab.

1. What is the difference between Monolithic and Microkernel? Do so by stating the advantages and disadvantages of both.
2. Create a file named “foo.txt” using the following shell command in your shell (not xv6) using “touch foo.txt”. Now use “ls -al” and explain what permissions it has. Also how would you make the file readable, writable, and executable for ONLY the user that owns the file, and zero permission for everything else?
3. In your own words describe what is the boot loader, boot sector, and what does the BIOS do at the start of the computer?

4. What are file descriptors and how are they assigned? Can a file opened by the user have a file descriptor value as 2? If yes, explain how. If no, why not?
5. Consider the following Assembly code and answer the questions in the comment

```
// Note: this is x86 32 bit (not 64 bit)

movl $0x41000000, %eax

movl $0x420000, %ebx

movl $0x4300, %ecx

xor %ebx, %eax

xor %ecx, %eax

// 1. What is the integer value in %eax at this point of code execution?

pushl %eax            // move value of eax onto stack

movl $0x04, %eax      // syscall number for write into eax

movl $0x01, %ebx      // file descriptor for stdout into ebx

movl %esp, %ecx       // address of stack (what was in eax before)

movl $0x04, %edx      // number of bytes to write

 int $0x80            // syscall

// 2. What will be printed to the screen at this point of code execution?
```

Extras (Not part of the assignment)

If you're feeling particularly adventurous, you can try running your programs on a real PC by writing them to a USB stick and then booting from the USB stick.

Rubric

We ask that you thoroughly comment your code and compile your code without errors to receive full credit for the assignment.

Part 1 Hello World - 25 points

Part 2 Guessing Game - 50 points

Part 3 Short answers 25 points, 5 points for each question