Operating Systems — Project 0
Completing a BIOS

1 Overview and motivation

This project will begin your work with *Fivish*, our (mostly) RISC-V-based assembler and system simulator. The goal is to get working with these tools and see a small group of sample assembly programs. The real work, though will be in the BIOS; it is mostly written, but you will need to complete it.

2 Getting started

2.1 System requirements and options

The *Fivish* environment requires *Java*, `bash`[1], and some typical UNIX-based command-line tools (e.g., `make`, `dd`, etc.) These are tools that you can install (if needed) and use on your own computer, but I leave that as a project/adventure for you to explore, perhaps with your fellow students. If you don't have these tools, I recommend that you use `ssh` to connect to `systems.cs.amherst.edu`, which already has them. If you choose to do your work on this server, I recommend installing an *X11 server* on your computer—*XQuartz* on macOS, *cygwin* with X11 packages on Windows. Talk to me if you want to try this, but it will be a bit of an adventure of its own.

Expect to work significantly on the command-line and with fundamental text/code editors. Specifically, `vim`, `emacs`, *Sublime Text*, even *Notepad++* are reasonable options for editing code. Maybe *VSCode*—maybe. Do **not** expect to use IDE's like *Eclipse* or *IntelliJ*; most of your work will be in assembly and, later, a language that these IDE's won't recognize and can't work with.

2.2 Getting *Fivish*

On whatever system your using, use `git` to checkout a copy of the assembler and simulator:

```
$ git clone git@gitlab.amherst.edu:sfkaplan/fivish.git
```

You should see two directories in this repository: `assembler` and `simulator`. Change into each directory in turn, building the source code:

```
$ cd fivish/assembler
$ make
$ cd ../simulator
$ make
$ cd ..
```

[1]The Bourne Again Shell that is common to Linux, macOS, etc.

2.3 Getting the project source

Download the source assembly files:

`bit.ly/cosc-277-p0`

You can download and then unpack the source files from the command-line like so, and then take a look at the files.

```
$ wget -i https://bit.ly/cosc-277-p0
$ tar -xJvpf project-0.tar.xz
$ cd project-0
$ ls
bios.asm  conditional.asm  do-nothing.asm  find-max.asm  loop.asm
```

The latter four files are small assembly examples of basic programming constructs that we will go over in class. The `bios.asm` file is where you will do your real work.

2.4 Assembling and running

As an example, we can use the `do-nothing.asm` code to see how to use the assembler and the simulator. First, open the source code in your favorite text editor; you will see that this little program loads a couple of constants into registers, and then it halts the processor.

To assemble this program, do the following:[2]

```
$ cd ../assembler
$ ./assemble ../project-0/do-nothing.asm
```

You should see the output of the assembler, which shows how each line of code is interpreted and translated into machine code. There should also be, in the `project-0` directory, a file named `do-nothing.vmx` that contains the actual RISC-V machine code. We can now try running this code in the simulator:

```
$ cd ../simulator
$ ./simulate ../project-0/do-nothing.vmx
[...]
[pc = 0x00009000]: step 10
```

You will see the processor carry out the instructions of the program (which actually takes fewer than 10 steps, but the halted processor ignores the request for any additional steps). You can match these steps to what the assembler showed. You can also examine the registers used to see that they contain the desired values, and then exit the simulator.

[2]These instructions assume that your `project-0` directory is in your `fivish` directory, alongside the `assembler` and `simulator` directories. Adjust as needed.

```
[pc = 0x00009018]: showregister a0
a0 = 0x0000000d
[pc = 0x00009018]: showregister a1
a1 = 0x1a2b3c4d
[pc = 0x00009018]: exit
```

Congratulations! You have used our little system. Now try the other sample programs and make sure that you see how they work.

3 Your assignment

Your goal with this assignment is to complete, in assembly, the code needed to bootstrap our simulated, hard-disk-less system. Much of this BIOS is written, but it lacks one critical portion of its code that you must fill in. So, to complete this assignment, you must:

1. **Grok:** Read the code already in `bios.asm` and grasp, to the greatest extent possible, what it does and how it does it. I recommend also writing down questions about parts you don't fully understand, and then asking about them. As a broad outline, here's what the BIOS, when complete, must accomplish:

   (a) Find the **RAM** in the physical address space.

   (b) Find the **second ROM** in the physical address space. Both your BIOS and users of it assume that this second ROM is the *kernel*.

   (c) Copy the kernel (2^{nd} ROM) into main memory (RAM) [see **Complete**, below].

   (d) Jump to the copied kernel's first machine code instruction.

2. **Complete:** You will find, in a procedure named `_procedure_copy_kernel`, an incomplete chunk of code (clearly marked with comments). This portion of the code must perform the actual copying of bytes from the kernel ROM and into RAM.

Notice that you must, for now, use a *dummy kernel*. Since we have not yet written any part of a true OS kernel, then any executable image (e.g., `do-nothing.vmx`, assembled from the provided `do-nothing.asm`) will suffice.

4 How to submit your work

Instructions forthcoming! You will be sending me, one way or another, your complete `bios.asm` file. I'll post details on what mechanism to use for this task.

This assignment is due at **11:59 pm** on **Sunday, February** 11^{th}**.**