CPSC 213: Assignment 1

Due: Sunday, Sep 19, 2010 at 6:00 pm

Goal

The goals of this assignment are for you to get some of the preliminary stuff under your belt:

- familiarize yourself with the Unix command line
- write, compile and run a very simple C program
- understanding endianness and how to detect it using a program
- install the SimpleMachine simulator on your machine
- implement the MainMemory class used by the SimpleMachine simulator
- familiarize yourself with the course policies, including plagiarism guidelines.

Unix Development

Much of the programming you will do in this course will be in Java using Eclipse or Apple's XCode (your choice). But, for some of what you will do, you will need a Unix development environment. One of the requirements of this first assignment is that you get a Unix environment setup and become familiar with it. You have some choices.

You can just use the departmental machines, if you like. Machines starting with lin and then followed with a two-digit number are IA-32 machines running Linux (e.g. lin02.ugrad.cs.ubc.ca). Most other undergrad machines are Sparc machines running SunOS (e.g., galiano.ugrad.cs.ubc.ca). Any of these will be fine. You can access these machines remotely using ssh or SecureCRT or XShell. (**Note that XShell is already setup on your Ugrad Windows environment** - simply look on the desktop for the XShell icon to start the program. You can then select which department machine to connect to. Use your 4 letter UNIX ID as the user name and the associated password as the password. As needed, make new session profiles to connect to the Linux machines like lin02.ugrad.cs.ubc.ca)

If you want to use your own Windows desktop/laptop you need to install the Cygwin (Microsoft's C Visual Studio environment and their C development tools will not work for this class). Cygwin provides your Windows machine with a Unix command line and various development tools. Be sure to specify that you want "gcc" included when you install it.

If you want to use a Mac you need to install Apple's Development Tools (XCode and related command line tools). This is an optional install included on the MacOS distribution DVD and is also available for download from Apple by joining their developer network (which is free); details at developer.apple.com.

C and Endianness

In class I gave you the outline of a C program that casts an `int` into an array of bytes. Modify this program (or start from scratch) to create a program that prints "Big Endian" on big-endian architectures and “Little Endian” on little-endian architectures. Test your program by compiling and running it on an IA32 (Intel) machine and on a Sparc machine. Undergrad machines with the lin## prefix (e.g., lin01.ugrad.cs.ubc.ca) are IA32 machines running Linux. The other undergrad machines are Sparc machines. You might guess that IA32 and Sparc have different endianness.

A part of the goal here is to get comfortable with the UNIX command line, with editing, compiling and running C programs etc. by starting with a simple example. Here are some useful commands.

command	**purpose**
`uname -a`	display the ISA and other characteristics of the current system
`emacs`	a text editor for writing source code (Eclipse will work too)
`gcc`	compile a program into an executable
`man`	display the manual page (documentation) for a unix command
`gdb`	the gnu symbolic debugger (for debugging C programs)

For more UNIX commands that will be useful to learn and remember, please refer to the course website under the Resources section. There is a command summary PDF and a book recommendation if needed.

SimpleMachine

Included in the lab material is the source code for the SimpleMachine Simulator. The file sm.zip contains three directories: src, doc and lib. In src is the entire source tree for the simulator. In doc is the complete java doc. In lib is a single jar file that must be included with the project. Unzip the sm.zip file somewhere outside of your eclipse workspace directory, for example, inside Z:\cs213\a1\sm.

Now you need to load this code into Eclipse or your preferred development environment. Ensure that you did this properly by building the project and executing the default class. You should see the GUI if everything was successful. That is all that will work until you implement the Memory class correctly.

If you use Eclipse, the following steps will set you up to run the simulator.

1. Create a new Java Project and give it a name such as ‘SM213’. In the package explorer view, right click on the ‘src’ folder in your project and select Import. Select ‘File System’ from the ‘General’ folder and select the sm/src directory. Select the entire ‘src’ directory and make sure you are importing into the src folder of your project. Click “Finish”. Ignore all the build errors because we’ll fix it in step 2. *Once again,* be *sure*

that your sm/src directory is not located inside of your Eclipse Workspace directory to ensure successful import.

2. This step varies by platform and Eclipse version.

   On Windows it is probably like this.

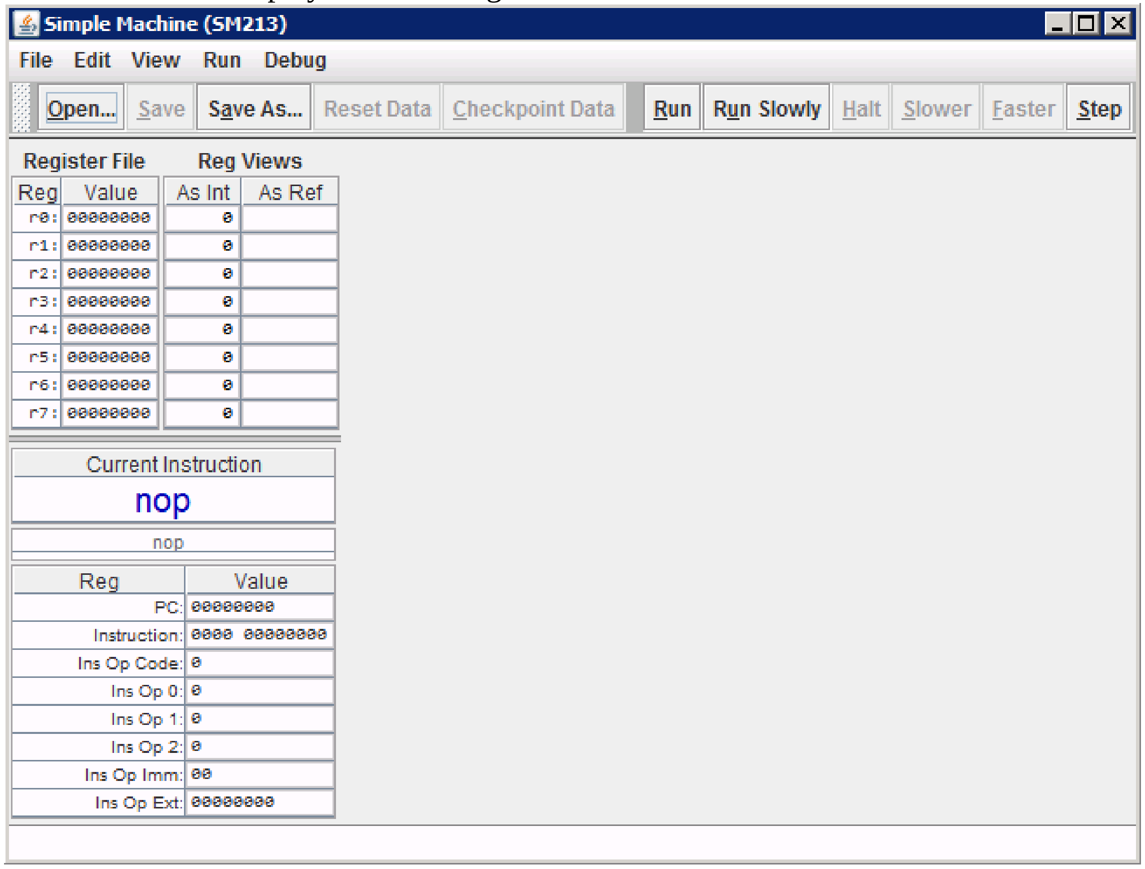
   a. Select “Java Build Path” from the “Project | Properties” menu. Click the “Libraries” tab and then select “Add External JARs”. Find the antlr-runtime-3.1.tar file included in the lib directory of the distributed code and add this library.

   On Mac there are two steps.

   a. From the “Preferences” menu, select the “Java | Build Path | User Libraries” pane. Click “New” to create a new library, naming it “antlr”. Then click “Add JARs” and select the file sm/lib/antlr-runtime-3.1.jar. Click “OK”.
   b. Right click on the project name in the Eclipse workspace and select “Build Path | Add Libraries”. Select “User Library”. Then you get a list with the “antlr” library. Check the box next to this library. The “antlr” library should now appear in the workspace for this project.

3. Try to run the SimpleMachine simulator. The best way is to right click on the SimpleMachine.java file in the default package and selecting “Java Application”. Note that the main() function is inside the “SimpleMachine” class.

4. It should run and display the following GUI:


Simple Machine (SM213)
File Edit View Run Debug
Open... Save Save As... Reset Data Checkpoint Data Run Run Slowly Halt Slower Faster Step
Register File
Reg Views

Reg	Value	As Int	As Ref
r0:	00000000	0	
r1:	00000000	0	
r2:	00000000	0	
r3:	00000000	0	
r4:	00000000	0	
r5:	00000000	0	
r6:	00000000	0	
r7:	00000000	0	

Current Instruction

nop

nop

Reg	Value
PC:	00000000
Instruction:	0000 00000000
Ins Op Code:	0
Ins Op 0:	0
Ins Op 1:	0
Ins Op 2:	0
Ins Op Imm:	00
Ins Op Ext:	00000000

5. The file you edit is MainMemory.java in Arch.SM213.Machine.Seq.Student.

Implementation of Memory Class

Finally, implement the missing methods of the Memory class. In doing so, you will determine how memory is stored, implement accessor methods (set and get), implement a method that determines whether an address is aligned and implement methods that covert between an array of bytes and four-byte Big Endian integers. This class will be used for all memory access in subsequent assignments by the SM213 machine implementation that you will build.

To do these things, modify the class MainMemory in the package Arch.SM213.Machine.Student. The simulator has a large number of other classes that you can completely ignore for this assignment.

Implementing Big Endian conversion requires the use of Java's bit shifting operators (**`>>`** and **`<<<)`** and bit-wise logical operators (**`|`** and/or **`&`**). You can tell Java that you only want the low order byte of an integer by casting it to a byte (e.g., **`byte b = (byte) (i);`**).

One caution is that Java bytes are signed numbers and so shifting to the right or assigning a **`byte`** into an **`int`** will sign extend the number, which might not be what you want. For example:

```
    byte b = (byte) 0xff;
    int i = b;
```

results in **`i`** storing **`-1`** which in hex is **`0xffffffff`** not **`0x000000ff`**.

Add a "Main" method to this class so that you can test the methods you implement. Be sure to test your code fully, including error cases in that should throw InvalidAddressException.

Now, further test your implementation by loading the file S1-global-static.s into the simulator. You should see the code for this file displayed in the simulator and you should be able to modify it. You will not be able to execute the code until you implement the Machine itself in subsequent assignments.

Course Policies

Read through the course policies posted at http://www.ugrad.cs.ubc.ca/~cs213/winter10t1/policies.html and ensure that you understand all of them, especially the material on plagiarism. If you have any questions, ask the course staff (instructor or TAs).

Material Provided

The sm.zip that contains the src, lib and doc parts of the SimpleMachine simulator and code.zip that contains S1-global-static.s.

What to hand in

Use the **`handin`** program to hand in the following:

1. A statement explaining which type of Unix environment you will use for this course and saying that you've got it setup and working.

2. The C program you write.

3. A list of the machines on which you ran your C program and what it said about their Endianness. For each of these machines, use the **`uname`** program to determine the ISA of the machine and include this in your report.

4. Your implementation of the Arch.SM213.Machine.Student.MainMemory class, including your test code. Carefully comment your test code to indicate why it provides test coverage.

5. A description of the results of your testing. If your test cases provide full coverage and all tests passed, it is sufficient to just say this. But, if certain things do not work, you must indicate this.

6. A statement asserting that you have familiarized yourself with the course policies and understand the plagiarism guidelines.

The handin assignment directory name for this assignment is **`a1`**.