CPSC 213: Assignment 1

Due: Monday, September 19, 2011 at 7am.

Late assignments are accepted until Thursday, September 22 at 7am with a 20% penalty per day (or fraction of a day) past the due date. This rule is strictly applied and there are no exceptions.

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.

Unix Development

Much of the programming you will do in this course will be in Java using Eclipse. 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 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. Most other undergrad machines are Sparc machines running SunOS (e.g., galiano). Any of these will be fine. You can access these machines remotely using ssh or Xshell (google to find downloads for these).

If you want to use a Windows machine you need to install 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 instal 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)

SimpleMachine

Included in the lab material is the source code for the SimpleMachine Simulator. The file sm-student-213.zip contains two jar files and two zip files.

- SimpleMachine213.jar

  An reference implementation of the SM213 simulator with classes obfuscated so that you can't decompile it to see the implementations of MainMemory.java and CPU.java, which are the classes that you will implement over the next several assignments.

  Click on the jar or type **`java –jar SimpleMachine213.jar`** at the Unix command line to run the reference implementation.

- SimpleMachineStudent.jar

  Contains the "student" version of the simulator that has only stubs for MainMemory.java and CPU.java.

- SimpleMachineStudentDoc213.zip

  Javadoc for the student version.

- SimpleMachineStudentSrc.zip

  Source for the student version.

Also included in the lab material is a zipped Eclipse project called sm-student-213-eclipse.zip that contains all of the simulator code, including jar and zip files listed above.

The second part of the assignment is create and Eclipse (or similar) development environment for the simulator. Instructions that describe how to install the simulator code into Eclipse are provided as part of the assignment material found on the course web page. As an alternative, you can, for example, use Apple's Xcode development environment, or any other tool you like.

Implementation of Memory Class

Finally, implement the missing methods of the Memory class. In doing so, you will implement accessor methods for access to memory (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.

Material Provided

On the course web page you will find the following additional material for this assignment.

1. sm-student-213.zip

2. sm-student-213-eclipse.zip
3. install.pdf (also included as an appendix to the Companion)
4. code.zip which includes S1-global-static.s

What to hand in

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

1. A single file called "README.txt" that includes your name, student number, four-digit cs-department undergraduate id (e.g., the one that's something like a0b1), and all written material required by the assignment as listed below.

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

3. The C program you write.

4. 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. Include this in the README file.

5. Your implementation of the arch.sm213.machine.student.MainMemory.java, including your test code. Carefully comment your test code to indicate why it provides test coverage.

6. 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. Include this in the README file.

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