CPSC 213: Assignment 1

Last Modified: September 8, 2014 at 1:20pm (fixed due date time)

Due: Sunday, September 14, 2014 at 11:59pm

After an eight-hour grace period, no late assignments accepted.

Goal

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

- get comfortable with hex, integers, and endianness
- 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.

Part I – Endianness and Hex Questions

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
7000	73	C3	DA	3D	53	78	13	C7	F4	02	7D	2D	41	A0	C8	4B
7010	18	8D	25	06	9A	FC	35	70	87	B0	32	EF	41	1B	63	6C
7020	5D	29	FE	43	A7	B4	26	06	6D	A4	46	84	C8	1B	48	75
7030	57	C2	3E	78	C7	09	22	37	82	4B	0A	CC	37	53	7B	34
7040	64	07	A3	76	21	ED	DE	B2	21	B2	50	3A	CE	12	4D	E1
7050	52	45	AD	0A	FD	5C	8D	E7	15	EB	65	E6	05	6D	5E	02
7060	08	FD	01	2D	4D	70	A2	69	E1	66	13	56	B4	30	FF	A1
7070	04	21	C3	18	EE	B9	3C	5B	B0	37	0E	C2	CA	F8	4E	3A
7080	6C	53	AC	F3	4D	D6	2B	D2	57	C8	C6	C7	BD	5D	E2	FD
7090	EE	73	04	65	9B	8F	58	C0	A4	70	90	60	B1	2C	52	A6
70A0	A0	D7	1F	68	0F	0C	18	F4	8A	52	5F	1F	0E	1E	81	5E
70B0	7F	7E	4F	E3	35	F6	34	91	0A	D4	97	F7	FB	24	AE	0E
70C0	6E	5E	9E	20	19	75	B5	60	BC	66	74	91	BE	FF	7A	5A
70D0	7E	7C	E4	CE	37	43	0B	DA	6D	0D	45	9E	0C	C9	4A	6B
70E0	94	D9	4C	83	F5	82	66	7F	88	92	8A	9E	80	4A	8A	C9
70F0	37	45	BD	09	EA	B9	A9	75	29	80	5D	C3	EC	41	CD	62

The table below lists the content of memory between address 0x7000 and 0x70FF. Every number is shown in hex. For example the value of the byte at address 0x7058 is 0x15.

Answer the following questions about this table.

1. What is the value (in hex) of the little-endian 4-byte integer stored at location 0x7064?

2. What is the value (in hex) of the big-endian 4-byte integer stored at location 0x7048?
3. What is the value (in hex) of the big-endian 8-byte long long stored at location 0x7090?
4. At what address is stored the 4-byte little-endian integer 0x609070A4?
5. Give an example of a 4-byte integer whose big- and little-endian representations are identical. Can you generalize this example?
6. Consider a hypothetical situation in which two different processors (possibly made by different manufactures) are accessing the same memory. The first processor computes "1 + 65536" and stores this in memory at location X. Then the second processor reads the value at location X and adds 1 to it, only to discover that this value is actually 16777473, not 65538, as expected. What has gone wrong? HINT: Start by converting the base-10 numbers involved into hex. Use a calculator or the computer (e.g., `printf ("0x%x \n",i);` prints the value of i in hex preceded by "0x").

Part II — Install the Simple Machine

Download http://www.ugrad.cs.ubc.ca/~cs213/cur/Assignments/a1/sm-student-213.zip. It contains two jar files and two zip files.

- SimpleMachine213.jar

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

  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.

Most of you will use Eclipse as your IDE for the course. If you will, also download the file http://www.ugrad.cs.ubc.ca/~cs213/cur/Assignments/a1/sm-student-213-eclipse.zip.

Follow the instructions in Appendix B of the Companion to install the Simulator into Eclipse (or if you prefer, another IDE). If you are using Eclipse, then the simplest instructions are in Section~B.1.

Most of this is just preparation for next week when we start to use the Simulator. But, there is one thing to do for this assignment once you have installed the simulator.

Part III — Implement the Memory class.

Like a real machine, the simulator has a Memory and a CPU. You are going to build both of them. This week, the Memory.

The **`MainMemory`** class is in the package **`arch.sm213.machine.student`**. This class simulates main memory by storing its content in an array of bytes. That part is already done for you. You must implement the five methods flagged with "TODO" comments.

Create a set of JUnit tests to test your implementation in the class MainMemoryTests in the same package. Do not worry about running the simulator itself just yet. Just get your unit tests to pass. Comment each test to explain what it is testing and to demonstrate test coverage.

Part IV – A Simple C Program

Unix Development

Much of the programming you will do in this course will be in Java using Eclipse or in assembly language using the Simulator. 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. 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 that is available as a free download from Apple https://developer.apple.com/xcode/.

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. Name this program "`endian.c`"

Test this program by running it on at least two different processors —- Sparc and Intel x86 —- running Unix-like operating systems (e.g., Mac OS, Linux, SunOS etc). Intel x86 machines are easy to find. Sparc machines are a little harder (galiano.ugrad.cs.ubc.ca is one). You can use the "uname" command shown below to determine a machine's type. You need to recompile your program for each machine you try.

One way to compile your program is using this command (see also the gcc man page):

```
gcc -o endian endian.c
```

If you do this, then you run your program by typing:

```
./endian
```

Part of the goal here is to get comfortable with the UNIX command line, for 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)

What to hand in AND HOW

You must use a program called **`handin`** to hand in your assignments in this course. To use this program you need a CS account. If you don't have one you must get one. Its easy. Here's a how: https://www.cs.ubc.ca/students/undergrad/services/account.

There are two ways to use **`handin`**: from the UNIX command line or from the Web.

You will find the instructions for using the command line version of **`handin`** here: https://my.cs.ubc.ca/docs/handin-instructions *(that page tells you everything you need to know, but note that the page currently has a bug: you don't type **`man handin`** at the command-line prompt to get a description of how to run the program, you simply type **`handin`**).*

You will find the instructions for using the web version here: https://my.cs.ubc.ca/docs/hand-in.

Whichever one you use, please double check that it worked correctly; the instructions show you how. We will not accept homework submitted any other way (e.g., email).

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. Your answers to questions 1-6 in README.txt.

3. The C program you write as a separate file called "`endian.c`".

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`** and your JUnit test class. Include only two java files: **`MainMemory.java`** and **`MainMemoryTests.java`**. Do not include any other java files or any class files. Do not include any directories or sub directories.

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`**.