CPSC 213
Lab 5
SM213 Simulator: Flow Control Instructions

Done in labs: July 26 & 29
Due: Tuesday, July 30, 10:00pm.

OBJECTIVES

The assignment has two parts. The goal of the first part of the assignment is to extend the machine simulator to simulate shift operations and control flow changes. Memory load and store operations and other arithmetic operations are already provided. This part builds on top of assignment 1, so we'll continue working on a version that has a completed `MainMemory` class. The second part involves translating conditional statements and loops from C to machine code and vise versa. .

SOFTWARE PROVIDED

- lab4-sm213files.zip - Containing the following :
  - **lab4-sm213.zip** - sm213 simulator code with incomplete CPU which has to be completed
  - **SimpleMachineStudent.jar** - a jar file that you have to include in your project for the simulator to run
- lab4-snippets.zip - a set of 1 C snippet and 1 sm213 snippet for part 2.

EXTENDING THE ISA

You will implement the following instructions:

opcode	format	semantics	example	example assembly

shift	`7rss`	**r[r] <— r[r] << ss, or r[r] <— r[r] >> ss**	`7102`
			`71fe`
				`Shl $2, r1 (shift left, ss is positive)`
				`Shr $2, r1 (shift right, ss is negative)`

branch	`8-oo`	**pc <— pc + 2*oo**	`1000: 8003`	`br 0x1008`
branch if equal	`9roo`	**if r[r] == 0, pc <— pc + 2*oo**	`1000: 9103`	`beq r1, 0x1008`

branch if greater	`aroo`	if r[r] > 0, pc <— pc + 2*oo	`1000: a203`	`bgt r2, 0x1008`
jump	`b--- aaaaaaaa`	pc <— aaaaaaaa	`1000: b000 00008000`	`j 0x8000`
jump indirect	`croo`	pc <— r[r] + 2*oo	`c102`	`j 0x4(r1)`
jump double indirect, b + o	`droo`	pc <— m[r[r] + 4*oo]	`d102`	`j *0x8(r1)`
jump double indirect, b+ index	`eri-`	pc<— m[r[r] + 4*r[i]]	`e120`	`j *(r1,r2,4)`

YOUR IMPLEMENTATION

The file sm213-a2.zip contains the sm213 simulator that was completed in Lab1. The simulator accepts and simulates all the load, store and arithmetic operations except the shift operation. In this lab/assignment you must extend the simulator to execute the above shift and branching instructions.

You will deal with two methods of the CPU class in the arch.sm213.machine.student package of the SM213 Simulator:

1. `fetch()` loads instructions from memory into the instruction register, determines their length, and adds this number to the pc register so that it points to the next instruction; it then loads the various pieces of the instruction into the registers `insOpCode, insOp0, insOp1, insOp2, insOpImm and insOpExt` (for 6 byte instructions). The meaning of each of these registers was given in class and is also part of the online lecture slides and the Companion notes. ***No changes are required to your fetch() stage, but do read it thoroughly and understand how it works.***
2. `execute()` uses the register values stored by the fetch stage to execute the instructions, transforming the register file (i.e. `reg`) and main memory (i.e. `mem`) appropriately.

First, you should check the CPU code for the load, store and ALU instructions and try to understand what it does. Your code will be similar to that. Here are some hints that will help you understand the code.

Reading and writing values from/to the memory locations is done by the memory methods implemented in Lab 1. To read and write values from/to the special registers you can use the `Register.Port` methods:

```
void set(long aValue)
int get()
int getUnsigned()
```

To read and write from/to the general registers you can use the `RegisterSet` methods:

```
int get(int regIndex)
void set(int regIndex, long value)
```

where `regIndex` is the index (0..7) of the register in the register set.

TASKS TO DO FOR PART 1 OF THE LAB

1. Download the incomplete version of the 213 machine and create a project as you did for assignment 1:
   - Download the `lab4-sm213files.zip` and unzip it. You will get two files: `lab4-sm213.zip` and `SimpleMachineStudent.jar` . Do not unzip them.
   - Create a new Java project in Eclipse and give it a name, say `lab4-sm213`.
   - Right click on the project name in Eclipse's Package Explorer tab, select Import --- Archive File – Browse and find and select the `lab4-sm213.zip` file. Make sure that you have checked "Overwrite existing resources… and click Finish.
   - Right click again on the project name in Eclipse's Package Explorer tab, select Properties -- Java Build Path -- Libraries. Click "Add External JARs", find the `SimpleMachineStudent213.jar` file and click Open. The jar will be added to the libraries. Click OK to close the window. The new jar should now appear in the workspace of the project in the Referenced Libraries section. Now the project is complete and you can run it.
   - To run the program expand the project in the Package Explorer, expand `src` and the default package. Select `SimpleMachine.java` in the default package, right click and select Run As and then Java Application. In the pop-up window select `SimpleMachine$Sm213Student - SimpleMachine`. You will get the initial window.
2. Run the simulator with `S1-global-static.s` file that you'll find on the Lectures web page. This file contains only load and store instructions and should work well with the simulator. You may also run any of the files you used in lab 1 and 2 for this.
3. Extend the code of the `execute()` method in `Arch.SM213.Machine.Student.CPU` to implement the new instructions shown in the table above.
4. Run your simulator with snippets `S5-loop.s` and `s6-if.s` which you will find on the Lectures web page, to test your code. Run these snippets with the complete sm213 simulator (which you already have set up in the previous lab) first to see how they should behave.

TASKS TO DO FOR PART 2 OF THE LAB

- Download **lab4-snippets.zip** and unzip it. You will find two files: `lab4_1.c,` and `lab4_2.s`.

- Translate `lab4_1.c` into sm213 assembly code and store in file `lab4_1.s`. Run it with your simulator and test it.
- Translate `lab4_2.s` into C and store in file `lab4_2.c` file. You may run it with gcc and test it. In this case you should define a main function that includes your code and include the standard library and io header files using
  #include <stdlib.h>
  #include <stdlio.h>

WHAT TO HAND IN

1. The file `Arch.SM213.Machine.Student.CPU.java` with the code for the new instructions.

2. Files `lab4_1.s` with your sm213 Assembly code for the C snippet and `lab4_3.c` with the C code for the Assembly snippet.

3. The assignment cover page filled in for this assignment by all partners in the group.

HAND IN INSTRUCTIONS

Only one person in each group will submit the assignment using the department's handin tool. To submit it you should do the following:

- Create a directory cs213 in your Unix home directory, if you have not done so. In your cs213 directory, create a directory **lab4**.
- Place all the files you have to submit in the lab4 directory.
- At the Unix command line execute
  > handin cs213 lab4
- Type **handin help** to see the options that are available for the handin tool.

Last Updated: Jul. 24, 13