CPSC 213: Assignment 4

Due: Monday, February 13, 2012 at 6pm.

Late assignments are accepted until Wednesday, February 15 at 6pm with a 25% penalty per day (or fraction of a day) past the due date. This rule is strictly applied and there are no exceptions.

Goal

In this assignment you extend the SM213 implementation to support static control flow, including static procedure calls. You use this expanded machine to examine the compiler implementation of for-loops, if-then-else statements, procedure calls and return. One goal is to understand the role of pc-relative addressing, conditional and unconditional branch statements in support of high-level language constructs such as loops and `if`, connecting your existing understanding of these language features to what you now know about the execution of programs in hardware. Another goal is to understand the role of the PC register in implementing jumps to see that a jump is really just an assignment of a new value to this register. The final goal is to contrast the use of static and dynamic jumps in implement procedure call and return and to understand how the return instruction knows to which instruction it should jump.

Extending the ISA

You will implement these six control-flow instructions.

Instruction	Assembly	Format	Semantics
branch	br aaaaaaaa	8-pp	pc ← (R = pc + pp*2)
branch if equal	beq rc, R	9cpp	pc ← (R = pc + pp*2) `if` r[c]==0
branch if greater	bgt rc, R	acpp	pc ← (R = pc + pp*2) `if` r[c]>0
jump	j label .pos aaaaaaaa label:	b--- aaaaaaaa	pc ← aaaaaaaa
get pc	gpc $o, rd	6fpd	r[d] ← pc + (o = p*2)
indirect jump	j o(rd)	cdpp	pc ← r[d] + (o = pp*2)

Code Snippets Used this Week

As explained in detail below, you will use the following code snippets in this assignment. There are C, Java and SM213 Assembly versions of each of these (except the C-loop-unrolled file, for which there is no Java).

- `S5-loop`
- `S5a-loop-unrolled`
- `S6-if`
- `S7-static-call[-regs]`

Requirements

Here are the requirements for this week's assignment.

1. Implement the six control-flow instructions one at a time. Extend your test program to include tests for each of them. Test one before starting to implement the next.
2. Compare the C versions of snippets `S5` and `S5a`, and then compare their assembly code by running them in the simulator, `S5a` first. Document the changes you see in memory and registers for the first few iterations of the loop and the last one.
3. Examine `S6` and `S7`. Step the assembly through the simulator. Document the changes you see in memory and registers.
4. Using `S5`, `S6` and `S7` as models, write, test and document the execution of an assembly code program that implements the following C program.

```
int min=1000000;
int i;
int a[] = {2,40,6,80,10,120,14,16,18,20};

void foo() {
  for (i=0; i<10; i++)
    if (min > a[i])
      min = a[i];
}

void bar() {
  foo();
}
```

Material Provided

This assignment includes snippets 5-7 in the file code.zip.

What to Hand In

Use the **handin** program. The assignment directory is **a4**. Please hand in exactly the following files with the specified names. Do not hand in class files, or your entire Eclipse project, or a README in formats like .doc or .rtf.

1. `CPU.java`.

2. `test-lab4.s` that tests all newly implemented instructions.

3. `min.s` that contains assembly implementation of the C code for Requirement 4 above.

4. `README.txt` that contains:

   - header with your name, student number, four-digit cs-department undergraduate id (e.g., the one that's something like a0b1)

   - statement that "I have read and complied with the collaboration policies" at http://www.ugrad.cs.ubc.ca/~cs213/winter11t2/policies.html

   - test description for `test-lab4.s`. Did all of the tests succeed? Does your implementation work?

   - description of the key things you noted about the machine execution while running snippets (`S5`, `S6`, and `S7`), including observations required by Requirements 2 and 3. Keep it brief, but point out what each instruction read and wrote.