ICSI404 – Assignment 9 – JUMP, JUMP!

This assignment builds on the previous assignment.

Part 1 – Jump

So far, our programs all run one instruction after another until we hit a “halt”. We will add an unconditional jump, first. You can think of this like a goto in C or Java.

The format for jump will be:

Jump 10 // This will jump to the instruction at address 10 (decimal).

We would encode this in bits like this:

0011 0000 0000 1010 // opcode is 3, 10 is the address. 0000 0000 1010 is 10 in decimal. Note that the address will always be positive since a negative address does not make sense.

Change your program to recognize this bit pattern and change the PC in the store() method.

Part 2 – Conditionals

Unconditional jumps, while necessary, don’t let us implement high level concepts like for() or while(), since they always jump. We need to implement conditional jumps. There are 6 possible branch types, based on logical comparisons: >, >=, <, <=, ==, != Six cases requires 3 bits (8 possibilities), leaving two left over. One trick that computer architects do to reduce the cases is to recognize that we can reduce the cases if we are clever. We will eliminate the < and <= cases by instructing assembly language programmers to reverse the order of the operands and use the corresponding greater than cases.

Instead of comparing a<b, we can say b>a; instead of saying a<=b we can say b>=a.

We will implement the following instructions:

Compare Rx Ry // compare (subtract) two registers.

0100 0000 xxxx yyyy

The results of this compare need to be stored in two bits in the CPU (your choice on exact formats and names).

BranchIfEqual 12 // 12 is, of course, an example.

The format of this instruction is a little more complex:

0101 – the opcode

The next two bits are the indicator of what conditions to branch on:

Bit 0 : 0 ifLessThan, 1 ifGreaterThan ; either one if equal

Bit 1: 0 if not equal, 1 if equal

This one opcode (Branch) can have many names, depending on which follow on bits we use.

The last 10 bits are the number of bytes to branch/jump. Note that these are a SIGNED value. This value, sign extended, should be added to the PC if the branch is taken.

In total, the instruction looks like this:
0101 CCSA AAAA AAAA

CC == condition code, S == sign of the address, A = address

The evaluation of the bits must take place in execute() and the change of PC must occur in store().

Part 3 – Add the new instructions to your assembler

Alter your assembler to include these new instructions.

It is, of course, critical to test these new instructions. Create small programs (and a new Java test file (cpu_test2). Your tests should consist of short assembly language programs that make sense and show that your code works. Please make sure that you test all 4 branch instructions, compare and jump.

Test Example:

String[] test1 = new string[] {“jump 4”,”move R1 5”,”interrupt 0”,”halt”};

String[] assembledCode = Assembler.Assemble(test1);

// now I run my assembled code. If R1 has 5 in it when the run happens, I know that my jump didn’t work…

<u>You must submit buildable .java files for credit.</u>

Rubric	Poor	OK	Good	Great
Comments	None/Excessive (0)	“What” not “Why”, few (5)	Some “what” comments or missing some (7)	Anything not obvious has reasoning (10)
Variable/Function naming	Single letters everywhere (0)	Lots of abbreviations (5)	Full words most of the time (8)	Full words, descriptive (10)
Unit Tests	None (0)	Partial Coverage (7)	All methods covered, needs more cases (13)	All methods/cases covered (20)
Jump	None (0)	Attempted (5)		Completely working (10)
Compare	None (0)	Attempted (5)		Completely working (10)
BranchIfEqual	None (0)	Attempted (5)		Completely working (10)
BranchIfNotEqual	None (0)	Attempted (5)		Completely working (10)
BranchIfGreaterThan	None (0)	Attempted (5)		Completely working (10)
BranchIfGreaterThanOrEqual	None (0)	Attempted (5)		Completely working (10)