Lab 6 - Y86 Version 2: Adding Conditional Instructions

Due Monday, Nov. 14 at 1:30 – No late turnins accepted

To implement the y86 version 2, you will add the conditional jump (JXX) and conditional move (CMOVXX) instructions to your y86v1 processor. As part of this, you will have to modify your ALU to generate the condition codes flags and add the condition code registers to your processor.

Getting Started

It is best to start out with a brand new design for your y86 version 2. You do this by making a copy of either your solution to version 1 or our solution. The best way to do this is to tell Aldec to add a ***new*** design for your y86 version 2 to your workspace using the Workspace/Add New Design command. In this new design, add all the existing files from the existing y86 version 1 design. You should start by compiling and testing this new design with your version 1 tests just to make sure you are starting with a good design.

There are seven branch instructions (JXX) and seven move instructions (CMOVXX) in the Y86 instruction set that you will now implement. Note that these instructions are quite similar: jumps test the condition codes to decide whether to change the program counter, while moves test the condition codes to decide whether to write to the register file.

	≤	<	=	≠	≥	>
jmp	jle	jl	je	jne	jge	jg
rrmovl	cmovle	cmovl	cmove	cmovne	cmovge	cmovg

1. ALU

The first step is to augment the ALU with three 1-bit outputs: ZF (zero flag), SF (sign flag, for negative results), and OF (overflow flag). I suggest that you now switch over to the ALU that we have included in the solution. You may continue to use the ALU you designed with the carry-lookahead adder, but it will be easier to use the Verilog version. If you use our ALU, you can write Verilog to compute these flags. Otherwise you will have to use gates.

OF should be high if adding or subtracting caused an arithmetic overflow (e.g., an overflow occurs is adding two positive numbers results in a negative number; what other cases are there?). You might want to refer to Chapter 3 of the CSapp text for a discussion of how to compute overflow.

After you have compiled your ALU, right-click on the ALU symbol in your CPU block diagram, choose “Compare Symbol with Contents…” and update the symbol in your

block diagram. You need to do this because you have added ports to this module and you need to let Aldec add these ports to the symbol for you. You will probably want to then right-click again and choose "Edit Symbol in Separate Window" in order to put the outputs in a reasonable-looking place.

You now need to save these condition codes in the condition code registers. Add a new module called "y86_conditional" to your cpu schematic to hold these registers. Of course, the only time you save the condition codes is when an OPl instruction is executed. That is, you will enable these registers only when an OPl instruction is executed. When you add the 3 registers for the condition codes, add a control signal for the enable signal, which will be generated by your controller module. Now, whenever your processor executes an OPl instruction, the condition codes generated by the ALU will be saved in the condition code registers.

2. Evaluating Conditional Instructions

Now that we have the condition codes saved, we need to use them to execute our conditional jump and register move instructions. For each of these instructions, we will either perform the jump/move, or ignore it. This decision is just a (complex) Boolean function of the instruction and the condition codes. For each of instruction types below, write the Boolean expression using just the three condition codes. The final decision is just the OR over all these cases. That is, the jump or move should be performed if any of these condition code expressions is true. Note that the Y86 instruction set was conveniently designed so that the jump and move instructions share a single set of function codes, so the decision can be used for either instruction type.

(You may find the Verilog exclusive-or operator ^ handy.)

B = A	**ZCC**
B ≠ A	
B < A	
B ≥ A	
B ≤ A	
B > A	

Now that you have the Boolean expression, add a Verilog assign statement to your y86_conditional module that computes this logic function. Add this signal as an output of this module.

3. Executing Conditional Moves

Let's implement the conditional moves first, since they are really just an extension of the register-register move. In fact, they share the same instruction code. Modify your y86_controller module to add y86_conditional output signal as an input, and

modify the controller logic so that it writes to the register file when there is a conditional move instruction and condition is true. This should be a simple extension to the logic you already have.

4. Executing Conditional Jumps

A jump instruction sends the program to a different address by putting that address in the program counter. That is, instead of just incrementing to the next instruction, the PC gets its next value from the instruction itself. In order to implement branches, you will add a new module (y86_selectPC) to the processor that selects either the incremented PC value, which you have already implemented, or—in the case of a (conditional) jump—the destination address provided by the jump instruction.

Modify the y86_selectPC module to implement the conditional jump logic. You already have the condition signal: if the instruction is a jump, and the branch condition is true, then instead of using the incremented PC value, you will instead pass along the address from the instruction to the program counter.

4. Test the new instructions

The **v2test.zip** archive contains unit tests for all the new instructions. You should use these to test and debug the instructions. You will find it useful to step through the test program and the Y86 ISA simulator side-by-side. There is a final test program in this archive called **v2testall.ys** which rolls up all the tests into one and produces a signature.

5. Write a program

Now use the program you wrote for Homework 4 to multiply M x N, where M and N are "input" via irmovl instructions at the beginning of the program. Run this program on your processor for the values 13 x 17 and submit the console log after running the simulator.

Turn In

For this assignment, you will be placing a lot of files in the Drop Box. Please put all these files, including zip files, into a single zip file with your name on it.

1. Run the v2testall.ys test program on your CPU design and turn in a copy of the console log via the Drop Box that shows your simulation run. Please clear the console log before running the simulation so that the log contains only the one simulation run, and truncate the log so it doesn't have lots of halt instructions at the end. The console log file is in the <project>\log directory.

2. Turn in via the Drop Box a copy of your multiply program, along with a copy of the console log for the simulation.

3. Archive your design using your name and submit it electronically via the 352 dropbox. Please use the Archive Design command in the Design menu when preparing your design for submission. If your current design uses blocks from a previous design, don't forget to add that to your current design that you submit. When you add existing files to your design, don't forget to check the "Make local copy" box.

To make sure that the TA will be able to run your design, take the zip file you are going to submit, create a new workspace and add your design to it. Then, try running everything. If it doesn't run there, the TA will not be able to grade your assignment. So it is in your best interest to double check that it runs in a different workspace before you submit it.

4. Print out and turn in all the schematics and Verilog files that you designed for this CPU design. Make sure your Verilog has the appropriate number of comments. You do not have to print any schematics or Verilog files that you did not modify.