ENGR 3410: MP#3
MIPS Single-Cycle CPU

Due: 5pm November 2nd, 2007

1 The Problem

The purpose of this machine problem is to complete your simple 32-bit MIPS single-cycle CPU. The CPU instructions to be implemented are:

LW, SW, J, JR, BNE, XORI, ADD, SUB, and SLT.

Chapter 5 will give some examples of how architectures are put together, and will be useful as you design your own CPU. For this CPU, you will use your two previous machine problems (the register file and the ALU) so you will need to have these fully functional before proceeding to work on your CPU.

I provide some fake memory Verilog modules, an assembler, some assembly code source, and some compiled machine code to get you started. You need to test more thoroughly!

2 Implementation Details

The data memory and instruction memory modules are provided in the files `datamem.v` and `instrmem.v`, respectively. Please find these files on the wiki. I also provide a bunch of test programs on the wiki to help you test the functionality of your CPU. You can change the program loaded by editing the `instr.dat` string in `instrmem.v`. You are responsible for coming up with the top-level testbench for this assignment—use previous machine problems' testbenches as guidance.

You will be given an assembler that will allow you to assemble your own assembly-language test benches into machine language that can be loaded into your system for testing. Please find a Windows command-line executable on the wiki. If you have trouble executing the assembler, please let me know immediately. *Please note, this is not a commercial assembler. Be gentle.*

The control logic for your CPU can (and should) be done in behavioral Verilog. If you do this, you can show the control logic on your schematics as black boxes. You need to indicate how the control logic hooks to the rest of the CPU, but do not have to draw the logic for the control logic itself.

This machine problem is significantly more complicated than the previous ones. Certainly, there are many more failure points in this problem than in the previous as there is much more integration of parts. Additionally, if your previous machine problems do not work correctly, you will need to make sure they are functional before you can test your CPU.

My estimate for completion of this machine problem is approximately 30 person-hours.

3 Requirements

There is no top-level test bench! You are designing your CPU, you must design a way to reliably test it. These tests will be used to prove to us that your CPU works. We have our own tests. These will take the form of **machine code** that gets loaded into memories.

As before, you should be writing test benches for **every** module you build. You are getting good at Verilog, I understand, however, this is a big machine problem, and you will get something wrong in putting it all together. Tests are your friend.

One good way to structure your test bench is to make a generic CPU test bench that instantiates your entire CPU (control and datapath) as well as both memories. These memories are simply Verilog files that fake memories and fill them with values from another file. **Look at the code for examples of using memories.** Then, you just change the contents of the memories and voila, your CPU executes a different piece of code. Observe the results as generated in the register file or in the input to the data memory write line. You have to instrument these "probes" for any useful data to come out.

Write some assembly code! Assemble it with the provided assembler and make sure your CPUs work!

4 Deliverables

4.1 Write-Up

I expect a semi-formal "lab write-up" of this machine problem. It does not need to be as rigorous as lab notebooks in other, more experimental classes. It should include, at a minimum:

- A brief write-up of the experiments
- Files of all Verilog code — modules *and* test benches
- Simulation output of the working CPU

We would prefer this packaged as a single document (Word, LaTeX, PDF) and supporting Verilog code, in a single, well-named archive file (ZIP or TAR).

Please name this file after your team. So if you are *Team Smack*, your directory would be "teamsmack", and you would ZIP that up into a file "teamsmack.zip".

Other notes:

- You may turn in one deliverable for all group members
- Please email your documents to myself and John Morgan
- We expect all group members to participate in every aspect of this machine problem
- Please check out the tutorials on the class wiki. They are actually useful, I promise.

4.2 Demos

Demos will be done in class, November 7.

5 Hints and Tips

Some of these are repeated from the last machine problem because they are so important.

- Did you notice that there were tons of demo assembly codes and no real CPU test harness? You have to write it. It literally instantiates the CPU parts and ticks a clock. Easy. Now build the CPU.
- You must write some assembly code and test it to make this complete!