ENGR 3410: Lab #3
MIPS Single-Cycle CPU

Assigned: October 26, 2005
Due: November 14, 2005

1 Introduction

For this lab you are to design a 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 labs (the register file and the ALU) so you will need to have these fully functional before proceeding to work on your CPU.

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 labs' 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.

2 Check-in Required

This lab is significantly more complicated than the previous labs. Certainly, there are many more failure points in this lab than in the previous as there is much more integration of parts. Additionally, if your previous labs do not work

correctly, you will need to make sure they are functional before you can test this lab.

My estimate for completion of this lab is approximately 30-45 person-hours. As in the previous lab, in order to keep you on track, your team *must demonstrate what you have done, in person, by November 3rd, and November 10th, outside of class.* This is a change from the previous lab where the intermediate evaluations were during class. It will be a quick assessment, and you will be graded based upon your overall progress toward the final deliverable.

3 Deliverables

For this lab you will demo the functionality of your CPU and must also turn in, during or before class, paper or electronic versions of the following:

- Your code (with test benches)
- A full schematic at the gate level. It will likely be multi-level (i.e. boxes on an upper level have a lower-level sheet with the details). Since this diagram will reappear in all subsequent labs, photocopy it or do it electronically.

DEMOS ARE REQUIRED, WHETHER YOUR LAB WORKS OR NOT

If you do not demo your assignment, you automatically get a 0. Missing your demo slot without prior approval will impose a late penalty on the entire lab.

4 Hints and Tips

4.1 Testing

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 myself and Mike Curtis 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 lab, 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 both 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.

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