ENGR 3410: Lab #2
MIPS 32-bit ALU

Assigned: October 12, 2005
Due: October 31, 2005

1 Introduction

The purpose of this lab is to create the arithmetic-logic unit of your MIPS-style microprocessor. You will be designing and implementing a simple 32-bit MIPS ALU. The ALU functions to implement are ADD, SUB, XOR, and SLT. Examples of this type of architecture are shown in our textbook. The overall block diagram of our design will look a little like the figure below (Fig. 1).

2 Check-in Required

My estimate for completion of this lab is approximately 20-30 person-hours. This lab is in some ways harder, and some ways easier than the last lab. In order to keep you on track, your team *must demonstrate what you have done, in person, on both October 19th, and October 26th, in class.* It will be a quick assessment, and you will be graded based upon your overall progress toward the final deliverable.

One way to easily break it up is to develop the ADD/SUB in the first week, the XOR/SLT in the following week, and put it all together in the last several days.

3 Implementation

The ALU has 7 ports. These ports are the two input ports A and B, the output port, ALU control, zero detect output, overflow detect output, and the carryout output. The ALU control line assignments are given below. Please use these inputs to select the ALU function. Remember that the turn-in format and design requirements from the previous lab apply to this lab as well.

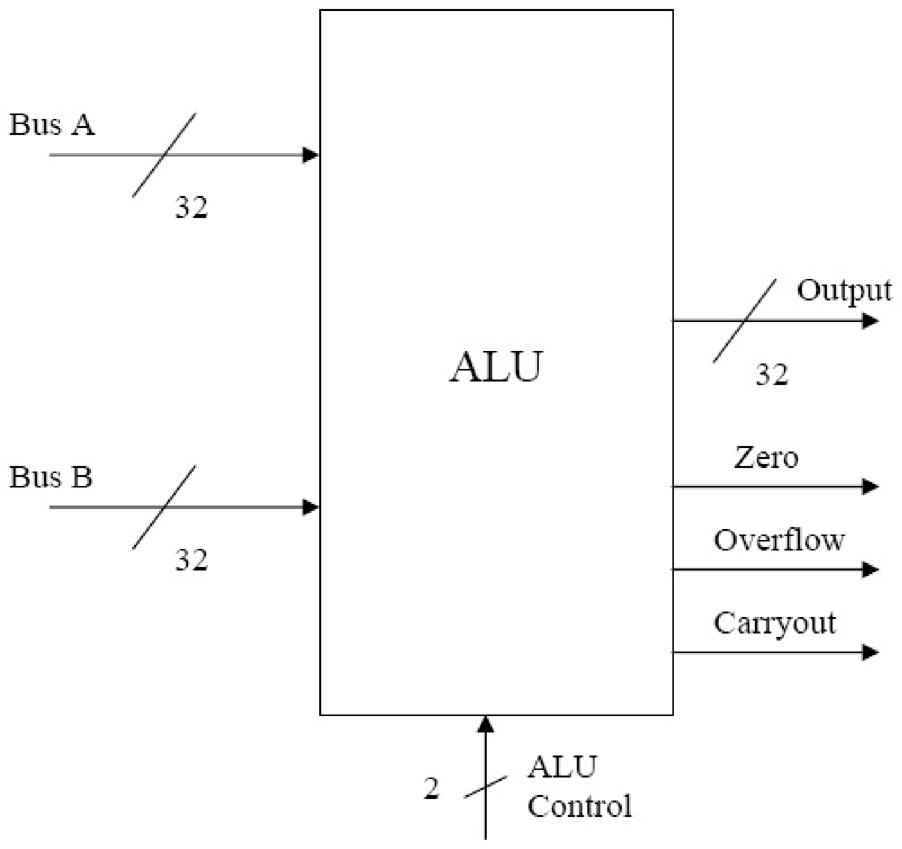

Figure 1: A block diagram of the ALU.

ALU CONTROL LINES	**FUNCTION**
00	ADD
01	SUB
10	XOR
11	SLT

4 Lab requirements

- Use the file "alustim.v" as your test bench. You can find this file on the wiki. You should alter the testing as necessary to make sure your unit works. I have my own test bench for use during the demos, so you must make sure your ALU takes the same inputs and outputs, in the same order, as is presented in the provided test bench. Write a bunch of tests!

- All logic must be gate level, structural. That is, built from explicit AND, OR, NAND, NOR, XOR, etc. gates. No assign statements (except an assign to set a wire to a constant value), CASE statements, etc.
- You may use behavioral Verilog for your test benches.
- All gates have a delay of 50 units. Processor performance won't be a grading criteria for the class (unless you do really ridiculous things), but you need delay to show how things behave.

5 Deliverables

For this lab you will demo the functionality of your ALU 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.

6 Hints and Tips

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

- Test EACH MODULE you make. There is literally 0% chance that you will write all these pieces without testing them, then slap them together into an ALU and it will just work. Add to the fact that this is now a conglomeration of hundreds if not thousands of gates, it is hard to debug when it inevitably does not work.
- As with the last lab, the provided test bench is really just a skeleton for something **you** should be writing to test each module you make. The testing here is **far** from exhaustive, and **far** from acceptable. Heck, it doesn't even try and test the XOR or SLT instructions. This is *deliberate.* I give you enough examples to see how to construct the tests, while having you figure out the test cases yourself.
- Check for typos. Verilog won't really tell you when you've used a signal that doesn't exist.
- Use the concatenation operation. Check the verilog tutorial.