Lab 4 – EC413 Computer Organization

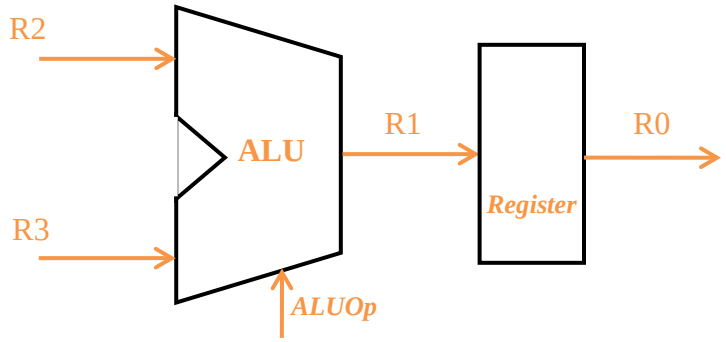
The purposes of this lab are to continue to gain experience with

- hierarchical design,
- useful components (i.e., an ALU), and
- creating good test benches that have a convincing set of test cases and automatic verification.

This lab also adds

- a sequential component
- the need for a more complex testbench to handle sequential logic and multiple functions
- parameterization using the Verilog constructs **for**, **generate**, and **parameter**.

In this lab you will design, implement, and test a 32-bit ALU and output register:



The wires (buses) are highlighted in the diagram.

R0	32 bit	Register output
R1	32 bit	ALU output
R2	32 bit	ALU input 1
R3	32 bit	ALU input 2
ALUOp	3 bit	ALU operation

The ALU has the following functions:

AOp2	AOp1	AOp0	Output	Function Name
0	0	0	R1 = R2	MOV – Pass Through
0	0	1	R1 = ~R2	NOT
0	1	0	R1 = R2 + R3	ADD
0	1	1	R1 = R2 - R3	SUB
1	0	0	R1 = R2 | R3	OR
1	0	1	R1 = R2 & R3	AND
1	1	0	R1 = 1 if R2 < R3, else 0	SLT (signed)

Behavioral requirements:

- The ALU generates a final carry out. This is necessary to help compute SLT (overflow cases) but otherwise does not need to be output.
- (extra credit) The SLT must handle a number of different cases appropriately, including when the result of the subtraction results in an overflow.

Implementation requirements:

1. No gate delays this week!
2. As in Lab 3, use structural (gate level) Verilog for the ALU. The register and the MUX can be behavioral.
3. Use only AND, OR, and NOT gates (as primitives).

4. Design must be hierarchical, i.e., using blocks which you can reuse. In particular, you must use a "bit-slice" style design for the ALU (as we covered in class) with 1-bit modules.
5. Your ALU should be parameterizable to support any word size.
   Your timing diagrams should include ***representative examples and corner cases for <u>each</u> function***.
   For example:
   MOV & NOT & OR & AND → random value
   ADD →big values w/ and w/o overflow, small values, negative + positive
   SUB → same values, difference is a negative/positive, subtract 0, random values
   SLT → R2 is bigger (same sign - positive/negative), R2 is smaller (same sign - positive/negative), R2 = R3 (both negative/positive/0), (R2-negative & R3-positive) / (R2-positive & R3-negative), R2 = 0 & R3 - positive/negative, R2 = positive/negative & R3 = 0
6. Your functions must be verified in the test bench with the given behavioral code.

Implementation recommendation. During prelab you will learn about parameterization and create a parameterized Adder. Recommended order for the rest of the lab is as follows.

1. Create the complete structural 1-bit ALU module (single "slice"), but without SLT. This module should have a 3-8 1-bit MUX. [1-bit so that it is part of the slice, as in the text; 3-8 means three selects to choose among up to eight inputs.]
2. Test the module thoroughly making sure all functions and the MUX are working.
3. Create your top module (based on the prelab FA top) for the ALU and test.
4. Add the register and test.
5. Add SLT and test.

To be provided: The behavioral code for test.

To be covered in discussion and Prelab:

- Parameterization using FOR, GENERATE, and PARAMETER
- A parameterized full adder based on a 1-bit FA module
- A parameterized register
- Testbench for parameterized FA with register
- SLT

An important component of your grade will be style. For example:

- Avoid unnecessary copy-pasting of code.
- Keep your top level module simple.
- Include a reasonable amount of comments in your code. Describe inputs and outputs of modules.
- A name given to a signal should explain its function: e.g., clocks should be labeled as clk and not C.

Report guidelines:

- Briefly describe each module
- Give the hierarchy of your design
- Include representative simulation results and waveforms

Grading guidelines:

- Demonstrate ALU functionality for all functions [40 points]
- Report
  - Code and Timing diagrams showing all ALU functionality except for SLT [40 points]
  - Report formatting, code style [20 points]

Extra Credit -- SLT: Getting this to work often takes a little more time than the others. A good strategy may be to add this last after the rest of the ALU is working. Also, the solution is not strictly bit-slice . [10 points]