EC413 Computer Organization
Lab 5 – ALU

Overview: The purpose of lab 5 is to continue to gain experience with:

Hierarchical design

Useful components (i.e., an ALU)

Creating good test benches that have a convincing set of test cases and automatic verification

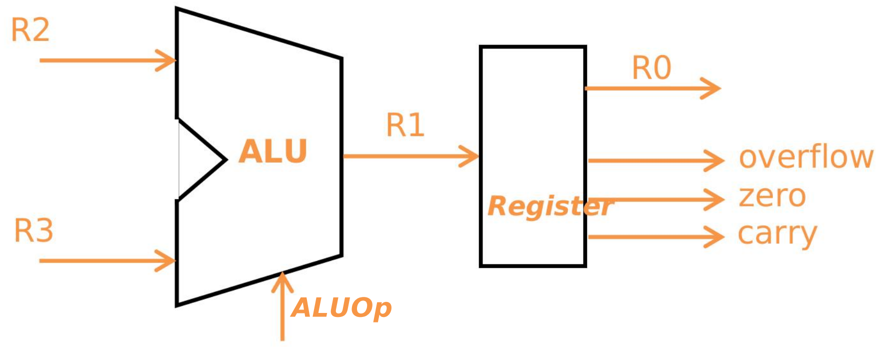
This lab also adds:

A sequential component

The need for a more complex testbench to handle sequential logic and multiple functions

Parameterization using the Verilog constructs **generate**, and **parameter**

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



The lines used as inputs and outputs to the ALU are highlighted in the diagram.

R0	32 bit	Register output
ZERO	1 bit	Register output
OVERFLOW	1 bit	Register output
CARRY	1 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
0	0	1	R1 = ~R2	NOT
0	1	1	R1 = R2 & R3	AND
1	0	0	R1 = R2 | R3	OR
1	0	1	R1 = R2 - R3	SUB
1	1	0	R1 = R2 + R3	ADD

Behavioral requirements:

The ALU generates a final carry out, overflow and zero output

Zero output flag has to be set when output is 0

Carry output flag has to be set when the register cannot properly represent the result as an unsigned value (no sign bit required)

Overflow output flag has to be set when the register cannot properly represent the result as a signed value (you overflowed into the sign bit)

Implementation Requirements:

1. No gate delays this week!
2. As in Lab 4, use structural (gate level) Verilog for the ALU. The register 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 each 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

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.
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.

To be provided: The behavioral code for test.

Deliverables:

All of your .v files.

A lab report, including:

- A description of the modules, and how they satisfy each of the implementation requirements.
- Waveforms showing all ALU functionality, along with a brief explanation.

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

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.

Grading guidelines:

Demonstrate ALU functionality for all functions [40 points]

Report

- Code and Timing diagrams showing all ALU functionality [40 points]
- Report formatting, code style [20 points]