Lab 2: Arithmetic Logic Unit

Course: COEN 122L
TA: Brandon Quant (bquant@scu.edu); Andrei Negulescu (anegulescu@scu.edu)
TA Office Hours: Brandon - Tuesday, Wednesday 1:15pm - 2:15pm; Andrei - Thursday 1:15pm - 2:15pm

Description

The Arithmetic Logic Unit or ALU is a hardware component responsible for performing certain arithmetic operations on a set of inputs. In our case, the ALU needs to handle the following operations: ADD, INCREMENT, NEGATE, SUBTRACT, and PASS A. These are detailed in the last page of your project description. The ALU determines which operation it needs to perform based on a set of input control signals. For this lab the ALU we will use has two data inputs (A and B), four control inputs (ADD, INC, NEG, and SUB), one data output (OUT), and two flag outputs (Z and N).

Assignment

In this lab you will write a description for an ALU using the System Verilog language. Your ALU should follow the specifications given on the last page of the project handout. You will write both the module description and a testbench to verify that your circuit functions correctly. For this assignment doing a full gate level implementation will be very tedious as each of the data inputs is 32 bits. Therefore I am giving you two options.

1. You may do a gate level implementation with 8 bit data inputs for this week and then use a dataflow implementation in your final project (you will have to write 2 versions of the code).

2. You may do a gate level implementation with 32 bit data inputs for this week and then re-use that code in your final project (you will only have to write 1 version of the code).

To receive full credit you will need to demo your working code and turn in copies of your source code and testbench.

Approach

There are several ways to design the ALU, and you may design it in any way that works. Here I will provide some thoughts you may find helpful to guide your design.

Looking at the truth table for the ALU we see that the ALU output data can have 1 of 5 forms (we don't need to worry about the don't care case because whatever data is being outputted in that case is unimportant). Therefore we could design 5 smaller circuits to generate the data in the 5 different cases and then use a multiplexer to select the appropriate output based on the input control signals. While this approach is very modular, we can condense the design by making some observations about the ALU's operations.

One key observation about the ALU is that the output data is always driven by both of the input data signals A and B. Thus, the output data of the ALU is always a combination of two distinct signals. To see what we mean consider re-writing the ALU truth table OUT column as follows:

add	B + A	→	B + A
increment	B + 1	→	B + 1
2's complement	-A	→	0 + (-A)
subtract	B - A	→	B + (-A)
pass A	A	→	0 + A

From this new table, we can see that the ALU output is always the sum of two signals, the first being either B or 0 and the second being A, 1, or -A. From this point we can begin to list the actual circuit components the ALU needs to generate these signals: combinational logic to generate -A, a way to generate signal 1 (B or 0), a way to generate signal 2 (A, 1, or -A), and a full adder to sum the two signals. Let's look at each individually.

Combinational Logic to Generate -A

For this class we will represent negative numbers using the 2's complement form. So to get the value - A we need to write some logic to obtain the 2's complement of the input signal A. To do this we need to flip the bits of A, and then add 1 to the resulting value using an adder.

A Way to Generate Signal 1

Signal 1 is either the input signal B (which we already have) or 0 (which we also have). The question is how do we determine which option we want to pick. Looking back at the ALU truth table we see the combinations of the control input signals that select B as well as the combinations of control input signals that select 0. We can use these combinations to drive the select for a 2 to 1 multiplexer whose output will be our signal 1.

A Way to Generate Signal 2

Signal 2 is either the input signal A, 1, or the output of the 2's complement logic that results in -A. Again we can look at the ALU truth table to see which combinations of control input signals select which of the values. We can use these combinations to drive the select for a 3 to 1 multiplexer whose output will be our signal 2.

A Full Adder

This component is a standard combinational adder circuit. You may choose to implement either a ripple-carry or a carry-lookahead adder. You can use this same adder design for the adder needed in the 2's complement logic.

Putting It All Together

Now we have a good idea of how the ALU circuit will look. We also know what modules we need to design to build the ALU:

- Full Adder
- 2 to 1 Multiplexer
- 3 to 1 Multiplexer
- 2's Complement Generator

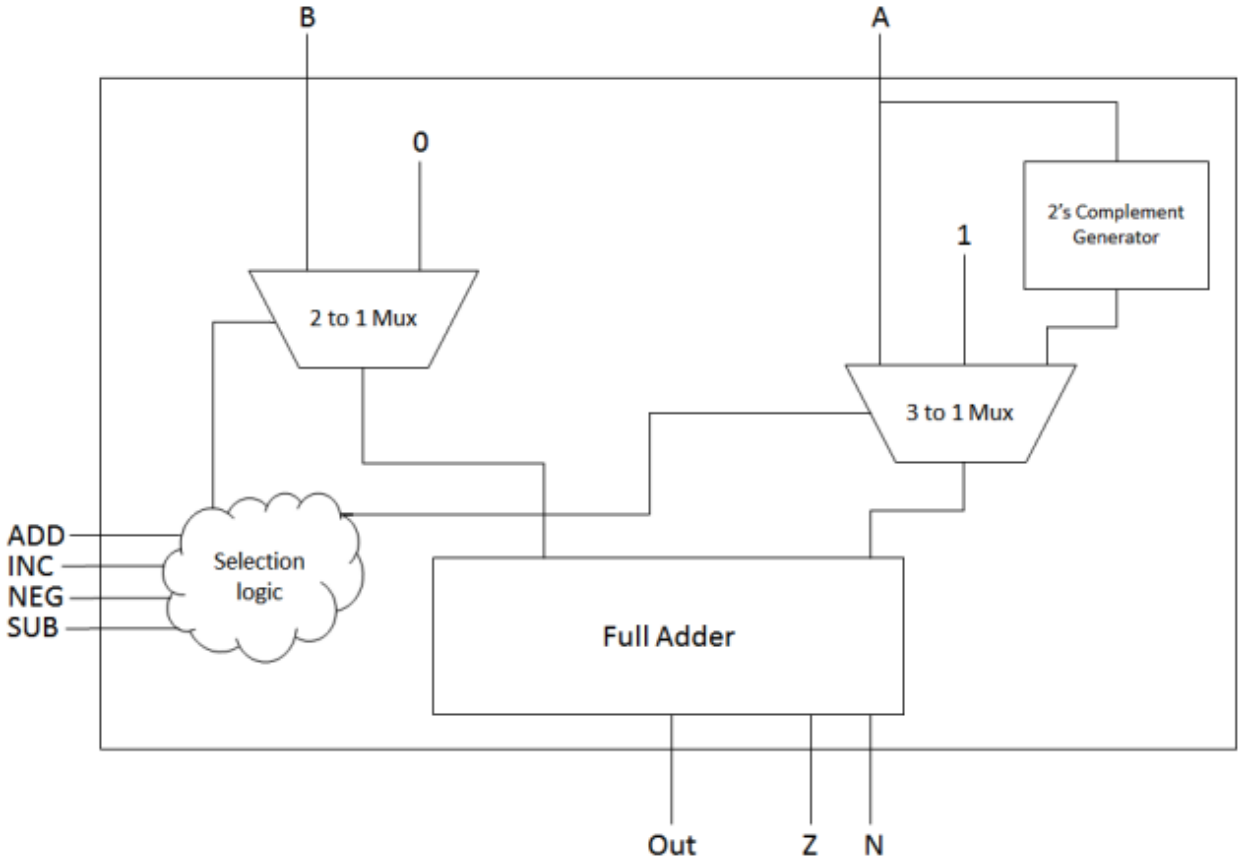

Figure 1: Block diagram of ALU

The final component of the ALU circuit is the logic for the two output flags. The output flag Z should be set to 1 if the output data is equal to zero. The output flag N should be set to 1 if the output data is negative. The diagram shows the signals as outputs from the full adder but you may want to set them in the ALU module based on the output of the full adder rather than inside the adder module itself.