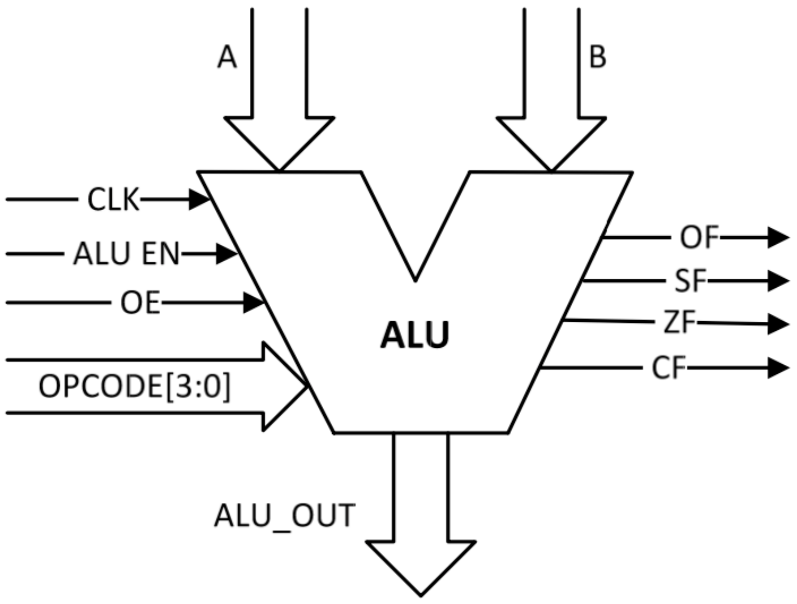
ECE 526L Spring 2021

Lab 8

Experiment #8, Arithmetic-Logic Unit Modeling

In this experiment, you will model an arithmetic-logic unit, or ALU.



1. Create a Verilog module of the ALU. Use a header similar to the one below or write one using SystemVerilog constructs:

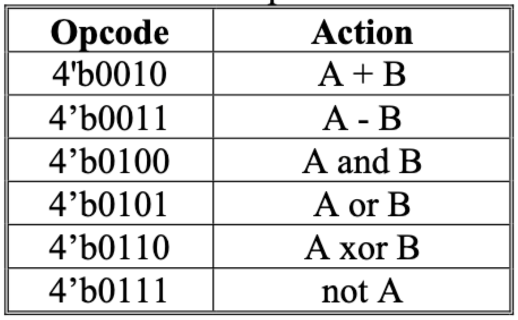
```
`timescale 1ns/100ps
 module alu(CLK, EN, OE, OPCODE, A, B, ALU_OUT, CF, OF, SF, ZF);


parameter WIDTH = 8;
 output [WIDTH - 1:0] ALU_OUT; output CF, OF, SF, ZF;
 input [3:0] OPCODE;
 input [WIDTH - 1:0] A, B;
 input CLK, EN, OE;
.

.

.

endmodule
```

Save your file as *alu.sv*.

Use localparams for all the opcodes, but avoid use of any Verilog keywords for any opcodes. Note that the built-in Boolean primitives do match some of the opcodes and these names should thus not be used for any localparam names. Remember that while the simulator is case sensitive, downstream tools may not be.

The ALU will support the instructions listed in Table 1. All opcodes not listed will not cause any change in ALU output values.

If the ALU is not enabled (EN is logic 0), the ALU will maintain its previous state.



Lab 8 Table 1: Opcodes and Actions

Opcode	Action
4'b0010	A + B
4’b0011	A - B
4’b0100	A and B
4’b0101	A or B
4’b0110	A xor B
4’b0111	not A

The ALU has Overflow, Negative, Zero and Carry flags. These flags are to hold their values between operations that update them. Overflow and Carry are only updated on arithmetic (addition or subtraction) operations. Both arithmetic and Boolean operations will update the other two flags.

Overflow is an indication that a *signed* operation was too big for the ALU. It is detected by having the sign bit take an incorrect value. Adding two positive numbers and getting a negative result would cause the overflow flag to be set, as would adding two negative numbers and getting a positive sum. Note that since this is a clocked module, the output and the operands that are used to form the output are never available at the same time.

Getting the overflow flag to operate correctly for subtraction is convoluted if Verilog subtraction is used. However, if the two's complement of B is first taken and that is then added to A, the flag operation is unchanged from that used for addition.

Carry is set when an addition or subtraction results in a carry out of the MSB position. While the definition for addition carry is clear and universally accepted, that is not the case for subtraction borrow. In this lab, use the Intel definition, which is that the bit is set for subtractions where A < B.

The carry flag only has meaning when the operands are considered to be *unsigned*. Interpretation of the bit values (signed or unsigned) is up to the user of the computer. This applies to both the overflow and carry flags as well as to the data output.

The Zero flag is set when the result is all zeros for any ALU operation.

The Negative flag is set when the MSB of the ALU output is logic one. Again, negative is subject to interpretation by the user. As the hardware designer, you do not know or care how the operands will be interpreted by a programmer.

The ALU has an Output Enable (OE) input. When it is disabled (logic zero), the data outputs must all go to high impedance mode. The flags are unaffected by OE. The data bus is *not* a bidirectional port, so it does not need to be a net. The tristate function can be coded by using an "if" or "case" construct or the conditional operator with one condition simply setting the outputs to high impedance.

Enabling the ALU (ALU_EN asserted) and enabling the outputs (OE asserted) are two distinct functions. Neither is dependent on the other. The ALU can operate internally even if OE is not asserted.

Try your design first with unsigned numbers, then check it again with signed (this requires declaring A and B to be signed variables). Explain the differences. You can use `ifdef or similar for this.

Write a test plan to non-exhaustively but thoroughly verify your design.

Implement your test plan in Verilog/SystemVerilog.

Analyze your results. Turning in pages of simulation without explanation of what each test is doing is not of value. Structure your simulation to include statements on what is being verified. Be sure your design actually works. Pay close attention to flag operation, especially overflow.