1 Introduction

For this lab, you are going to continue the construction of your simulated computer. The resulting component of this assignment is a 32 bit Arithmetic and Logic Unit (ALU).

2 Requirements

The following requirements must be met by your ALU:

1. The assignment will be written in VHDL. The exact signature of the ALU should conform to the following definition:

```
ENTITY ALU IS
    PORT(Value1, Value2: IN STD_LOGIC_VECTOR(31 DOWNTO 0);
         Operation: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         ValueOut: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
         Overflow, Negative, Zero, CarryOut: OUT STD_LOGIC);
END ALU;
```

2. The ALU will provide 5 outputs:

   Value out is the value computed based on the inputs and selected operation of the ALU. The value is 32 bits wide.

   Overflow is 1 if the computed value requires more than 32 bits to accurately represent the correct computation based on the operation and operands involved; otherwise the value is 0.

   Negative is 1 if the result of the computation results in a negative value; otherwise the value is 0.

   Zero is 1 if the result of the computation is zero; otherwise the value is 0.

   Carry out is the carry out bit from the most significant digit.

3. The ALU will take 3 inputs:

   Value 1 is the value of the first operand. The value is 32 bits wide.

   Value 2 is the value of the second operand. The value is 32 bits wide.

   Operation is the encoded choice of operation for the ALU. See Table 1 for a listing of all the possible values. You may assume that no illegal value will be sent to the ALU by another part of the CPU. The operation is 3 bits wide.

Encoding	Operation
0000	Add
0001	Unsigned add
0010	Left shift
0011	Right shift
0100	Subtract
0101	Set less than
1000	xor
1001	nor
1010	nand
1100	complement

Table 1: Table of operation encodings

4. The ALU will perform 10 operations. For the following descriptions, a, v_1, and v_2 will refer to the computed value, input value 1, and input value 2 respectively. All descriptions of the operations use C syntax.

   Add causes $a = v_1 + v_2$ to be performed. This operation may need to set the overflow, negative, zero, and carry out bits.

   Unsigned add performs the same operation as add but interprets the overflow, negative, zero, and carry out bits differently.

   Subtract causes $a = v_1 - v_2$ to be performed. This operation may need to set the overflow, negative, zero, and carry out bits.

   Complement causes $a = \tilde{}\, v_1$ to be performed on a bit-by-bit basis. This operation may need to set the negative and zero bits.

   Xor causes $a = (v_1 \& \tilde{}\, v_2) | (\tilde{}\, v_1 \& v_2)$ to be performed on a bit-by-bit basis. This operation may need to set the negative and zero bits.

   Nor causes $a = \tilde{}\, (v_1 | v_2)$ to be performed on a bit-by-bit basis. This operation may need to set the negative and zero bits.

   Nand causes $a = \tilde{}\, (v_1 \& v_2)$ to be performed on a bit-by-bit basis. This operation may need to set the negative and zero bits.

   Left shift causes $a = v_1 << v_2$ to be performed. This operation may need to set the overflow, negative, zero, and carry out bits. Note: for this operation only, the carry out bit is the value of the least significant bit that was shifted off of v_1.

   Right shift causes $a = v_1 >> v_2$ to be performed. This operation may need to set the zero and carry out bits. Note: for this operation only, the carry out bit is the value of the most significant bit that was shifted off of v_1.

   Set less than causes $a = (v_1 < v_2)$ to be performed. This operation may need to set the zero bit.

5. All gates that are used must have four or fewer inputs.

6. Each gate should have a 5ps delay.

3 Implementation ideas

You may want to develop a 1 bit ALU that you can extend for 32 bits to perform many of the operations. This will not work for all the operations, so you can't do this blindly, but it does work for some.

Note that the ALU does not take a clock input. In your final design, the ALU will have to perform the specified operation within a single clock cycle so that it can have the final result ready to store back in the register file or memory at the end of the instruction. This implies that you cannot build a simple single-bit shifter that happens to execute the right number of times before you go to the next instruction. You must build some kind of barrel or giant mux shifter.

You need to create a test bench that thoroughly exercises your ALU. You do not need to include a clock for the test bench, but it may be helpful for you to determine how long it takes for the ALU to stabilize on a value. This may be the limiting factor for how small you can make the clock period in your final CPU implementation.

When you create your final CPU, you will be connecting the outputs of your register file from the previous assignment to the ALU at times. You may wish to test that you are able to perform this connection after completing all your other testing. This will not be part of your grade for this assignment, but will make your future assignments easier.

4 Deliverables

You should turn in an electronic copy of your VHDL, including all the components you created to construct the ALU hierarchically. Additionally, you must demonstrate your lab to me at a scheduled time.

The project is worth 100 points as follows:

- 25 points overall structure
  - 10 points good hierarchical design
  - 15 points correct connections between components
- 75 points ALU operations perform correctly
  - 15 points Add
    - 9 points handles unsigned values
    - 6 points handles twos complement values
  - 4 points Subtract
  - 3 points Nor
  - 12 points Left shift
    - 3 points handles 1 bit shift
    - 3 points handles 2-3 bit shift
    - 3 points handles 4-7 bit shift
    - 3 points handles 8+ bit shift
  - 4 points Right shift
  - 12 points Set less than
    - 8 points handles unsigned values
    - 4 points handles twos complement values
  - 5 points Overflow set correctly
  - 5 points Negative set correctly
  - 5 points Zero set correctly
  - 10 points Carry out set correctly
    - 5 points Carry correct for shifts
    - 5 points Carry correct for all other operands