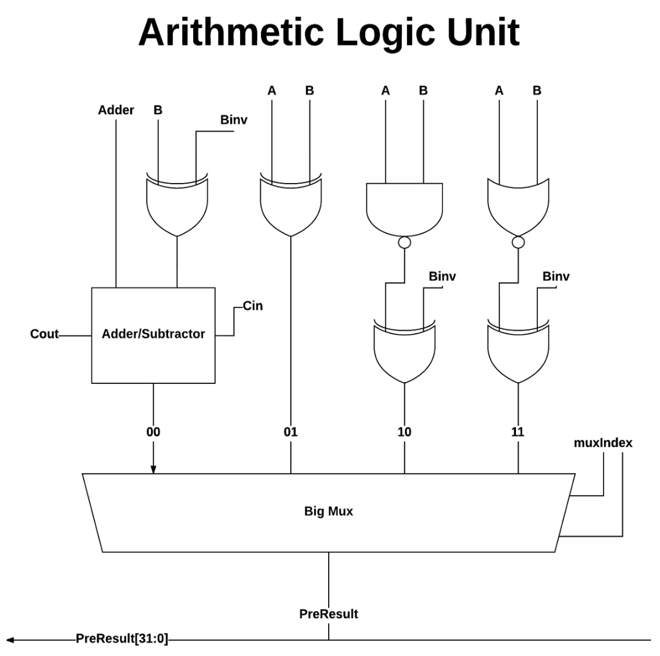
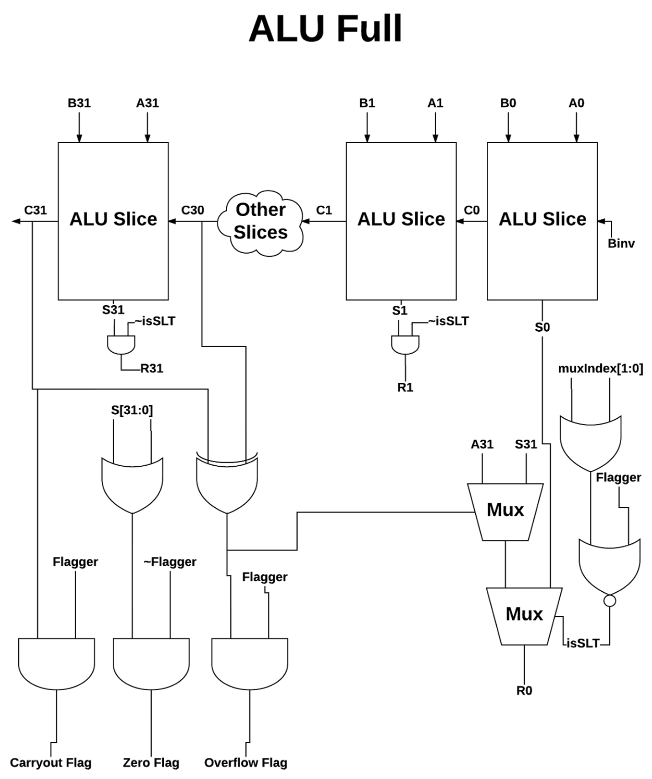
Report

In this lab your team is designing the ALU subsystem. Write a report to the project manager in charge of the entire CPU demonstrating that your ALU design is complete, correct, and ready to be included in the CPU.

Implementation

Discuss any interesting design choices you made along the way.





We implemented our circuit using a Bit-Slice ALU approach. This allowed us to use small and understandable circuits for each bit and compile them together into an answer. Each Bit-Slice ALU was comprised of 4 operations, Add, Xor, Nand and Nor. Of these, Add, Nand, and Nor were invertible into Subtract, And and Or. Lastly, the Set Less Than operation simply used subtract with some extra logic outside of the function. This logic sets all of the output bits to zero except the least significant bit which is then set to the value of set less than. Set less than is calculated based on the sign of the Result and Operand A. We use a mux to simply select between these two. The truth table below shows that the Less than should simply be a selection between the signed bit of the result and Operand A.

Overflow	Sign of Operand A	Sign of Result	SLT
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	X
1	0	1	0
1	1	0	1
1	1	1	X

The Xs in the SLT result are values we don't care about simply because they cannot happen.

Lastly, the overflow flag is calculated through the XOR of Cout[31] and Cout[30] and the zero flag is the negation of the OR of all of the results.

Test Results

For each ALU operation, include the following in your report:

1. A written description of what tests you chose, and why you chose them. This should be roughly a couple sentences per operation.
    a. Add: 6 cases testing both overflow and values
        i. With no overflow: (+, +), (-, -), (+, -), (-, +)
        ii. With overflow: (+, +), (-, -)
    b. Subtract: 6 cases testing both overflow and values
        i. With no overflow: (+, +), (-, -), (+, -), (-, +)
        ii. With overflow: (+, -), (-, +)
    c. XOR: 2 non-zero case randomly chosen and 2 zero cases to test the zero flag
    d. SLT:

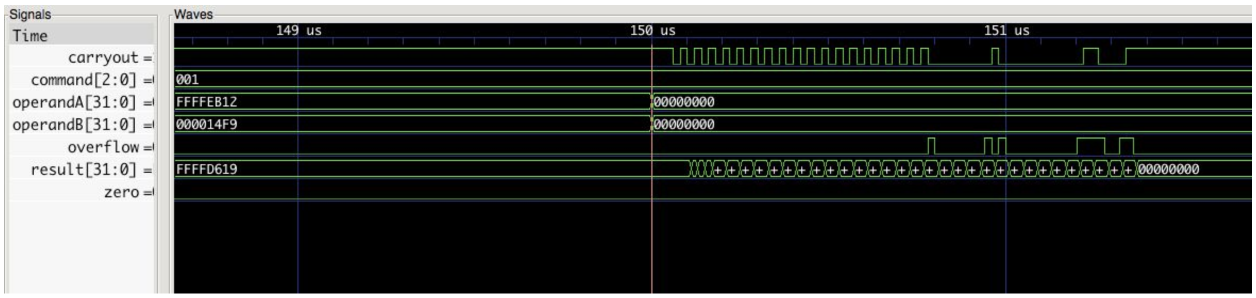
        i. A<B: (+, +), (-,-), (-,+)
        ii. A=B: +, -, 0
        iii. A>B: (+, +), (-,-), (+,-)
        iv. Subtraction overflow cases: b.ii (this is newly added because we had trouble with overflow)
    e. AND: 2 non-zero case randomly chosen and 2 zero cases to test the zero flag
    f. NAND: 2 non-zero case randomly chosen and 1 zero cases (all 1s) to test the zero flag
    g. NOR: 2 non-zero case randomly chosen and 2 zero cases to test the zero flag
    h. OR: 2 non-zero case randomly chosen and 2 zero cases to test the zero flag

2. Specific instances where your test bench caught a flaw in your design.
    a. When we were testing different values for strictly less than, the result was always 1 independent of the truth. This made us find a bug in the code that we reversed the order of our input array for the 2 muxes we were using for SLT. Unlike in other programing language, the first (left most) element in an array is the most significant bit with the highest index. Thus, our 0 and 1 input were reversed. Additionally, InvertB was set as 0 for SLT, which needed a 1 for subtraction. After we fixed those two bugs, the SLT worked correctly.
    b. For add and subtract the overflow cases was failing because I gave around 2^32 operands, which by themselves overflows, because our system is signed. After changing them to be less than 2^31 values, the overflow worked.

Timing Analysis

Simulated analysis in gtkwave:

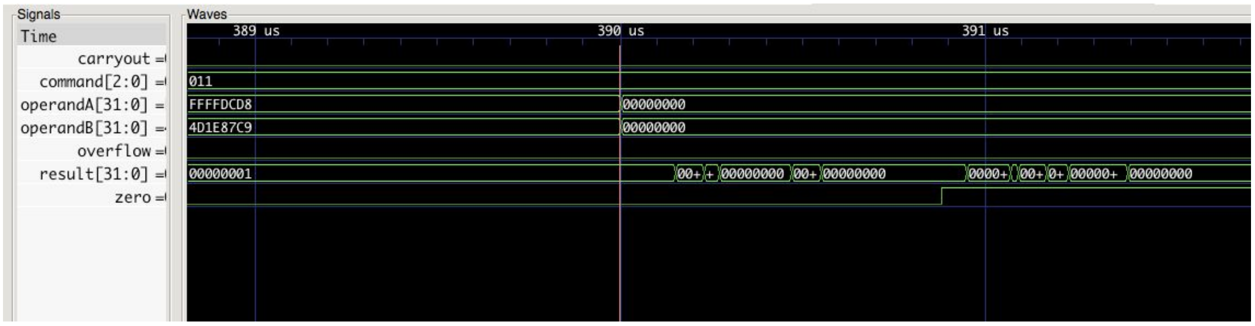
The worst time delay is about #1400 units for result, overflow and carry out to settle.
Below are two scenarios with the worst time delay:
1st Scenerio: 0 subtracts 0:



Because our program convert the second 0 to 11111111111111 and then gives a carry in on the LSB, every bit slice has a carryout to the next one and that delays the process.

Second Scenario:

Comparing 0 against 0. This is actually the same scenario because our SLT uses subtract. But in this case the carryout is not changing (because it is handled in the end).



In general, our ALU has a much lower delay of between #200-1000 units of delay for result to settle. As a reference, a single input NAND, NOR gate is #10 units of delay, where OR and AND are #20 units per input.

Work Plan Reflection

Compare how long each unit work actually took to how long you predicted it would take. This will help you better schedule future labs.

I predicted implementing the logic to take about 2 hrs. In reality it took William 2.5 hrs to implement a basic model, then we discussed and drew out our model for 1.5 hrs, then we worked together to implement all functionalities for about 2 hrs. Then we found out SLT was not working, so it took an extra 2 hrs to fix that, in total taking William 7.5 hrs and me around 4 hrs.

For the testing I also estimated 2 hrs. I spent about 4 hrs making testing automated and completing and writing out all test cases. Before that william did a bunch of testing to make sure the logic worked, which took about 2hrs

For the report it was the most accurate as both of us spent about 1hr on it.
So the whole lab took both of us around 9 hrs to finish rather than 5 hrs.