CMPSC 210
Principles of Computer Organization
Fall 2016
John Wenskovitch
http://cs.allegheny.edu/~jwenskovitch/teaching/CMPSC210

Lab 10 – 8-bit ALU
Due (via Bitbucket) Monday, 12 December 2016
30 points

Lab Goals

- Practice creating circuits using Logisim
- Build an 8-bit ALU!

Assignment Details

For this project, you will be building an 8-bit ALU that can carry out operations ADD, AND, OR, NOR, and NAND. You will accomplish this by creating a subcircuit that works as a 1-bit ALU, then chaining together those 1-bit subcircuits to create the final larger circuit (see Figure 1).

Control

Because the ALU needs to handle 5 operations, you need a minimum of 3 bits of input as a control signal into your circuit. You should encode the operations as follows:

- 000 (0x0) = ADD
- 010 (0x2) = AND
- 011 (0x3) = OR
- 110 (0x6) = NAND
- 111 (0x7) = NOR
- All other signal inputs should light an error LED (Figure 2).

Registers

Rather than creating separate input pins for all eight bits of input A, eight bits of input B, and three bits of control signal, you will be using the Register tool (see Figure 3) found in the Memory menu, and can separate the register value into individual bits using a Splitter (as in Lab 9). You

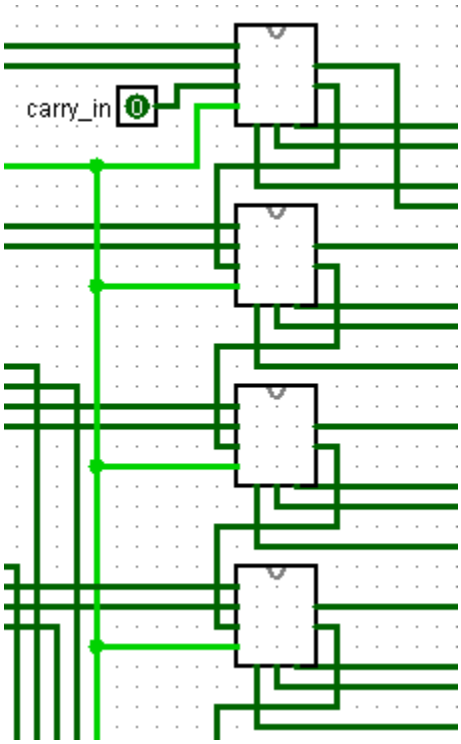

Figure 1: A screenshot from Logisim showing several 1-bit ALUs chained together. Note that the uppermost ALU has a 0 carry_in field, while all of the following ALUs receive their carry_in from the previous ALU's carry_out.

can read further about the Register tool in the Logisim documentation, but in short, the Register tool allows you to store values that are more than a single bit in size. The Q output on the right side of the Register allows you to read the value stored in the Register. The Register value can be set manually (using the hand tool) or through input from other parts of a circuit (say, from your ALU). Note that the value displayed on the register is in hexademical.

Operations

In class, we already created a 1-bit adder, so the ADD operation is already partially completed. Each 1-bit ALU that you create should take 4 inputs: A, B, carry_in, and Control (which is a 3-bit

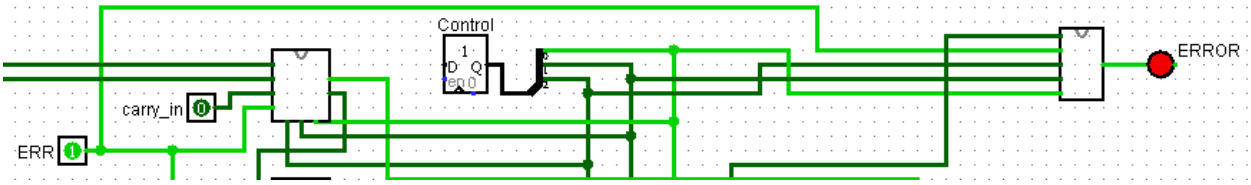

Figure 2: The error LED is lit when control signal 001 = 0x1, an invalid instruction, is received.

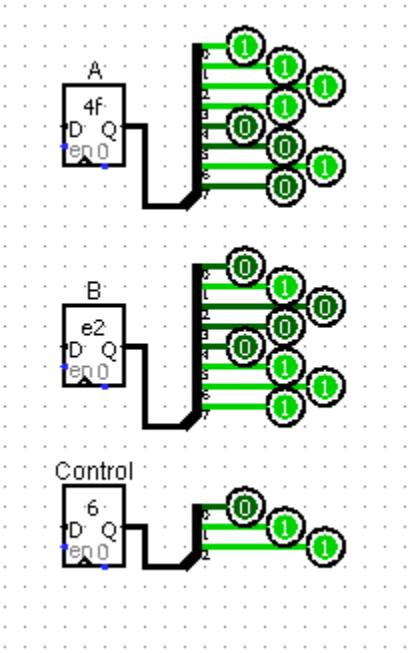

Figure 3: A screenshot of several Registers in Logisim. The upper two, A and B, are 8-bit registers, while the lowest, Control, is a 3-bit register. The individual bits in the Register can be obtained with the use of a splitter (as shown with the output pins in the Figure).

input). Naturally, the carry_in input is only used for the ADD function; the other four functions will only make use of the A and B inputs.

The Control signal will control a multiplexer that selects which output will be provided by the 1-bit ALU. In this way, the ALU is essentially computing all five operations at the same time, and selecting the correct output based on the control signal provided. You must create this multiplexer from logic gates yourself; do not use the multiplexer provided by Logisim. You may find it easier

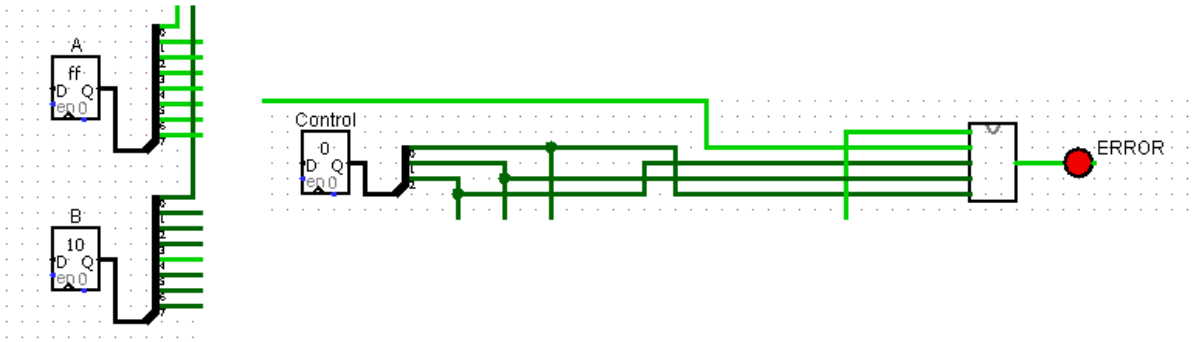

Figure 4: The error LED is lit when control signal 000 = 0x0, the ADD instruction, is received. The values in the registers A and B are 0xFF (255) and 0x10 (16). Adding these two values results in an overflow.

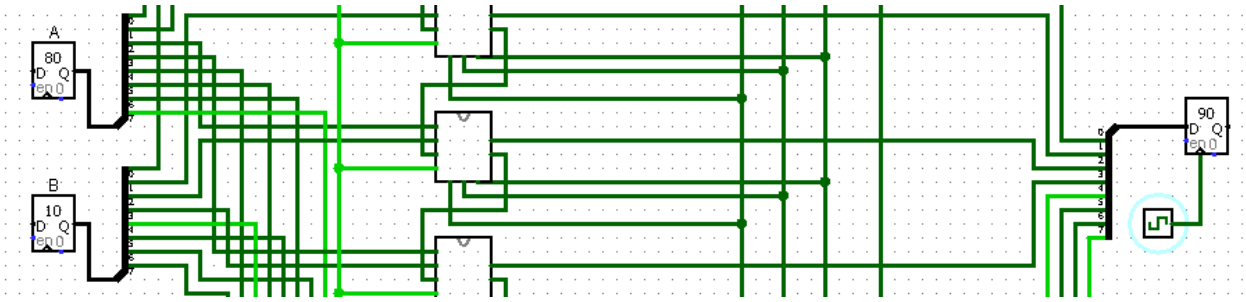

Figure 5: The ADD function performed on input 0x80 and 0x10, resulting in output 0x90. The result Register did not update until the clock was activated.

to create the multiplexer as its own subcircuit.

You must also use 2-input AND and OR gates throughout the project.

Your 8-bit ALU should also light an error LED when overflow is detected. In other words, if the carry_out result of the most significant bit adder is 1, an error should be thrown (see Figure 4).

Finally, you will need to add a Clock to your result Register. The register will only update on a clock tick, so you can either tick manually, enable ticks in the Simulate menu (see Figure 5).

Circuit Development

Recall from the course material that the easiest way to develop a circuit is to follow these steps:

1. Construct a truth table with all possible inputs and the target output.
2. Convert that truth table into a Karnaugh Map and retrieve the simplified expression from the K-Map.
3. Encode that expression using logic gates.

Submission Details

For this assignment, your submission to your BitBucket repository should include the following:

1. Your ALU Logisim file.

Before you turn in this assignment, you also must ensure that the course instructor has read access to your BitBucket repository that is named according to the convention `cs210f2016-<your user name>`.