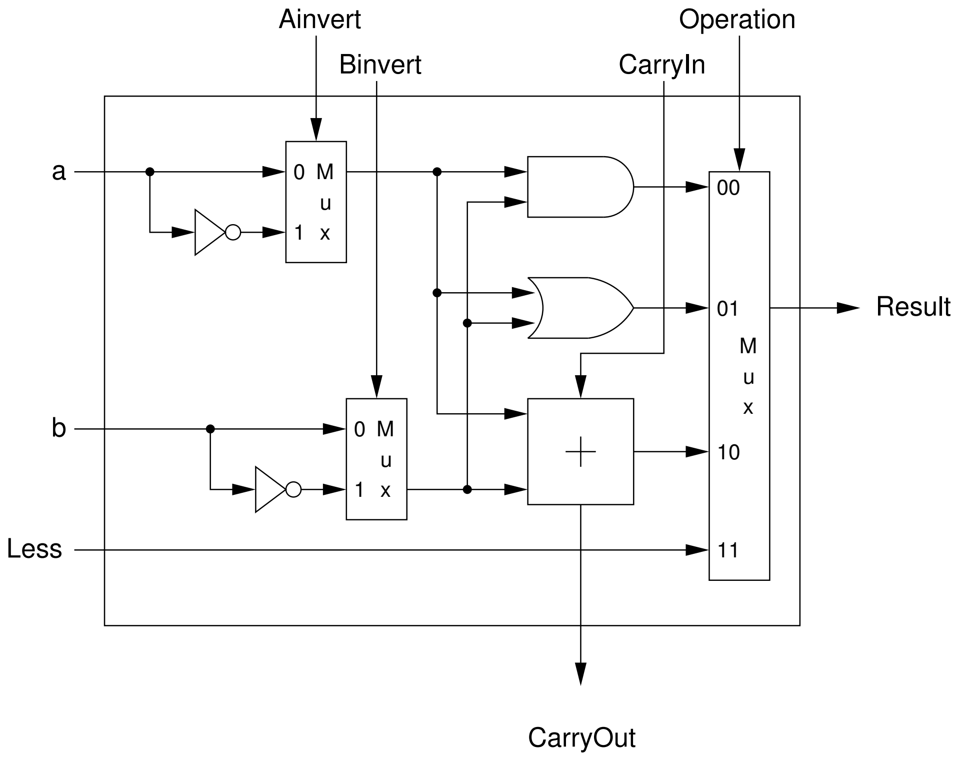
Lab 8: ALU

Submission timestamps will be checked and enforced strictly by the CourseWeb; **late submissions will not be accepted**. Check the due date of this lab on the CourseWeb. Remember that, per the course syllabus, if you are not marked by your recitation instructor as having attended a recitation, your score will be cut in half.

In this lab, you will build a basic 8-bit arithmetic and logic unit (ALU), one of the major workhorses of any computational processor.

Building a One-bit ALU

Recall the following diagram from class for a one-bit ALU, which can perform AND, OR, addition, subtraction, and set if less than operations as shown below:

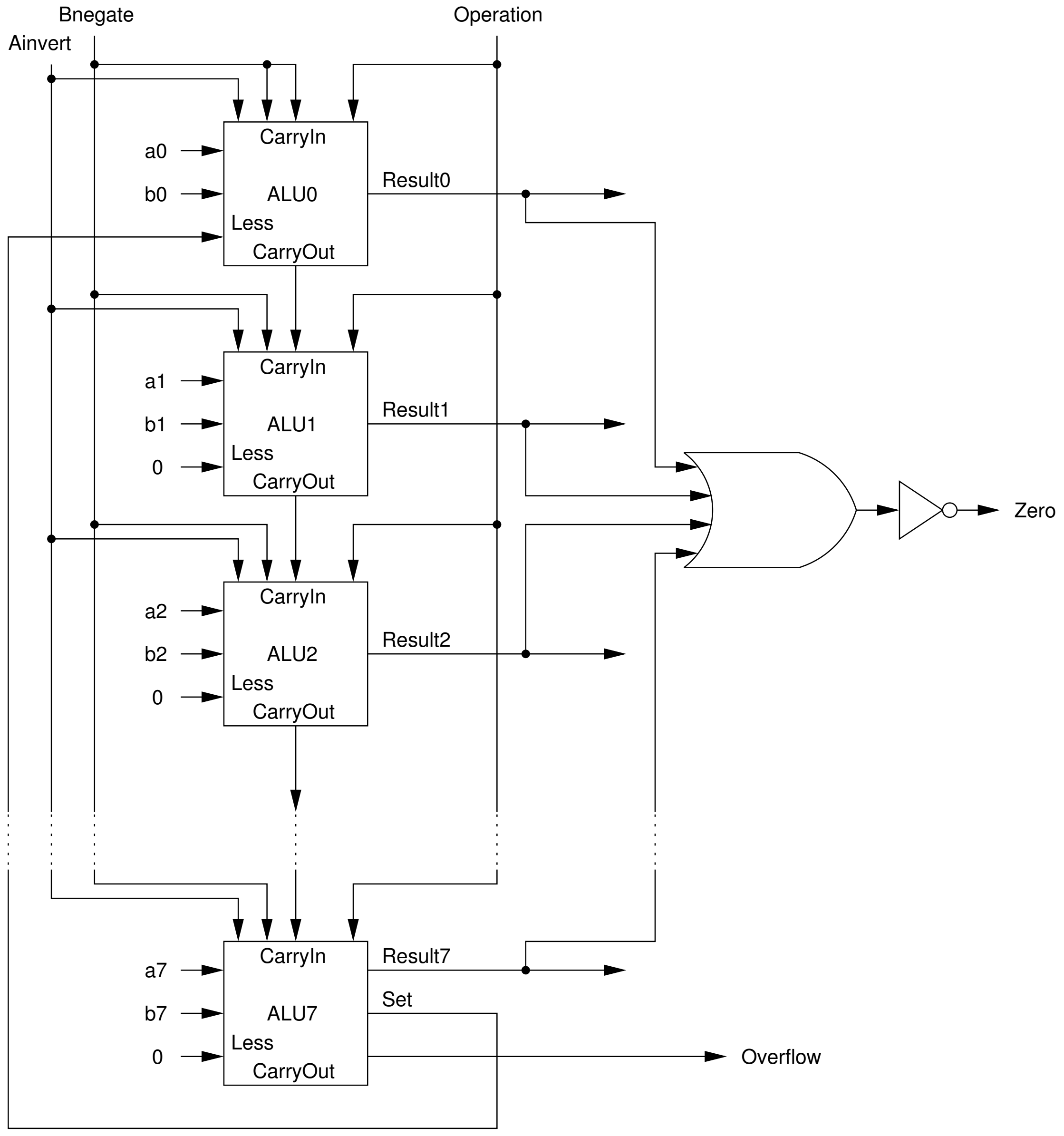


Start a new Logisim project, named it `lab08.circ`, and build the "1-bit ALU" subcircuit as shown above. Make sure you have all sevven inputs (*a*, *b*, *Ainvert*, *Binvert*, *CarryIn*, *Less*, and *Operation*), and all three outputs (*Result*, *Set*, and *CarryOut*). Note that all inputs and outputs are one bit, except for the *Operation* which is a two-bit input.

You are permitted to use the built-in Adder component from the Arithmetic Library as a useful abstraction to keep your circuit file simpler. Note that you can also built a one-bit adder from scratch just like in Lab 6.

Chaining together a multi-bit ALU

A one-bit ALU is nice, but it is not very useful for computation by itself. We can chain one together with its likeminded friends to get some real work done! In the picture below, we chain together eight one-bit ALUs to form an eight-bit ALU:



Using your 1-bit ALU subcircuit, build the "8-bit ALU" subcircuit detailed above. Your input will include a four-bit *ALUOperation* code, (consisting of *Ainvert*, *Bnegate*, and a two-bit *Operation*), as well as two eight-bit operands, *A* and *B*. Your outputs will include *Zero*, *Overflow*, and an eight-bit *Result*. **Be sure to use splitters** to split an combine your input and output values inside this subcircuit, so that you main circuit will stay as simple as possible.

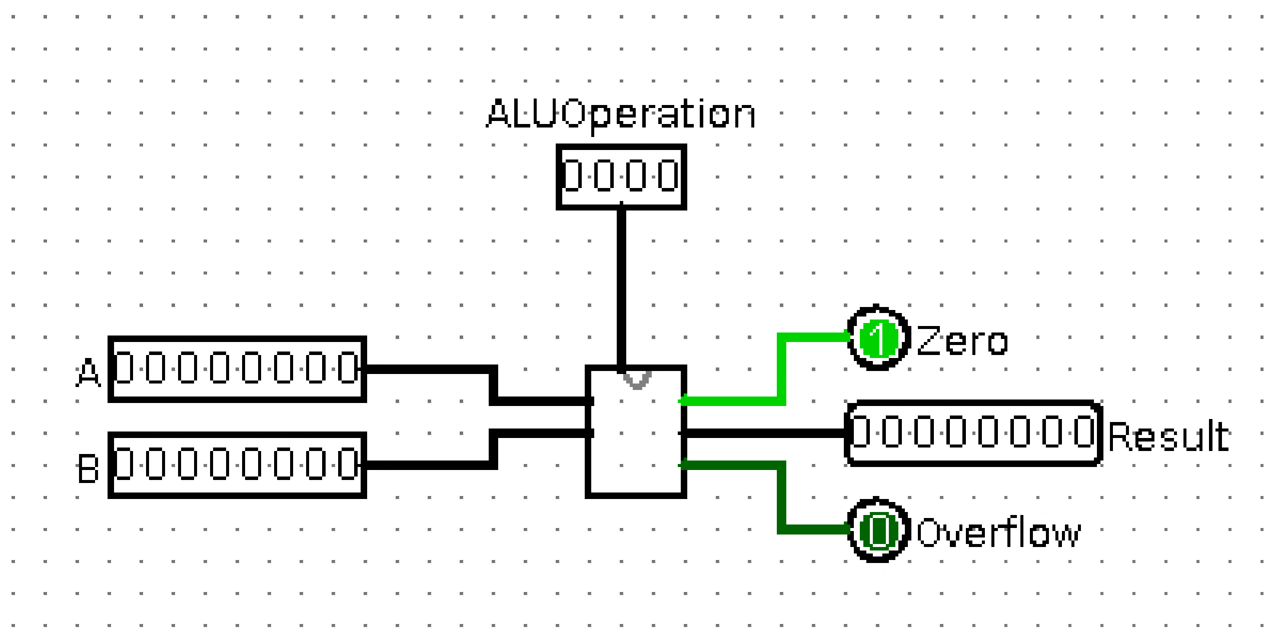
Some things to note:

- The *Bnegate* input serves as **both** the *Binvert* signal to all of your 1-bit ALUs **and** the *CarryIn* to the ALU for the least significant bit (LSB). This is used so that our ALU can support subtraction, since, in two's complement, $A - B = A + (\neg B + 1)$.
- The *Zero* output should be true (1) **if and only if** the value of *Result* is 0. (That is, the value of all of the *Result* bits is 0.)

- The *Overflow* output is derived from (but not equal to) the most significant bit (MSB) of the *Result*; that is, the sign bit. You also need to use the sign bits of the two operands A and B. (See slide 60 of the ALU side deck.)
- Recall that the *Set* output is **only** used from the adder of the MSB. For simplicity, we will allow all of our one-bit ALUs to generate this signal as shown in the Figure above, but we will not connect this signal anywhere when it is not needed.
- The *Set* output is dependent on the *Overflow* condition and must be inverted if overflow occurs. (This is not shown in the figure; see slide 61 of the ALU slide deck.)
- Only the *Set* signal from the MSB is tied to the *Less* input of the LSB, as shown in the Figure above. All other *Set* signals are **wired to a constant 0**. (You can find constants under the "Wiring" library.)

Trying it all together

Finally, **create a main circuit** which uses your 8-bit ALU subcircuit and simple takes in two eight-bit operands A and B and a four-bit *ALUOperation* code, and returns an eight-bit *Result* alongside *Zero* and *Overflow* signals. Remember that the *ALUOperation* consists of *Ainvert*, *Bnegate*, and *Operation*, which are used by different parts of your ALU.



Test your circuit with a few sample inputs to make sure everything is working properly. Congratulations! You've just built a critical component of any processor!

Submission

Submit the file `lab08.circ` via CourseWeb before the due date stated on the CourseWeb.