COMP2611: Computer Organization

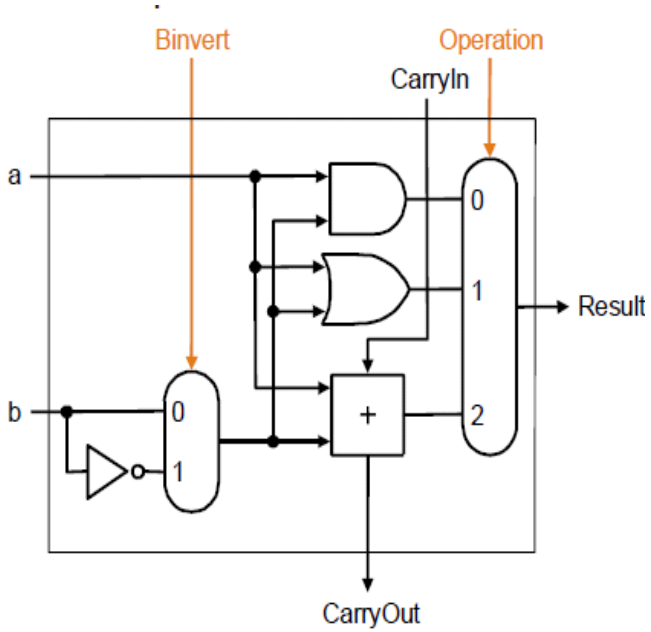
Building an 1-bit ALU with Logisim

Overview

- You will learn the following in this lab:
  - building an 1-bit adder,
  - building a 3-to-1 multiplexor,
  - building an 1-bit ALU.

Reminder : an 1-bit ALU that does AND, OR, Addition, Subtraction 1/2

- The 1-bit ALU we are going to build can perform AND, OR, Addition and Subtraction operations on two 1-bit inputs.



- A X-bit ALU can be built using X of the 1-bit ALUs shown above. Each 1-bit ALU will take care of the operations for exactly one bit.

Reminder : an 1-bit ALU that does AND, OR, Addition, Subtraction 2/2

- The inputs of the 1-bit ALU are:
  - an 1-bit operand "a"
  - an 1-bit operand "b"
  - an 1-bit input "CarryIn"
  - an 1-bit control "Binvert"
  - an y-bit control "Operation", where $y = \lceil \log_2(n) \rceil$, n is the number of inputs to the multiplexor.
- The outputs of the 1-bit ALU are:
  - An 1-bit output "CarryOut"
  - An 1-bit output "Result"
- We need to build ourselves two components, the adder, and a 3-to-1 multiplexor in order to implement the ALU.

Building the adder for the ALU 1/4

- Observation:
  - the subtraction operation can be done using the 2's complement scheme (i.e. through the CarryIn and Binvert inputs), so we just need to consider implementing the ADD operation.
- The inputs to the adder are:
  - the input "a", the input "b", the input "CarryIn"
- The outputs of the adder are:
  - an output to the multiplexor, let's denote it by "SumOut"
  - an output "CarryOut" (this output indicates whether there will be a bit to be carried out to the next digit after the addition operation).

Building the adder for the ALU 2/4

- The truth table for the previous inputs and outputs is:

Inputs			Outputs	
CarryIn	a	b	SumOut	CarryOut
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Building the adder for the ALU 3/4

- The corresponding logic expressions for the truth table using the Sum-Of-Product representation scheme are :

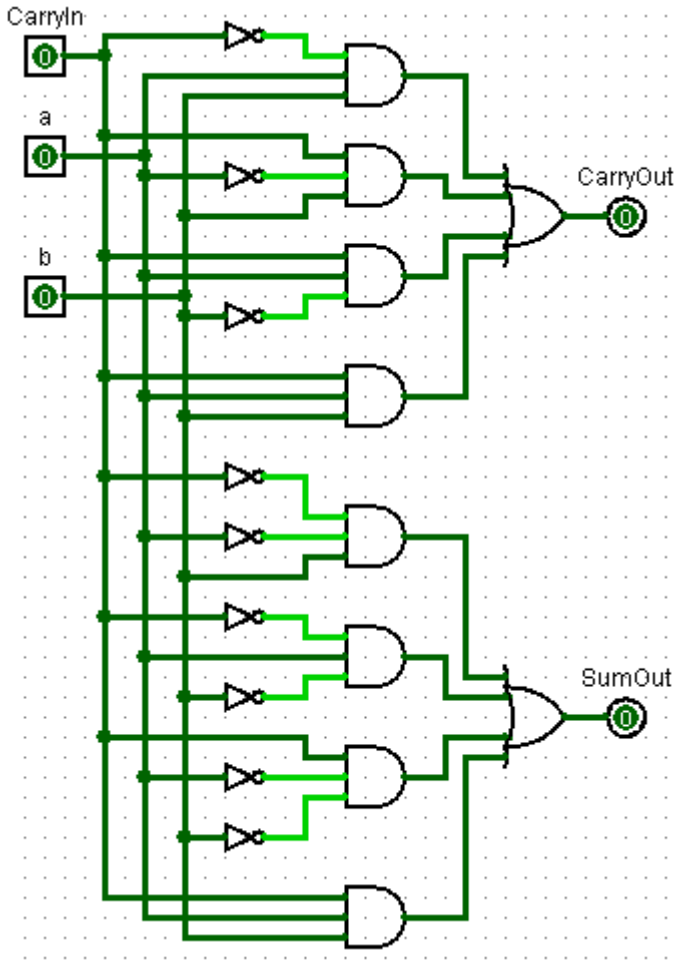
SumOut = (~CarryIn·~a·b) + (~CarryIn·a·~b)
+ (CarryIn·~a·~b) + (CarryIn·a·b)

CarryOut= (~CarryIn·a·b) + (CarryIn·~a·b)
+ (CarryIn·a·~b) + (CarryIn·a·b)

- Do let me know if you have problem to derive the above expressions.
- A question to ask: Is it possible to simplify the expressions for "SumOut" and "CarryOut"?

Building the adder for the ALU 4/4

- Using the logic expression, design the circuit for the adder.
- We have an example circuit at the right.
- Please build your own circuit based on the logic expressions and verify its correctness by poking the input values.
- Save the circuit as "1-bit-adder.circ" for future uses.



Building the multiplexor for the ALU 1/5

- After building the adder, we now need to build the multiplexor.
- The multiplexor selects the output of the ALU among the outputs of the AND, OR, and Addition/Subtract operations.
- We have 3 possible outputs to select from, thus we need $\lceil \log_2(3) \rceil = 2$ bits for the control signal (do ask me if you are not clear on this).
- You can define the way the multiplexor selects input as you wish.

Building the multiplexor for the ALU 2/5

- In the following, we define the truth table for the multiplexor as:

Input	Output
Operation	Result
00	AND (indexed by 0)
01	OR (indexed by 1)
10	Add/Sub (indexed by 2)
11	undefined

- You can define your truth table differently, as long as you remember how you defined it.

Building the multiplexor for the ALU 3/5

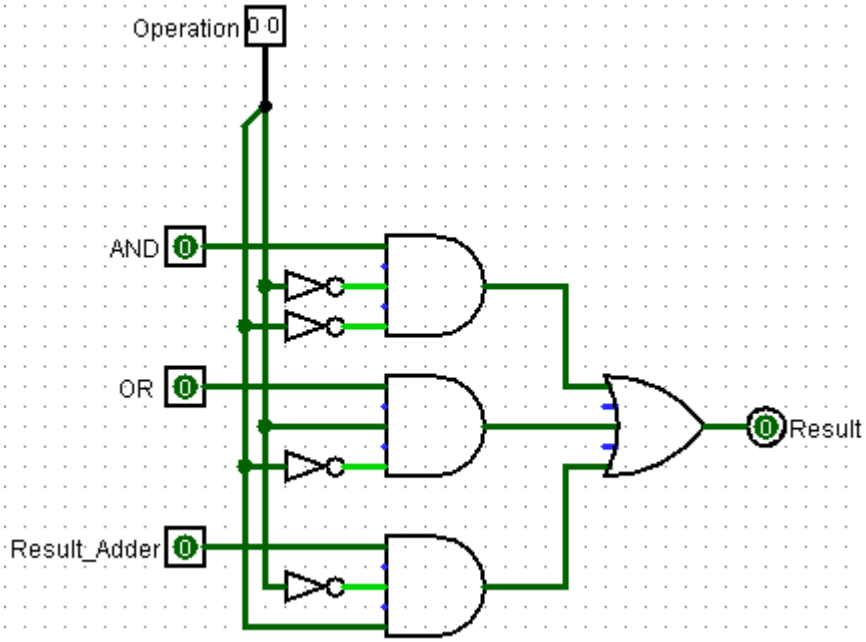
- The truth table tells us that
  - whenever the control signal, "Operation", equals to 00, we should generate an "1" and supply it properly to let the result of "AND" operation go through the multiplexor (suppress all other results),
  - whenever "Operation" equals to 01, we should generate a an "1" and supply it properly to let the result of the "OR" operation get through (suppress all other results),
  - whenever "Operation" equals to 10, we should generate a an "1" and supply it properly to let the result of the adder get through (suppress all other results).

Building the multiplexor for the ALU 4/5

- Start a new Logisim project (press Ctrl+N). Design a multiplexor according to the description. Feel free to ask me questions.
- Hint: you need 4 inputs and 1 output:
  - an input that accepts the result from the "AND" operation,
  - an input that accepts the result from the "OR" operation,
  - an input that accepts the result from the "Adder",
  - a 2-bit control input "Operation",
  - an output that forwards one of the three results to "Result".
- After you have implemented it, poke the inputs to verify it is correct. For example, if the control signal "Operation" is 00, when you flip the bit for the input from "AND", you will notice the output of the multiplexor, "Result", flips accordingly. Verify also the correctness for the control signal values of 01 and 10.
- Save the multiplexor as "3-input-multiplexor.circ" for future uses.

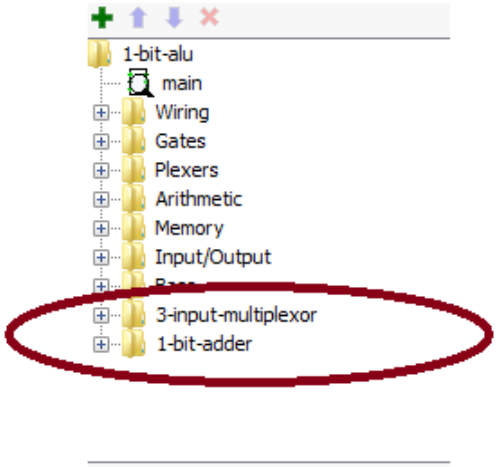
Building the multiplexor for the ALU 5/5

- The following is a sample circuit for the multiplexor, it is for your reference.



Putting the pieces together: the 1-bit ALU 1/3

- Now we have all the required pieces and we can put them together to make the 1-bit ALU.
- Load the files "1-bit-ALU.circ" and "3-input-multiplexor.circ" as Logisim library ("Project", "Load Library", "Logisim Library…").
- After that, you should be able to see the entries for the two sub-circuits "1-bit-adder" and "3-input-multiplexor" in the "Explorer pane":



- Drag them to the canvas and use them.

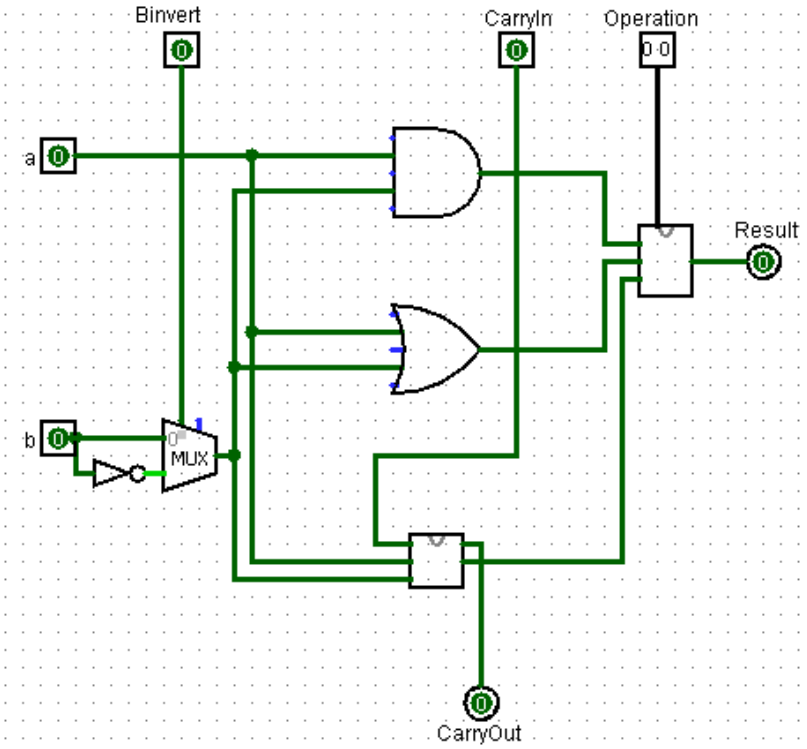
Putting the pieces together: the 1-bit ALU 2/3

- We still need another (2-to-1) multiplexor for the Adder for inverting the input bit to make 2's complement.
- You can use the Logisim built-in multiplexor for that.
- Just drag a multiplexer from the "Plexers" folder of the "Explorer pane", and edit the attributes in the "Attributes table" accordingly.
- It is a good chance for you to get familiar with the documents of Logisim. Make sure you read the multiplexor description before using it:

  http://www.cburch.com/logisim/docs/2.7/en/html/libs/plexers/mux.html

Putting the pieces together: the 1-bit ALU 3/3

- Implement the 1-bit ALU as shown on slide 3. Poke the inputs to verify the correctness of the ALU.
- Save it as "1-bit-alu.circ" for future uses.
- Here is an example implementation



Conclusions

- Today we have learnt:
  - building an 1-bit adder,
  - building a 3-to-1 multiplexor,
  - building an 1-bit ALU.