
# North South University

Department of Electrical & Computer Engineering

## LAB REPORT

## Computer Organization and Architecture Lab

Experiment Number: **Lab - #06**

Experiment Name: **Design of an ALU**

Experiment Date: 01-12-2021

Report Submission Date: 07-12-2021

Section: **02**

| Student Name: **Asfaria Islam Chowdhury** | Score |
|---|---|
| Student ID: **1931741642** | |
| Remarks: | |

## Objectives

The objective is to build a 16 bit ALU using sixteen 1-bit ALUs. Equality and overflow are detected. Each 1-bit ALU can AND, OR, add, and subtract.

## List of Equipment

Logisim

## Theory

An ALU performs arithmetic and logical functions. The inputs of the ALU are received from the register file's multiplexer outputs called Read Registers. The values of the Read Registers can be added, subtracted, AND and OR in this experiment. The output is then connected to all the registers, and when load = 1 for a specific register, the value stored in the register is rewritten (without writing this new value in other registers).

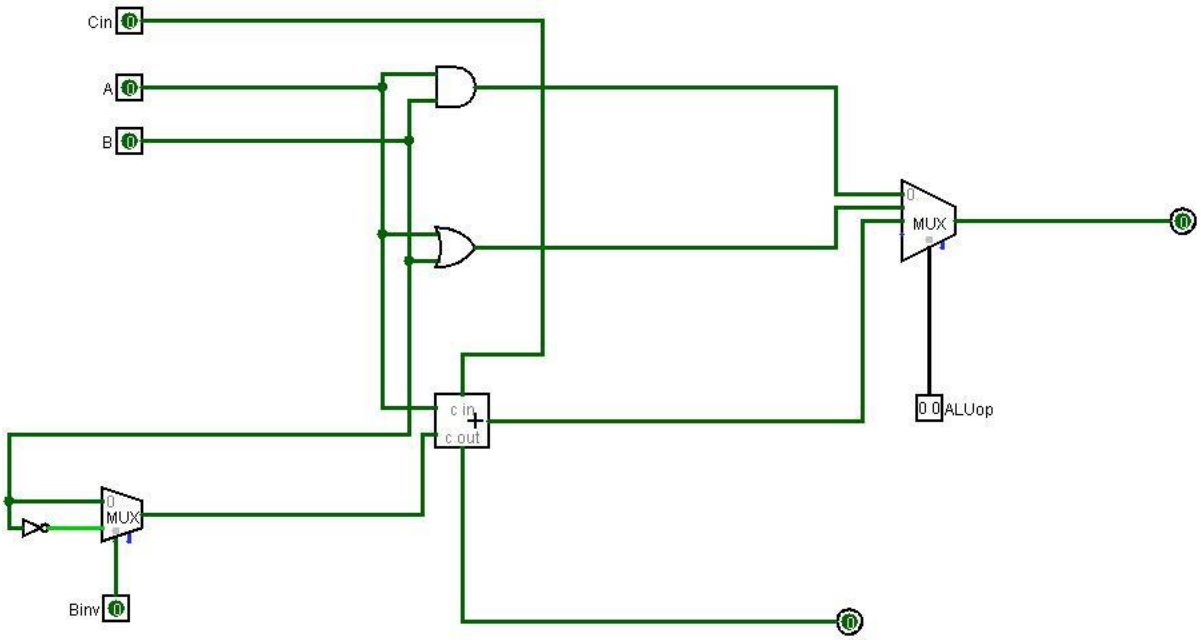
*Relationship between the Register File and ALU*

In the register file (connected to 16-bit ALU), there is a common clock for all the 16 D flip flops because this is a synchronous circuit.

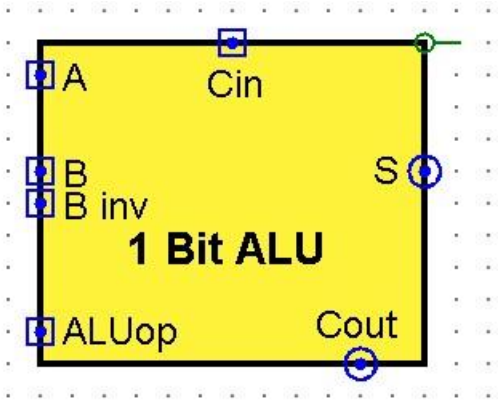
A common 16 bit write data input button is connected to all registers (made of D FFs). This write data input is either user-provided, or comes directly from the ALU output. The enable key of register takes connection from a (4X16 decoder AND a write-enable button), the result of which if 1 allows data to be written to register (if both the input bits of decoder produces a particular minterm output 1 connected to a corresponding register AND if write-enable is 1 – however, for other registers the minterm output of decoder is 0, so 0 AND 1 = 0, so data is only written to one register at a time). The input bits on the decoder are also called write register, and the enable key on the register is called load of the register, where it is observed that only load = 1 allows data to be written for only a single register while other registers have load = 0. (The write-enable is also common for all registers.)

All the outputs of the registers are connected to two 16X1 multiplexers. Each multiplexer has 4 bit selection bits, which are also known as Read Registers. The outputs of the multiplexers show the 16-bit values read from two different specific registers – the selection bits determine which registers to read from. These two values of Read Registers are connected as inputs to the 16-bit ALU.

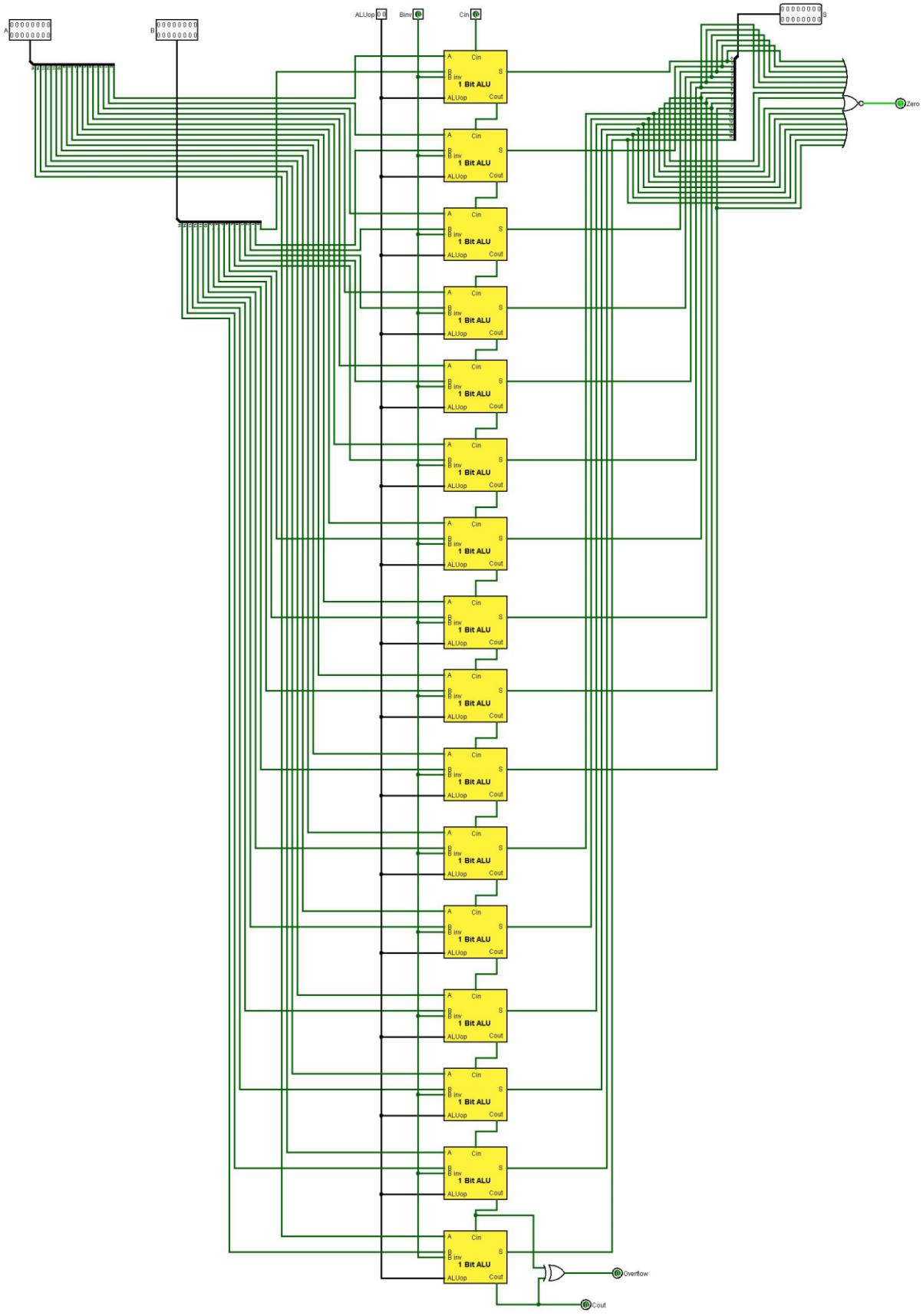
**Logisim Diagram of 1-bit ALU**



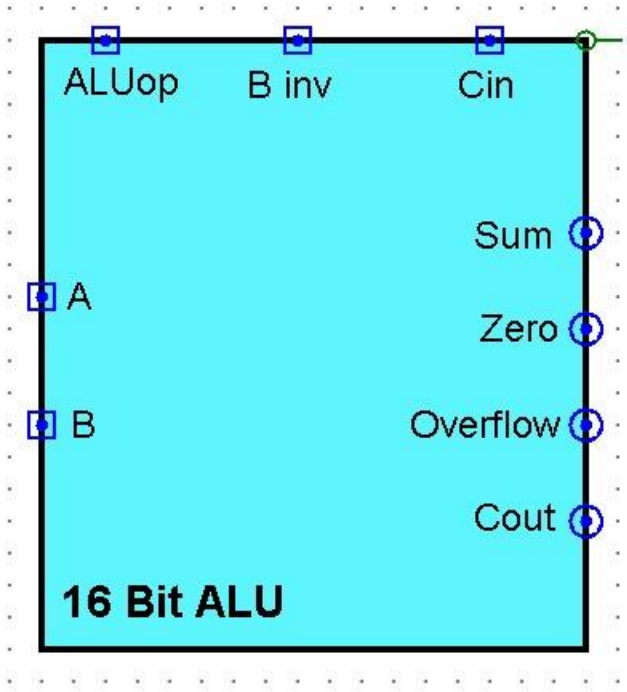

**Subcircuit of 1-bit ALU**



## Logisim Diagram of 16-bit ALU



**Subcircuit of 16-bit ALU**




**Control Selection Table**

| Operations | ALUop | $B_{inv}$ | $C_{in}$ |
|---|---|---|---|
| AND | 00 | X | X |
| OR | 01 | X | X |
| ADD | 10 | 0 | 0 |
| SUB | 10 | 1 | 1 |

**Discussion**

First, a 1-bit ALU is built. A and B are taken from the Read Registers of the register file. A 2:1 MUX is used to invert B when the selection key = 1 during subtraction (1's complement). $C_{in}$ = 1 (2's complement = 1's complement + 1). A 1-bit full adder takes A, the output of smaller MUX, and $C_{in}$ as inputs, and the sum of A+B and carry out as outputs. This $C_{out}$ is connected to $C_{in}$ of next 1-bit ALU. A AND B, A OR B, and the summation/subtraction are the three inputs connected to 4:1 MUX (the fourth input is kept empty in Logisim – there is no operation in the table for ALUop = 11). The selection keys of this multiplexer are known as the ALUop, and the control selection table is followed to get the suitable output. We place don't care as values of $B_{inv}$ and $C_{in}$ when we AND and OR – the value of $B_{inv}$ and $C_{in}$ do not affect these two operations.

Then, a 16-bit ALU is built. The values of A and B are taken from read registers of the register file, which are 16-bit values. A bit splitter is used to split the bit, 1 bit per wire. These bits pass into 16 subcircuits of 1-bit ALU, i.e. $A_0$ and $B_0$ are connected to the first 1-bit ALU … $A_{15}$ and $B_{15}$ are connected to the sixteenth 1-bit ALU. ALUop is still 2-bit, since individual subcircuits require 2 bits of ALUop and that all subcircuits must be connected to same ALUop. The same is true for the 1-bit $B_{inv}$. The $C_{in}$ is a one-bit number, and is only connected to the first subcircuit. The rest of the sub-circuits have the previous subcircuit's $C_{out}$ as their own $C_{in}$, until the very last subcircuit produces a final $C_{out}$. All the sum/result bits are connected to a bit splitter, 1 bit per wire, to a single 16-bit output of S.

All the result bits are connected to a 16-bit NOR gate, the output of which if 1 shows equality, and if 0 shows inequality.

| X | Y | X OR Y | X NOR Y |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |

Here, two 1-bit numbers are used to explain equality. It is only when all the bits are 0 when the result of NOR is 1, so the equality output is 1. In all other cases, having at least a single bit which is 1 among 16 bits causes OR output to be 1, and inverts that to be 0 in equality output.

The final carry out and the 2nd last 1-bit ALU's carry out are both connected to an XOR to check overflow. Overflow is a situation when the result is too large or too small to represent within our 16-bit output. If two operands with different signs are added, overflow does not occur. This is because any carry out that takes place can easily be represented with a final carry out output. If two positive numbers are added to produce a negative number (last bit is 1 in signed representation), the number is too large to show the output in S (adding two positives produces a much larger summation than adding a positive and negative). If two negative numbers are added to produce a positive number (last bit is 0 in signed representation), the number is too small (less than 16 bits) to show the output in S (adding two negatives produces a much smaller summation than adding a positive and negative).