Lab 5 – RISC ALU

Objectives

In this lab, we will develop a RISC arithmetic and logic unit and verify its functionality using simulation.

Section	
a) Introduction A brief overview of this lab.	**02**
b) Implementation This section is divided into two lab tasks. First, you will implement and test a 1-bit RISC ALU. Second, you will connect 6 instances of 1-bit ALUs to construct a 6-bit ALU followed by its verification using test bench.	**40 + 60**
Exercise You will develop a behavioral model of a 64-bit ALU.	**45**

a. Introduction

The arithmetic logic unit (ALU) is the brawn of the computer, the device that performs the arithmetic operations like addition and subtraction or logical operations like AND and OR. This lab constructs an ALU from four hardware building blocks (AND and OR gates, inverters, and multiplexors) and illustrates how combinational logic works.

Because the RISC-V registers are 64 bits wide, we need a 64-bit-wide ALU. Let's assume that we will connect 64 1-bit ALUs to create the desired ALU. We'll therefore start by constructing a 1-bit ALU, shown below.

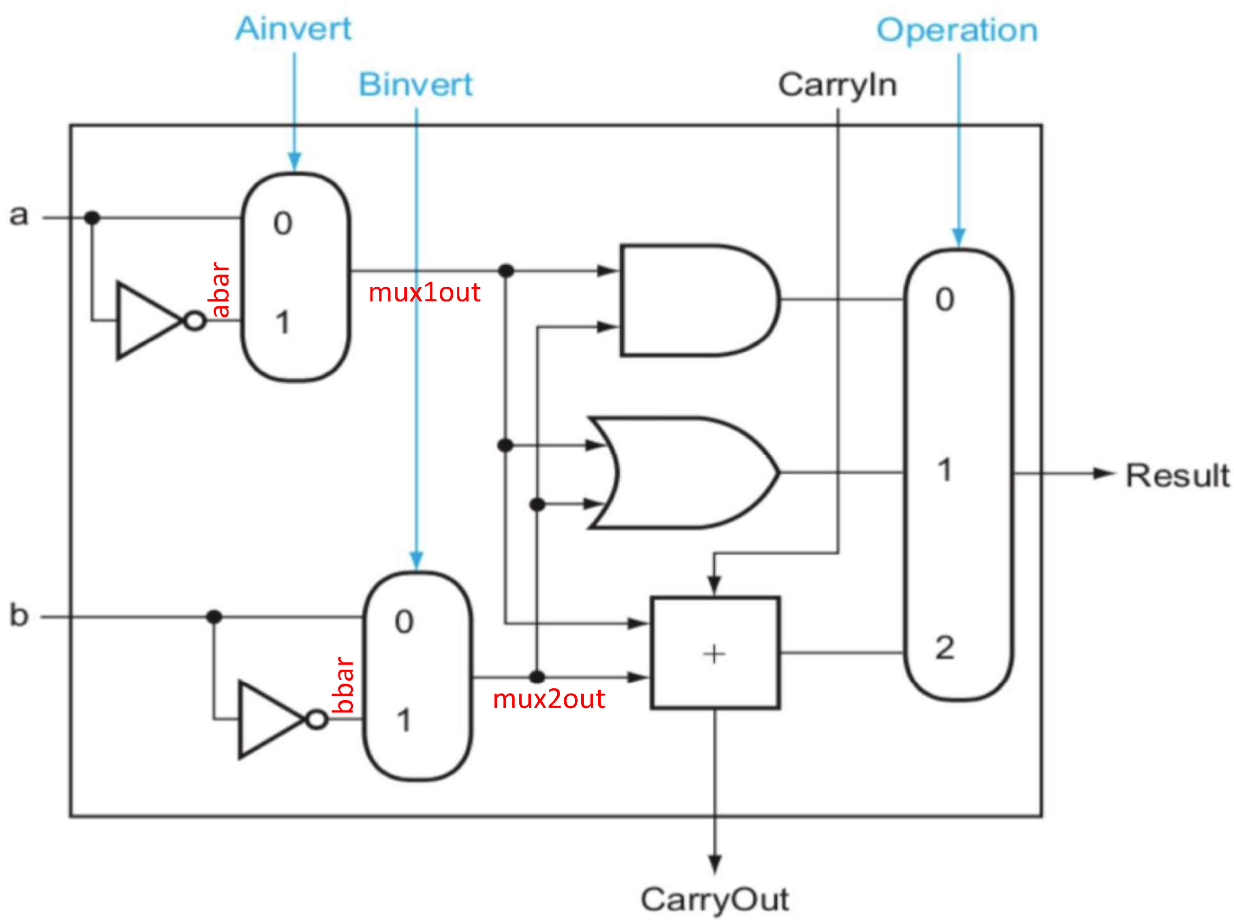

Fig. 5.1. 1-bit ALU with four control signals – Ainvert, Binvert, [1:0] Operation.

b. Implementation

In this lab, we will design an ALU that will be able to perform a subset of the ALU operations of a full RISC-V ALU. In this exercise, we will develop an ALU that will take two 1-bit inputs, `a` and `b`, and will be able to execute the following five instructions:

add, sub, and, or, nor

The 1-bit ALU will generate a 1-bit output, named “Result”, and a “CarryOut” signal. The different operations will be selected by a 4-bit control signal, named “ALUop”. The description of these 4 control signals are shown in the table below.

[3:0] ALUOp				Function
Ainvert	Binvert	Operation [1]	Operation[0]	
0	0	0	0	AND
0	0	0	1	OR
0	0	1	0	Add
0	1	1	0	Subtract
1	1	0	0	NOR

i. *Lab Task 01*

Design a module named “`ALU_1_bit`” to incorporate the functionality shown in Fig. 5.1. This module should have 1-bit inputs `a`, `b`, `CarryIn`; 4-bit input `ALUOp`; and two 1-bit outputs `Result` and `CarryOut`.

The text shown in red in Fig. 5.1. indicate the name of corresponding wires. For example, the wire at the output of upper inverter (having input `a`) is given a name `abar`.

As discussed in lecture, the output `CarryOut` should be implemented according to the following equation:

```
CarryOut = Input1.CarryIn + Input2.CarryIn + Input1.Input2
```

where,
`Input1` and `input2` are the inputs to the adder shown in Fig. 5.1.

The inputs to the adder shown in Fig. 5.1. are not the input signals `a` and `b`. Choose the appropriate signals carefully.

Write a testbench and verify the functionality of all five operations.

ii. *Lab Task 02*

Now you are supposed to use this 1-bit ALU to implement 64-bit ALU by instantiating the previously developed module, `ALU_1_bit`, 64 times. Since it will become too difficult to debug these many modules, therefore, for now, extend this 1-bit ALU to 6-bit ALU by instantiating `ALU_1_bit` module 6 times in a separate top module and make the appropriate connections as shown in Figure 5.2 (only for 6-bits ALU).

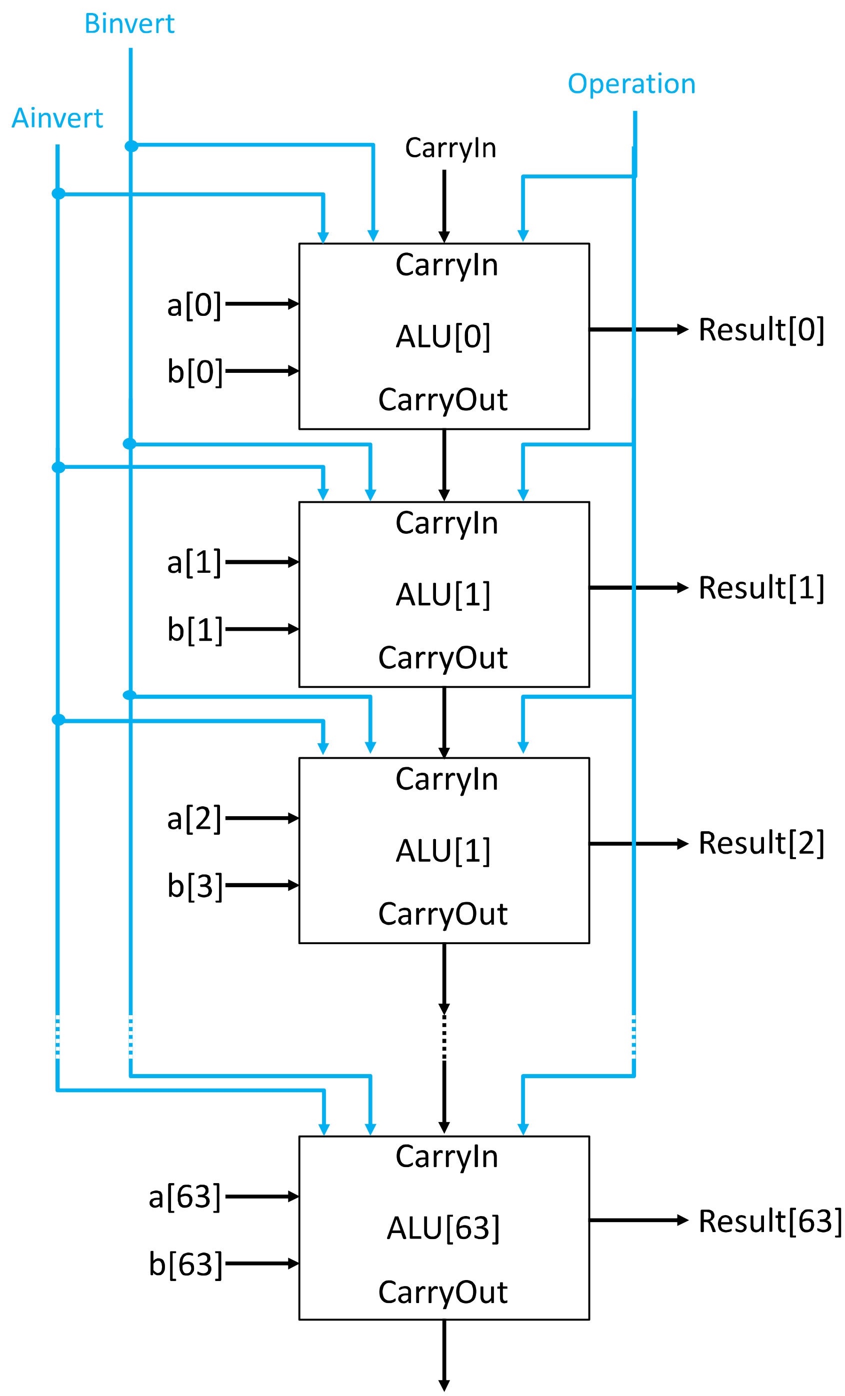

Fig. 5.2. Extension of 1-bit ALU to 64-bit ALU. In this lab, you are required to use 1-bit ALUs 6 times to construct a 6-bit ALU.

Write a testbench to test the functionality of 6-bit ALU and verify the correct functioning of all five operations.

Exercise

At this point, you have already experienced that extending `ALU_1_bit` module 64 times will result in a 64-bit ALU. However, this process would be too lengthy and cumbersome. In contrast, you are required to develop a behavioral model of 64-bit ALU just by declaring `a` and `b` 64-bits wide and declare the corresponding operations using a single multiplexer. You also need to add an additional output named ZERO in your 64-bit ALU, as shown in Fig. 5.3 below.

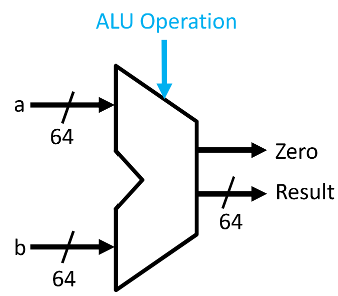

Fig. 5.3. 64-bit ALU with ZERO output.

This ZERO output should be set to 1 if the `Result` is 0, else set it to 0. The circuitry for triggering ZERO is shown in the figure below.