Computer Architecture Lab 05

Arithmetic Logic Unit

Introduction

This lab focuses on designing and understanding the Arithmetic Logic Unit (ALU) — the core component responsible for performing computations inside a processor. Every instruction that involves arithmetic (like addition or subtraction) or logical operations (like AND, OR, or XOR) ultimately passes through the ALU.

By implementing an ALU in Verilog, you will gain insight into how a processor executes operations at the hardware level, how control signals determine the operation performed, and how status flags (like zero, carry, or overflow) are generated as a result.

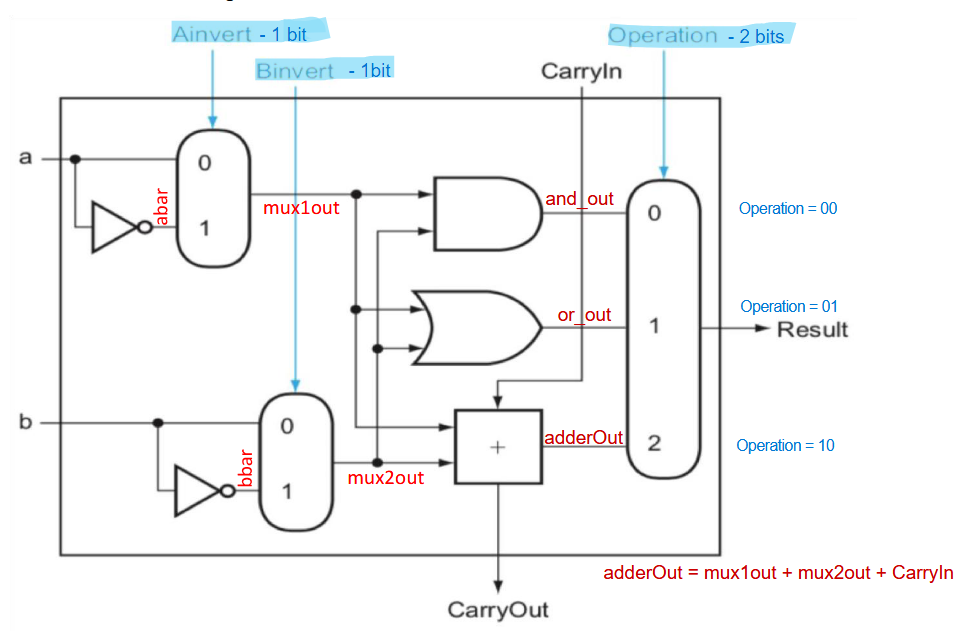
This lab also serves as a foundation for later topics such as instruction execution and datapath design. You’ll use combinational logic to perform different operations based on an operation select input, and later integrate your ALU with other components like registers and control units.

Objectives

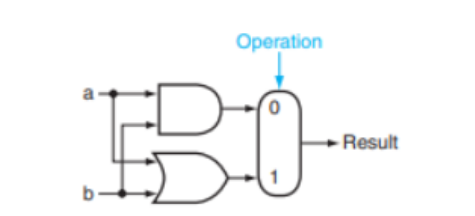
* Understand the role and structure of an ALU in a processor datapath.
* Implement arithmetic and logical operations using Verilog.
* Learn to design hardware modules that use control signals to select between multiple operations.
* Observe how status flags are generated and interpreted.

Task 1: Single Bit ALU

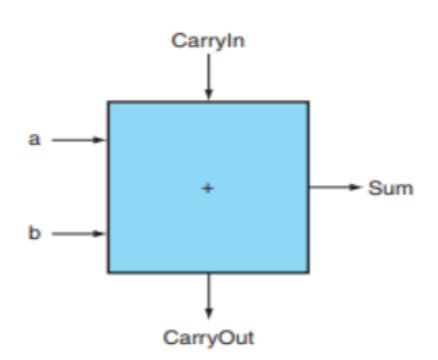
We start by constructing a 1-bit ALU



The 1-bit logical unit for AND and OR looks like this. The multiplexor on the right then selects a AND b or a OR b, depending on whether the value of Operation is 0 or 1. The line that controls the multiplexor is shown in colour to distinguish it from the lines containing data. Notice that we have renamed the control and output lines of the multiplexor to give them names that reflect the function of the ALU.



The next function to include is addition. An adder must have two inputs for the operands and a single-bit output for the sum. There must be a second output to pass on the carry, called CarryOut. Since the CarryOut from the neighbor adder must be included as an input, we need a third input. This input is called CarryIn. The inputs and the outputs of a 1-bit adder look like this:



Provide your module here:

|  |
| --- |
|  |