



The Islamic University of Gaza
Faculty of Engineering
Department of Computer Engineering



LCOM 3010 - Computer Architecture

Lab 2

ALU

Prof. Wesam M. Ashour

Eng. Hashem M. Hejazy

Summer 2022/2023

Introduction

What is ALU?

The Arithmetic Logic Unit (ALU) is a fundamental component of a CPU (Central Processing Unit) that performs arithmetic and logical operations on binary data. It is responsible for carrying out the majority of the computations that occur within a CPU, including addition, subtraction, multiplication, division, and logical operations such as AND, OR, and NOT.

The ALU is typically designed to operate on fixed-length binary data, which is represented as a series of bits. The number of bits that the ALU can process at any given time is known as its word size. Common word sizes include 8 bits, 16 bits, 32 bits, and 64 bits.

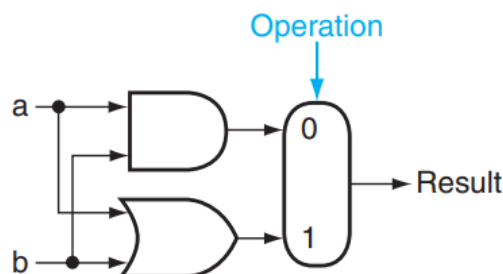
The ALU is typically designed to operate on fixed-length binary data, which is represented as a series of bits. The number of bits that the ALU can process at any given time is known as its word size. Common word sizes include 8 bits, 16 bits, 32 bits, and 64 bits.

The ALU typically consists of several subunits, including:

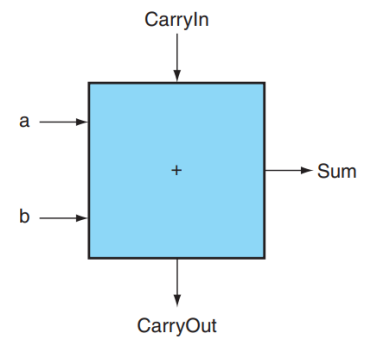
1. **Arithmetic unit:** This unit performs arithmetic operations such as addition, subtraction, multiplication, and division. The arithmetic unit usually includes a set of registers that hold the operands and the result of the computation.
2. **Logical unit:** This unit performs logical operations such as AND, OR, and NOT. The logical unit is typically implemented using a set of logic gates.
3. **Control unit:** This unit is responsible for directing the operation of the ALU. It determines which operation to perform based on the instruction provided by the CPU.
4. **Shifter unit:** This unit is responsible for shifting the bits of a binary number left or right. The shifter unit is typically used in conjunction with the arithmetic unit to perform multiplication and division operations.

A 1-Bit ALU

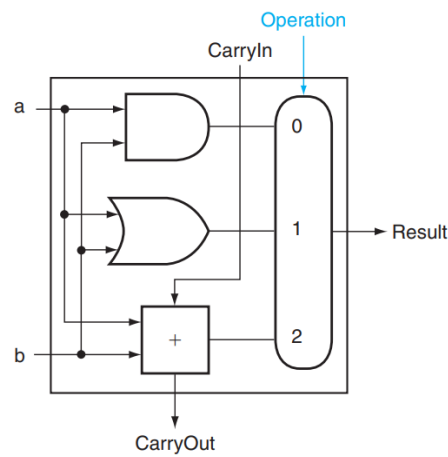
The 1-bit logical unit for AND and OR looks like Figure. The multiplexor on the right then selects a AND b or a OR b, depending on whether the value of Operation is 0 or 1.



The 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.



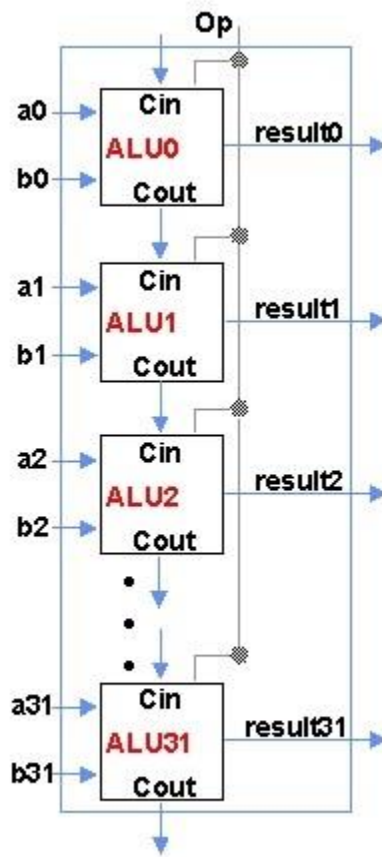
Now, to design A 1-bit ALU collect the above components in one circuit as in Figure.



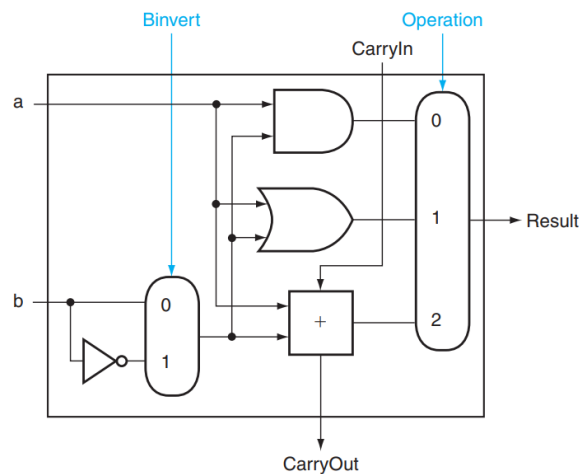
Sometimes designers also want the ALU to perform a few more simple operations, such as generating 0. The easiest way to add an operation is to expand the multiplexor controlled by the Operation line.

A 32-Bit ALU

Now that we have completed the 1-bit ALU, the full 32-bit ALU is created by connecting adjacent “black boxes.”

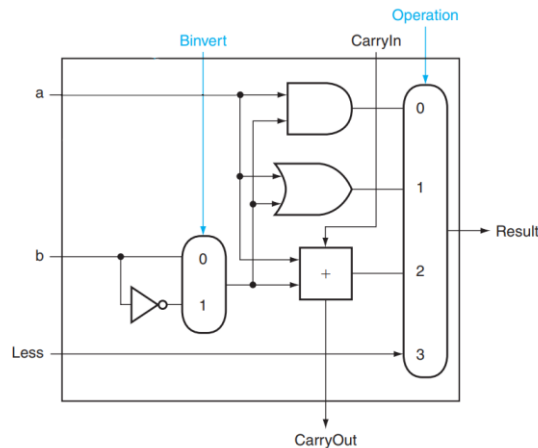


Subtraction is the same as adding the negative version of an operand, and this is how adders perform subtraction. Recall that the shortcut for negating a two's complement number is to invert each bit (sometimes called the one's complement) and then add 1. To invert each bit, we simply add a 2:1 multiplexor that chooses between b and $\bar{b}$, as Figure.



Tailoring the 32-Bit ALU to MIPS

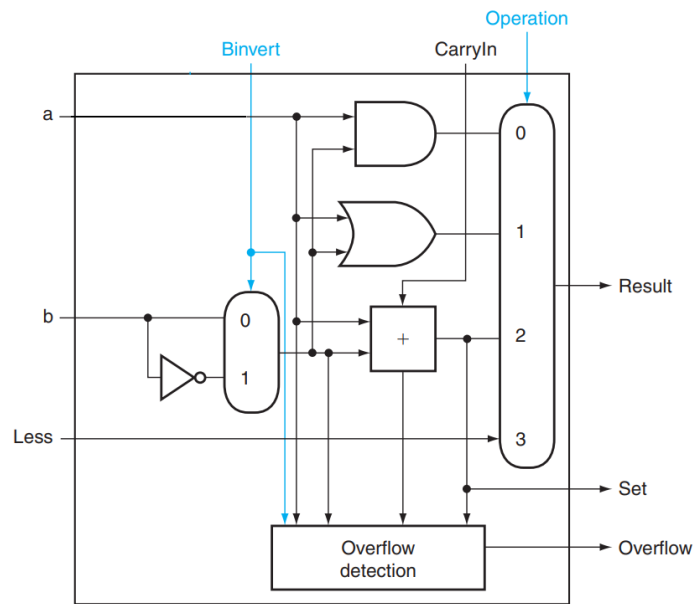
These four operations—add, subtract, AND, OR—are found in the ALU of almost every computer, and the operations of most MIPS instructions can be performed by this ALU. But the design of the ALU is incomplete. One instruction that still needs support is the set on less than instruction (slt). Recall that the operation produces 1 if $rs < rt$, and 0 otherwise. For the ALU to perform slt, we first need to expand the three-input multiplexor in Last Figure to add an input for the slt result. We call that new input Less and use it only for slt.



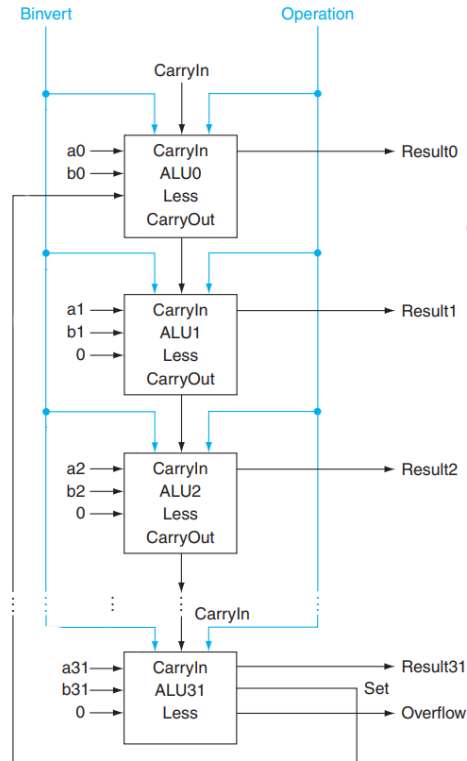
From the description of `slt` above, we must connect 0 to the Less input for the upper 31 bits of the ALU, since those bits are always set to 0. What remains to consider is how to compare and set the least significant bit for set on less than instructions.

We want the least significant bit of a set on less than operation to be a 1 if $a < b$; that is, a 1 if $a - b$ is negative and a 0 if it's positive. This desired result corresponds exactly to the sign bit values: 1 means negative and 0 means positive. Following this line of argument, we need only connect the sign bit from the adder output to the least significant bit to get set on less than.

Thus, we need a new 1-bit ALU for the most significant bit that has an extra output bit: the adder output. The Following Figure shows the design, with this new adder output line called Set, and used only for slt. As long as we need a special ALU for the most significant bit, we added the overflow detection logic since it is also associated with that bit.



A 32-bit ALU constructed from the 31 copies of the 1-bit ALU. The *Less* inputs are connected to 0 except for the least significant bit, which is connected to the *Set* output of the most significant bit. If the ALU performs $a - b$ and we select the input 3 in the multiplexor in Figure then $\text{Result} = 0 \dots 001$ if $a < b$, and $\text{Result} = 0 \dots 000$ otherwise.



LAB TASK

Use Logisim Program to implement and simulate the an 1-bit ALU then Use this block to design an 4-bit ALU.

