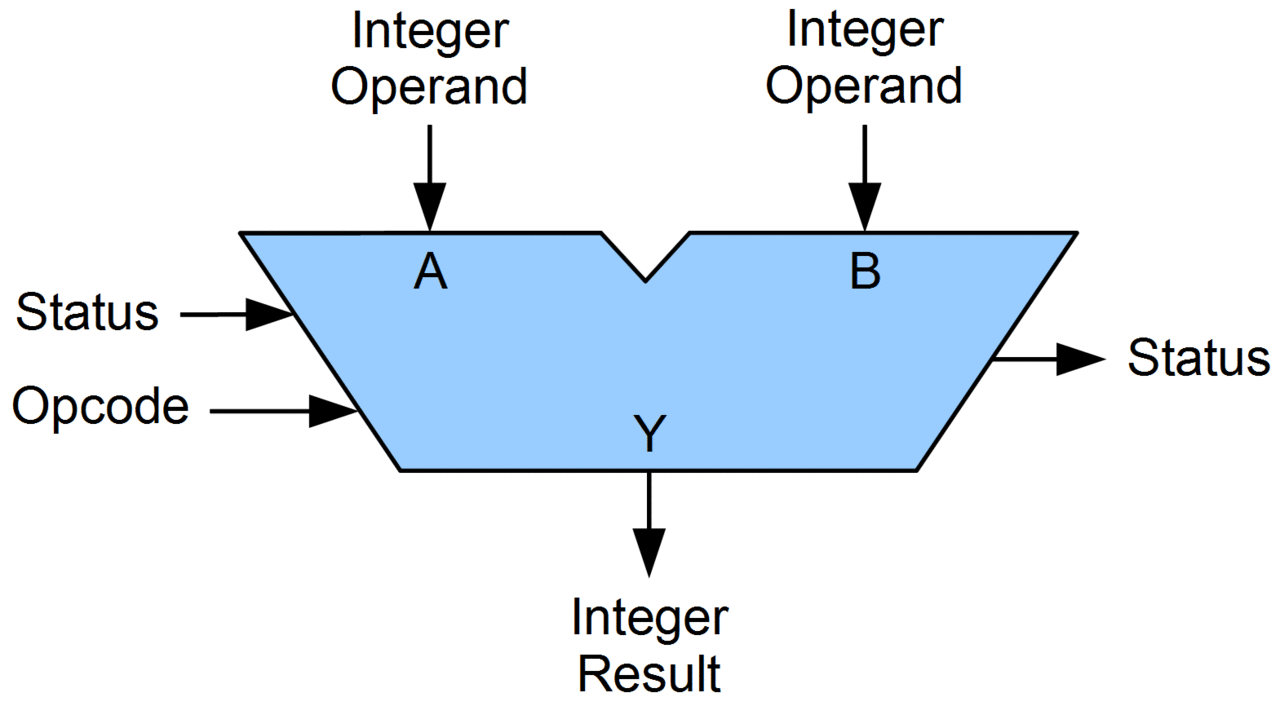
# Week 4 Lab A: Arithmetic Logic Unit (ALU)

## Objectives

Develop understanding and experience of:

1. What is meant by an ALU and its opcode
2. Building a simplified ALU

## ALU



1. General information about ALUs.
2. How many operands does an ALU have and what are they for?

Operand is the input for the ALU and there are 2 operands in an ALU.

1. What is the opcode for?

Controls which operation we want to see the result of.

1. We will use a 2-bit opcode. How many different operations can we include? 4

An ALU usually does all of its operations in parallel and the opcode is used to select which to pass to the output result.

1. What component can we use to select one from a set of signals? Multiplexor
2. Building an ALU

You will create a simple ALU in Logisim Evolution which is just an example of what an ALU can do, each architecture has its own set of instructions it can process. Although we will use built-in components from Logisim Evolution, if you have been working through including extensions, you have seen how we can build all of them from basic logic gates. Although the ALU in the coursework is different from the one used here, the principles are the same, so it is important that you understand the ideas here.

|  |  |
| --- | --- |
| ALU OP value | Output |
| 0 | The result of adding A and B together |
| 1 | The result of a bitwise A AND B |
| 2 | The result of subtracting 1 from **A** |

Create the ALU in Logisim Evolution as follows:

* 1. Have two 8-bit input pins (labelled A and B) for the data to operate on and a 2-bit input (labelled ALU\_Opcode) for the operation to carry out.
  2. Have one 8-bit output pin (labelled ALU\_OUT) for the result.
  3. The image below has both A and B going into both addition (operation 0) and bitwise AND (operation 1). Operation 2 should subtract 1 from the input value from A. Note that for you will need to use a constant for the value 1 (from the Logisim Evolution wiring tools) and set the Data Bits and value appropriately.

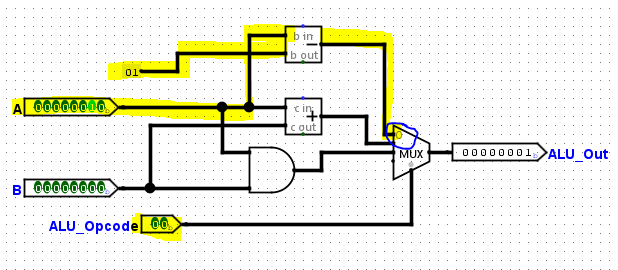
A picture containing text, clock, screenshot

Description automatically generated

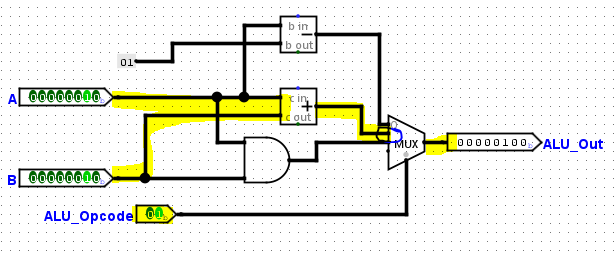
* 1. Use a suitable component to select the final output according to the value of the 2-bit ALU OP input according to the table above.
  2. Choose some values for A and B and work out what you expect the result to be after each of the three operations. Write down your values and expected results.

Test your circuit with each ALU\_Opcode and compare to your expected results. Add images to this worksheet.

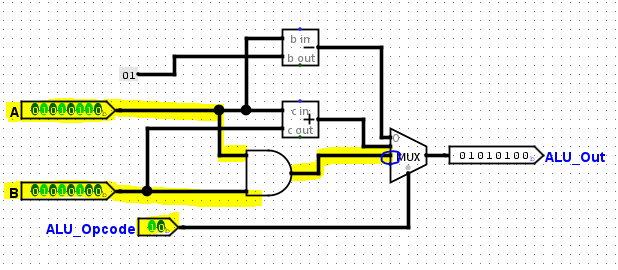
2 – 1 = 1



When a is 2 and b is 2 the output of a add b is 4



Value of a and b



## Extension exercises

### Manipulating binary numbers with two’s complement representation

Note that these tasks include an exercise using Logisim evolution and one on Java programming.

1. Suppose you have the following binary value stored in 8-bits but you don’t know if it is a signed or unsigned integer, what are the two possible decimal values it could have?

10011001 == 153

1. What is the decimal equivalent of the following 8-bit binary number using 2’s complement representation?

01111111

What do we get if we add 1 to that binary number using normal binary arithmetic?

What is the decimal equivalent of the resulting binary number? Remember that we are using two’s complement.

1. Using 8-bit two’s complement, use binary addition to do the equivalent calculation of 12 – 5, that is do 12 + (-5). Make sure that you check your answer.
2. What would the representation be if we store -5 using 16-bit two’s complement? What does that imply if we need to move a number (positive or negative) from 8-bit storage to 16-bit storage?

How would you use wiring and basic logic gates to cater for that issue? If you look in the Wiring tools in Logisim Evolution, you will see a built-in component that can be used to cater for this problem.

1. Problems with overflow. Using 8-bit two’s complement binary addition, find the sum of decimal 111 and 52. You may use a website to find the binary representation of those numbers. What problems do you find?

The issue you find with two’s complement arithmetic is known as overflow. In last week’s lecture we saw that some ALUs have additional output flags, one of those might be set to 1 when an overflow has occurred.

When working with unsigned integers and the result of addition is too big for the output register, that is called carry. Many processors have to deal with carry so that they can do calculations with numbers larger than can be stored in a register. For example, 8-bit computers are not limited to calculations on numbers up to 255 only. You could research how a carry bit output from an ALU could be used along with taking one byte of a larger number at a time. This investigation might give some explanation of why little-endian order of bytes is so common. We saw little-endian when the RGB values were in reverse order in the bmp image format.

1. See two’s complement in action in Java. Create a Java program (in Processing) to do the following:
2. Create an integer and assign it a value of 2147483640
3. Create a loop that iterates 20 times. Do **not** use the integer created in step 1 to control the iteration.
4. Inside the for loop
   1. Print the value of the integer created in step 1 to the console. You can use println()
   2. Add 1 to the integer created in step 1.

Add an image of your console and describe what happened. Given that an int in Java uses 32 bits, you should be able to relate the result to two’s complement representation.