
# Computer Organization/ Architecture (COMP2825)

Instructor: Maryam Tanha

Fall 2021

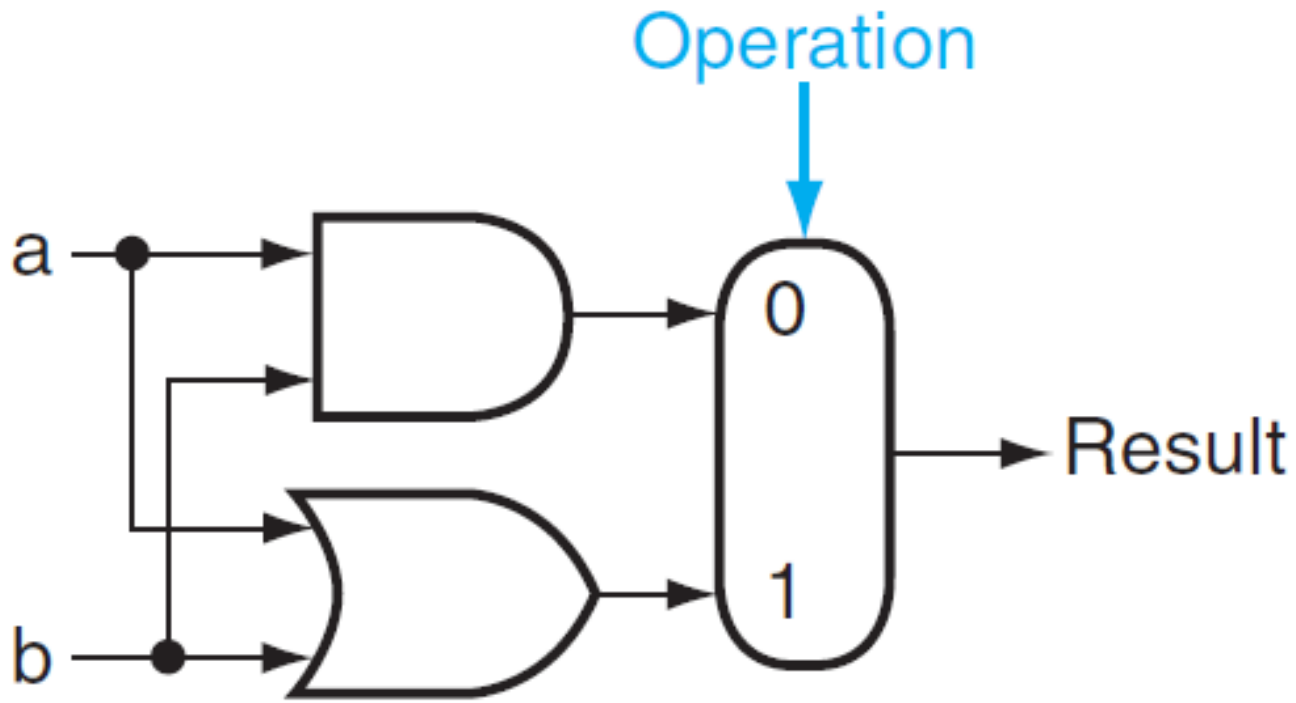
# Agenda

- **Constructing a Basic ALU**

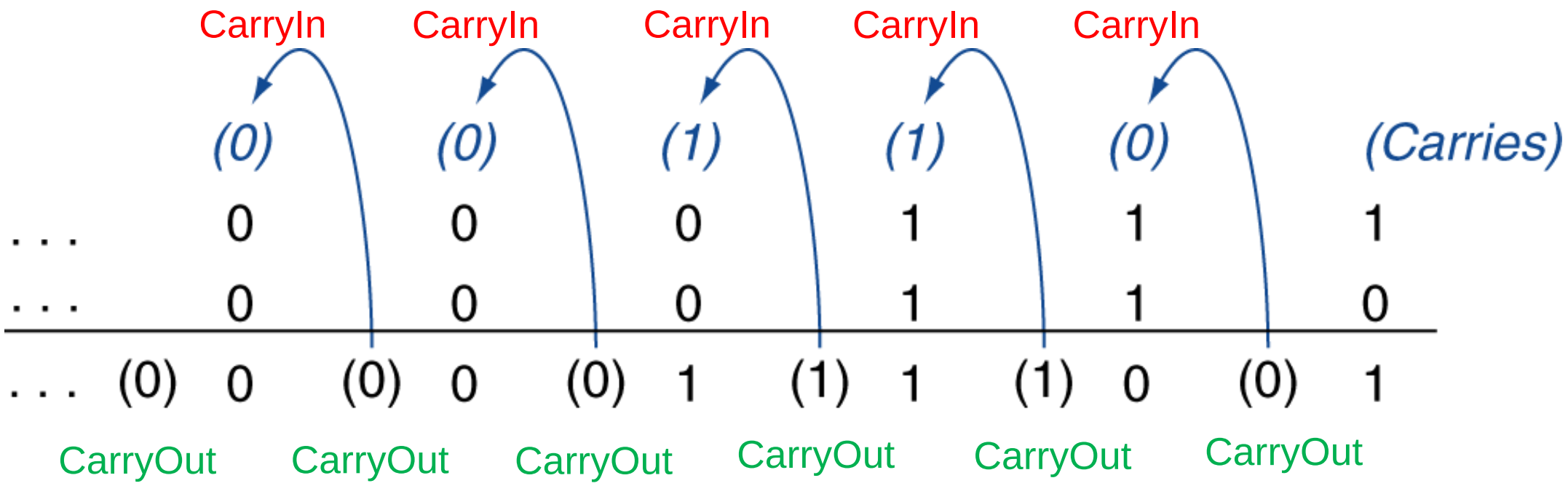
# Constructing a Basic ALU

- **ALU (Arithmetic Logic Unit)**
  - performs the arithmetic operations such as addition and subtraction or logical operations like AND and OR.
- An ALU can be built from four **hardware building blocks** (AND and OR gates, inverters, and multiplexers).

# 1-Bit Logical Unit for AND and OR

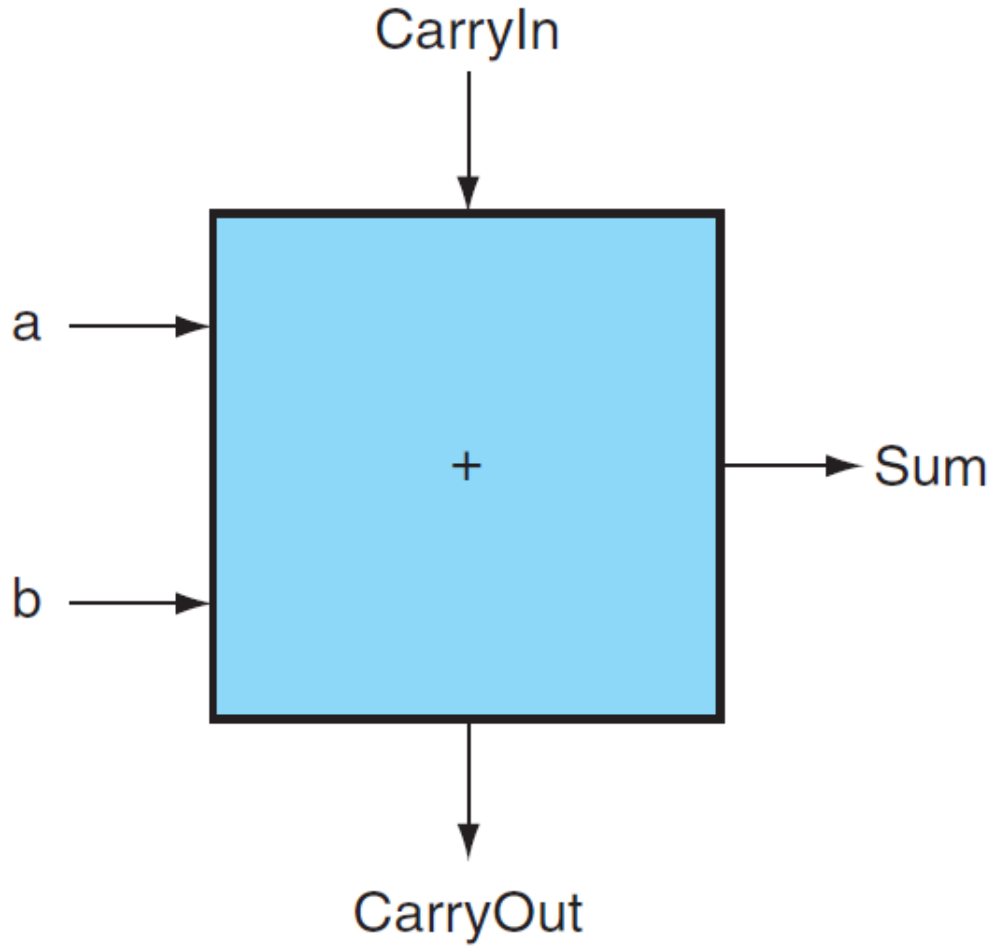


# Binary Addition – Review



# 1-Bit Adder

- A full adder (also called a (3,2) adder because it has 3 inputs and 2 outputs).
- **Two inputs** for the operands and **one input** (CarryIn) for the CarryOut from the neighbor adder.
- **Two outputs**, one for the sum and one for passing on the CarryOut.



# Input and Output Specification for a 1-bit Adder

| Inputs | | | Outputs | | Comments |
|---|---|---|---|---|---|
| a | b | CarryIn | CarryOut | Sum | |
| 0 | 0 | 0 | 0 | 0 | $0 + 0 + 0 = 00_{two}$ |
| 0 | 0 | 1 | 0 | 1 | $0 + 0 + 1 = 01_{two}$ |
| 0 | 1 | 0 | 0 | 1 | $0 + 1 + 0 = 01_{two}$ |
| 0 | 1 | 1 | 1 | 0 | $0 + 1 + 1 = 10_{two}$ |
| 1 | 0 | 0 | 0 | 1 | $1 + 0 + 0 = 01_{two}$ |
| 1 | 0 | 1 | 1 | 0 | $1 + 0 + 1 = 10_{two}$ |
| 1 | 1 | 0 | 1 | 0 | $1 + 1 + 0 = 10_{two}$ |
| 1 | 1 | 1 | 1 | 1 | $1 + 1 + 1 = 11_{two}$ |

# Input and Output Specification for a 1-bit Adder

| Inputs | | | Outputs | | Comments |
|---|---|---|---|---|---|
| a | b | CarryIn | CarryOut | Sum | |
| 0 | 0 | 0 | 0 | 0 | $0 + 0 + 0 = 00_{two}$ |
| 0 | 0 | 1 | 0 | 1 | $0 + 0 + 1 = 01_{two}$ |
| 0 | 1 | 0 | 0 | 1 | $0 + 1 + 0 = 01_{two}$ |
| 0 | 1 | 1 | 1 | 0 | $0 + 1 + 1 = 10_{two}$ |
| 1 | 0 | 0 | 0 | 1 | $1 + 0 + 0 = 01_{two}$ |
| 1 | 0 | 1 | 1 | 0 | $1 + 0 + 1 = 10_{two}$ |
| 1 | 1 | 0 | 1 | 0 | $1 + 1 + 0 = 10_{two}$ |
| 1 | 1 | 1 | 1 | 1 | $1 + 1 + 1 = 11_{two}$ |

# Expressing CarryOut as a Logical Equation

- **The values of the inputs when CarryOut is a 1**

| Inputs | | |
|---|---|---|
| **a** | **b** | **CarryIn** |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | 1 | 1 |

- **The corresponding logical equation is as follows:**

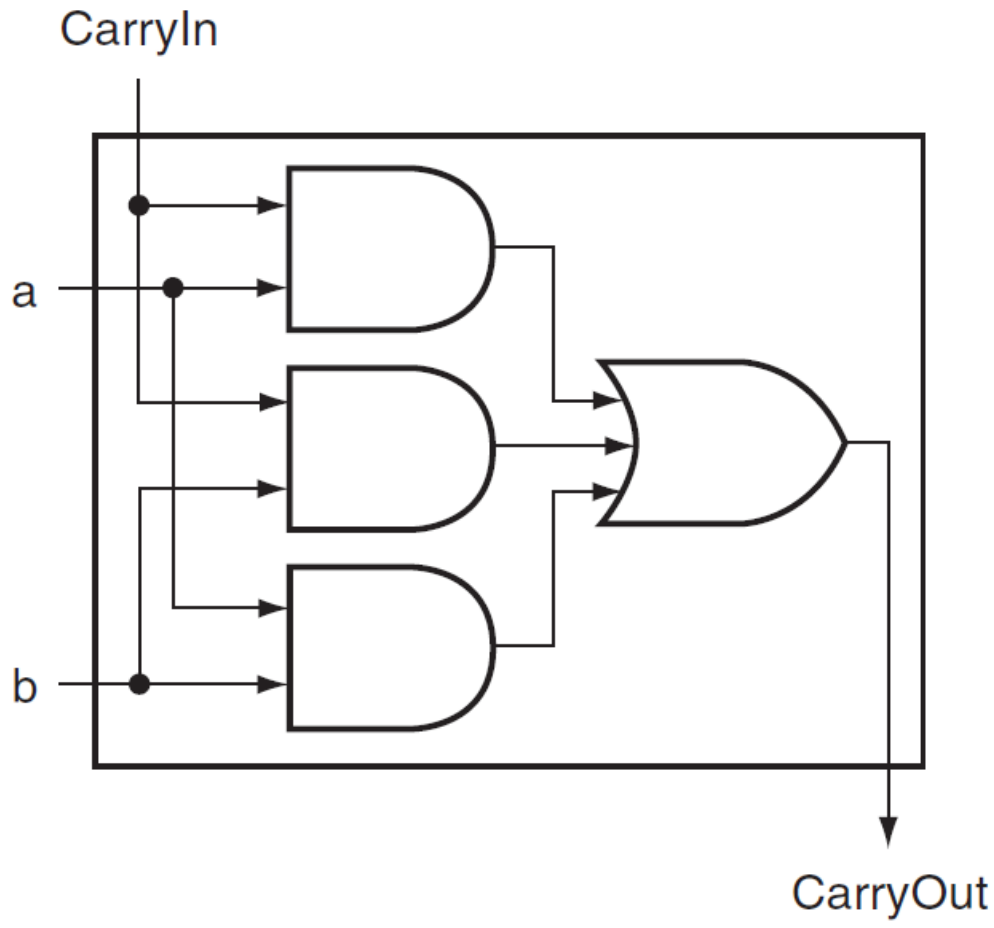
$$\text{CarryOut} = (\text{b} \cdot \text{CarryIn}) + (\text{a} \cdot \text{CarryIn}) + (\text{a} \cdot \text{b}) + (\text{a} \cdot \text{b} \cdot \text{CarryIn})$$

- **The equation after simplification (due to absorption law):**

$$\text{CarryOut} = (\text{b} \cdot \text{CarryIn}) + (\text{a} \cdot \text{CarryIn}) + (\text{a} \cdot \text{b})$$

# The Hardware within the Adder for CarryOut

$$\text{CarryOut} = (b \cdot \text{CarryIn}) + (a \cdot \text{CarryIn}) + (a \cdot b)$$

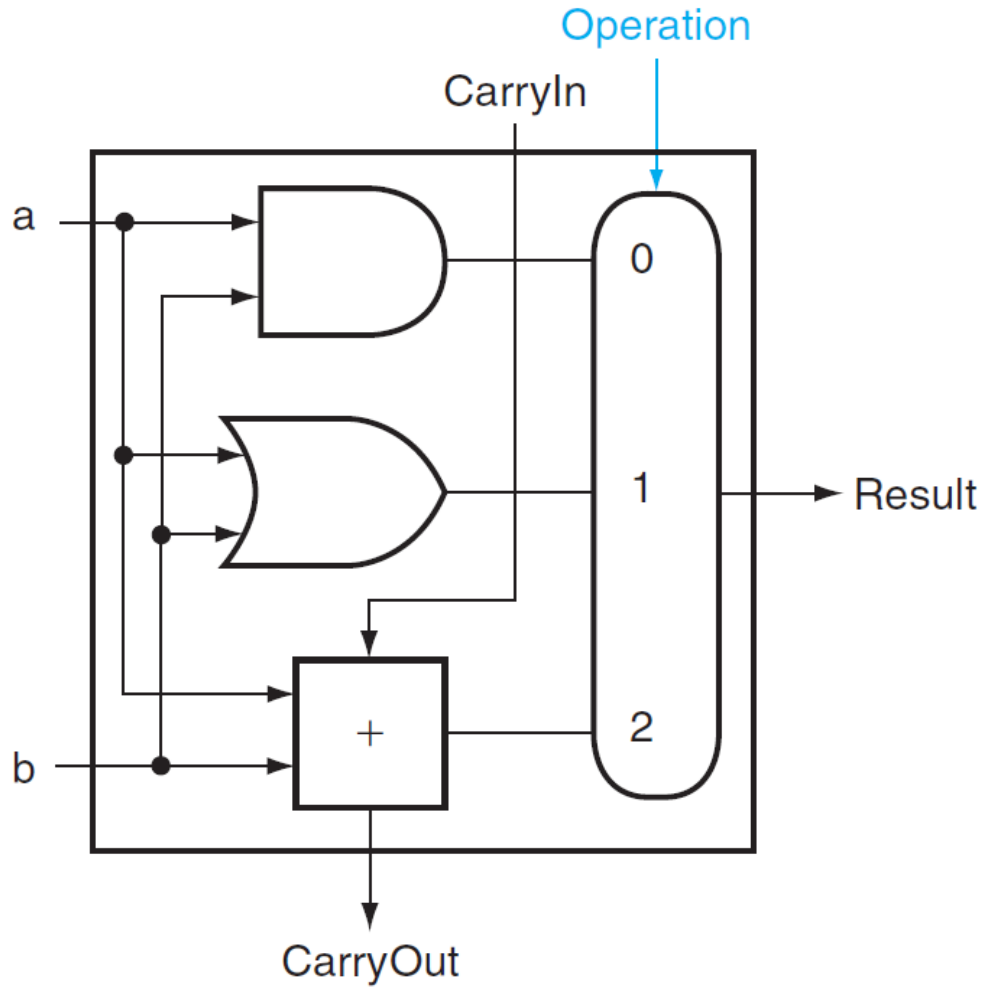


# Expressing Sum as a Logical Equation

$$\text{Sum} = (a \cdot \overline{b} \cdot \overline{\text{CarryIn}}) + (\overline{a} \cdot b \cdot \overline{\text{CarryIn}}) + (\overline{a} \cdot \overline{b} \cdot \text{CarryIn}) + (a \cdot b \cdot \text{CarryIn})$$

| Inputs | | | Outputs | | |
|---|---|---|---|---|---|
| a | b | CarryIn | CarryOut | Sum | Comments |
| 0 | 0 | 0 | 0 | 0 | $0 + 0 + 0 = 00_{two}$ |
| 0 | 0 | 1 | 0 | 1 | $0 + 0 + 1 = 01_{two}$ |
| 0 | 1 | 0 | 0 | 1 | $0 + 1 + 0 = 01_{two}$ |
| 0 | 1 | 1 | 1 | 0 | $0 + 1 + 1 = 10_{two}$ |
| 1 | 0 | 0 | 0 | 1 | $1 + 0 + 0 = 01_{two}$ |
| 1 | 0 | 1 | 1 | 0 | $1 + 0 + 1 = 10_{two}$ |
| 1 | 1 | 0 | 1 | 0 | $1 + 1 + 0 = 10_{two}$ |
| 1 | 1 | 1 | 1 | 1 | $1 + 1 + 1 = 11_{two}$ |

# 1-Bit ALU that performs AND, OR, and Addition

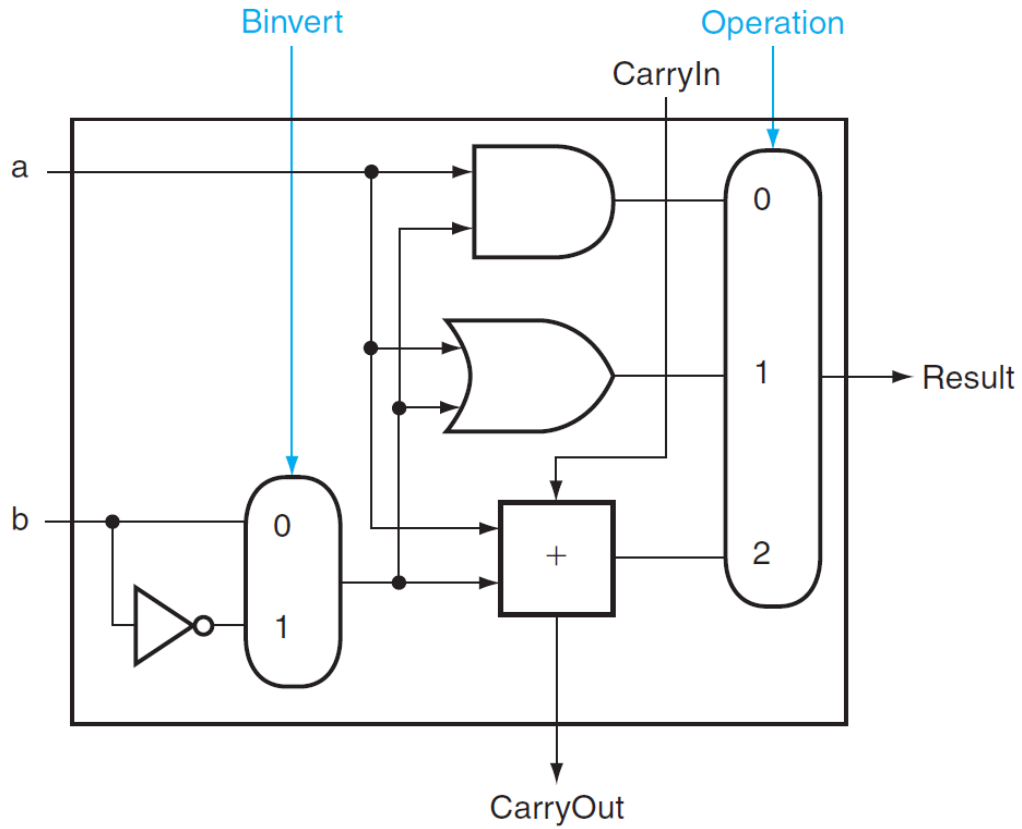


# Exercise

Design and draw a 1-Bit ALU that performs AND, OR, and addition on a and b as well as a and $\bar{b}$ .

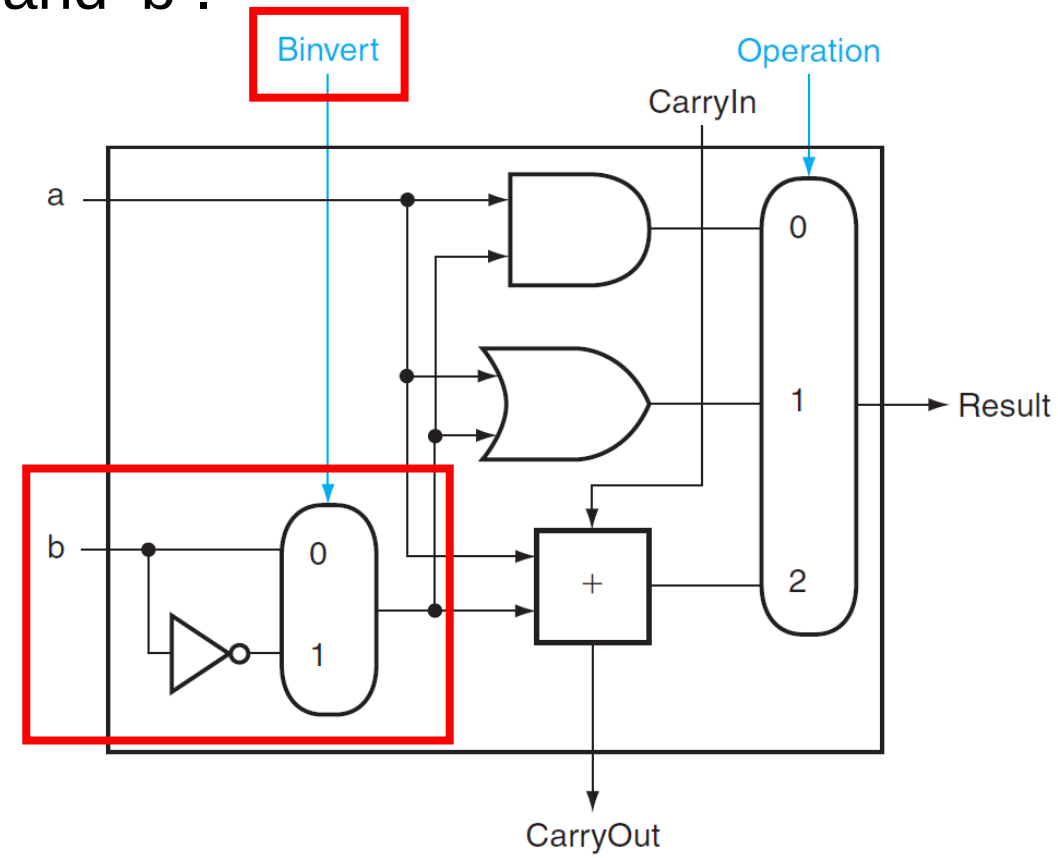
# Exercise – Answer

Design and draw a 1-Bit ALU that performs AND, OR, and addition on a and b as well as a and $\bar{b}$ .



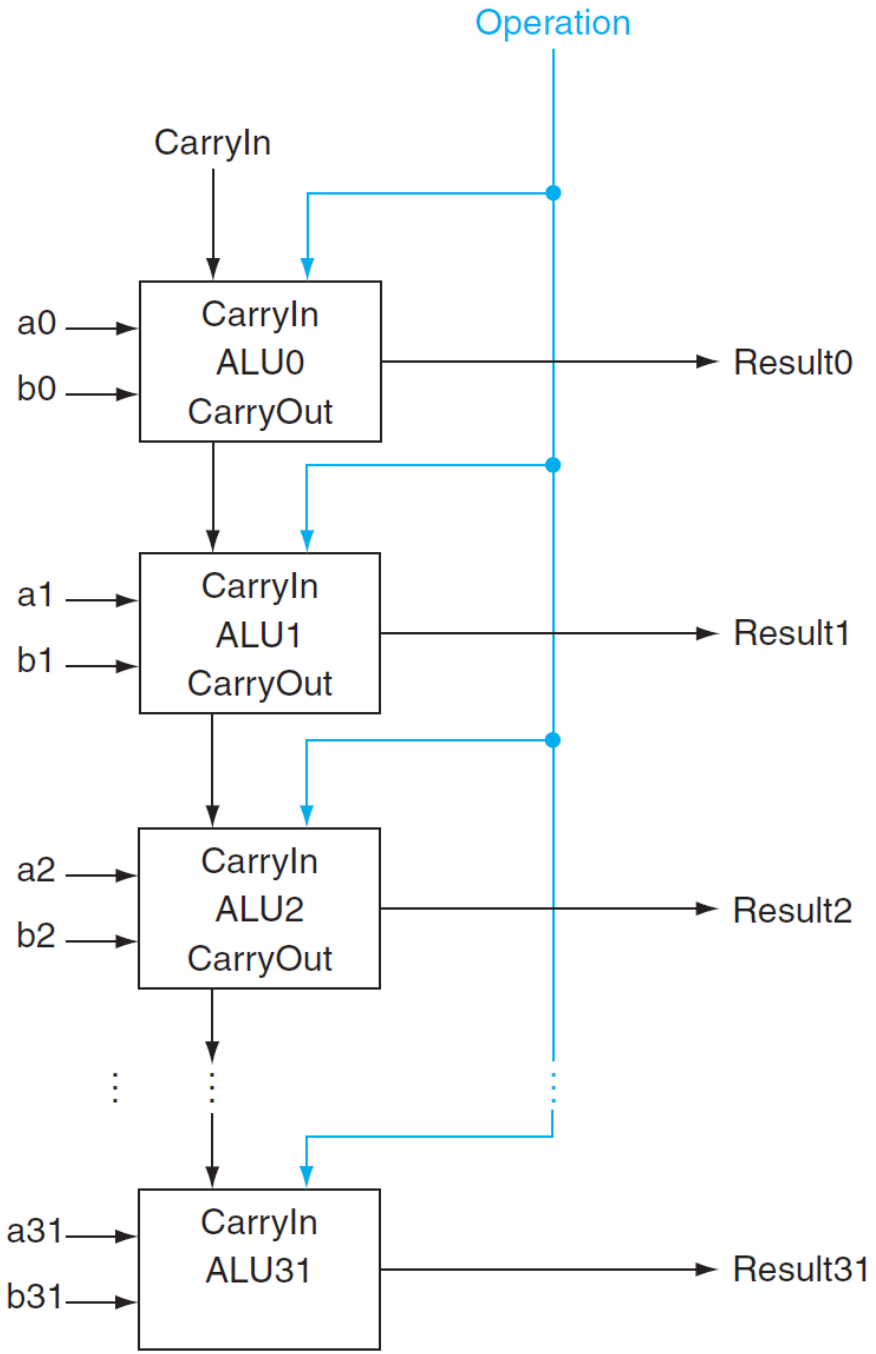
# Exercise – Answer

Design and draw a 1-Bit ALU that performs AND, OR, and addition on a and b as well as a and $\bar{b}$ .



# A 32-Bit ALU

- Constructed from 32 1-bit ALUs
- Ripple carry adder



# A 32-Bit ALU for MIPS

- The previous **32-bit ALU** can be tailored to **MIPS** by adding hardware for the following instructions:
  - **NOR**
  - **Conditional jump/branch**
  - **slt**

# A 1-Bit ALU for MIPS – NOR

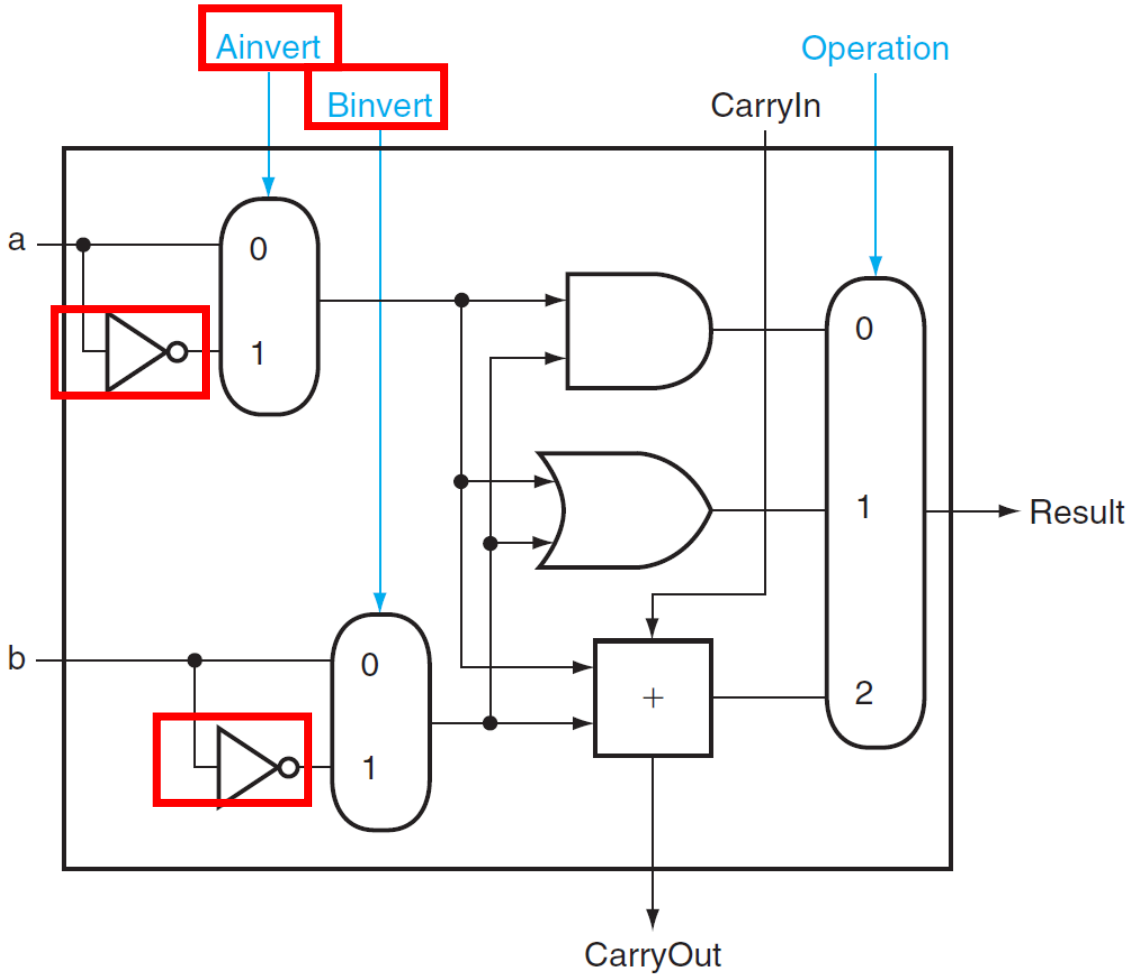
- **NOR** function
  - according to DeMorgan's theorem: $\overline{(a + b)} = \bar{a}.\bar{b}$
    - we only need to add NOT a and NOT b to the ALU.

# A 1-Bit ALU for MIPS – NOR

- **NOR** function
  - according to DeMorgan's theorem: $\overline{(a+b)} = \bar{a}.\bar{b}$
    - we only need to add NOT a and NOT b to the ALU.

**1-bit ALU with NOR function**

Ainvert = 1
Binvert =1
Operation = 00



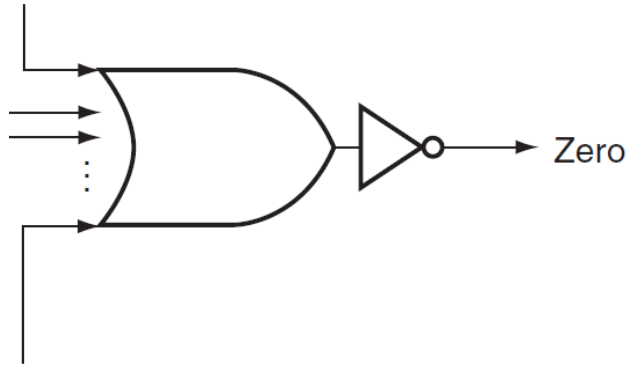
# A 32-Bit ALU for MIPS – Conditional Branch

- **How to determine if two binary numbers are equal to each other? (how to implement *beq* and *bne* instructions?)**
- **Which hardware components/gates needed in the datapath of the processor?**

# A 32-Bit ALU for MIPS – Conditional Branch – Continued

- **Conditional branch instructions**
  - Subtract b from a.
  - Test whether the result is zero by adding a zero detector to the ALU.
    - ✓OR all the outputs together and then send that signal through an inverter
    - ✓an output added to the ALU (called Zero)

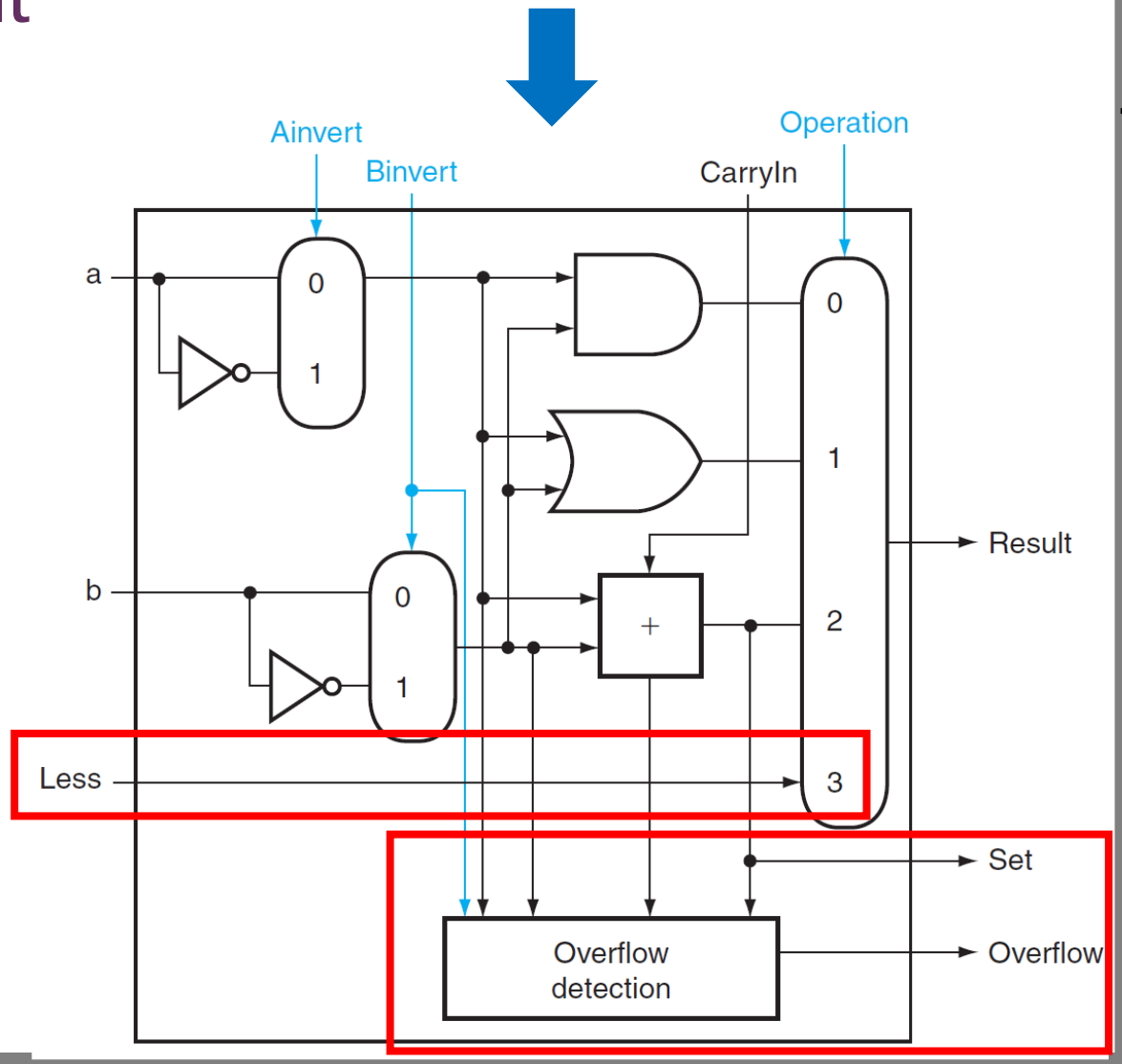
$$\text{Zero} = \overline{(\text{Result31} + \text{Result30} + \ldots + \text{Result2} + \text{Result1} + \text{Result0})}$$



# A 1-Bit ALU for MIPS – slt

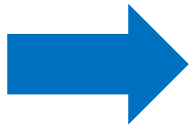
- **slt** (set on less than)
  - rs < rt → 1 otherwise 0
  - The least significant bit is set to 0 or 1 based on the result of the comparison and other bits must be set zero.
  - Add a new input (called ***less***) to the multiplexer.
  - Connect the result of comparison (**Set**) to the least significant bit.
  - Comparison is achieved by **subtraction**.
  - **Notice that every time we want the ALU to subtract, we set both CarryIn and Binvert to 1.**

**1-bit ALU with slt instruction**

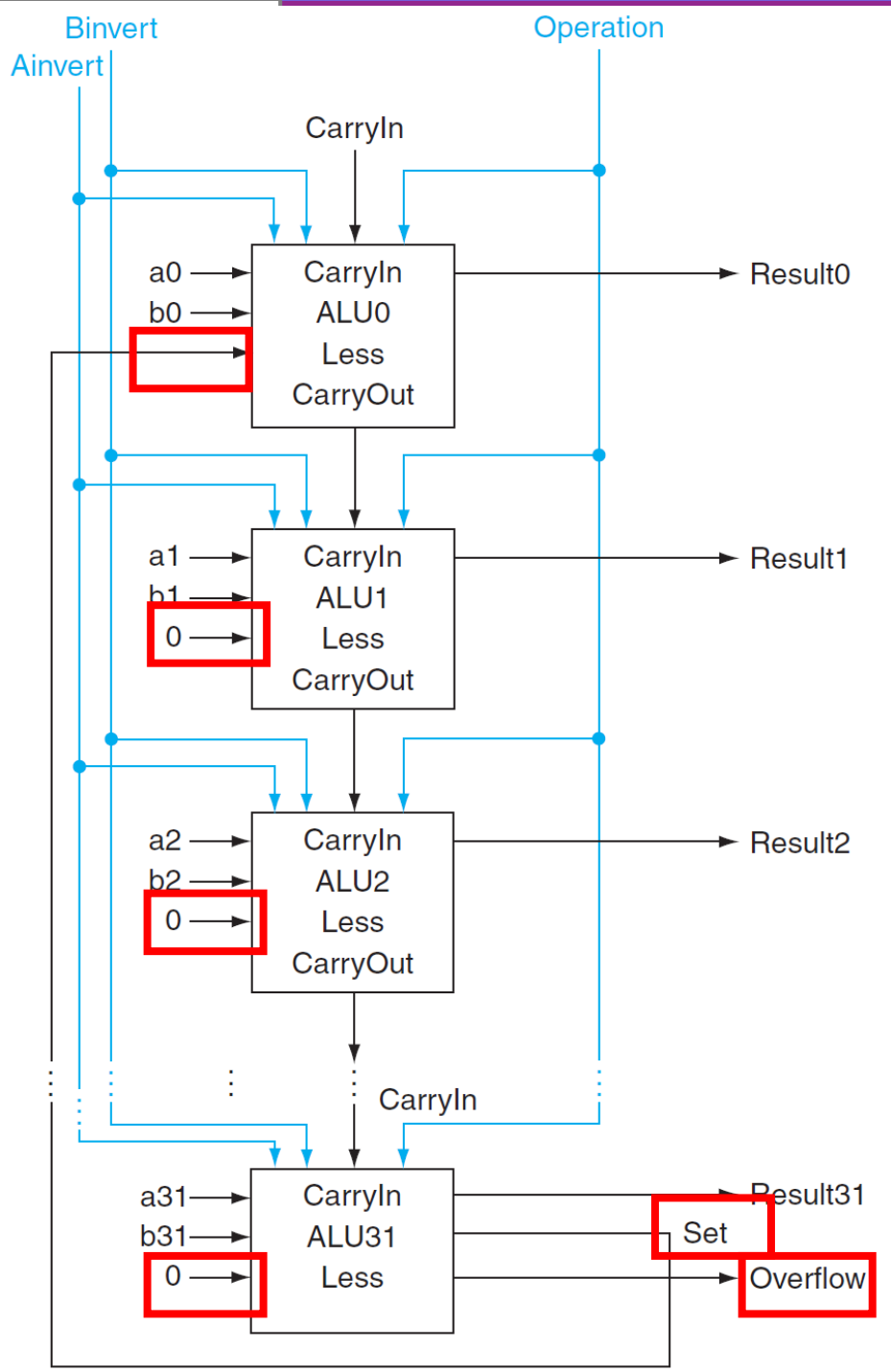


# A 32-Bit ALU for MIPS – slt

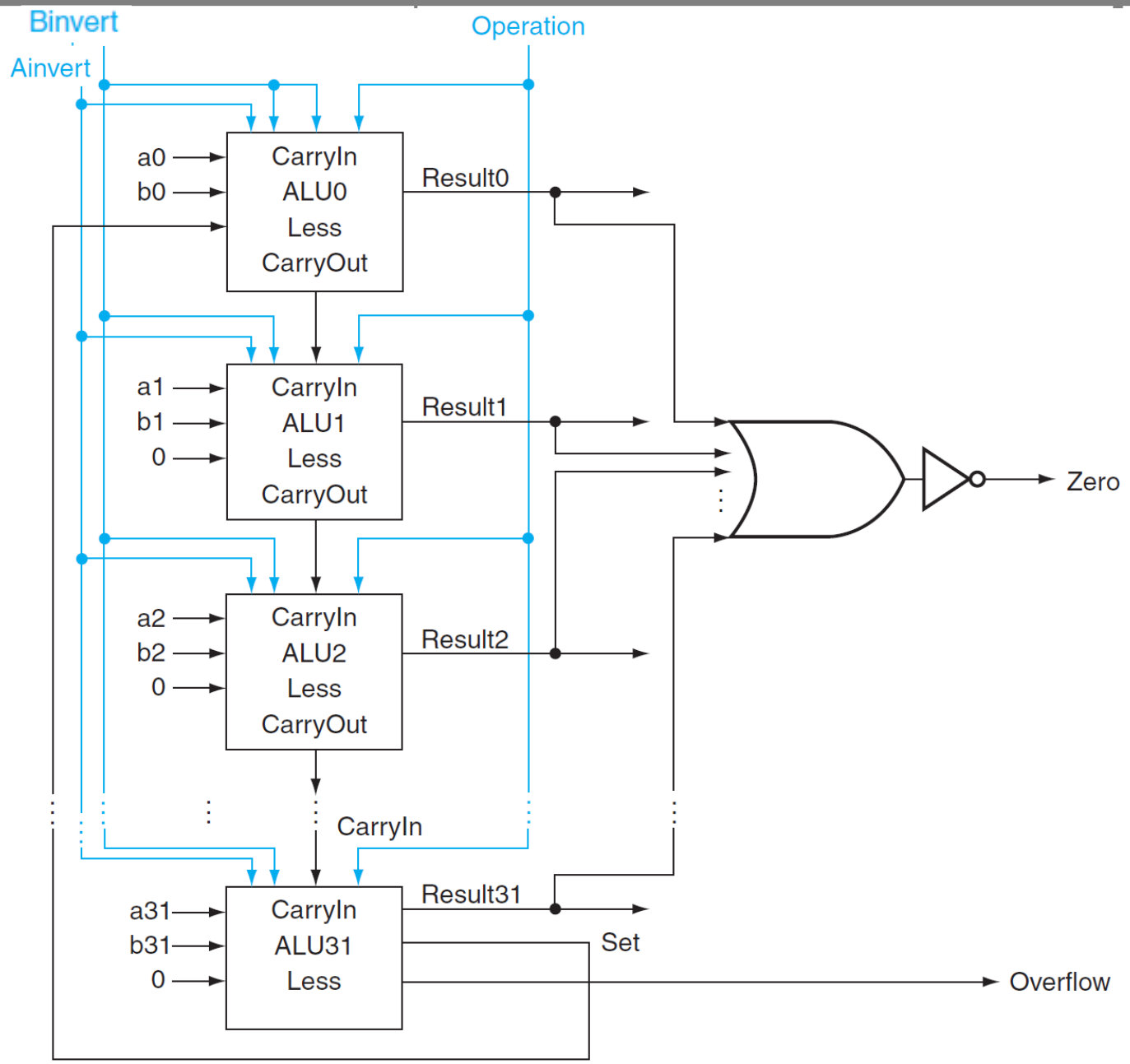
**32-bit ALU with slt instruction**



The desired result for *slt* corresponds to the result of **XOR** operation between the **sign bit** and **overflow**.



# The Final 32-Bit ALU for MIPS




**Control lines from left to right: Ainvert (1 bit), Binvert (1 bit), Operation (2 bits)**

| ALU control lines | Function |
|---|---|
| 0000 | AND |
| 0001 | OR |
| 0010 | add |
| 0110 | subtract |
| 0111 | set on less than |
| 1100 | NOR |

# Universal Symbol for a Complete ALU