# ARITHMETIC AND LOGIC UNIT

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

# Overview

- This lecture
  - How to design ALU
    - Design rules
    - Common logic blocks
    - Designing multi-function blocks

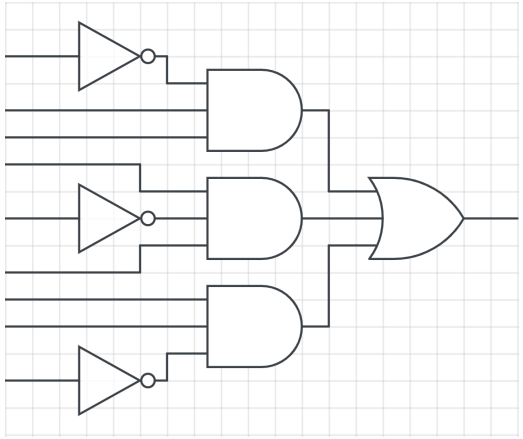
# Design Rules

- Any Boolean function can be represented by a truth table and sum of products (two gate levels)

| A | B | C | E |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Sum of Products

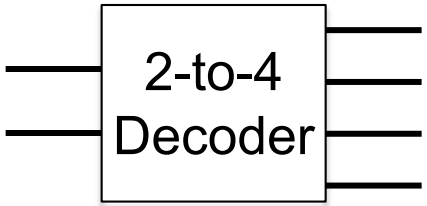
$E = (\overline{A}.B.C) + (A.\overline{B}.C) + (A.B.\overline{C})$



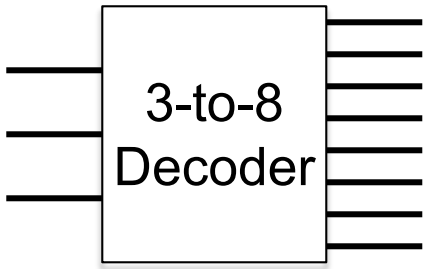
# Common Logic Blocks

- An n-input decoder takes n inputs, based on which only one out of $2^n$ outputs is activated

**2-to-4**



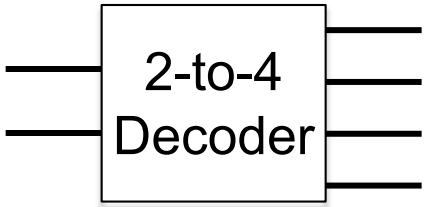

**3-to-8 (using two 1-to-4 units)**



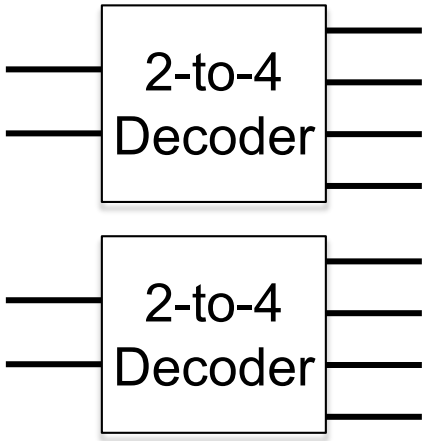
# Common Logic Blocks

- An n-input decoder takes n inputs, based on which only one out of $2^n$ outputs is activated

**2-to-4**



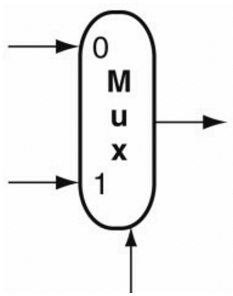

**3-to-8 (using two 1-to-4 units)**



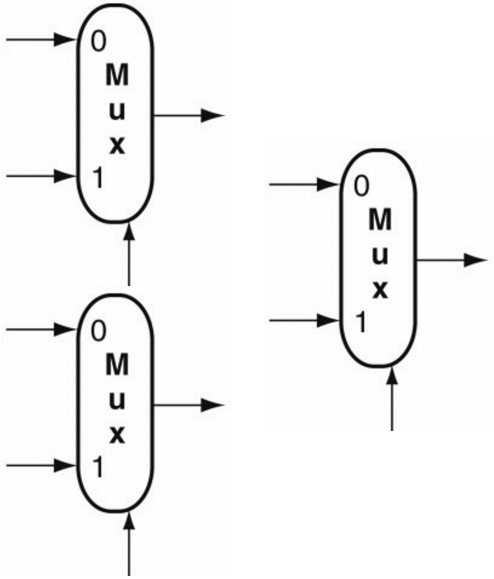
# Common Logic Blocks

- A multiplexer (or selector) reflects one of n inputs on the output depending on the value of the select bits

**2-to-1**

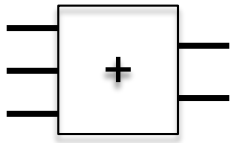



**4-to-1 (using there 2-to-1 units)**

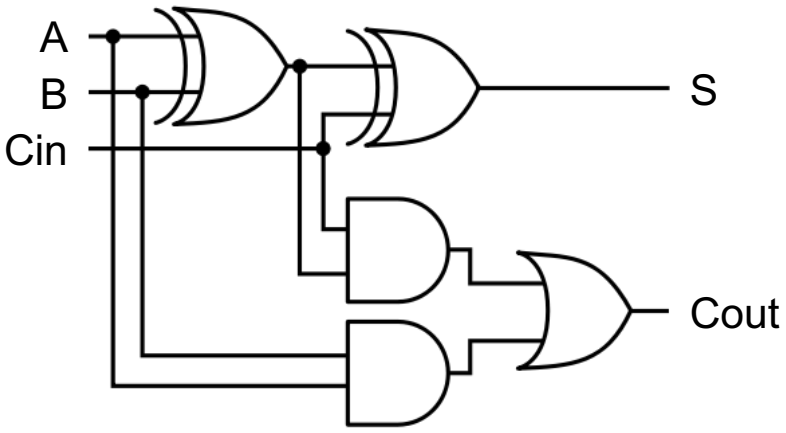


# Common Logic Blocks

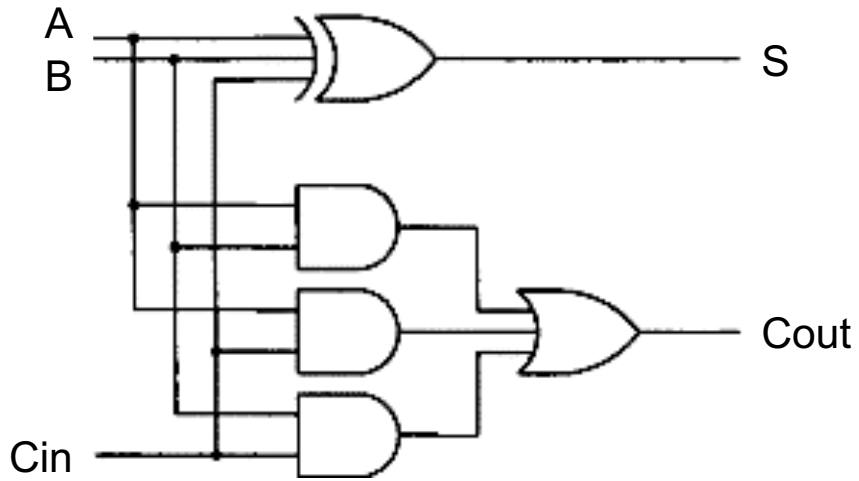
- A full adder generates the sum (S) and carry (Cout) for inputs A, B, and Cin.




**Example Ckt 1:**

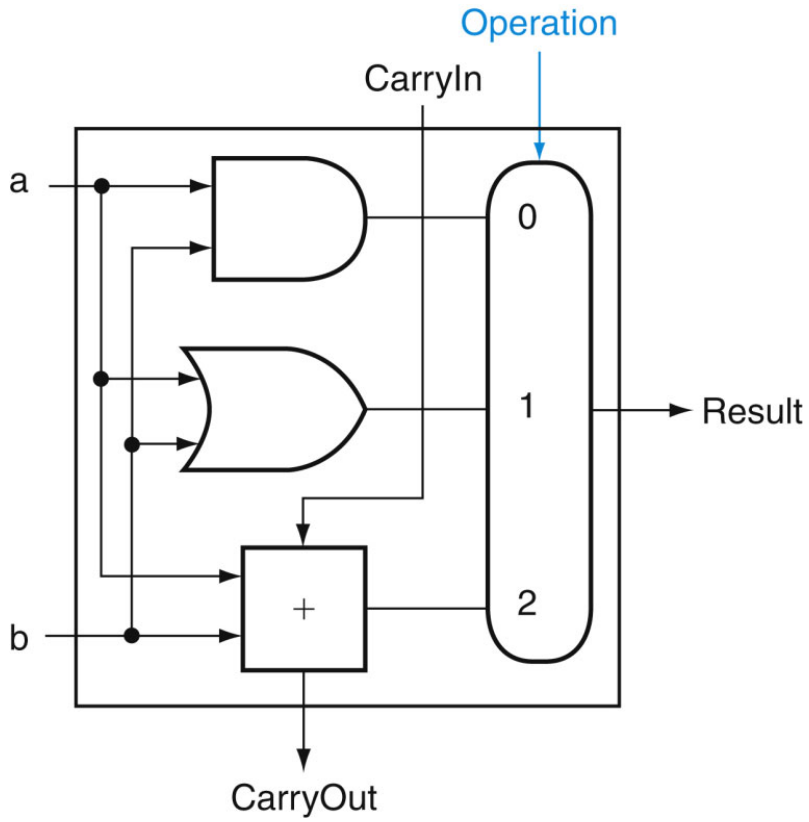



**Example Ckt 2:**



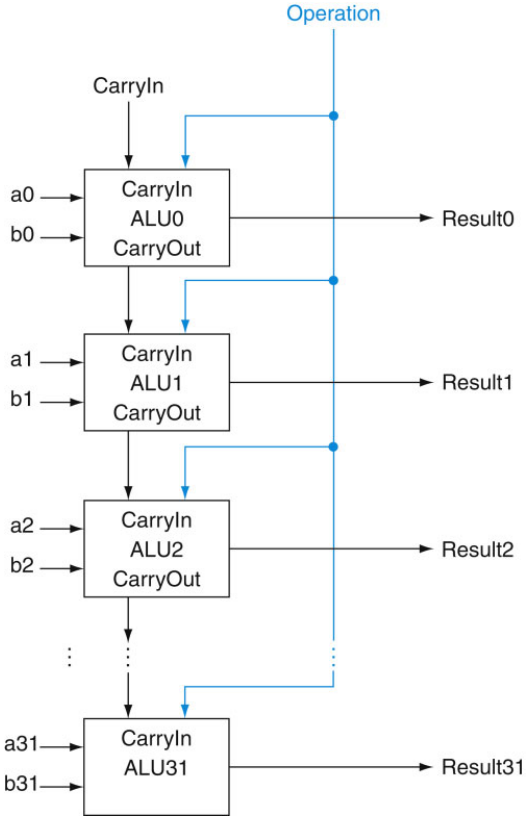
# Multi-Function Blocks

- A 1-bit ALU with ADD, OR, and AND operations
  - A multiplexer selects between the operations



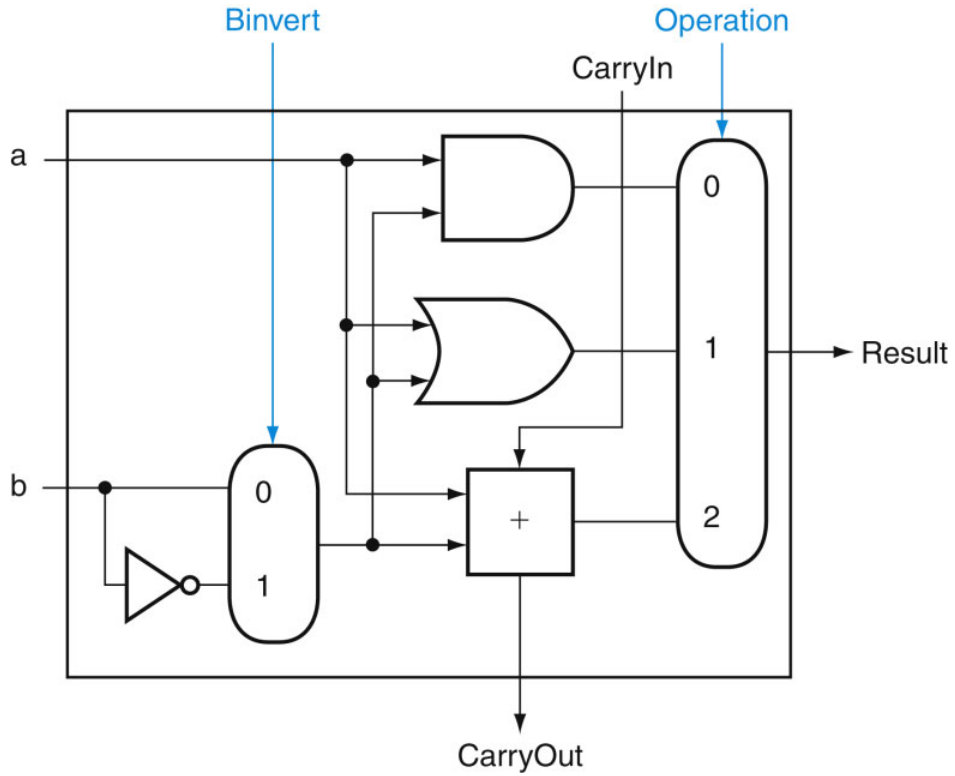
# Multibit ALU

- 1-bit ALUs are connected "in series" with the carry-out of 1 box going into the carry-in of the next box



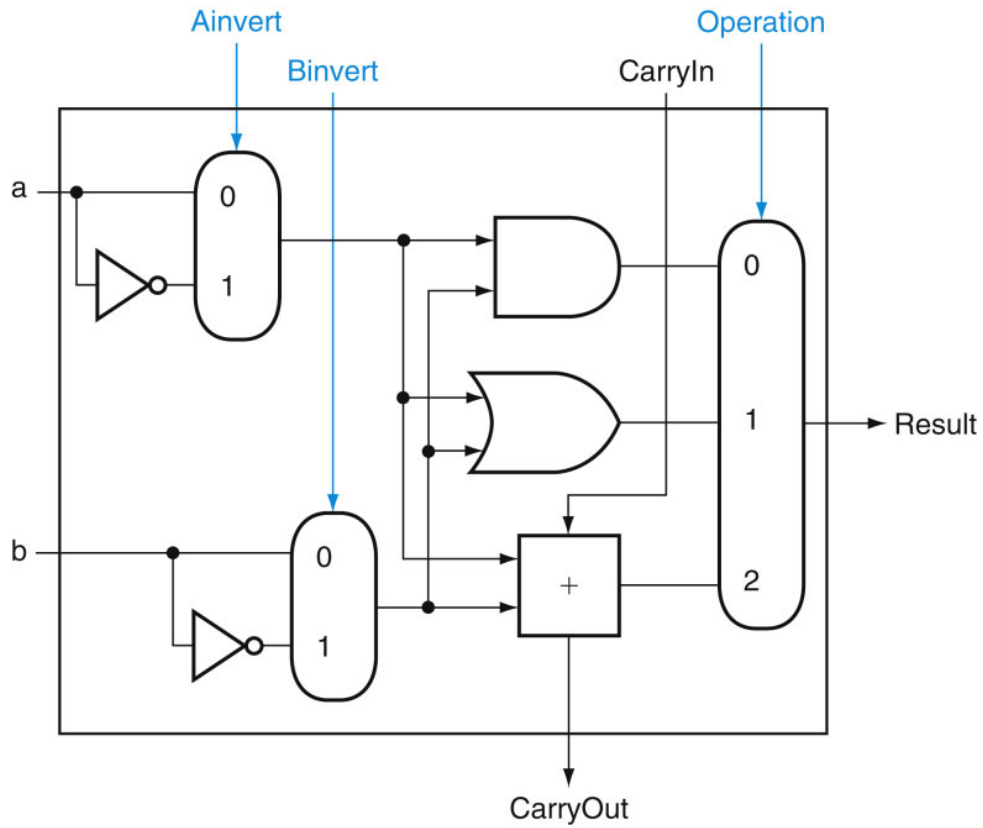
# Incorporating Subtraction

- Must invert bits of B and add a 1
  - Include an inverter CarryIn for the first bit is 1
- The CarryIn signal (for the first bit) can be the same as the Binvert signal



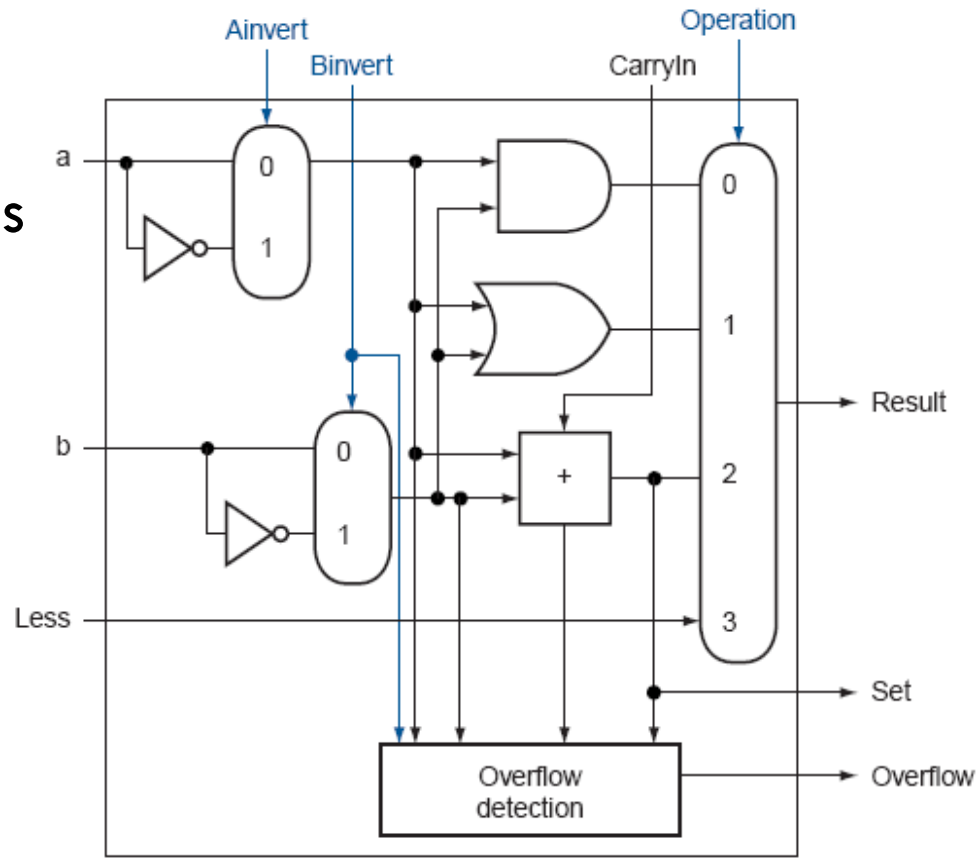
# Incorporating NAND and NOR

- Must invert bits of A and B to realize NAND and NOR
  - Recall DeMorgan's Laws



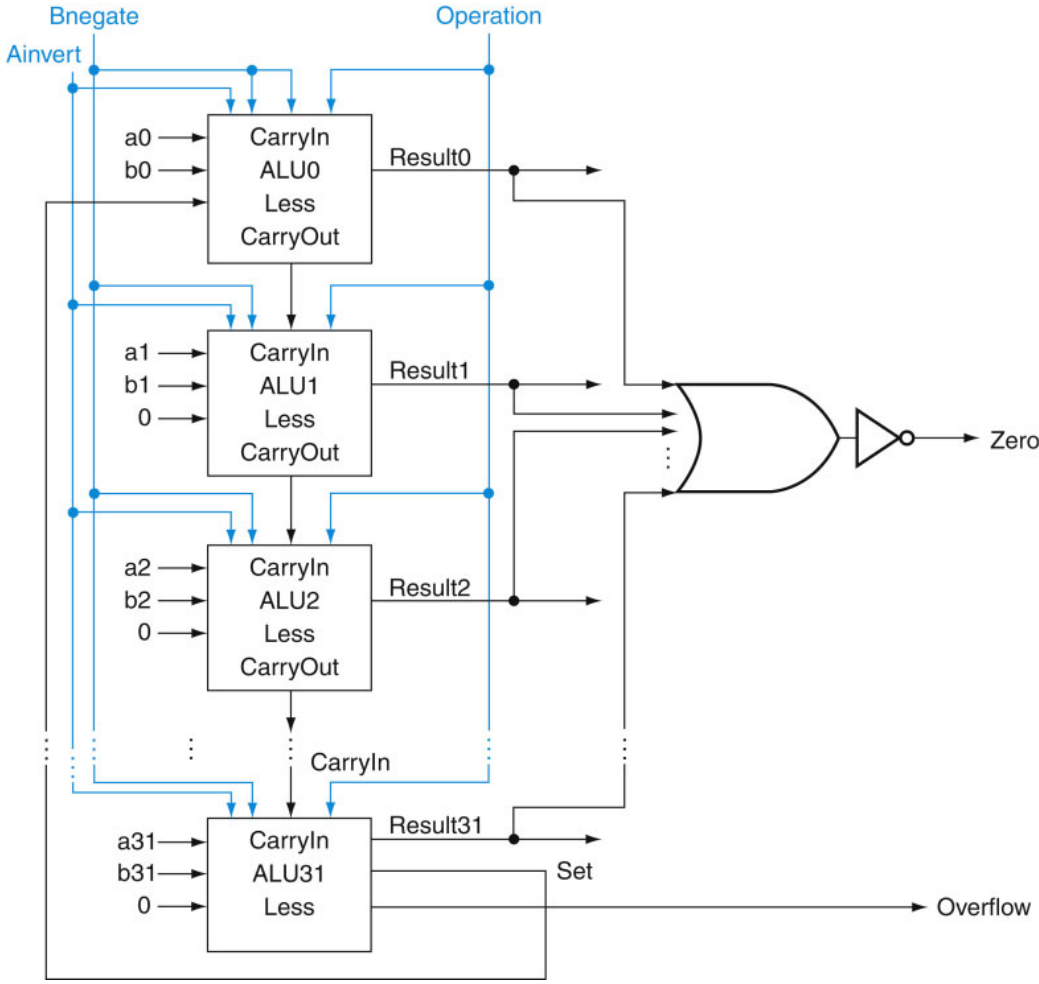
# Incorporating Comparison

- For example: the slt instruction
  - Perform a – b and check the sign
  - New signal (Less) that is zero for ALU boxes 1-31
  - The 31st box has a unit to detect overflow and sign – the sign bit serves as the Less signal for the 0th box



# Incorporating Comparison

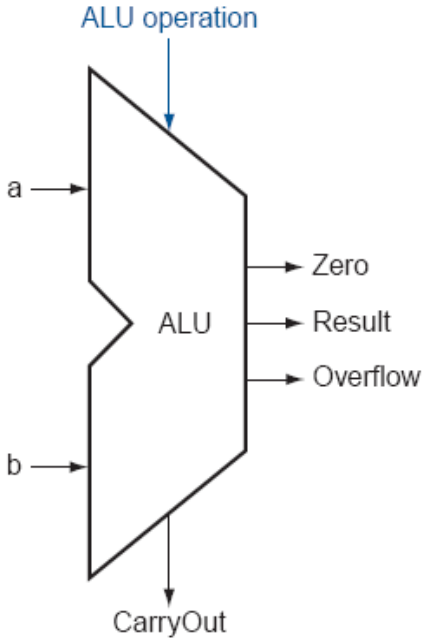
- For example: the beq instruction
  - Perform a – b and confirm that the result is all zero's
  - What about bne?
  - Control lines determine what operations to be done.



# Incorporating Comparison

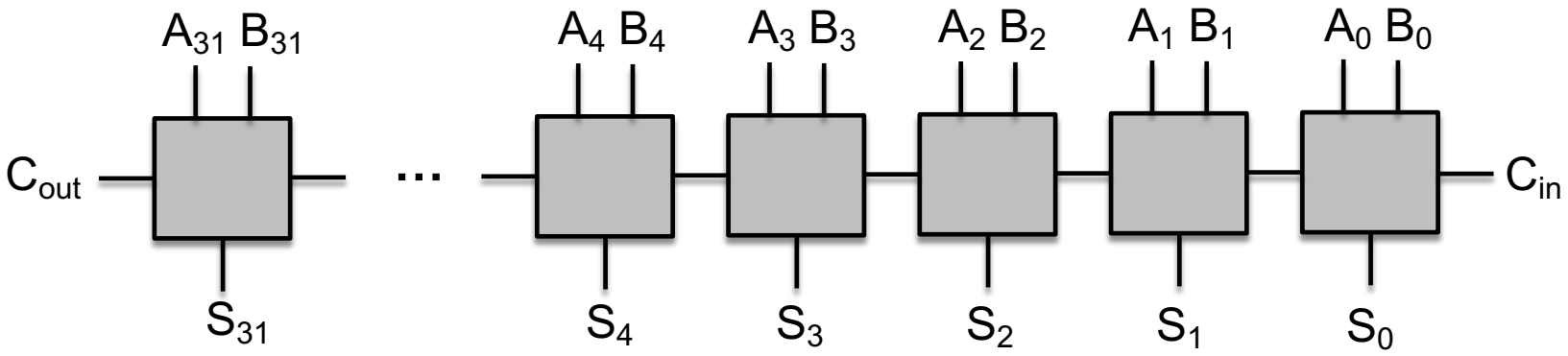
- For example: the beq instruction
  - Perform a – b and confirm that the result is all zero's
  - What about bne?
  - Control lines determine what operations to be done.

| | Ai | Bn | Op |
|---|---|---|---|
| AND | 0 | 0 | 00 |
| OR | 0 | 0 | 01 |
| Add | 0 | 0 | 10 |
| Sub | 0 | 1 | 10 |
| SLT | 0 | 1 | 11 |
| NOR | 1 | 1 | 00 |



# Carry Ripple Adder

- Simplest design by cascading 1-bit boxes
  - Each 1-bit box sequentially implements AND and OR
  - Critical path: the total delay is the time to go through 64 gates
- How to make a 32-bit addition faster?



# Carry Ripple Adder

- Simplest design by cascading 1-bit boxes
  - Each 1-bit box sequentially implements AND and OR
  - Critical path: the total delay is the time to go through 64 gates
- How to make a 32-bit addition faster?
- Recall: any logic equation can be expressed as the sum of products (only 2 gate levels!)
  - Challenges: many parallel gates with very large inputs
  - Solution: we'll find a compromise