ECE 485/585
Computer Organization and Design

Lecture 6: ALU Design
Fall 2022

Won-Jae Yi, Ph.D.

Department of Electrical and Computer Engineering
Illinois Institute of Technology

Outline

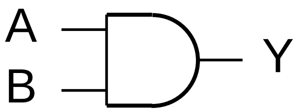
- 1-Bit ALU Design
    - Logical operations
    - Addition
    - Subtraction
    - Conditional branch
- 32-Bit ALU Design
- Faster Adder Implementation

Logic Design Basics

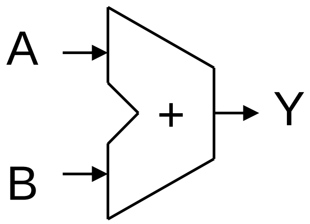
- Information encoded in binary
  - Low voltage = 0, High voltage = 1
  - One wire per bit
  - Multi-bit data encoded on multi-wire buses
- Combinational element
  - Operate on data
  - Output is a function of input
- State (sequential) elements
  - Store information

Combinational Elements

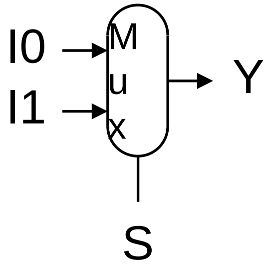
- AND-gate
  - Y = A & B



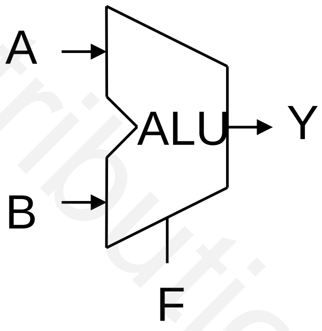
- Adder
  - Y = A + B



- Multiplexer
  - Y = S ? I1 : I0

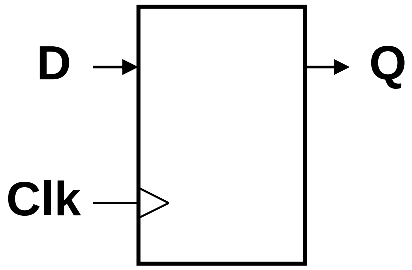


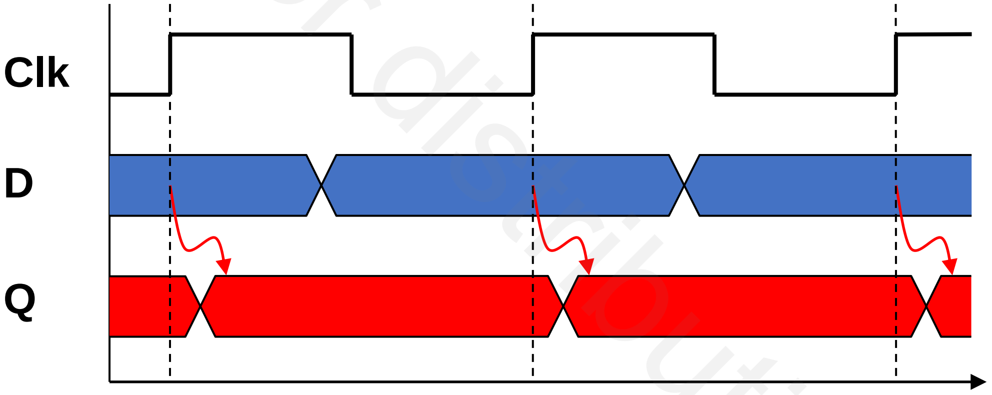
- Arithmetic/Logic Unit
  - Y = F(A, B)



Sequential Elements

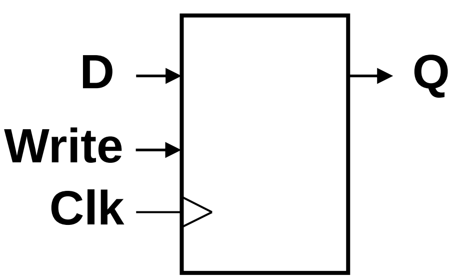
- Register: stores data in a circuit
  - Uses a clock signal to determine when to update the stored value
  - Edge-triggered: update when Clk changes from 0 to 1

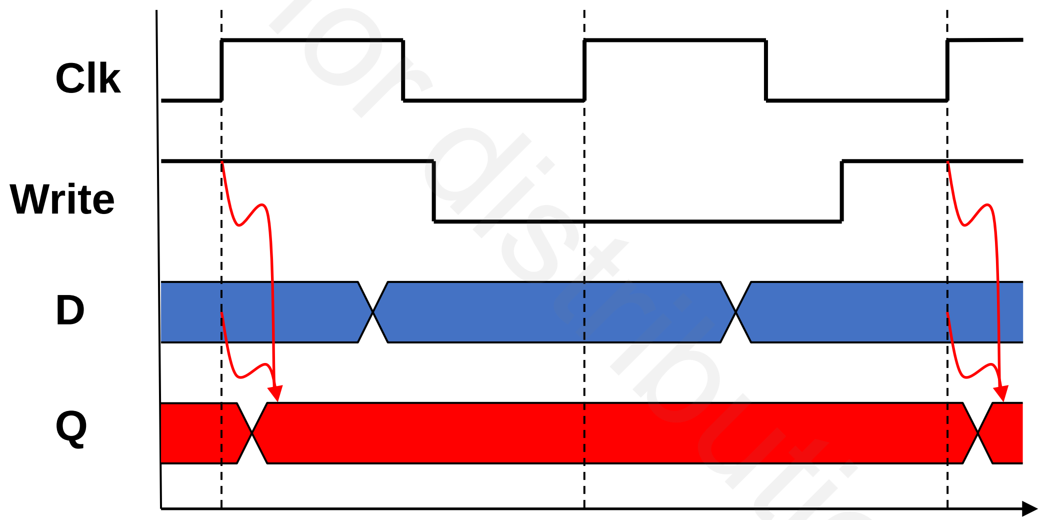




Sequential Elements

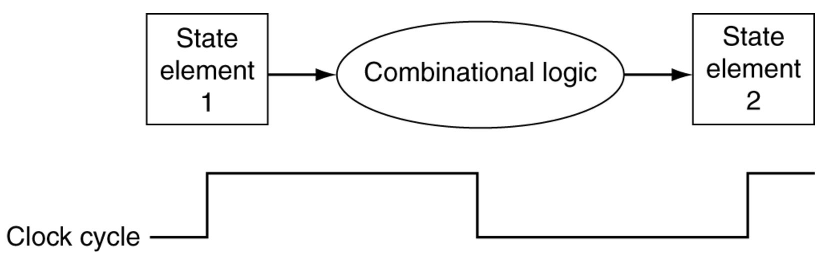
- Register with write control
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later

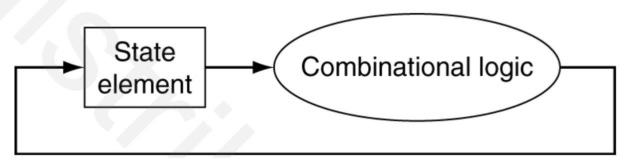




Clocking Methodology

- Combinational logic transforms data during clock cycles
  - Between clock edges
  - Input from state elements, output to state element
  - Longest delay determines clock period





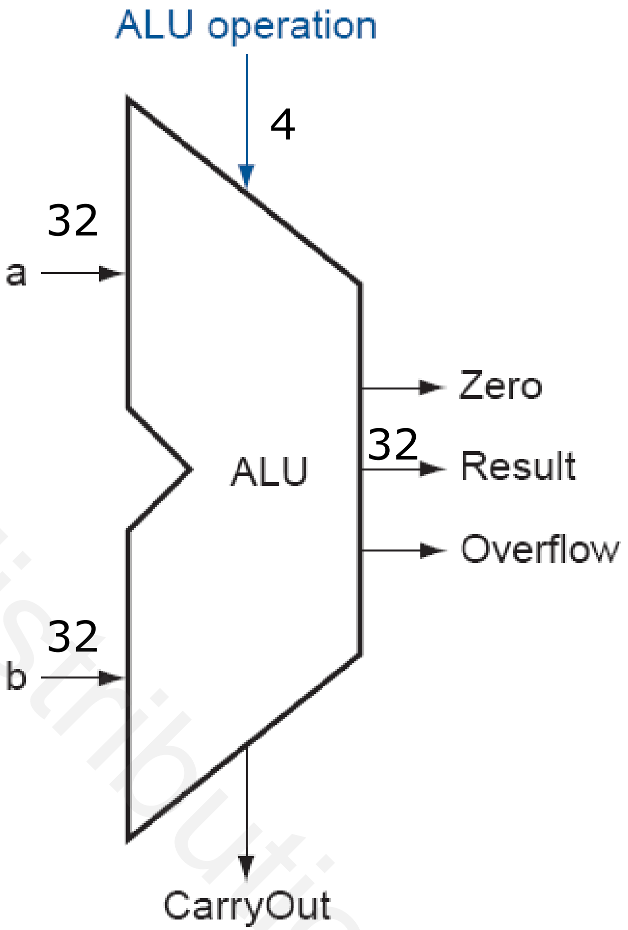
Arithmetic Logic Unit (ALU)

- Datapath and Control unit are the main components of any processing unit
  - ALU is part of the datapath system
  - Computations are done in ALU
- ALU design using only AND, OR, inverter gates and multiplexors
- 32-bit ALU for MIPS
  - First build a 1-bit ALU
  - Use 32 1-bit ALUs to form 32-bit ALU

ALU Interface

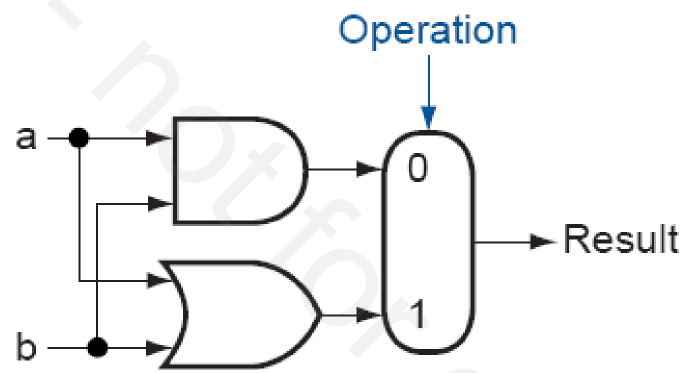
- We will be designing a 32-bit ALU with the following interface:

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



1-Bit ALU

- Start with simple operations and build upon it
- Logical Operations for 1-Bit ALU:
    - AND
    - OR

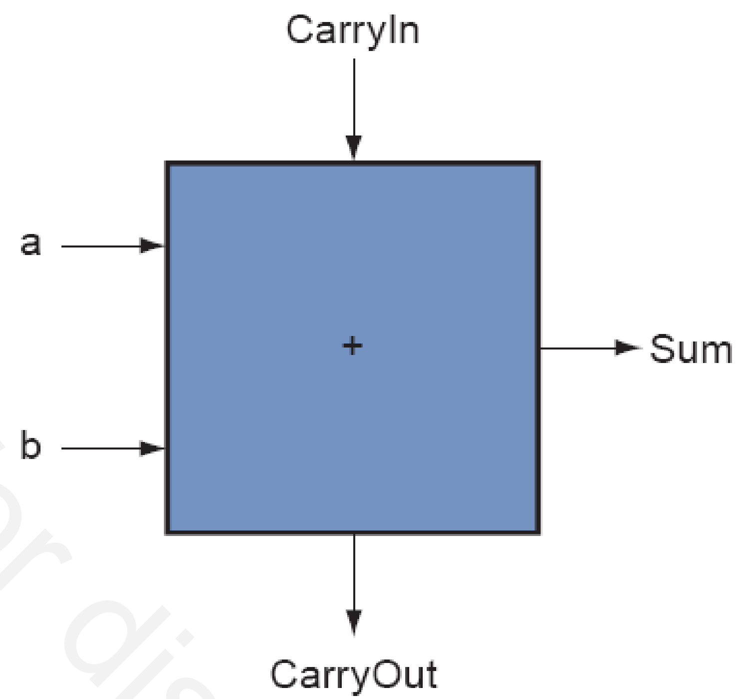


1-bit logical unit for AND and OR

- The function of the ALU is determined by the control signals for the multiplexor

Addition

- Full-Adder (3,2) adder:
  - Each adder has 3 inputs:
    - 2 inputs for the operands
    - 1 input for the *CarryIn*
  - Each adder has 2 outputs
    - 1 output for the result
    - 1 output for the *CarryOut*



- Half-Adder (2,2) adder
  - Each half-adder has 2 inputs (no *CarryIn*)

Full Adder

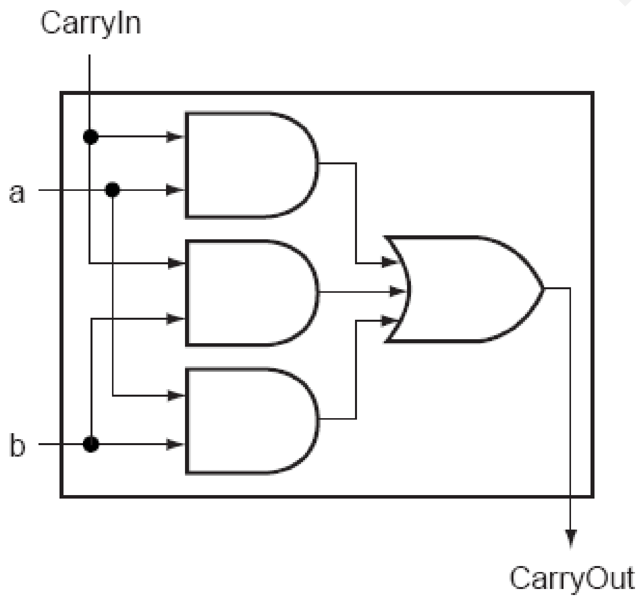
- Input/Output specifications for a 1-bit adder

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	$1 + 1 + 1 = 11_{two}$

Full Adder

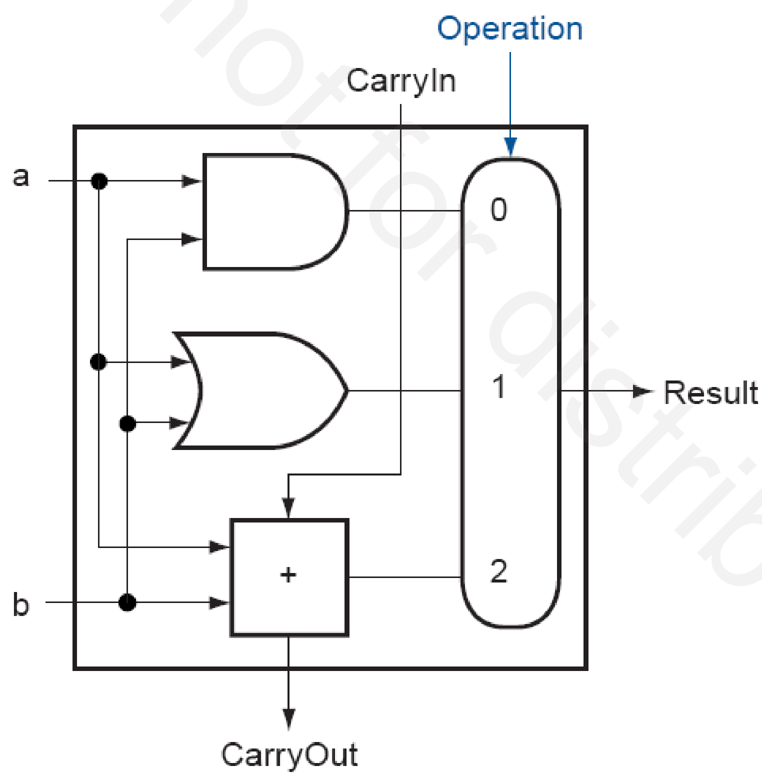
$$CarryOut = (b \cdot CarryIn) + (a \cdot CarryIn) + (a \cdot b)$$

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



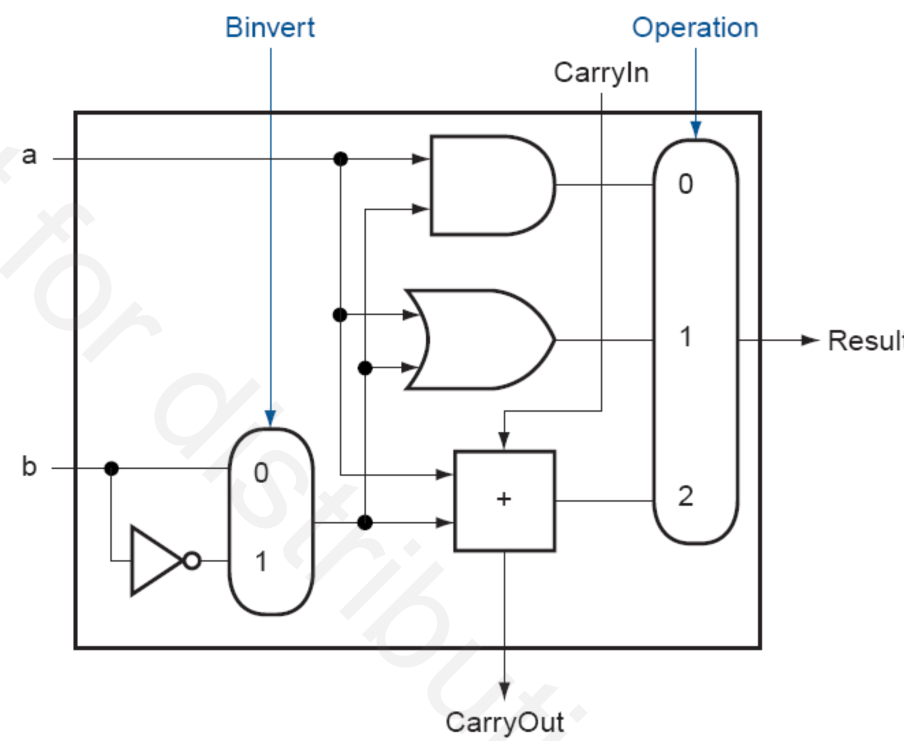
Expanding ALU Hardware

- Add more functionality by expanding the multiplexor
  - Additional inputs and control lines



Subtraction

- Subtraction is the same as adding the negative version of the operand
  - Invert each bit and add 1
- Add a new multiplexor to invert the operand **b**



NOR Gate

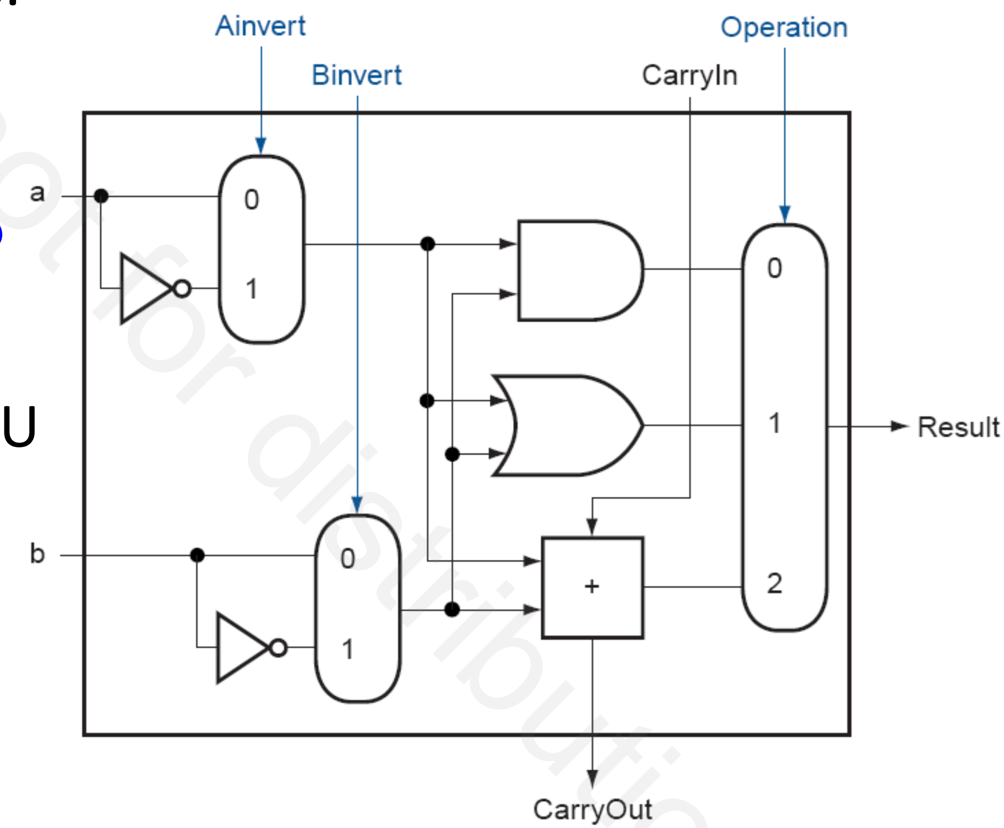
- Use the existing hardware in the ALU:

  According to DeMorgan's:

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

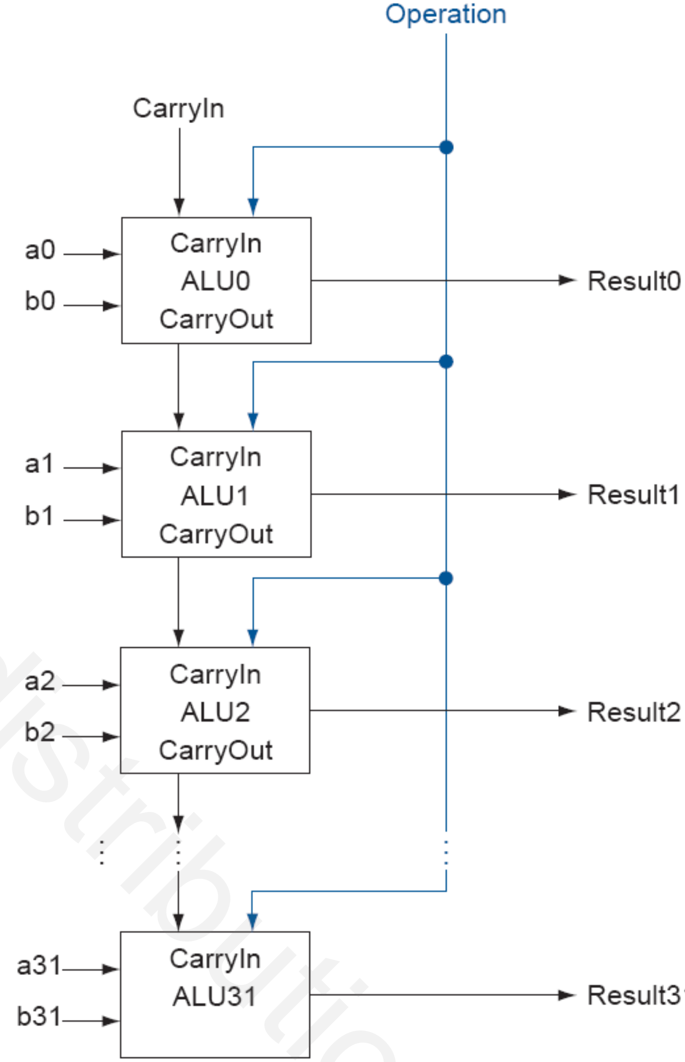
- **AND** gate and **NOT b** already exists;
  - add **NOT a** to the ALU

- Another Multiplexor and an inverter



32-bit ALU

- Full 32-bit ALU can be created by connecting the adjacent 1-bit ALU's
- Results will ripple from the least significant bit **`Result0`** to most significant bit **`Result31`**
- **`Ripple-Carry`** adder



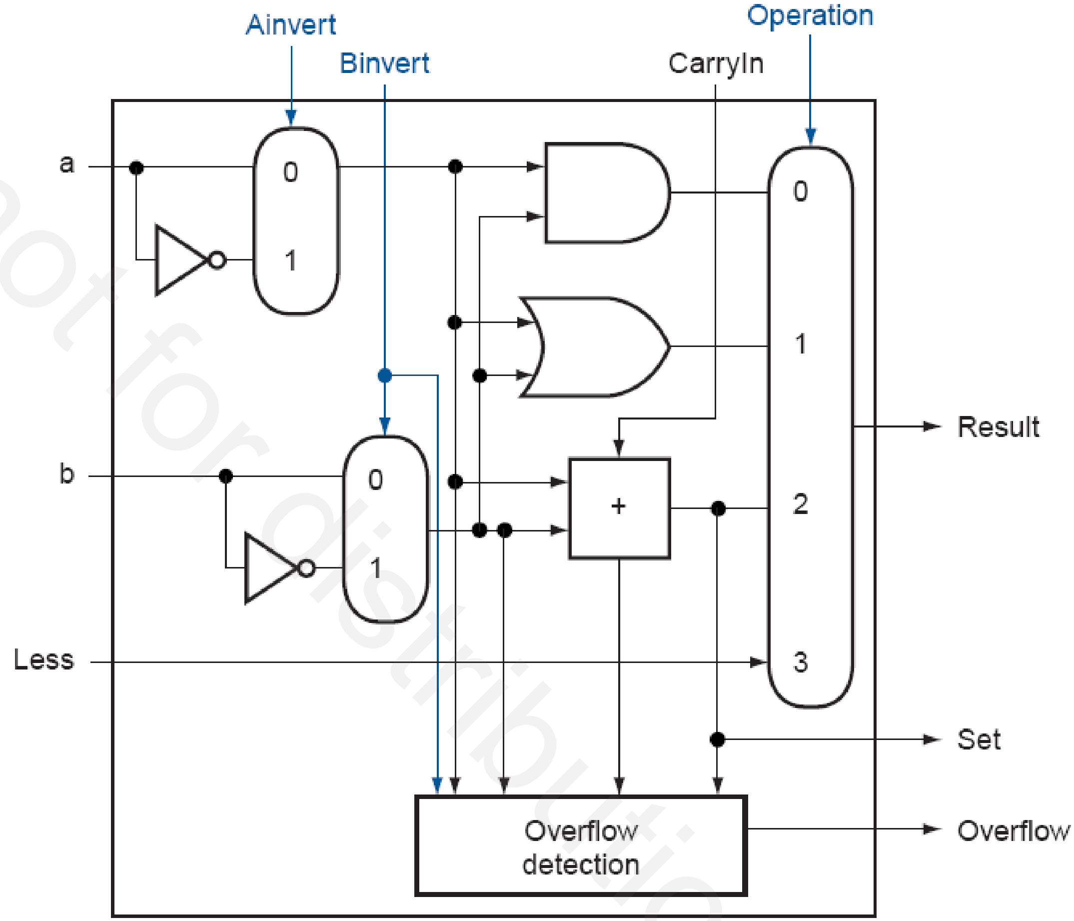
Set-on-Less-Than (slt) Instruction

- slt: compares two registers $r_s < r_t$
  - If $r_s < r_t$ then r_d set to 1; else r_d clear to 0.
- Another input (**`Less`**) required for the Result multiplexor:
  - connect 0 to **`Less`** input of upper 31 bits;
  - What about the least significant bit?
- Subtract **`(a-b)`** and the sign bit will determine the value for less input for the least significant bit
- A new output bit **`SET`** (adder output) is required for the most significant bit in the ALU

1-bit ALU for the MSB

- Overflow logic can also be integrated into most significant ALU bit

$$V = C_n \otimes C_{n-1}$$



Zero Detection

- Conditional branch instructions check if two registers are equal or unequal
  - do subtraction and see if the result is 0
- Needs simple hardware for NOR-ing 32 output bits:

$$Zero = \overline{(result31 + result30 + \ldots + result2 + result1 + result0)}$$

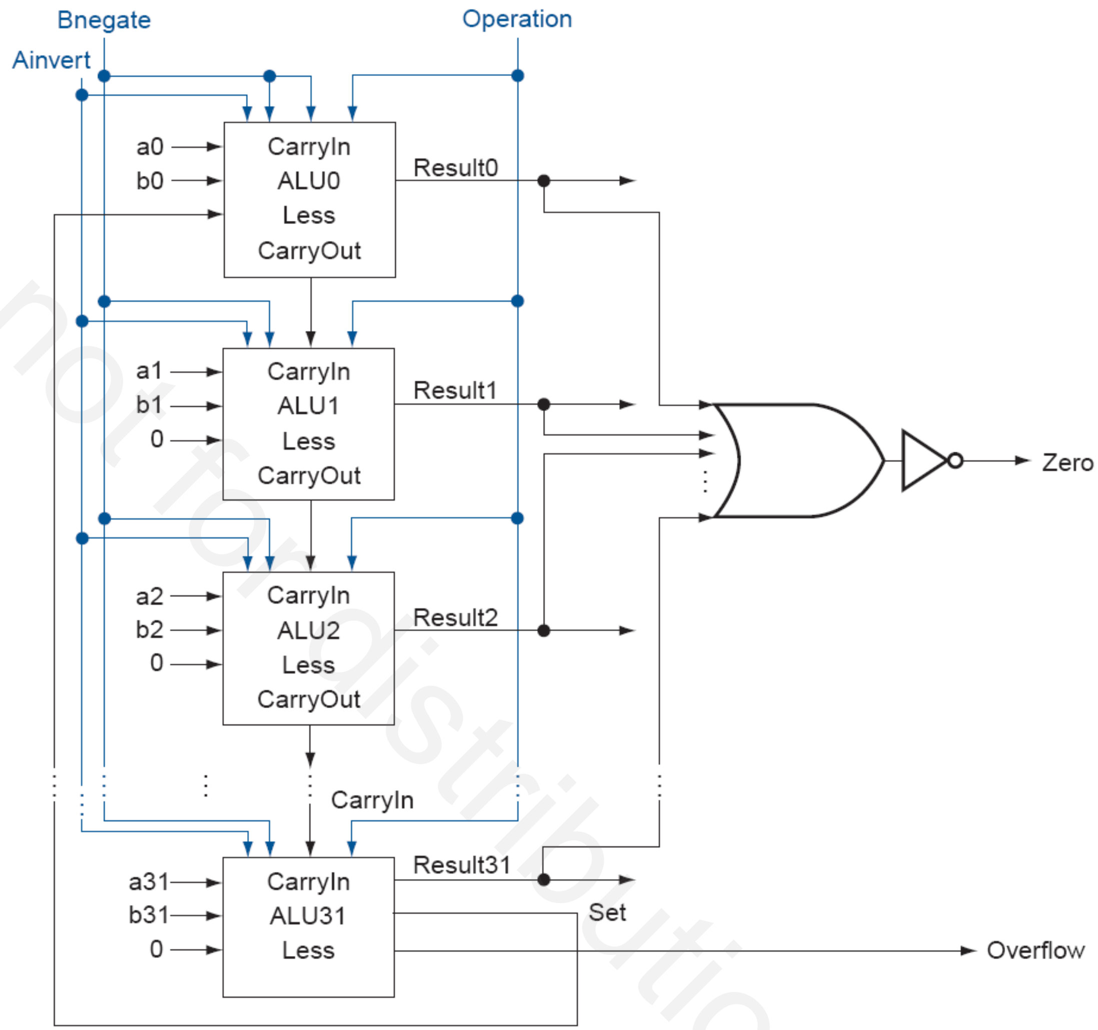
32-bit MIPS ALU

- 4-bit control signals:

  2-bits for Operation

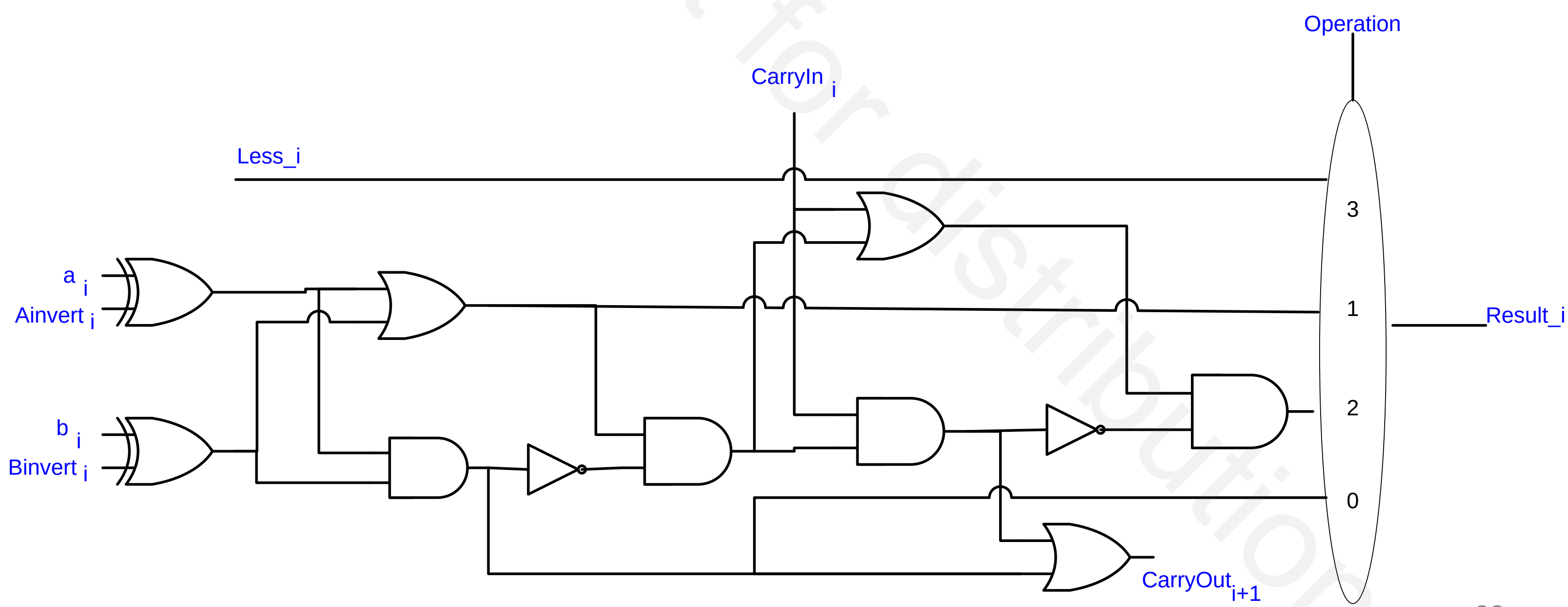
  1-bit for B_{negate} line

  1-bit for A_{invert} line



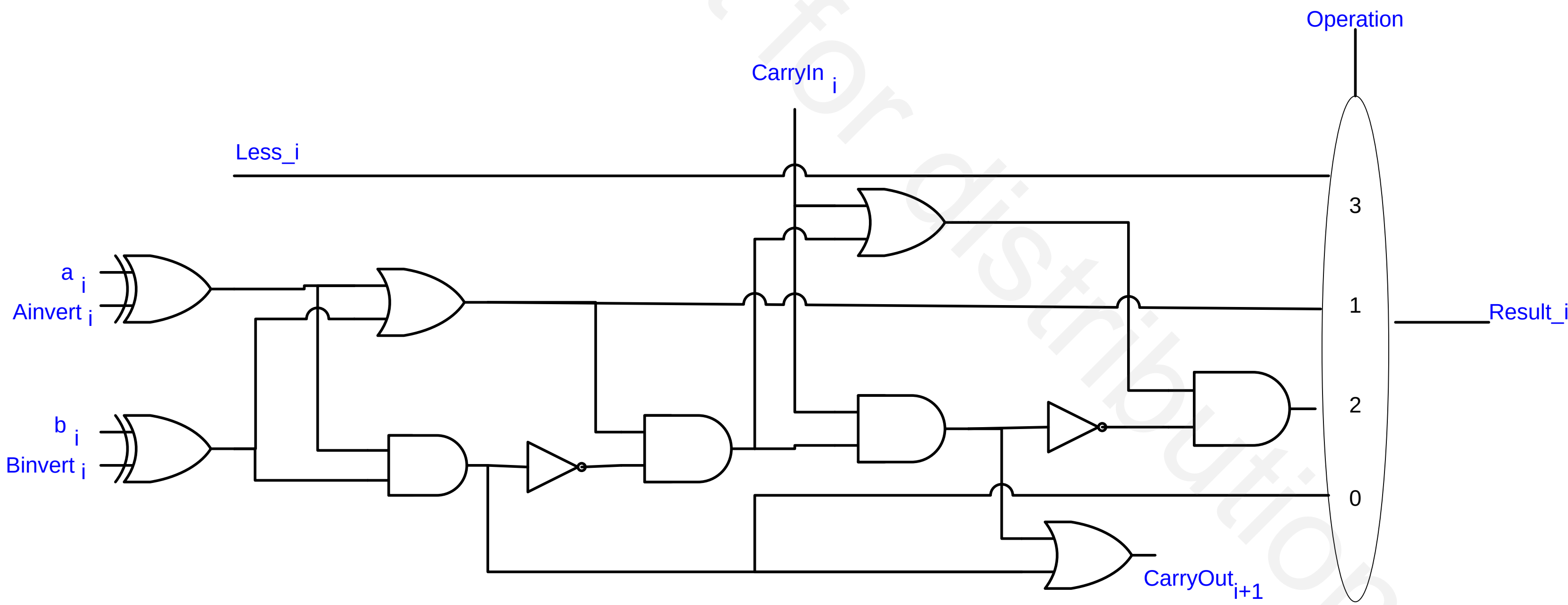
Gate Count

- How many gates are required by
  - 1-bit ALU?
  - 32-bit ALU?
  - n-bit ALU?
- Assume AND/OR =1gate, MUX=5gates, XOR=3gates, Inv=1gate



Gate Delays

- What is the delay of
  - 1-bit ALU?
  - 32-bit ALU?
  - n-bit ALU?
- Assume delays for AND/OR =1t, MUX=2t, XOR=2t, Inv=1t



Faster Addition

- The key to speeding up addition is determining the carry into the higher order bits sooner
- With Carry Lookahead Adder (CLA), carries are computed in parallel

Summary

- An n-bit ALU can be designed by cascading n 1-bit ALUs
- ALU functionality is increased by adding more control signals and inputs to multiplexors
- Carry lookahead adders are used for fast addition
- Different design options are available for ALU implementations
  - Decision based on area, power and speed requirements