

CSCI 210: Computer Architecture

Lecture 18: Arithmetic Logic Unit

Stephen Checkoway

Oberlin College

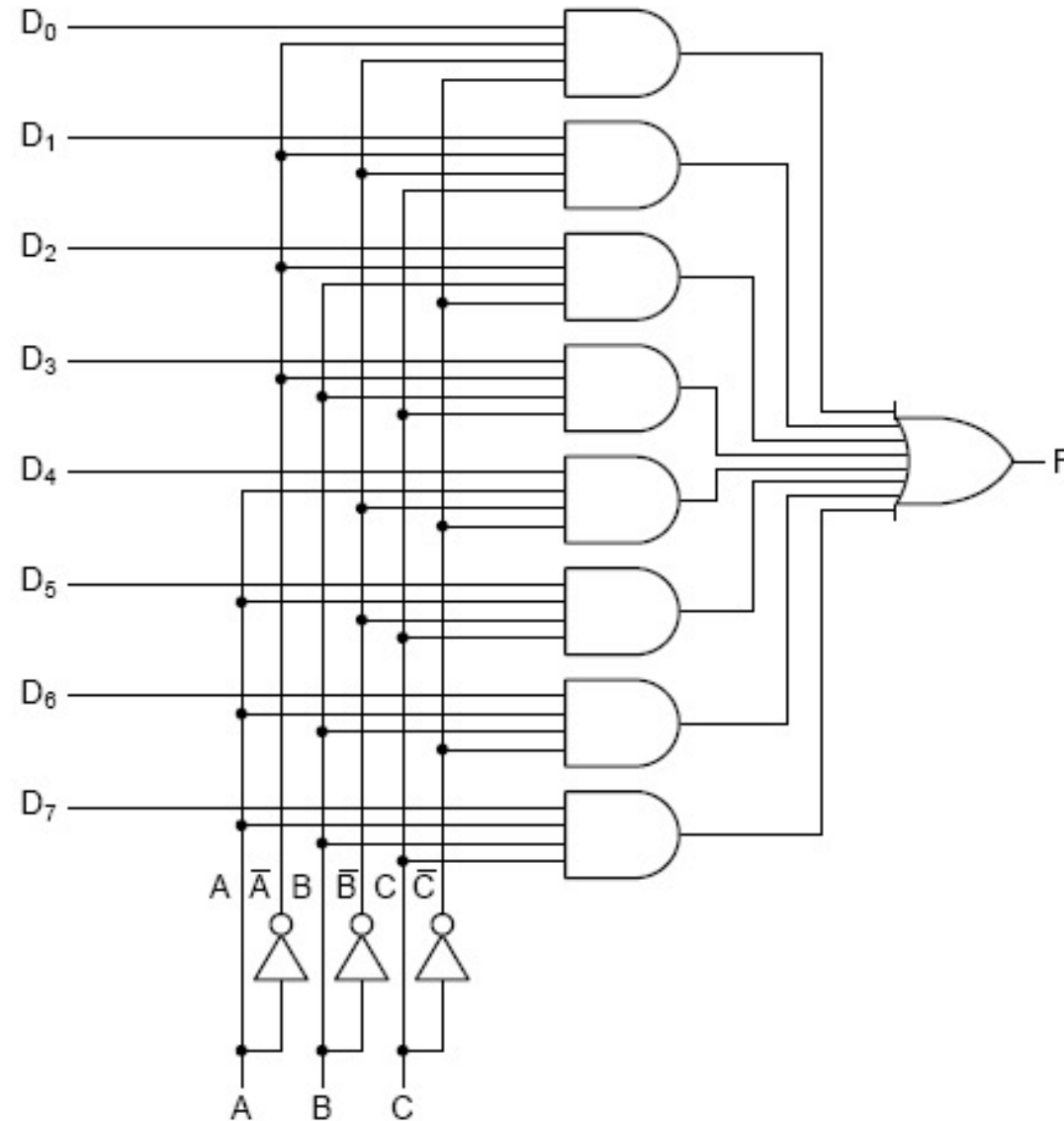
Nov 12, 2021

Slides from Cynthia Taylor

Announcements

- Problem Set 5 due tonight
- Lab 4 due Sunday
- Office Hours today 13:30 – 14:30

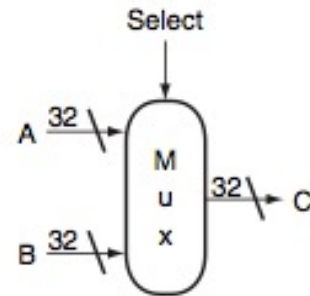
3-to-8 input multiplexer



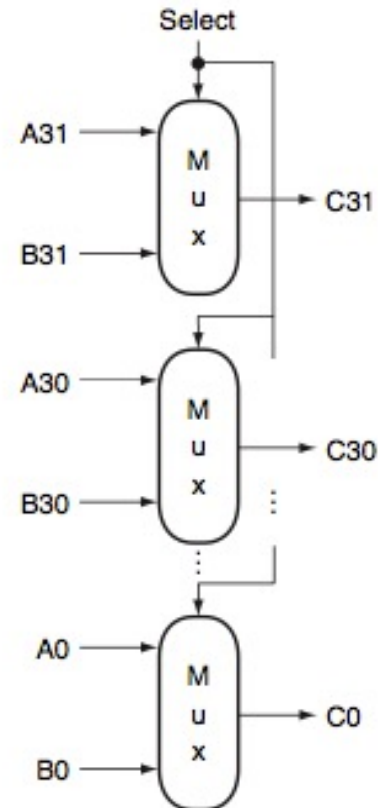
Scaling Up

- Have to perform combinatorial operations over an entire word (32-bits) of input.
- Bus: a collection of data lines that is treated together as a single logical signal.
- Example: A multiplexer is used to choose which of the two buses (each 32 bits wide) will be written into the Result register

Replicating a 1-bit Multiplexer 32 times



a. A 32-bit wide 2-to-1 multiplexor

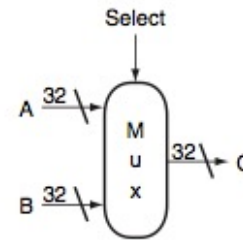


b. The 32-bit wide multiplexor is actually an array of 32 1-bit multiplexors

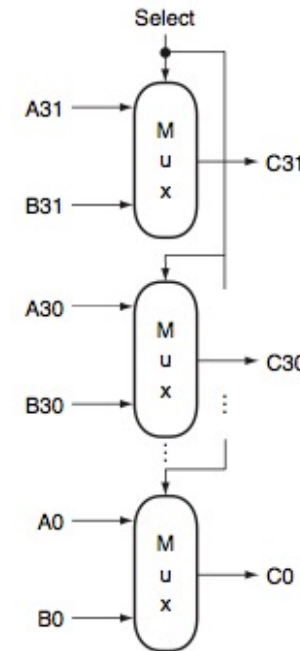
We want to choose which 32-bit bus value gets written to a register. Will the select value be the same for each 1-bit multiplexer?

A. Yes

B. No



a. A 32-bit wide 2-to-1 multiplexer



b. The 32-bit wide multiplexer is actually an array of 32 1-bit multiplexers

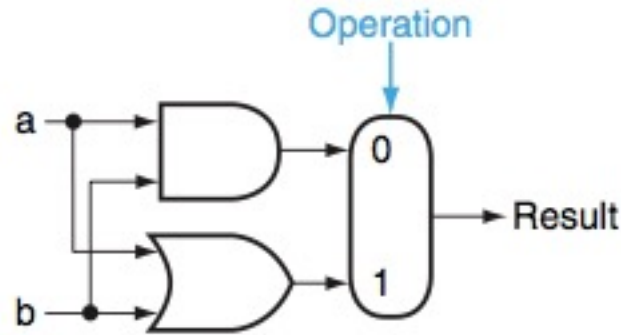
ALU: Basic Problem

- Need to use digital logic to build a unit that can do basic computation – math, logical operations, etc.
- Needs to be 32 bits wide, since MIPS has 32 bit words.
 - Build out of 1-bit ALUs

Our ALU will support the following instructions:

- Add/Addi
- Sub
- Or/Ori
- And/Andi
- Nor/Nori
- Nand/Nandi
- Set less than

1-bit ALU: AND and OR



- Inputs go to both AND and OR
- Multiplexer selects AND or OR function for output

1-bit Binary Adding

$$0 + 0 = 0$$

$$0 + 1 = 1$$

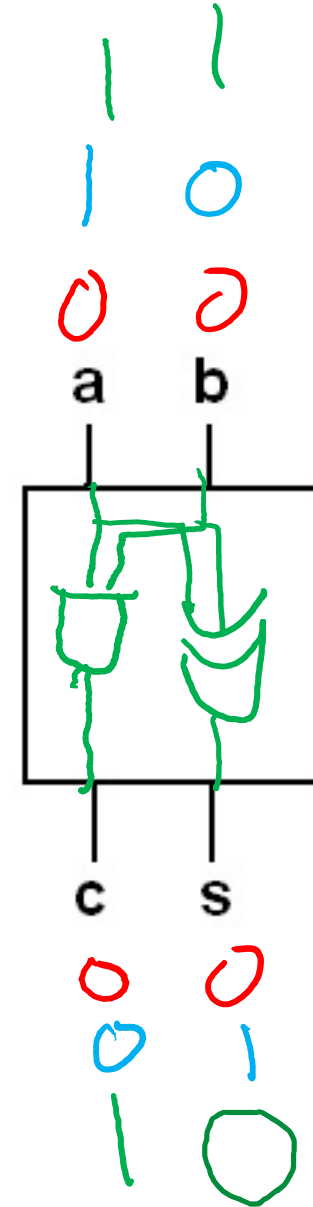
$$1 + 0 = 1$$

$$1 + 1 = 10$$

Need to account for two output bits!

Half Adder

- Inputs a, b
- Outputs sum and carry out.
- Sum is the result of adding a and b.
- Carry out is the overflow bit.



Below is the truth table for the SUM output of a half adder.
What is the Boolean algebra function that will give us this truth table?

a	b	Sum
0	0	0
0	1	1
1	0	1
1	1	0

A. $a \text{ OR } b$

D. $a \text{ NOR } b$

B. $a \text{ XOR } b$

E. None of the above

C. $a \text{ AND } b$

Below is the truth table for the CARRY output of a half adder.
What is the Boolean algebra function that will give us this truth table?

a	b	Carry out
0	0	0
0	1	0
1	0	0
1	1	1

A. $a \text{ OR } b$

D. $a \text{ NOR } b$

B. $a \text{ XOR } b$

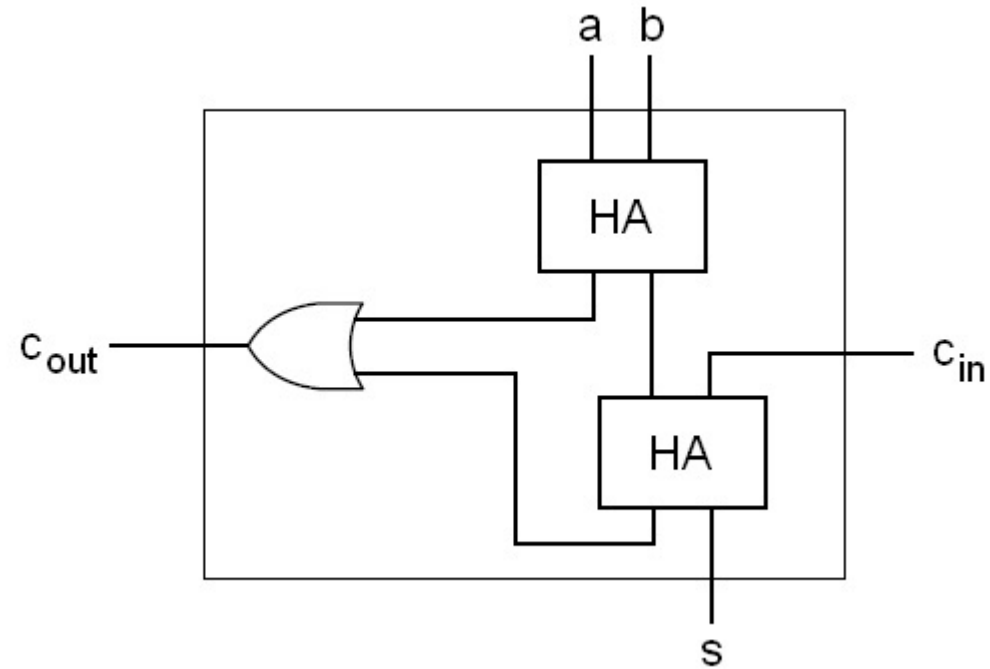
E. None of the above

C. $a \text{ AND } b$

Binary Addition with Arbitrary Number of Bits

- Just like regular, grade school addition
 - Make sure we carry a 1 to the next digit when needed
- Now we need to be able to account for the carry-in from the next least-significant bit
- Example: $7+5$

Full Adder from Half Adders



- Need carry-in, as well as carry-out

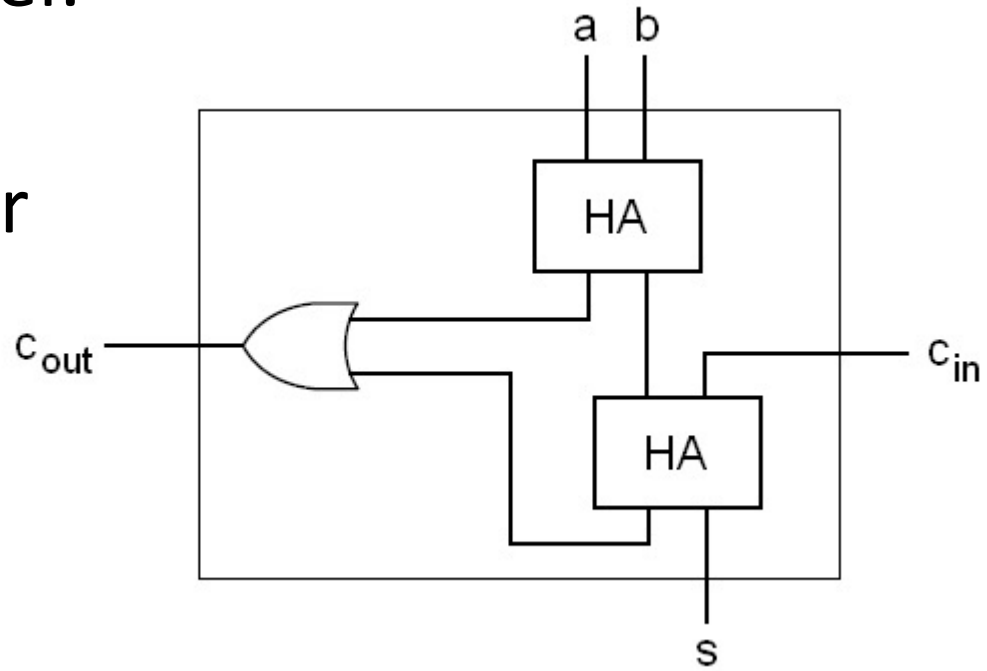
What if both half adders have carry-out?

A. We will get the wrong answer.

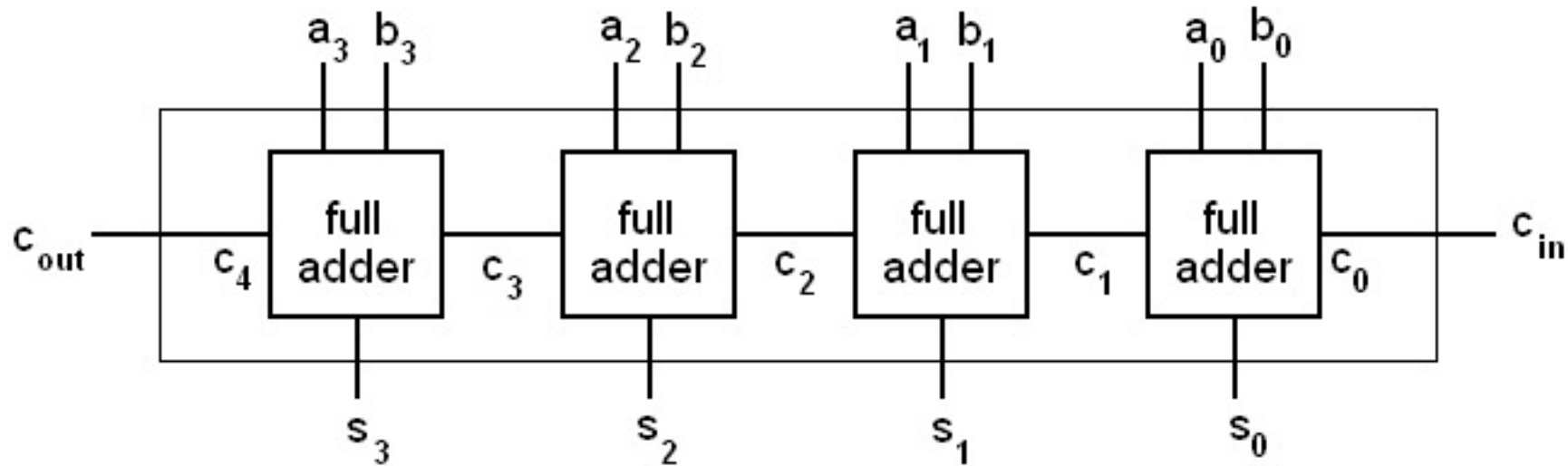
B. We will ignore it, the answer will still be correct.

C. That will never happen

D. None of the above

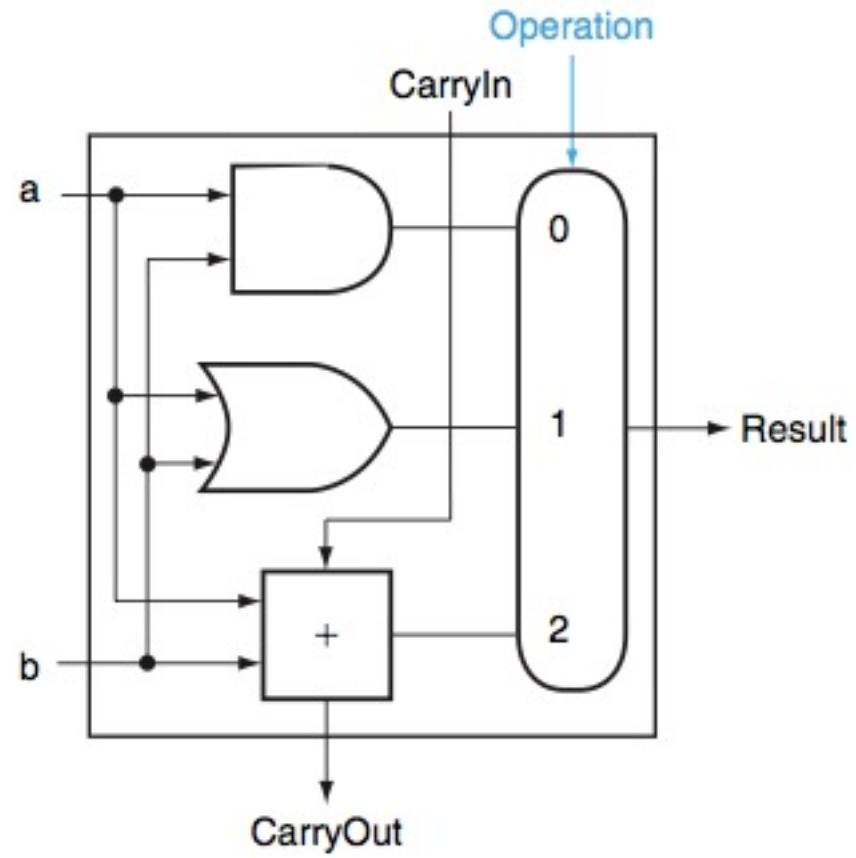


Ripple-Carry Adder



- Create adder for an arbitrary number of bits simply by connecting carry-out from adder $n-1$ to the carry-in for adder n
- Carry bit “ripples” up

1-bit ALU



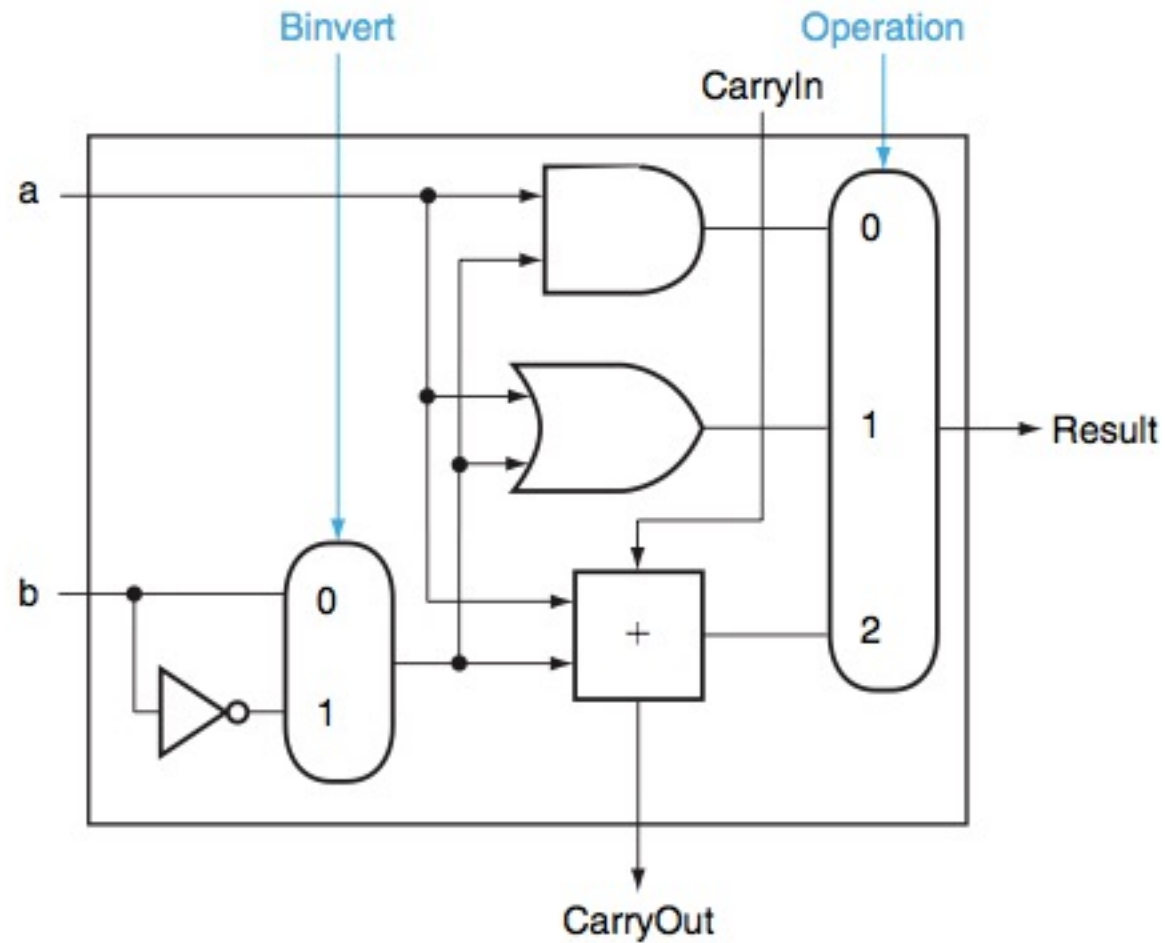
Subtraction: $a - b$

- Just add negative version of b !
- To negate operand, transform to two's complement
 - Invert each bit
 - Add one

We can use a NOT gate to invert the input. To add one to the input, we should

- A. Set the carry-in for the least significant bit to 1.
- B. Add a new “subtract” input that we set to 1 for subtraction.
- C. Do something else.

1-bit ALU with Subtraction

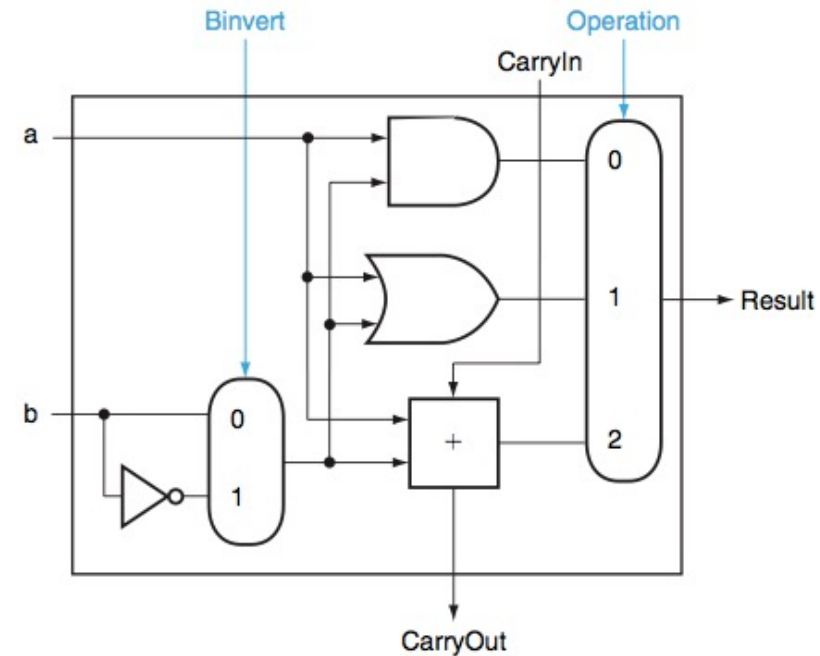


Adding NOR

- Want to add NOR functionality
- DeMorgan's Law
 - $\overline{(A+B)} = \bar{A} \bar{B}$

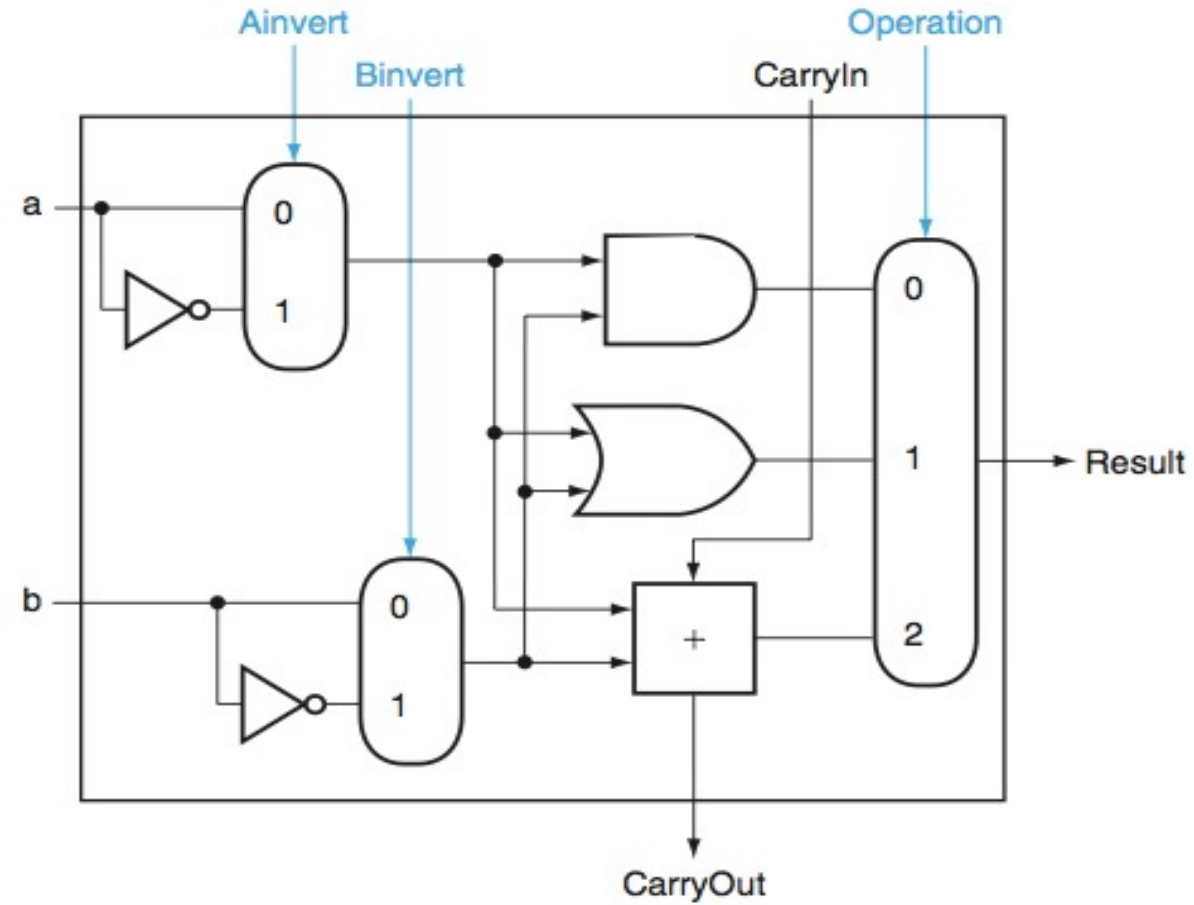
To add NOR to the ALU, we need to add

- A. Nothing
- B. The ability to invert A
- C. A NOR gate
- D. Something else



DeMorgan's Law
$$\overline{(A+B)} = \bar{A} \bar{B}$$

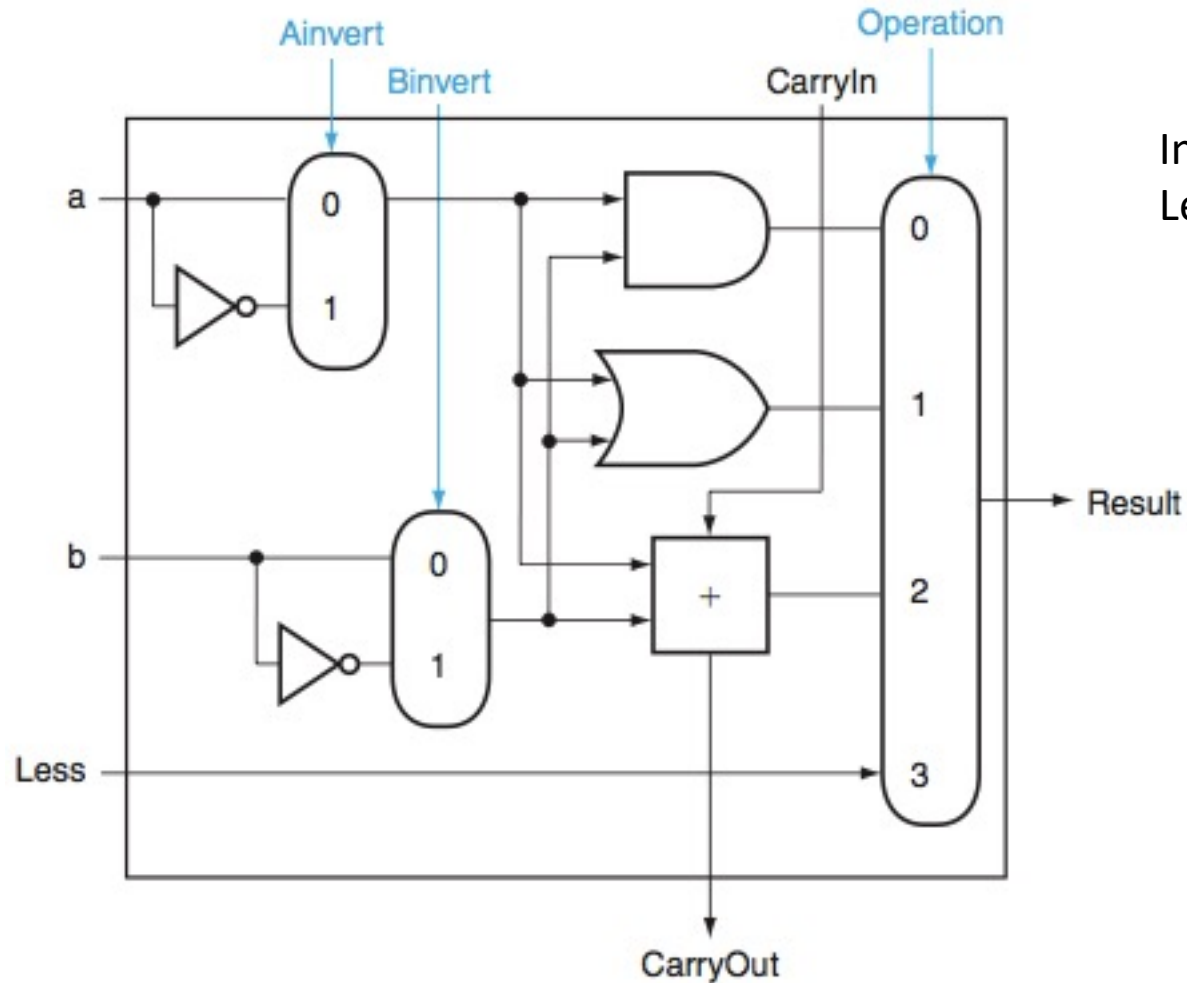
1-bit ALU with NOR



Adding slt

- `slt rd, rs, rt`
 - `rd = 1` if `rs < rt`, and `0` otherwise
- Only sets least significant bit
 - All other bits are `0`

1-bit ALU: Add new input for slt

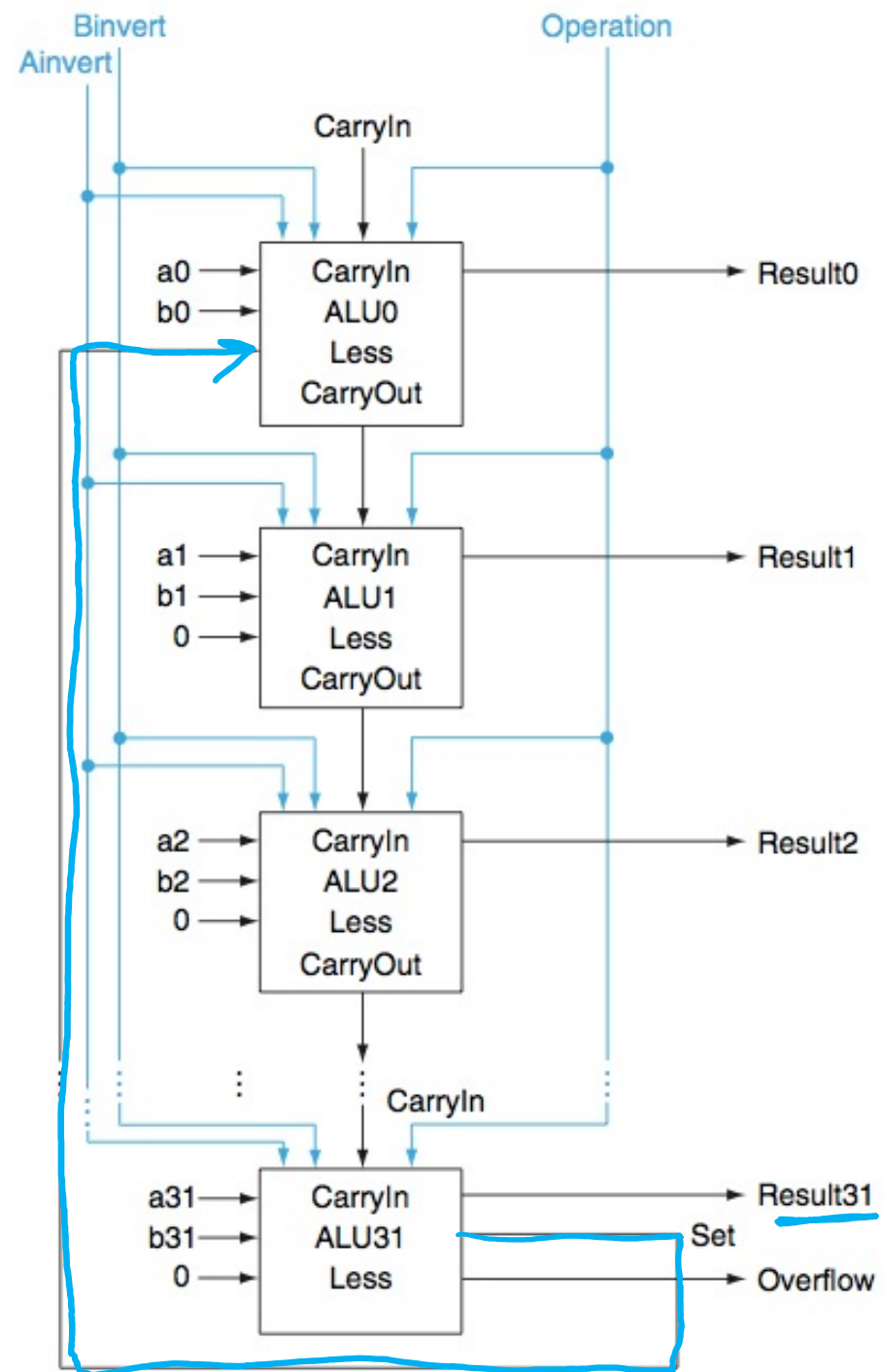


In all but the least significant bit,
Less will always be 0

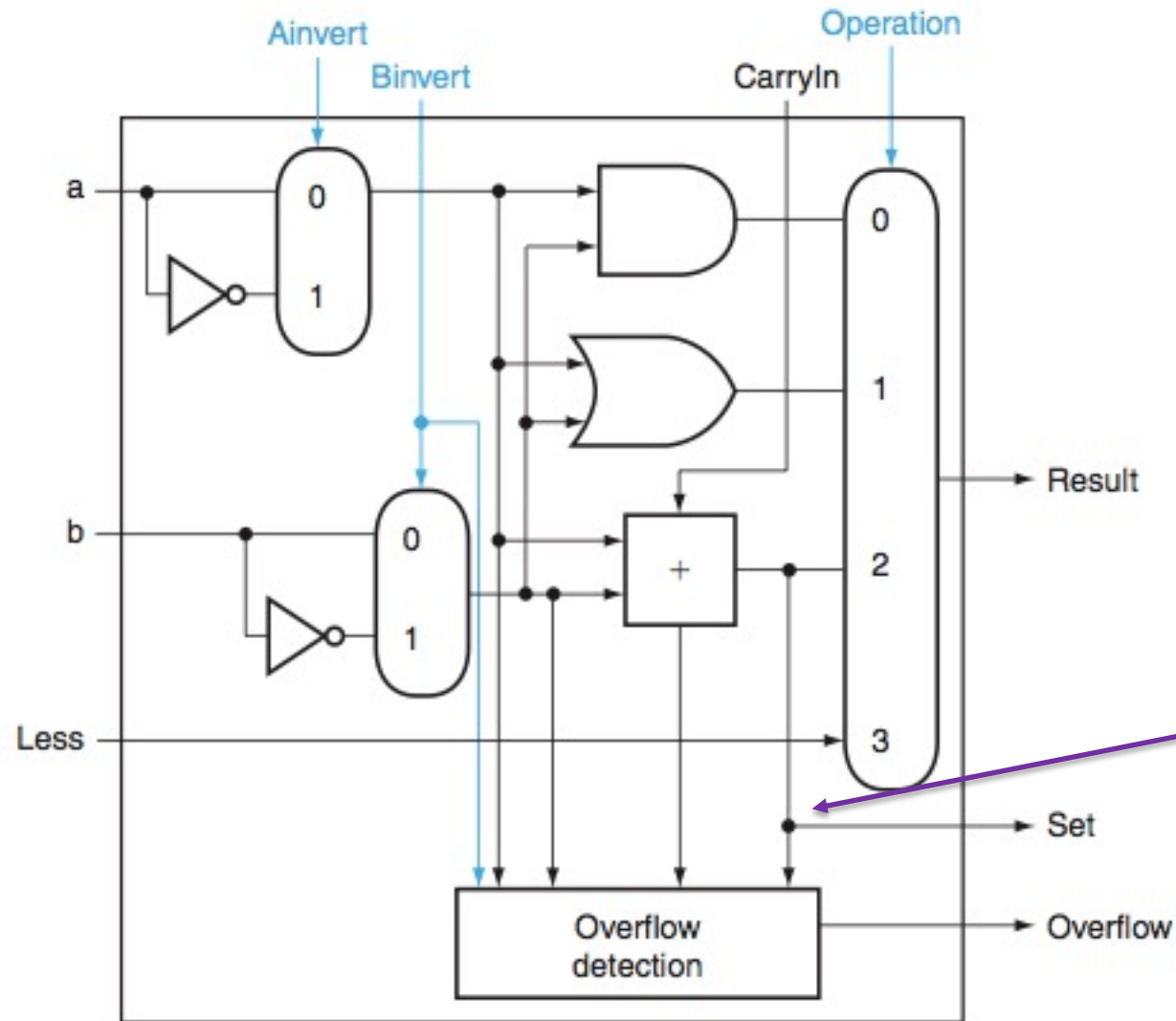
How do we tell if $a < b$?

- Subtract b from a
- If $a - b < 0$, then $a < b$
- We can check this by checking the most significant bit
 - $\text{MSB} = 1$, $a < b$

- Problem: Output is at Most Significant Bit, we need it at Least Significant Bit
- Solution: Special ALU for Most Significant Bit, with output for SLT
- Hook SET output into LESS input for Least Significant Bit



1-bit ALU for the Most Significant Bit



This doesn't
always work!
You'll fix it in
problem set 6

Reading

- Next lecture: Clocks, Latches and Flip flops
 - 3.6
- Problem set 5
 - Due Tonight
- Lab 4
 - Due Sunday