CSC 411

Computer Organization (Spring 2024)
Lecture 21: ALUs

Prof. Marco Alvarez, University of Rhode Island

Adder

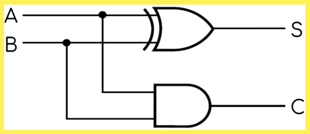
1-bit half-adder

- For now, lets ignore the carry-in (half-adder)
  - just add two bits and calculate the output/carry-out

A	B	C_{out}	S
0	0		
0	1		
1	0		
1	1		

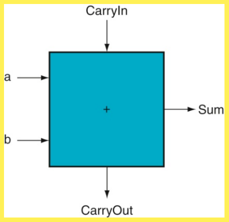
Write a boolean expression for C_{out}?

Write a boolean expression for Sum?



1-bit adder

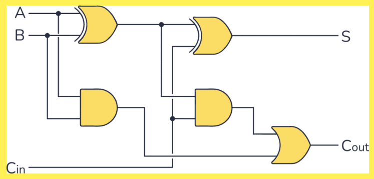
- Now considering carry-in



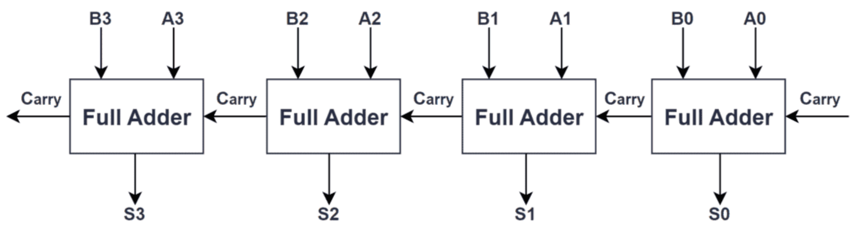
A	B	C_{in}	C_{out}	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Write a boolean expression for C_{out}?

Write a boolean expression for Sum?

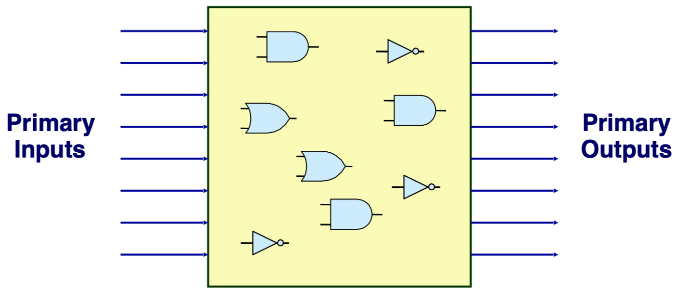


4-bit adder



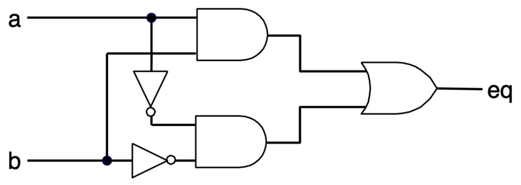
Delays in combinational circuits

- Circuit continually responds to input changes
- Outputs are calculated after some **delay**



Example: bit equality

- What is the delay?
  - assume NOT takes 1.1 units of time, AND takes 3.9, and OR takes 4.5



The delay of a circuit is determined by its "critical path", the path between an input and output with the maximum delay

Image credit: CSC 252: Computer Organization: Lecture 11, University of Rochester

Example: 64-bit equality

- What is the total delay?

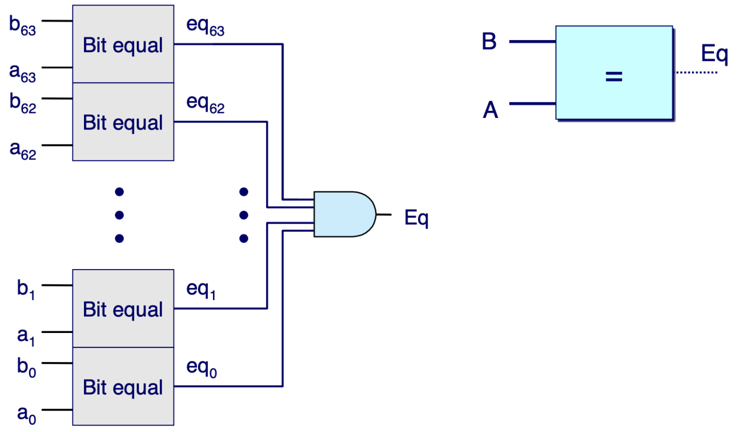
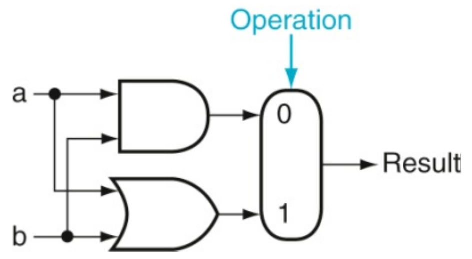


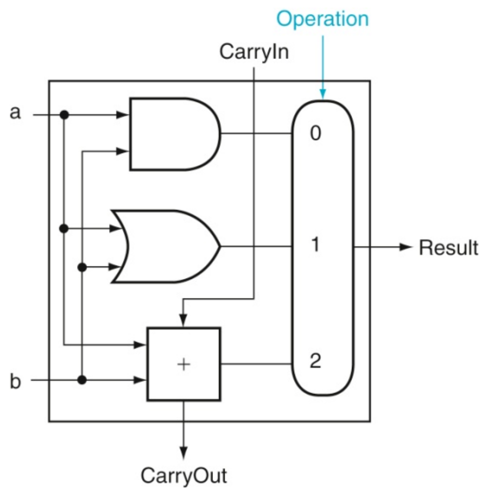
Image credit: CSC 252: Computer Organization: Lecture 11, University of Rochester

ALUs

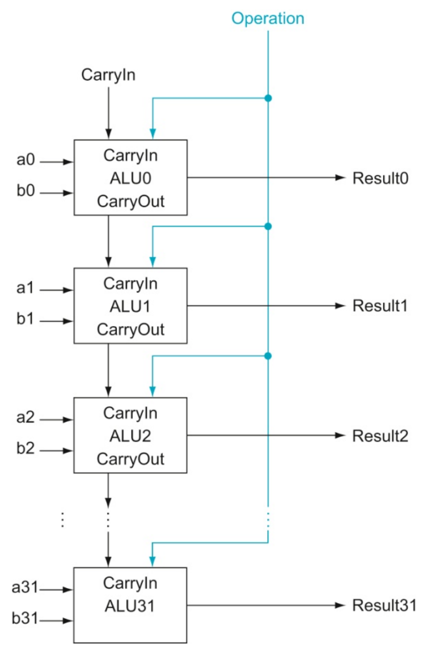
1-bit ALU that performs AND and OR



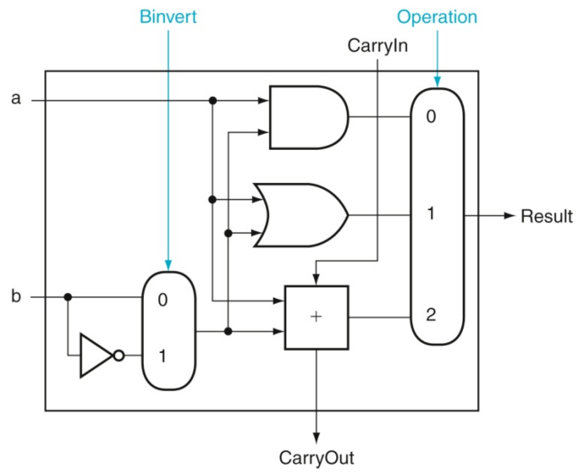
1-bit ALU (AND, OR, Add)



32-bit ALU



1-bit ALU (AND, OR, Add, Sub)



Subtraction: Invert and set CarryIn = 1

1-bit ALU (AND, OR, Add, Sub, NOR)

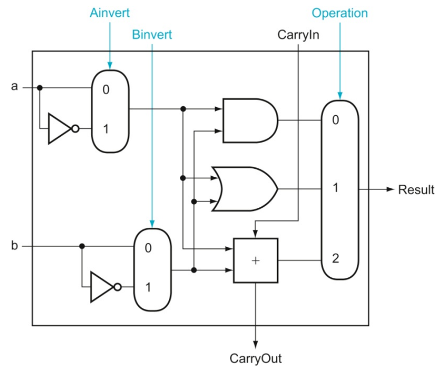
We can also add a NOR function.

Instead of adding a separate gate for NOR, we can reuse the hardware.

The insight comes from:

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

What would be the control bits for NOR?

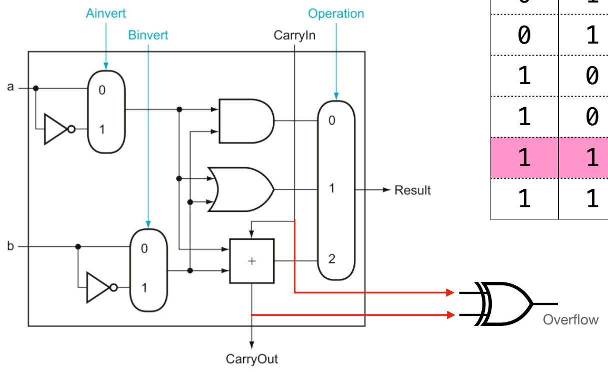


ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract

Overflow detection

C_{in} of MSB != C_{out} of MSB

A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

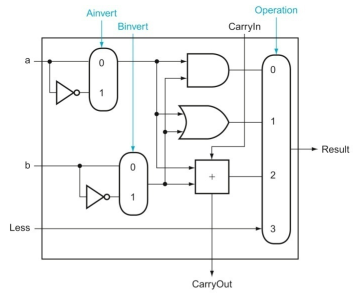


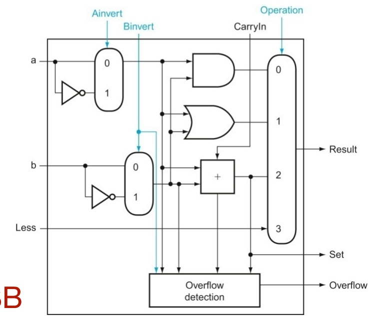
Adding SLT

Set less than instruction (1 if rs1 < rs2, and 0 otherwise)

Output should be all zeros, with the least significant bit set according to the comparison.

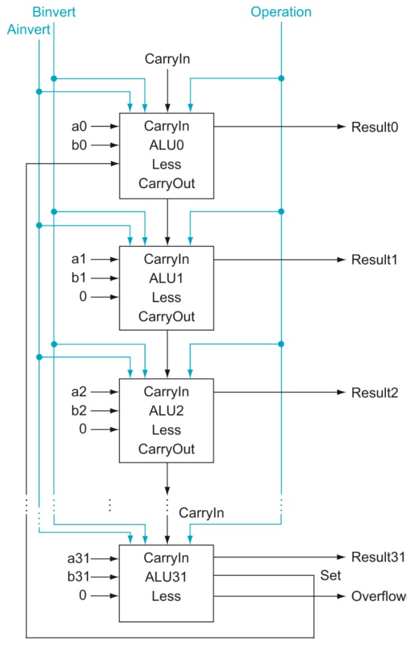
If the ALU performs a-b and we select 3 in the multiplexor, then Result = 0…001 if a<b, and Result = 0…000 otherwise.





MSB

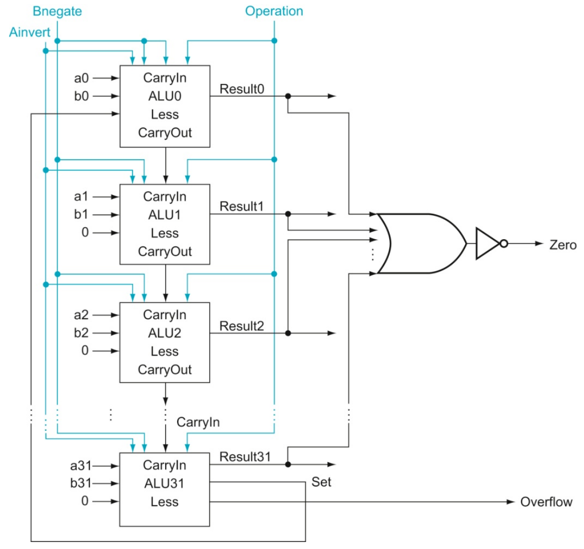
32-bit ALU



This is a 32 bit "ripple carry" ALU, connecting 32 1-bit ALUs sequentially.

The **Less** inputs are 0 except for the LSB, which is connected to the **Set** output of the MSB.

Adding a zero detector

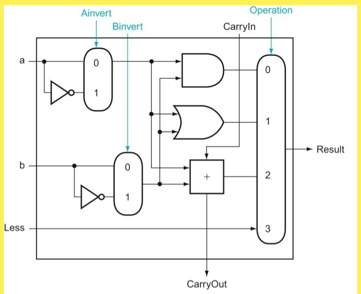


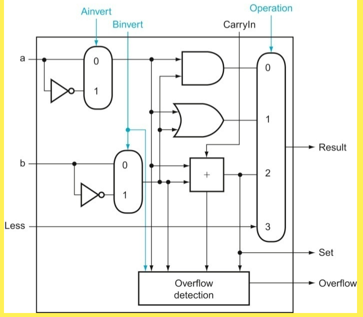
Checking if two inputs contain the same values (as in **beq**) can be done by performing **a-b** and checking the result is **all zeros**.

32-bit ALU

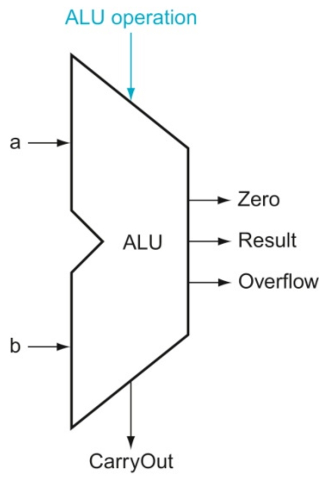
A	Ainv	Binv	Op
and			
or			
add			
sub			
nor			
nand			
slt			
beq			

Control lines?





32-bit ALU



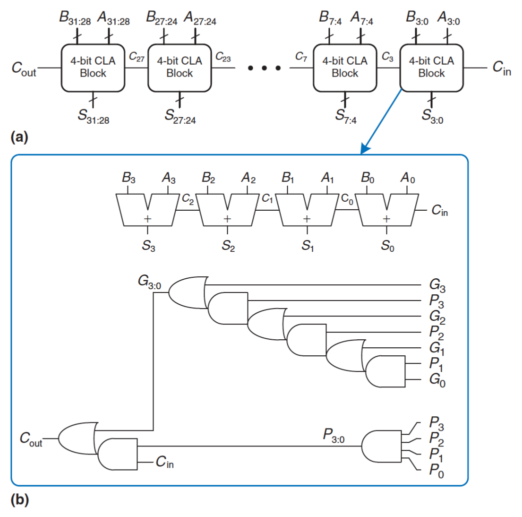
Ripple carry

- Total delay?
  - the time to go through 32*d gates!
- Any logic equation can be expressed as the sum of products
  - technically, it should be possible to compute the result by going through only 2 gates
  - **caveat**: need many parallel gates and each gate may have a very large number of inputs
- Can we find a compromise?
  - moderate number of gates and inputs to each gate
  - moderate number of sequential gates traversed

32-bit carry look-ahead adder

Eight 4-bit ALUs using carry lookahead to form a 32-bit adder

Takes less sequential steps than a ripple carry adder



$B_{31:28}$ $A_{31:28}$ $B_{27:24}$ $A_{27:24}$ $B_{7:4}$ $A_{7:4}$ $B_{3:0}$ $A_{3:0}$
C_{out}
4-bit CLA Block
C_{27}
C_{23}
C_7
C_3
C_{in}
$S_{31:28}$ $S_{27:24}$ $S_{7:4}$ $S_{3:0}$
(a)
B_3 A_3 B_2 A_2 B_1 A_1 B_0 A_0
C_2 C_1 C_0
+
S_3 S_2 S_1 S_0
$G_{3:0}$
G_3 P_3 G_2 P_2 G_1 P_1 G_0
$P_{3:0}$
P_3 P_2 P_1 P_0
(b)