

CSC 411

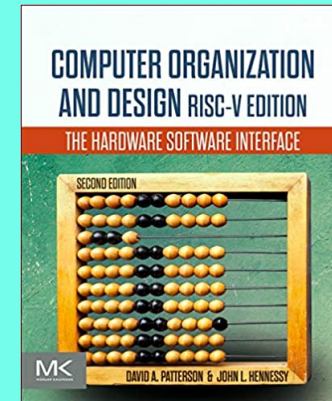
Computer Organization (Fall 2024)
Lecture 19: Adders and ALUs

Prof. Marco Alvarez, University of Rhode Island

Disclaimer

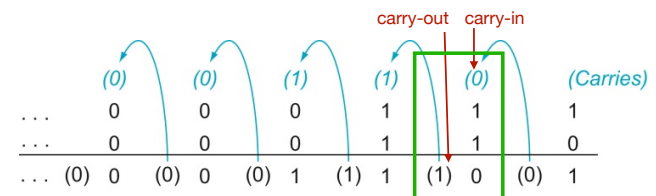
Some figures and slides are adapted from:

Computer Organization and Design (Patterson and Hennessy)
The Hardware/Software Interface



Adders

1-bit half-adder



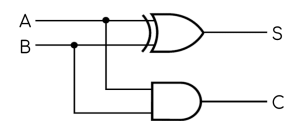
► For now, let's ignore the carry-in bit

- add two bits (A, B) and output (S) and carry-out (C)

Write a boolean expression for C?

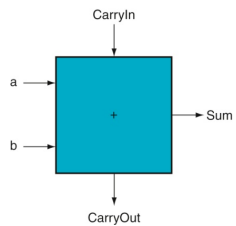
Write a boolean expression for S?

A	B	C	S
0	0		
0	1		
1	0		
1	1		

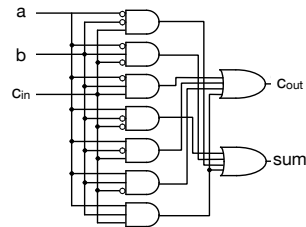


1-bit adder

- Now consider the carry-in bit



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

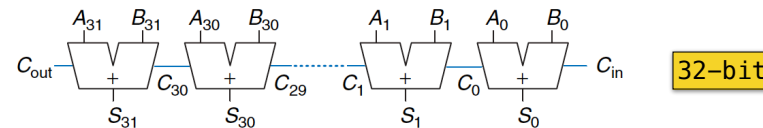
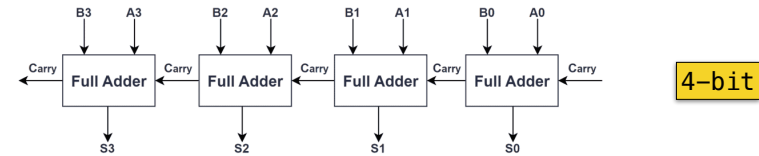


sum of products for 1-bit addition
(circuit can be simplified by using boolean algebra)

Write a boolean expression for C_{out} ?

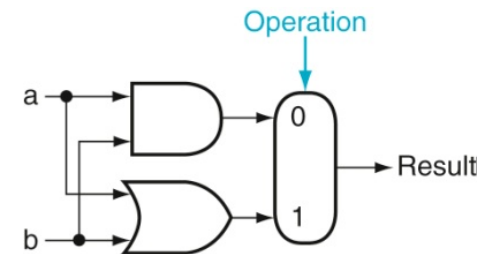
Write a boolean expression for S ?

Ripple carry adders

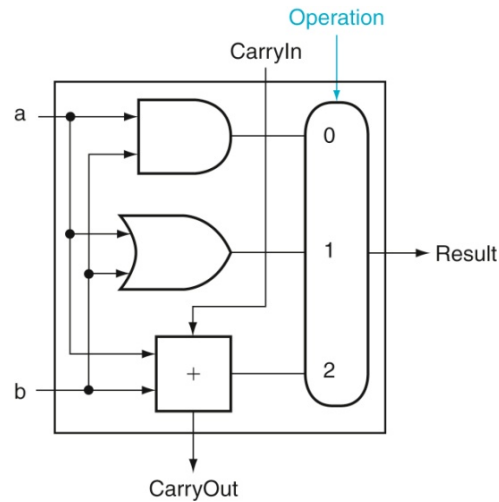


ALUs

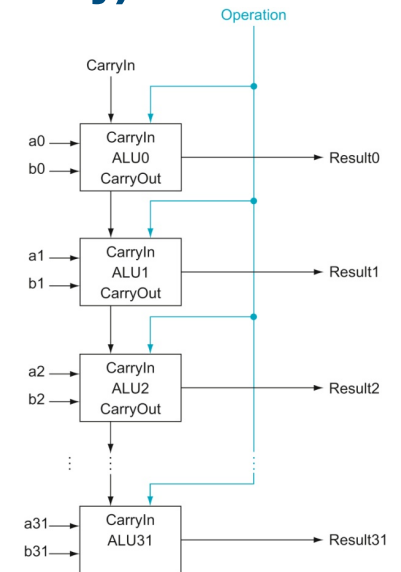
1-bit ALU that performs AND, OR



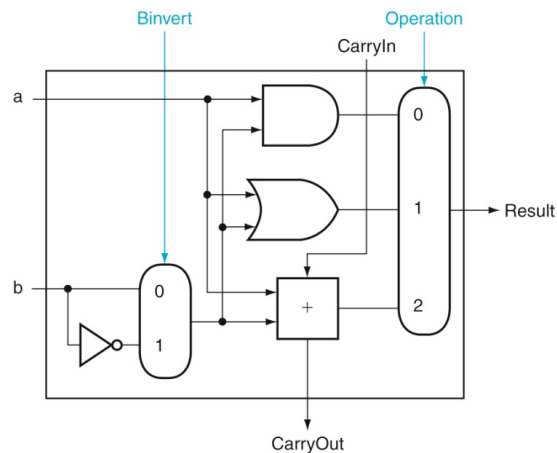
1-bit ALU that performs AND, OR, ADD



32-bit ALU (ripple carry)

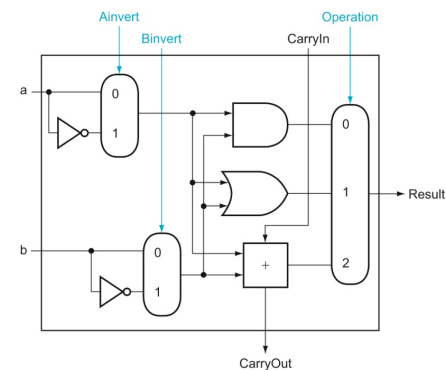


1-bit ALU that performs AND, OR, ADD, SUB



By selecting **Binvert = 1** and setting **CarryIn = 1** in the least significant bit of the ALU, we get the two's complement subtraction $a - b$ instead of the addition $a + b$

1-bit ALU that performs AND, OR, ADD, SUB, NOR



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

Assuming control lines are *Ainvert*, *Binvert*, *Operation*, how would you encode the NOR operation?

By selecting **Ainvert = 1** and **Binvert = 1**, we get $a \text{ NOR } b$ instead of $a \text{ AND } b$. The insight comes from the De Morgan's laws: $\overline{a + b} = \overline{a} \overline{b}$.

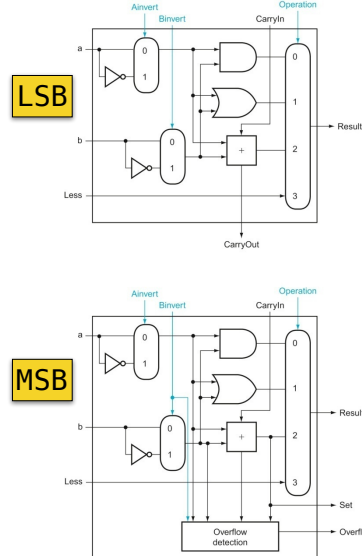
Adding **slt** to 32-bit ALU

- **slt** instruction produces 1 if $rs1 < rs2$, and 0 otherwise
- All outputs should be 0
 - except for the LSB, which can be 1 or 0 depending on the comparison

- Can use an input **Less equal to zero** for all ALUs

- except the LSB's which receives this input from the Sum value of the MSB's adder (see next slide)
- insight comes from the formula below, if $a - b < 0$ then $a < b$
 - argument only works if the result does not overflow

$$(a - b < 0) \Rightarrow (a - b) + b < 0 + b \Rightarrow a < b$$



Overflow detection

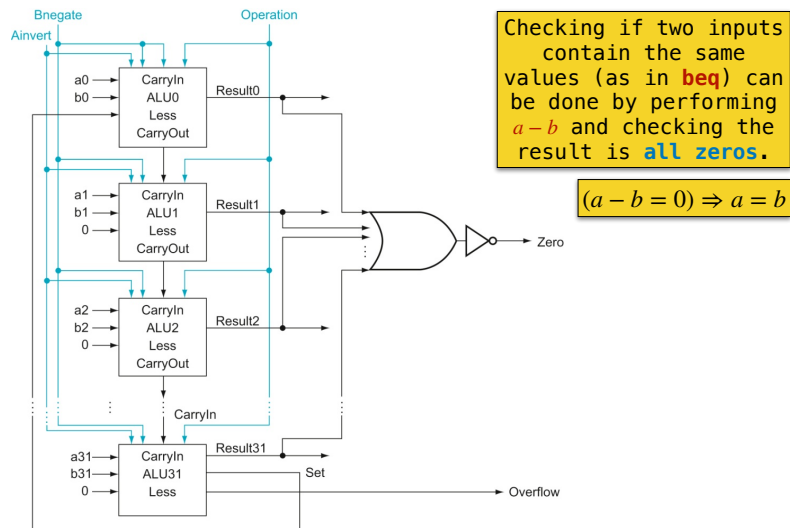
- A simple check for overflow during addition

- compare the carry-in of the most significant bit (MSB) with the carry-out of the MSB
- if both are different, overflow has occurred



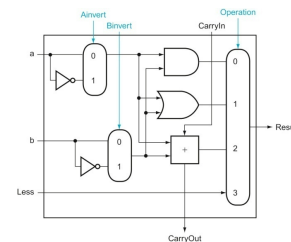
A	B	C _{in}	C _{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

32-bit ALU with a zero detector



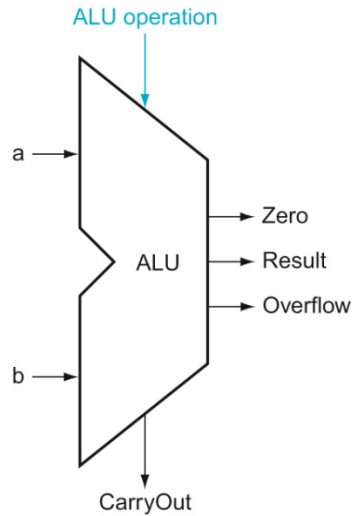
$$(a - b = 0) \Rightarrow a = b$$

Complete the control lines



	Ainvert	Binvert	Op
and			
or			
add			
sub			
nor			
nand			
slt			
beq			

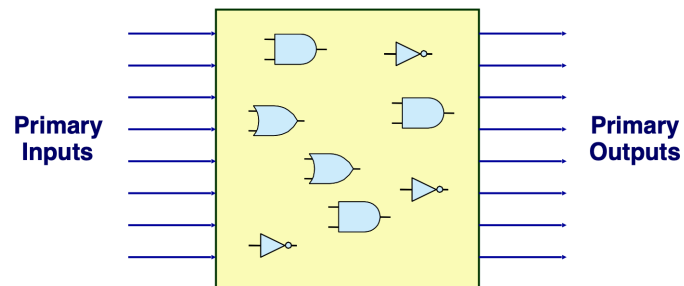
Symbol used to represent the ALU



Delays

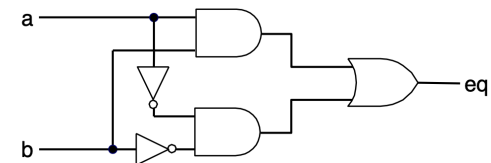
Delays in combinational circuits

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



Example: 1-bit equality

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



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

Example: 61-bit equality

- What is the total delay?

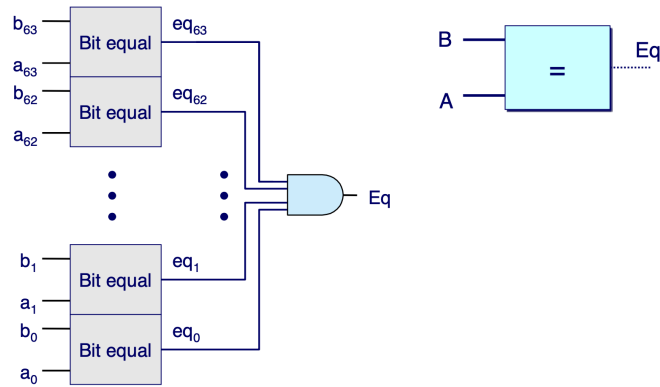


Image credit: CSC 252: Computer Organization: Lecture 11, University of Rochester

Delays in ripple carry ALUs

- Carry bit propagates from the LSB to the MSB sequentially
 - total delay is proportional to the number of bits (e.g., $b = 32$) and the delay of each full 1-bit ALU cell (d), and can be expressed $b * d$
 - each cell introduces a certain amount of delay (d), and this delay accumulates as the carry bit ripples through the chain
 - technically, it should be possible to compute the result by going through only 2 gates
 - remember any logic equation can be expressed as the sum of products
 - however, it may need many parallel gates and each gate may have a very large number of inputs
- To address this limitation, other carry propagation schemes are often used
 - e.g., carry-lookahead or carry-select
 - these more advanced architectures can achieve a logarithmic or constant-time delay, independent of the word size, at the cost of increased circuit complexity

16-bit ALU using carry-lookahead

- 4-bit ALUs connected with a carry-lookahead unit
 - requires $2+2+1$ gate delays
- A ripple carry ALU would take $16 * d$ gate delays

