CSC 411

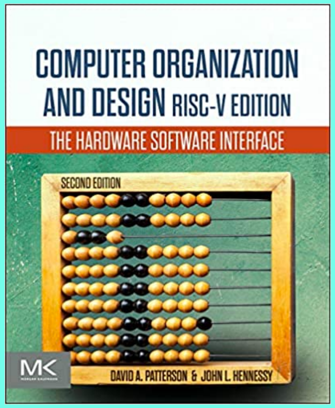
Computer Organization (Spring 2024)
Lecture 21: Adders and ALUs

Prof. Marco Alvarez, University of Rhode Island

Disclaimer

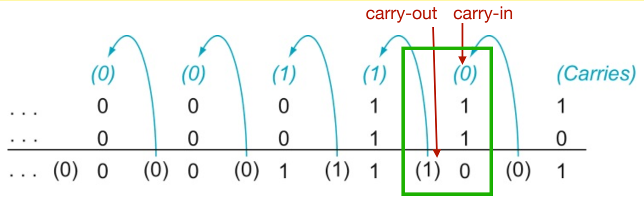
Some figures and slides are adapted from:

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


COMPUTER ORGANIZATION AND DESIGN RISC-V EDITION
THE HARDWARE SOFTWARE INTERFACE
SECOND EDITION
MK
DAVID A. PATTERSON & JOHN L. HENNESSY

Adders

1-bit half-adder

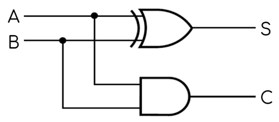


- For now, lets ignore the carry-in bit
  - add two bits (A, B) and output (S) and carry-out (C)

Write a boolean expression for C?

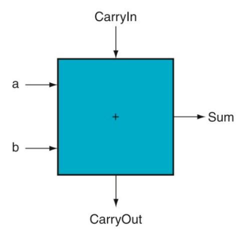
Write a boolean expression for S?

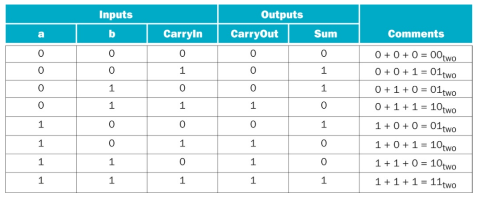
A	B	C	S
0	0		
0	1		
1	0		
1	1		



1-bit adder

- Now consider the carry-in bit

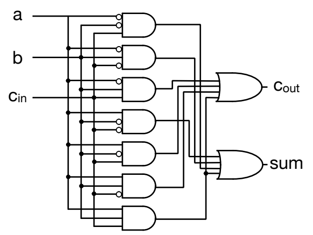




Inputs			Outputs		
a	b	CarryIn	CarryOut	Sum	Comments
0	0	0	0	0	$0 + 0 + 0 = 00_{two}$
0	0	1	0	1	$0 + 0 + 1 = 01_{two}$
0	1	0	0	1	$0 + 1 + 0 = 01_{two}$
0	1	1	1	0	$0 + 1 + 1 = 10_{two}$
1	0	0	0	1	$1 + 0 + 0 = 01_{two}$
1	0	1	1	0	$1 + 0 + 1 = 10_{two}$
1	1	0	1	0	$1 + 1 + 0 = 10_{two}$
1	1	1	1	1	$1 + 1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

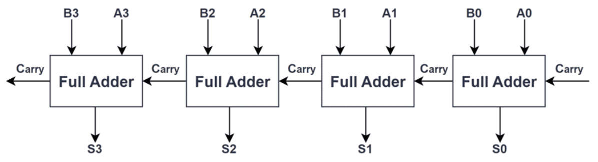
Write a boolean expression for C_{out}?

Write a boolean expression for S?

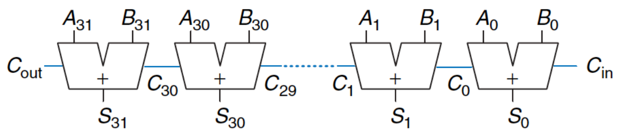


sum of products for 1-bit addition
(circuit can be simplified by using boolean algebra)

Ripple carry adders



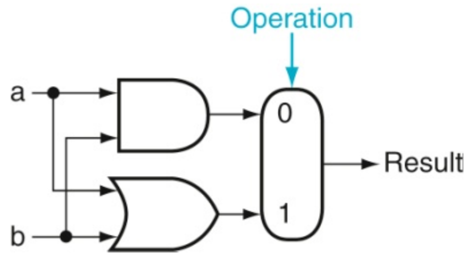
4-bit



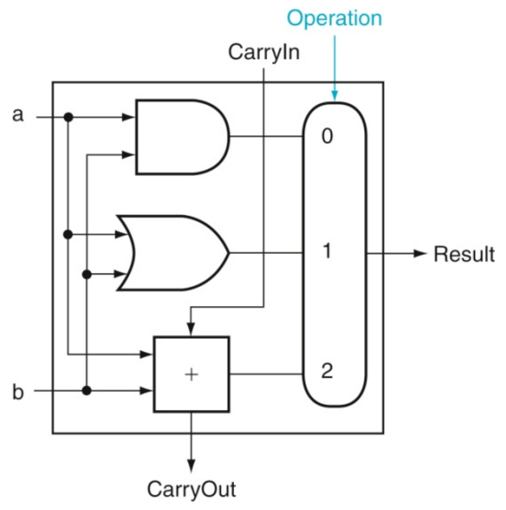
32-bit

ALUs

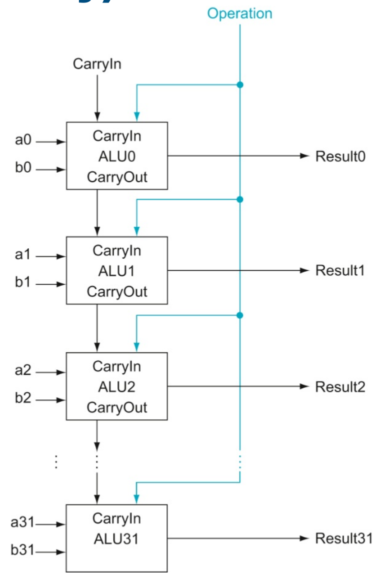
1-bit ALU that performs AND, OR



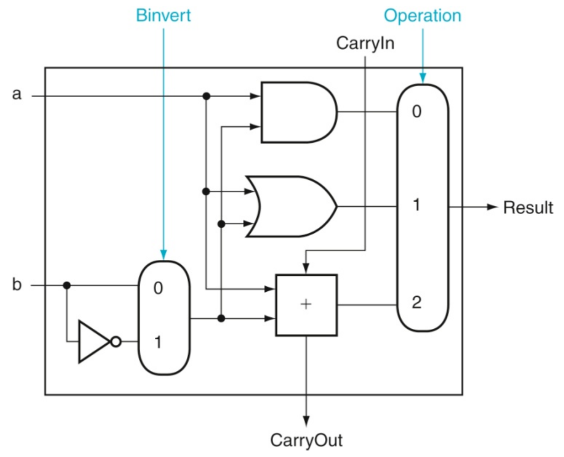
1-bit ALU that performs AND, OR, ADD



32-bit ALU (ripple carry)

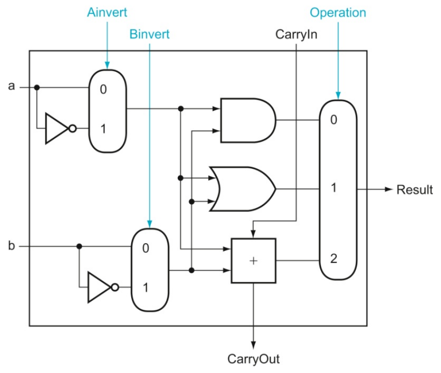


1-bit ALU that performs AND, OR, ADD, SUB



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?

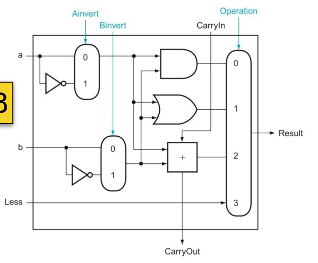
By selecting **Ainvert = 1** and **Binvert = 1**, we get $a\ NOR\ b$ instead of $a\ AND\ b$. The insight comes from the De Morgan's laws: $\overline{a+b} = \bar{a}\bar{b}$.

Adding `slt` to 32-bit ALU

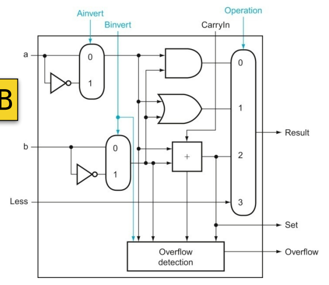
- **`slt`** instruction produces 1 if rs1 < rs2, and 0 otherwise
- All outputs should be 0
  - except for the LSB, which can be 1 or 0 depending on the comparison
- Can use an input **Less** equal to zero for all ALUs
  - except the LSB's which receives this input from the Sum value of the MSB's adder (see next slide)
  - insight comes from the formula below, if $a - b < 0$ then $a < b$
    - <u>argument only works if the result does not overflow</u>

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

LSB

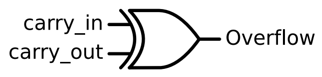


MSB



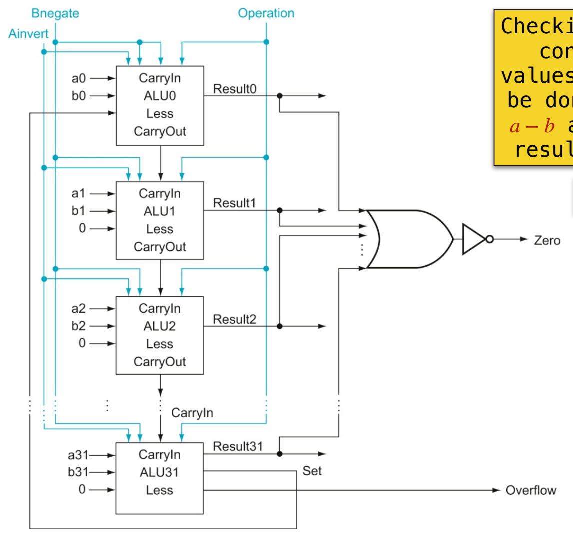
Overflow detection

- 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

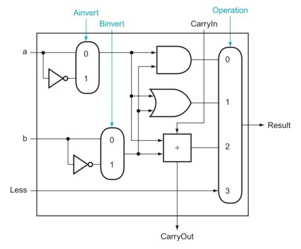
32-bit ALU with a zero detector

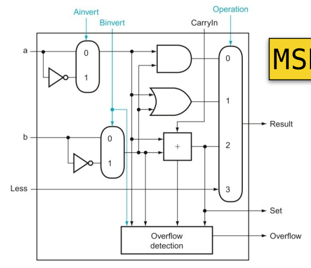


```
Checking if two inputs
   contain the same
values (as in beq) can
 be done by performing
 a − b and checking the
 result is all zeros.
```

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

Complete the control lines

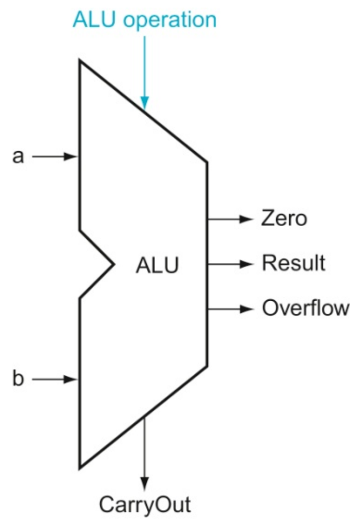




MSB

	Ainvert	Binvert	Op
and			
or			
add			
sub			
nor			
nand			
slt			
beq			

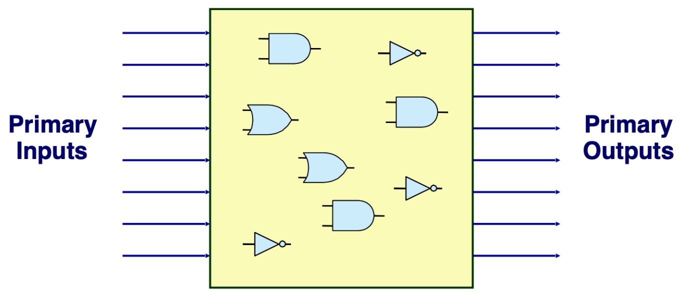
Symbol used to represent the ALU



Delays

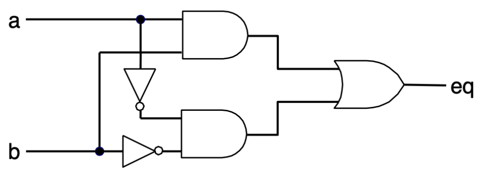
Delays in combinational circuits

- 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

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

Example: 61-bit equality

- What is the total delay?

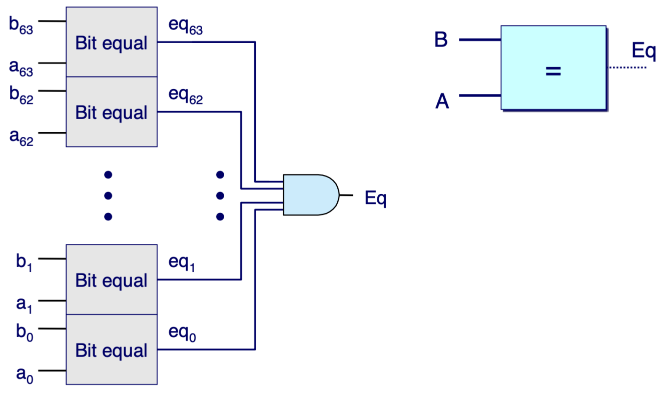


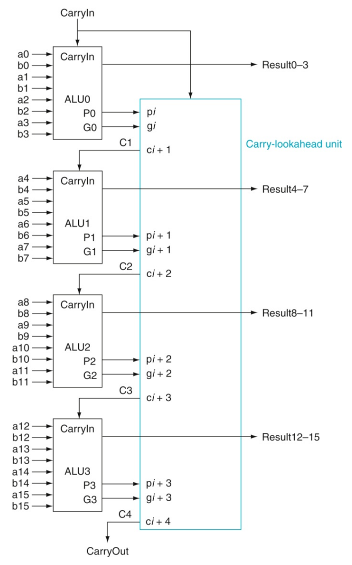
Image credit: CSC 252: Computer Organization: Lecture 11, University of Rochester

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., 32) and the delay of each full 1-bit ALU cell (d), and can be expressed $32 * d$
  - 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 looked

- 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



CarryIn
a0 b0 a1 b1 a2 b2 a3 b3
CarryIn
ALU0
P0
G0
Result0–3
pi
gi
C1
ci + 1
Carry-lookahead unit
a4 b4 a5 b5 a6 b6 a7 b7
CarryIn
ALU1
P1
G1
Result4–7
pi + 1
gi + 1
C2
ci + 2
a8 b8 a9 b9 a10 b10 a11 b11
CarryIn
ALU2
P2
G2
Result8–11
pi + 2
gi + 2
C3
ci + 3
a12 b12 a13 b13 a14 b14 a15 b15
CarryIn
ALU3
P3
G3
Result12–15
pi + 3
gi + 3
C4
ci + 4
CarryOut