

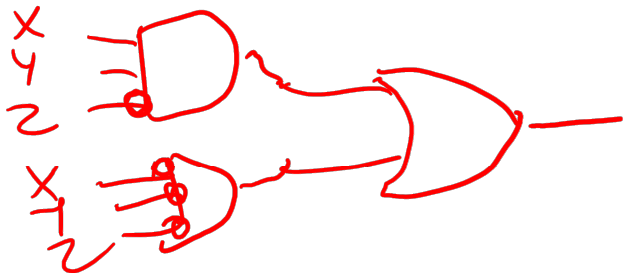
# Lecture 13: ALUs, Adders

- HW 5 due 2/25 Tue used / 2/26

• Today's topics:

- ALU
- Carry-lookahead adder

$$x \cdot y \cdot \bar{z} + \bar{x} \cdot y \cdot \bar{z}$$



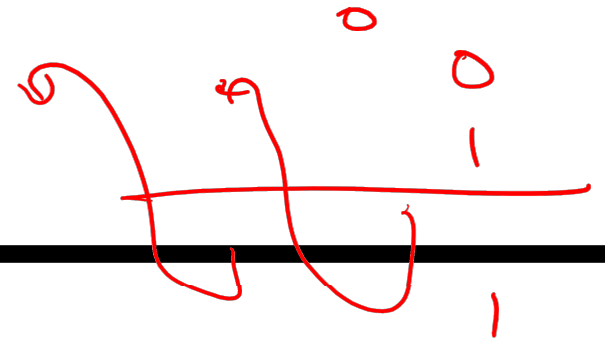
inp			out	
X	Y	Z	A	B
0	0	0	1	0
0	0	1	1	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

8 rows

Sum of products equation

# Adder Algorithm

32b Add  
a + b



Carryin:

a bits:	1	0	0	1
b bits:	0	1	0	1
Sum	1	1	1	0
Carry	0	0	0	1

Equations:

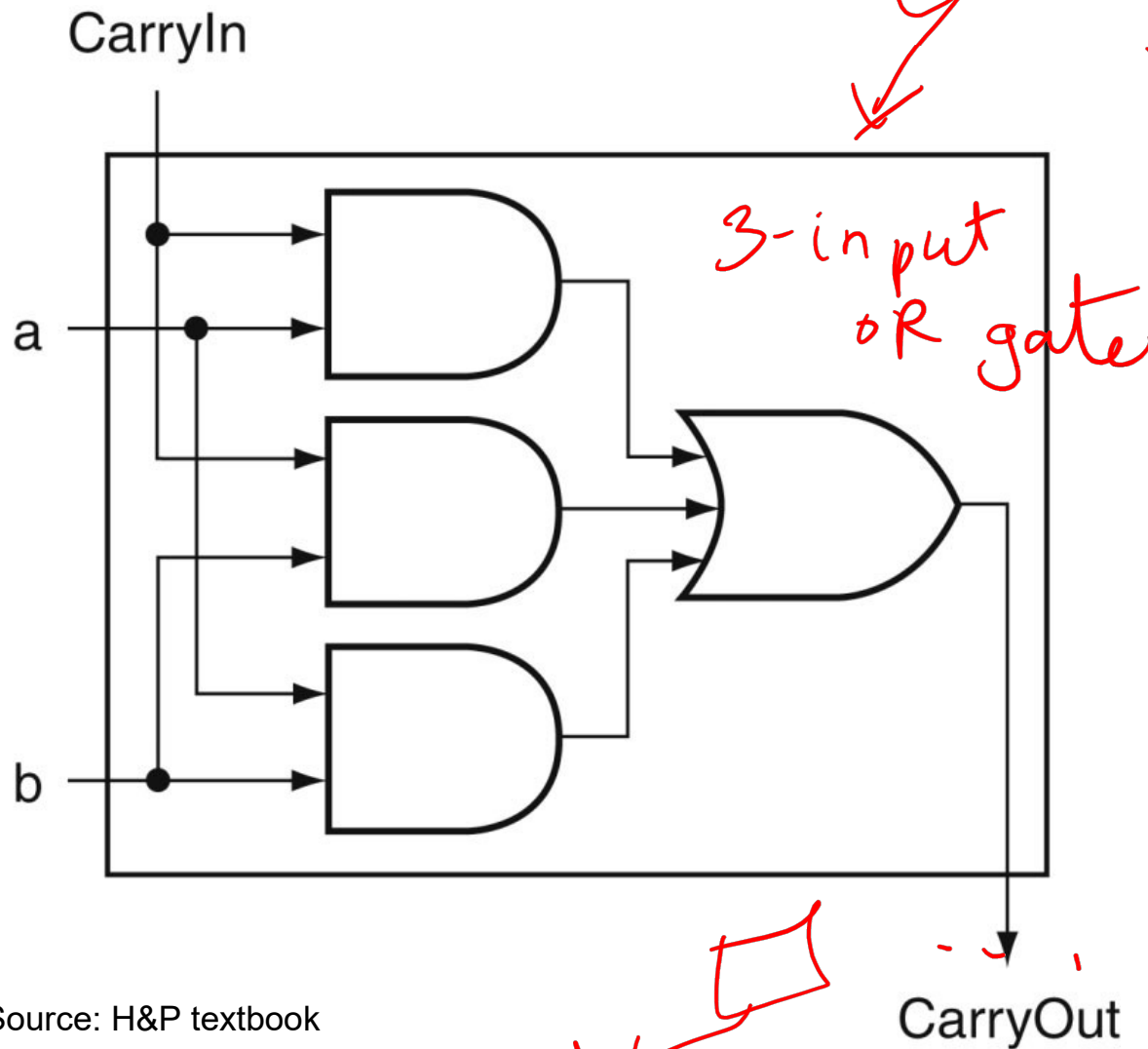
$$\begin{aligned} \text{Sum} = & \text{Cin} \cdot \bar{A} \cdot \bar{B} + \\ & B \cdot \bar{\text{Cin}} \cdot \bar{A} + \\ & A \cdot \bar{\text{Cin}} \cdot B + \\ & A \cdot B \cdot \text{Cin} \end{aligned}$$

Truth Table for the above operations:

A	B	Cin	Sum	Cout
0	0	0	0	0
<u>0</u>	<u>0</u>	<u>1</u>	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\begin{aligned} \text{Cout} = & A \cdot B \cdot \text{Cin} + \\ & A \cdot B \cdot \bar{\text{Cin}} + \\ & A \cdot \text{Cin} \cdot \bar{B} + \\ & B \cdot \text{Cin} \cdot \bar{A} \\ = & A \cdot B + \\ & A \cdot \text{Cin} + \\ & B \cdot \text{Cin} \end{aligned}$$

# Carry Out Logic

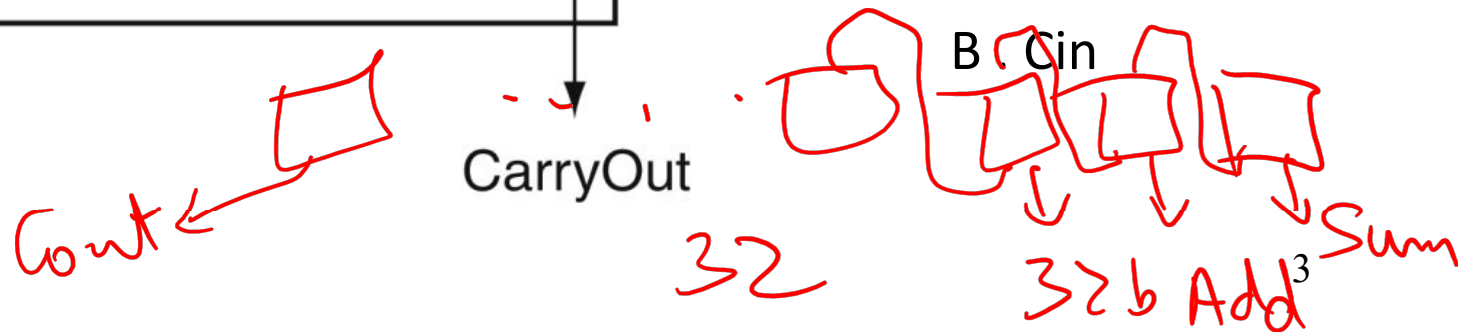


Source: H&P textbook

Equations:

$$\begin{aligned} \text{Sum} = & \text{Cin} \cdot \bar{A} \cdot \bar{B} + \\ & B \cdot \bar{\text{Cin}} \cdot \bar{A} + \\ & A \cdot \bar{\text{Cin}} \cdot B + \\ & A \cdot B \cdot \text{Cin} \end{aligned}$$

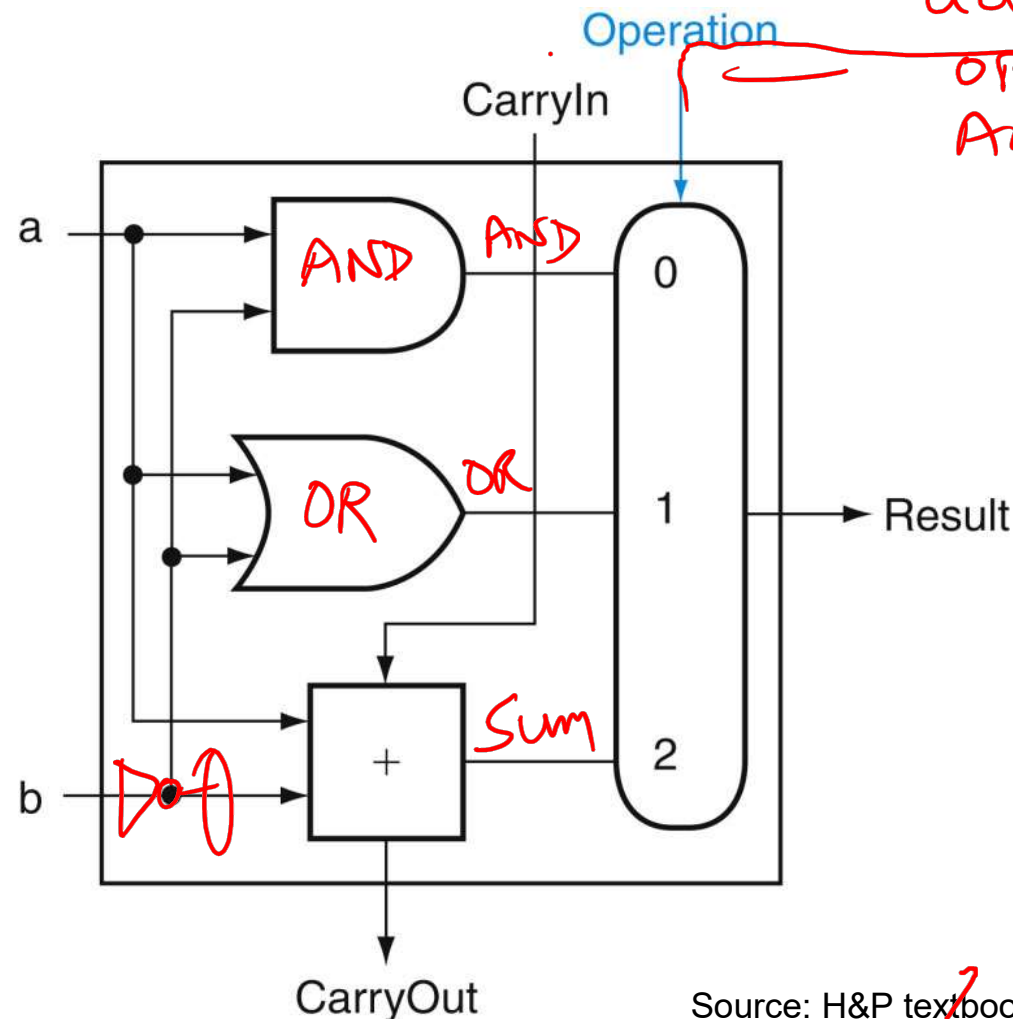
$$\begin{aligned} \text{Cout} = & A \cdot B \cdot \text{Cin} + \\ & A \cdot B \cdot \bar{\text{Cin}} + \\ & A \cdot \text{Cin} \cdot \bar{B} + \\ & B \cdot \text{Cin} \cdot \bar{A} \\ = & A \cdot B + \\ & A \cdot \text{Cin} + \\ & B \cdot \text{Cin} \end{aligned}$$



# → Arithmet & Logical Unit

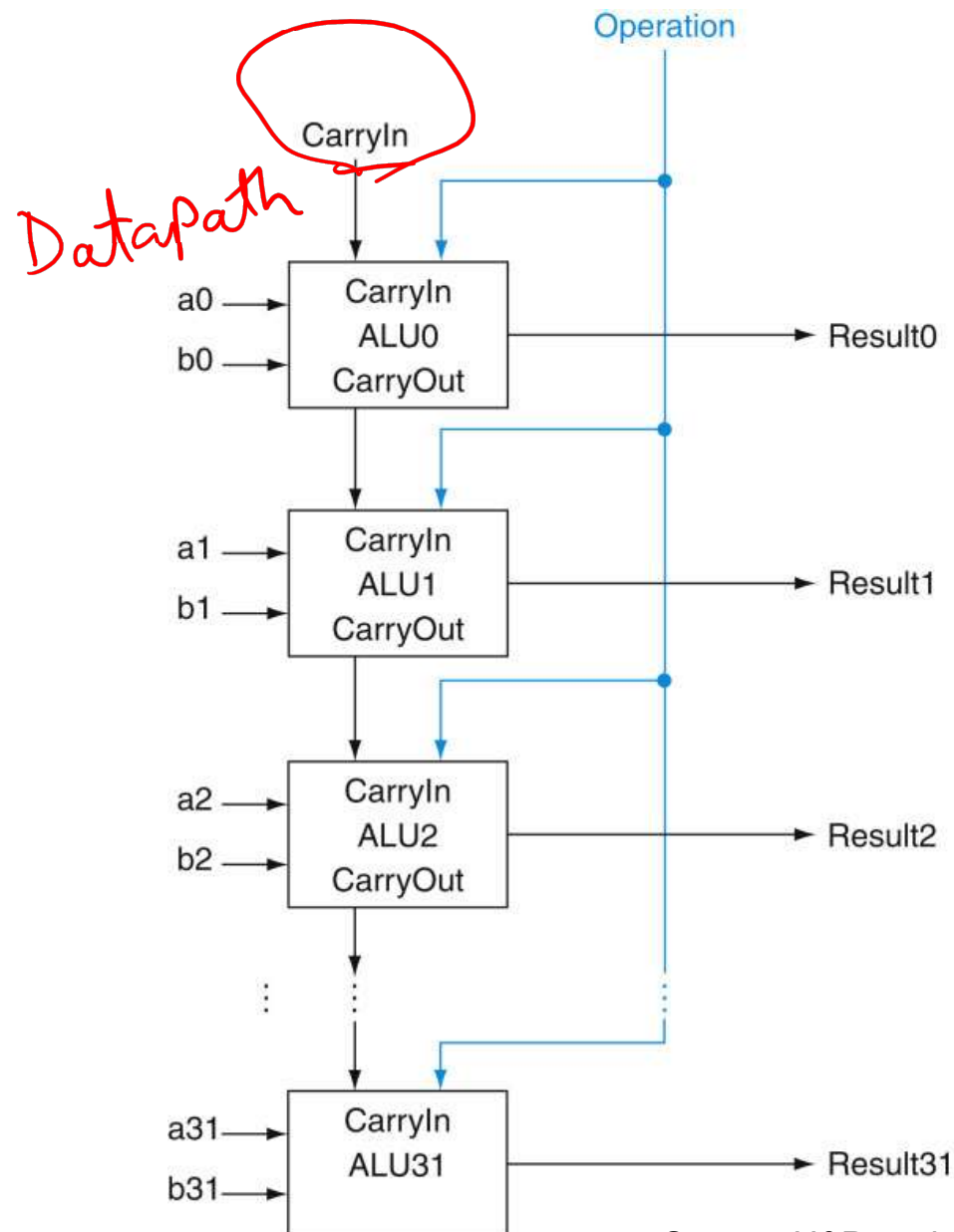
## 1-Bit ALU with Add, Or, And

- Multiplexor selects between Add, Or, And operations



# 32-bit Ripple Carry Adder

1-bit ALUs are connected  
“in series” with the  
carry-out of 1 box  
going into the carry-in  
of the next box



Source: H&P textbook

# Incorporating Subtraction

$b \Rightarrow -b$   
 $\hookrightarrow$  inv all bits, +1  $\rightarrow -b$

SUB

32b inst a - b

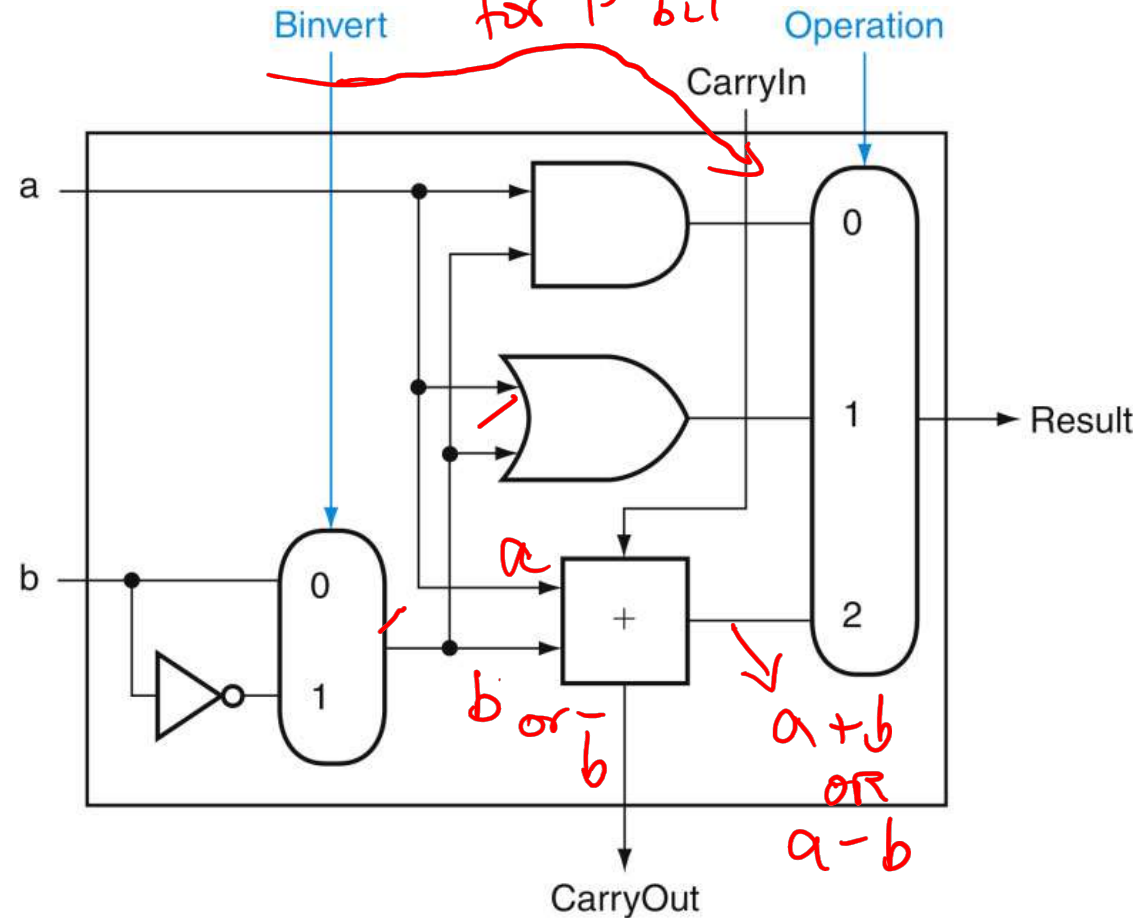
derived from optype for 1st bit

Must invert bits of B and add a 1

- Include an inverter
- CarryIn for the first bit is 1
- The CarryIn signal (for the first bit) can be the same as the Binvert signal

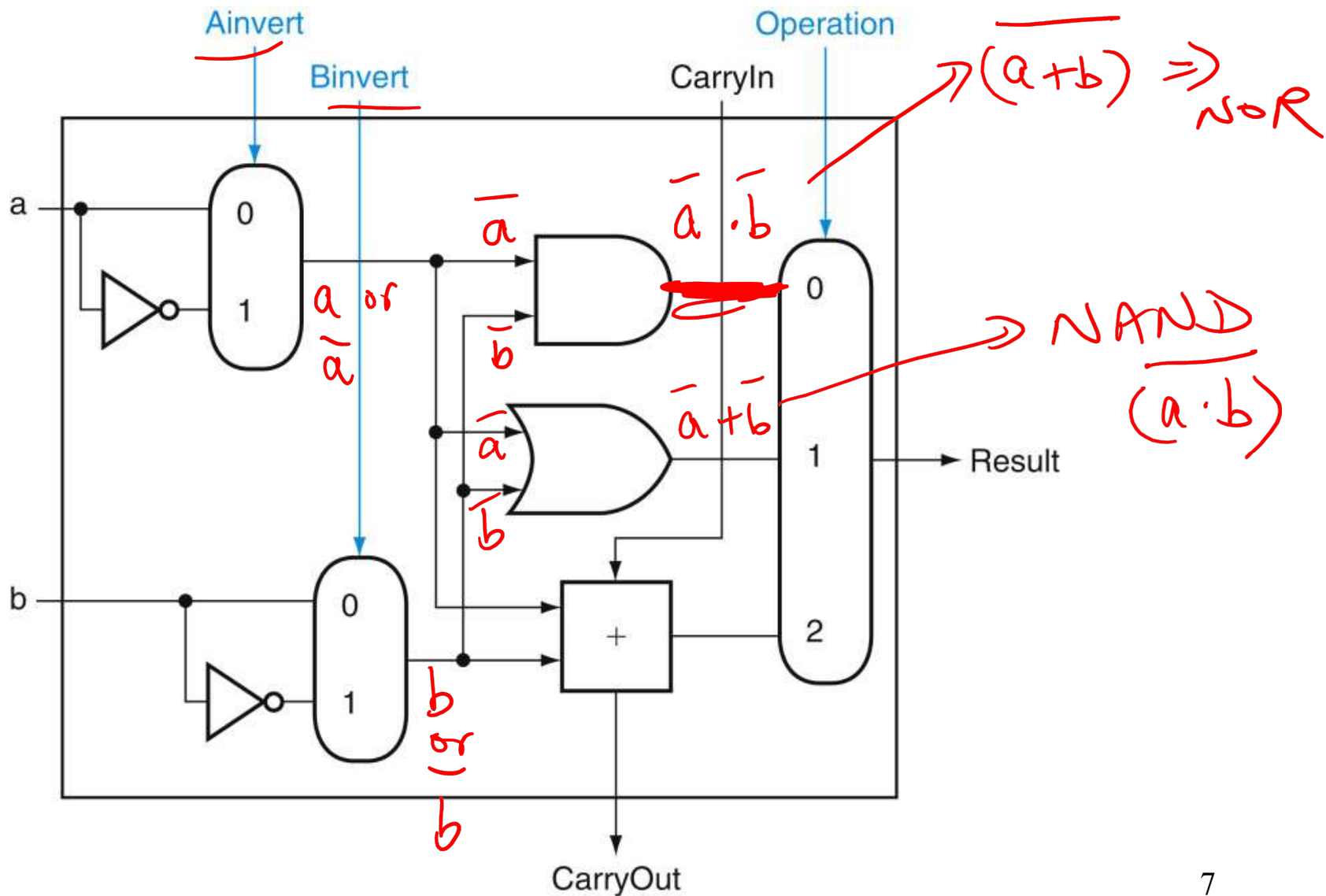
Add or AND  
Sub

ALU



Source: H&P textbook

# Incorporating NOR and NAND

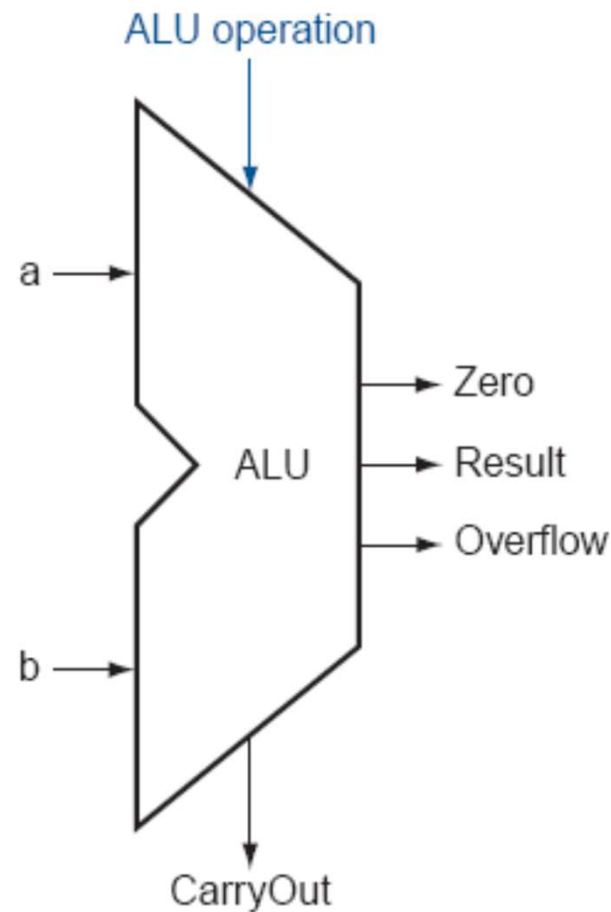


# Control Lines

---

What are the values of the control lines and what operations do they correspond to?

	<u>Ai</u>	<u>Bn</u>	<u>Op</u>
AND	0	0	<u>00</u>
OR	0	0	01
Add	0	0	10
Sub	0	1	10
NAND	1	1	01
NOR	1	1	00



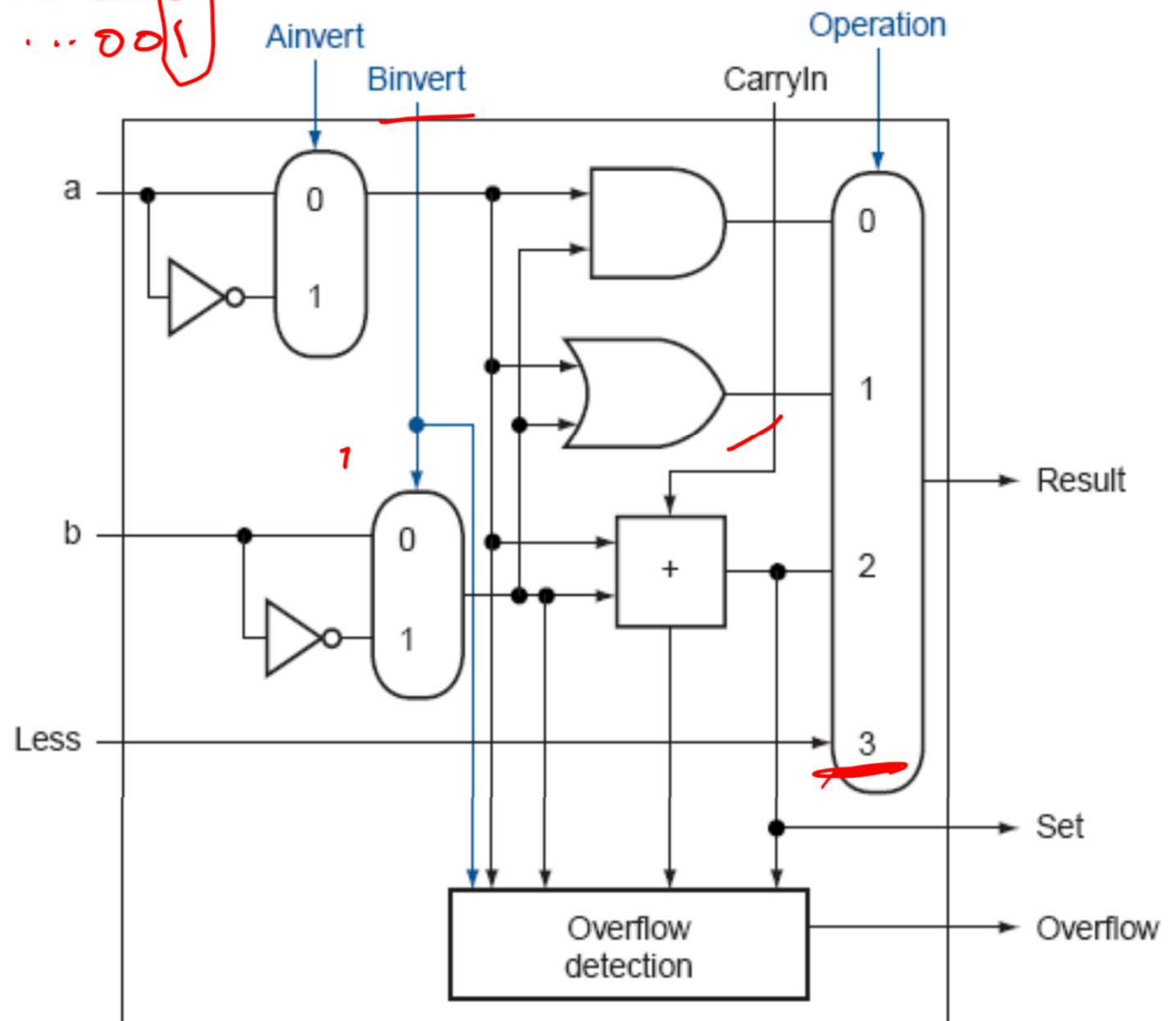


# Incorporating slt

if  $a < b$  then set out to 1  
else out to 0

out either 0000000...000 or 0000000...001

- Perform  $a - b$  and check the sign
- New signal (Less) that is zero for ALU boxes 1-31
- The 31<sup>st</sup> box has a unit to detect overflow and sign – the sign bit serves as the Less signal for the 0<sup>th</sup> box



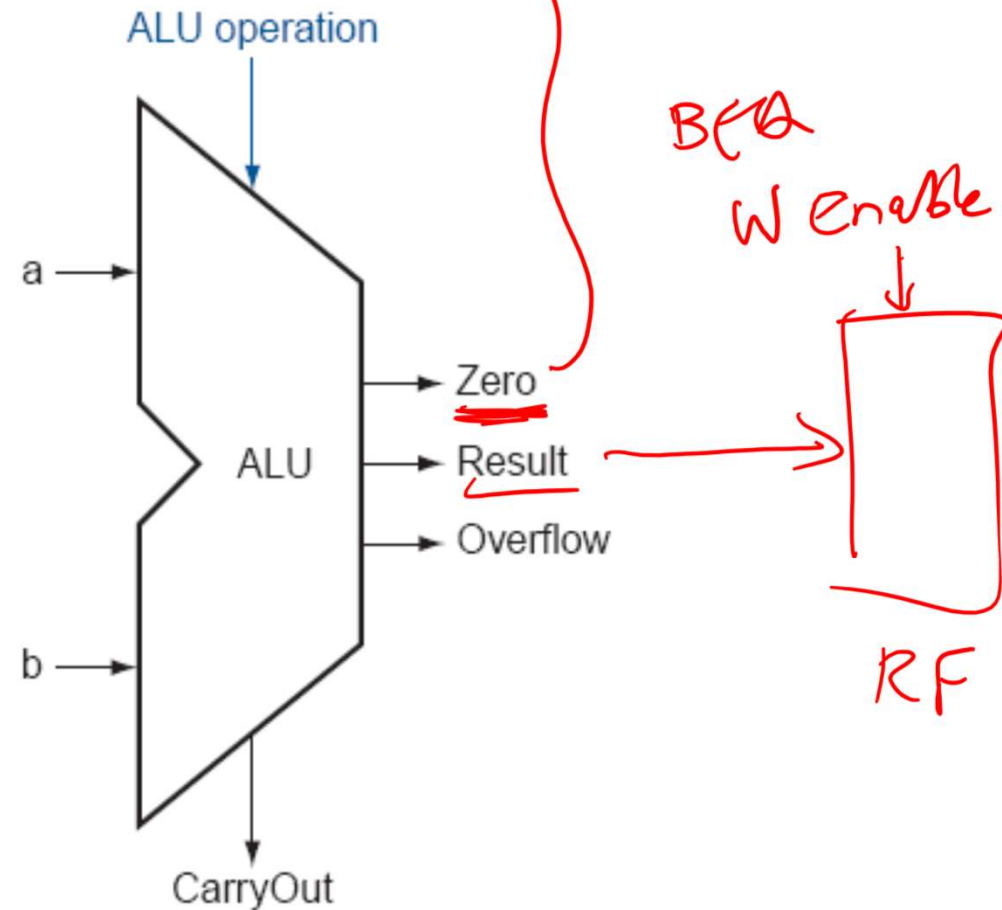
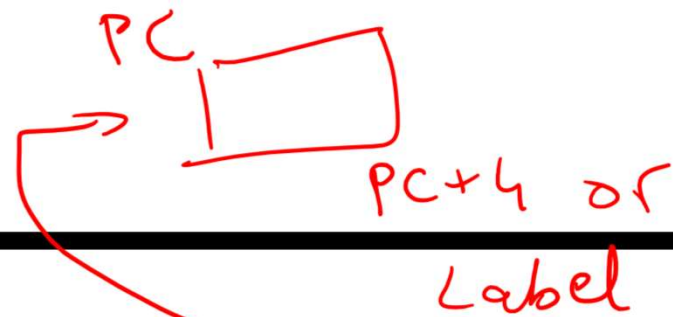
- Perform  $a - b$  and confirm that the result is all zero's



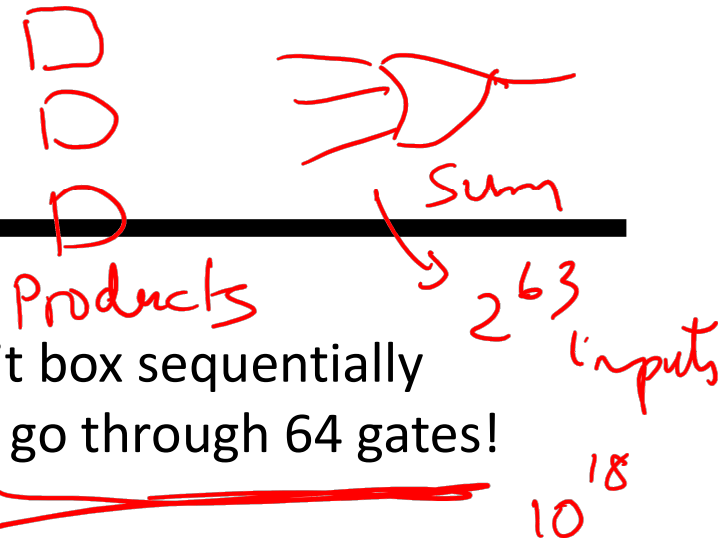
# Control Lines

What are the values of the control lines and what operations do they correspond to?

	Ai	Bn	Op
AND	0	0	00
OR	0	0	01
Add	0	0	10
Sub	0	1	10
NOR	1	1	00
NAND	1	1	01
<u>SLT</u>	<u>0</u>	<u>1</u>	<u>11</u>
BEQ	0	1	10



# Speed of Ripple Carry

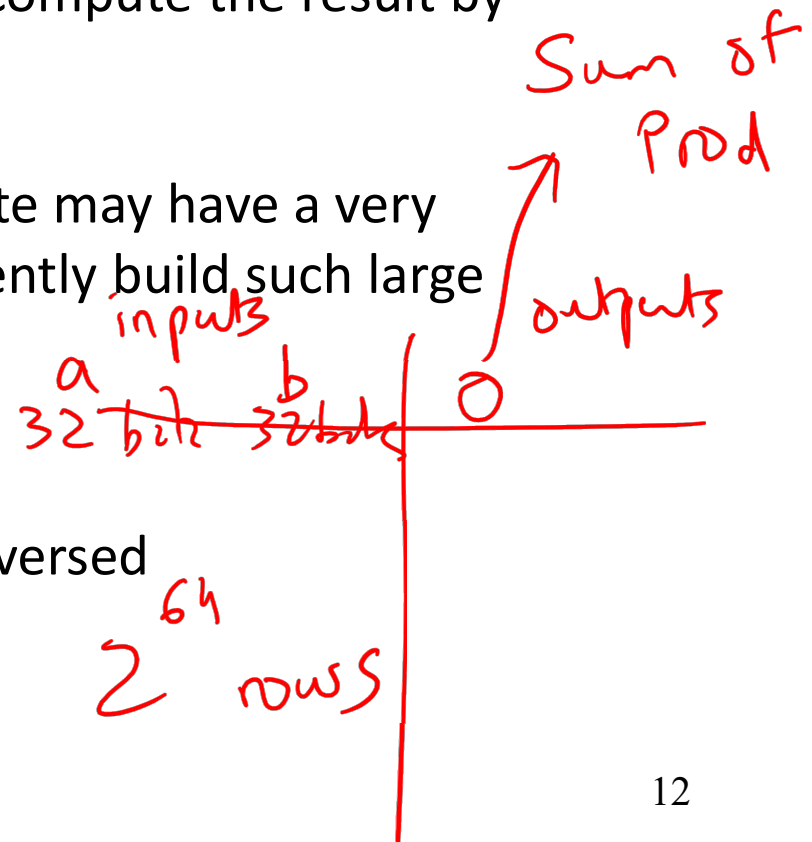


- The carry propagates thru every 1-bit box: each 1-bit box sequentially implements AND and OR – total delay is the time to go through 64 gates!

- We've already seen that any logic equation can be expressed as the sum of products – so 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 – it is difficult to efficiently build such large gates, so we'll find a compromise:

- moderate number of gates
- moderate number of inputs to each gate
- moderate number of sequential gates traversed



# Computing CarryOut

$$\text{Cout} = A \cdot B + A \cdot \text{Cin} + B \cdot \text{Cin}$$



$$\text{CarryIn1} = b_0 \cdot \text{CarryIn0} + a_0 \cdot \text{CarryIn0} + a_0 \cdot b_0$$

$$\text{CarryIn2} = b_1 \cdot \text{CarryIn1} + a_1 \cdot \text{CarryIn1} + a_1 \cdot b_1$$

$$= [b_1 \cdot b_0 \cdot c_0 + b_1 \cdot a_0 \cdot c_0 + b_1 \cdot a_0 \cdot b_0 + a_1 \cdot b_0 \cdot c_0 + a_1 \cdot a_0 \cdot c_0 + a_1 \cdot a_0 \cdot b_0 + a_1 \cdot b_1]$$

...

CarryIn32 = a really large sum of really large products

- Potentially fast implementation as the result is computed by going thru just 2 levels of logic – unfortunately, each gate is enormous and slow

Expl extremes  
(impractical)

# Generate and Propagate

Equation re-phrased:

$$\begin{aligned} C_{i+1} &= a_i.b_i + a_i.C_i + b_i.C_i \\ &= (a_i.b_i) + (a_i + b_i).C_i \end{aligned}$$

Gen Propagate

Stated verbally, the current pair of bits will *generate* a carry if they are both 1 and the current pair of bits will *propagate* a carry if either is 1

$$C_{out} = G + P.C_{in}$$

$$\text{Generate signal} = a_i.b_i = g_i$$

$$\text{Propagate signal} = a_i + b_i = p_i$$

$$\text{Therefore, } C_{i+1} = G_i + P_i . C_i$$

# Generate and Propagate

3 gate delays

$$c1 = g0 + p0.c0$$

$$c2 = g1 + p1.c1$$

$$= g1 + p1.g0 + p1.p0.c0$$

$$c3 = g2 + p2.g1 + p2.p1.g0 + p2.p1.p0.c0$$

$$c4 = g3 + p3.g2 + p3.p2.g1 + p3.p2.p1.g0 + p3.p2.p1.p0.c0$$

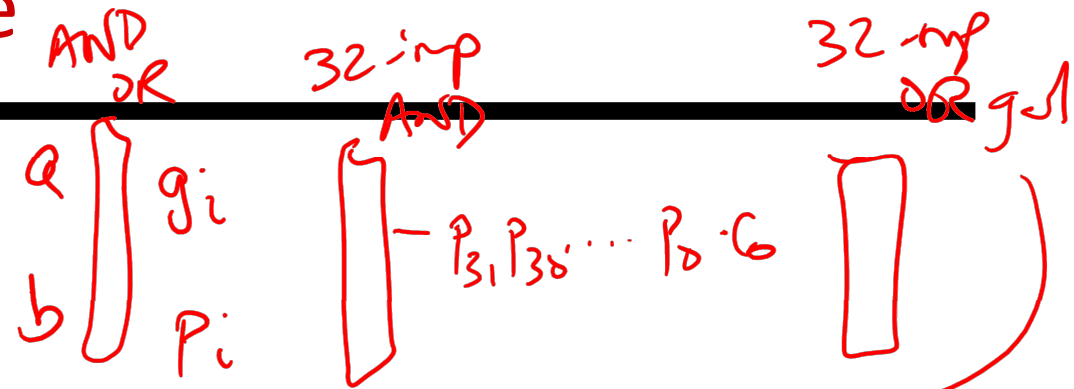
$C_{32} =$   
Either,

a carry was just generated, or

a carry was generated in the last step and was propagated, or

a carry was generated two steps back and was propagated by both the next two stages, or

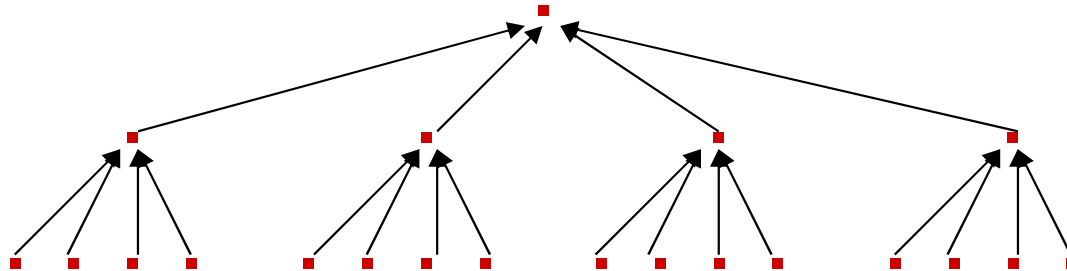
a carry was generated N steps back and was propagated by every single one of the N next stages



# Divide and Conquer

---

- The equations on the previous slide are still difficult to implement as logic functions – for the 32<sup>nd</sup> bit, we must AND every single propagate bit to determine what becomes of c0 (among other things)
- Hence, the bits are broken into groups (of 4) and each group computes its group-generate and group-propagate
- For example, to add 32 numbers, you can partition the task as a tree





# P and G for 4-bit Blocks



- Compute P0 and G0 (super-propagate and super-generate) for the first group of 4 bits (and similarly for other groups of 4 bits)

$$P0 = p0.p1.p2.p3$$

$$G0 = g3 + g2.p3 + g1.p2.p3 + g0.p1.p2.p3$$

- Carry out of the first group of 4 bits is

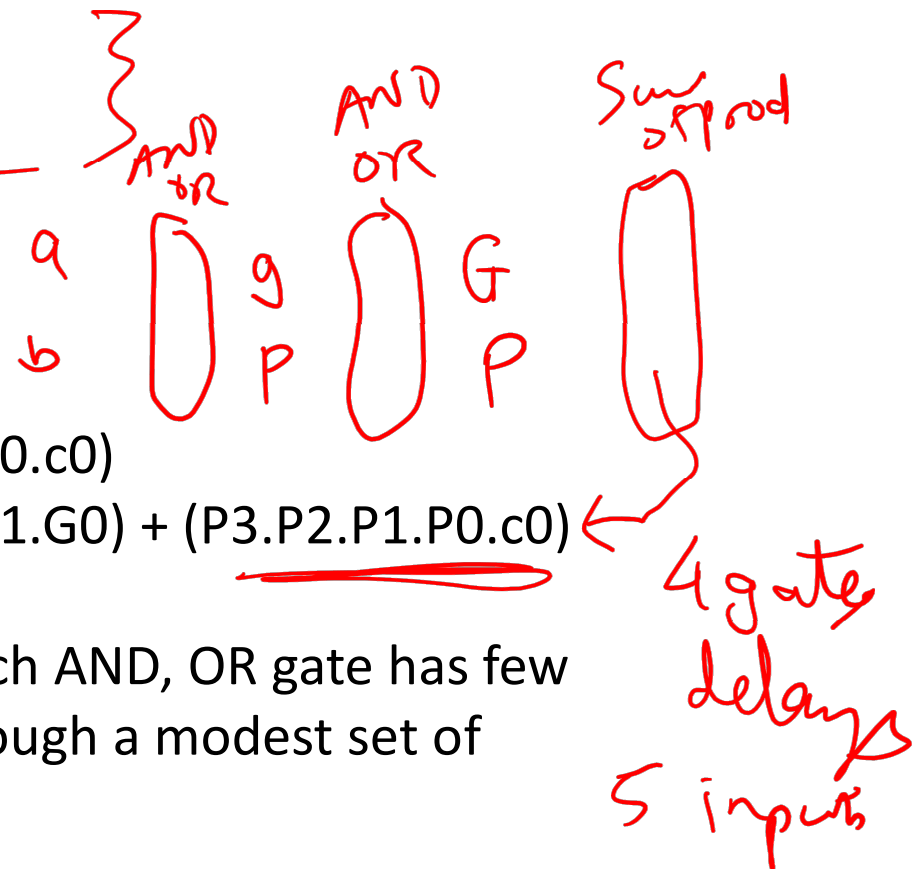
$$C1 = G0 + P0.c0$$

$$C2 = G1 + P1.G0 + P1.P0.c0$$

$$C3 = G2 + (P2.G1) + (P2.P1.G0) + (P2.P1.P0.c0)$$

$$C4 = G3 + (P3.G2) + (P3.P2.G1) + (P3.P2.P1.G0) + (P3.P2.P1.P0.c0)$$

- By having a tree of sub-computations, each AND, OR gate has few inputs and logic signals have to travel through a modest set of gates (equal to the height of the tree)



# Example

---

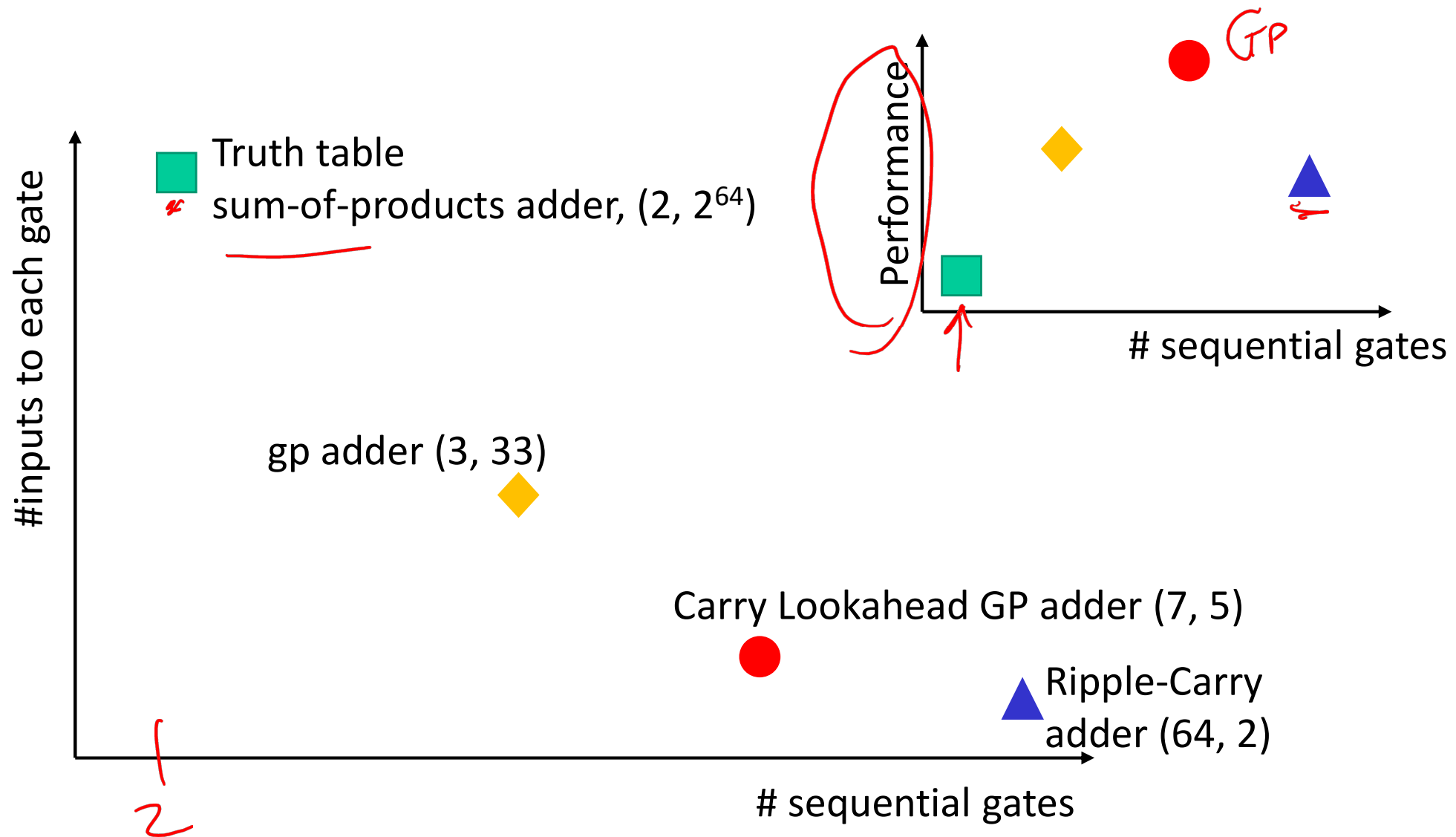
Add	A	0001	1010	0011	0011
	B	1110	0101	1110	1011
		<hr/>			
	g	0000	0000	0010	0011
	p	1111	1111	1111	1011

P	1	1	1	0
---	---	---	---	---

G	0	0	1	0
---	---	---	---	---

C4 = 1

# Trade-Off Curve



# Carry Look-Ahead Adder

- 16-bit Ripple-carry takes 32 steps
- This design takes how many steps?  
5 sequential steps

