Lecture 4 Arithmetic-Logic Unit

Arithmetic - Logic Unit ALU

- Handles integers
- Does the calculations

Arithmetic-Logic Unit ALU

- Performs arithmetic
  - add, subtract
- Performs logic
  - and, or, invert, complement
- Shifts
  - right, left, arithmetic, logical
- Provides result and status

Review Binary Addition

	Binary							Decimal		
Carry	1	1	1	1	0			0	0	
		1	1	0	1	1			2	7
		1	0	1	1	0			2	2
	1	1	0	0	0	1		0	4	9

Example Numbers

- **8 bit 2's complement**
  - **+127 = 01111111 = 2^7 -1**
  - **-128 = 10000000 = -2^7**
- **16 bit 2's complement**
  - **+32767 = 011111111 11111111 = 2^{15} - 1**
  - **-32768 = 100000000 00000000 = -2^{15}**

Sign Extension

- Positive number pack with leading zeros

  +18 = 00010010

  +18 = 00000000 00010010

- Negative number pack with leading ones

  -18 = 11101110

  -18 = 11111111 11101110

- i.e. pack with MSB (sign bit)

Addition and Subtraction

- Normal binary addition circuitry
- Take two's complement of subtrahend and add to minuend

  i.e. a - b = a + (-b)
- Need only addition and complement circuits

Consider Binary Addition

Assume 5 bits 2's complement arithmetic

12 - 7 = 12 + (-7) = 5

Binary

Carry	1	1	0	0	0	
		0	1	1	0	0
		1	1	0	0	1
	1	0	0	1	0	1

Carry out

Consider Binary Addition

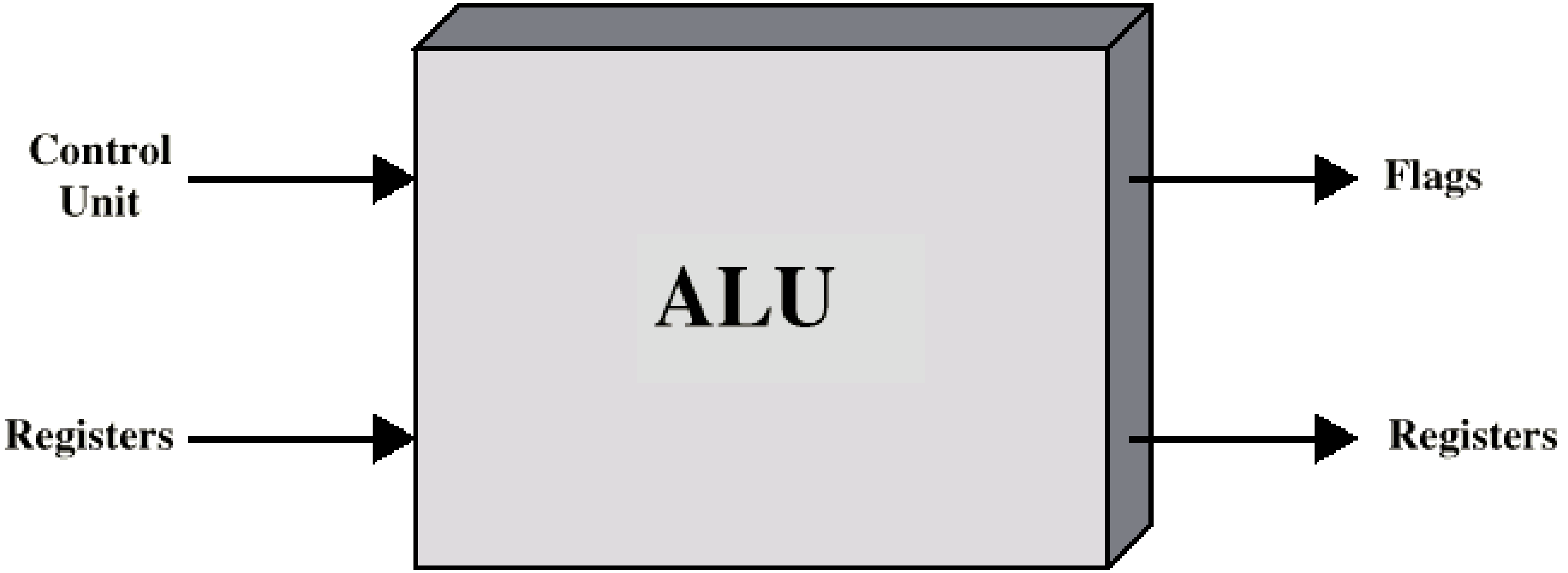
Assume 5 bits 2’s complement arithmetic

12 - 13 = 12 + (-13) = -1

		Binary				
Carry	0	0	0	0	0	
		0	1	1	0	0
		1	0	0	1	1
	0	1	1	1	1	1

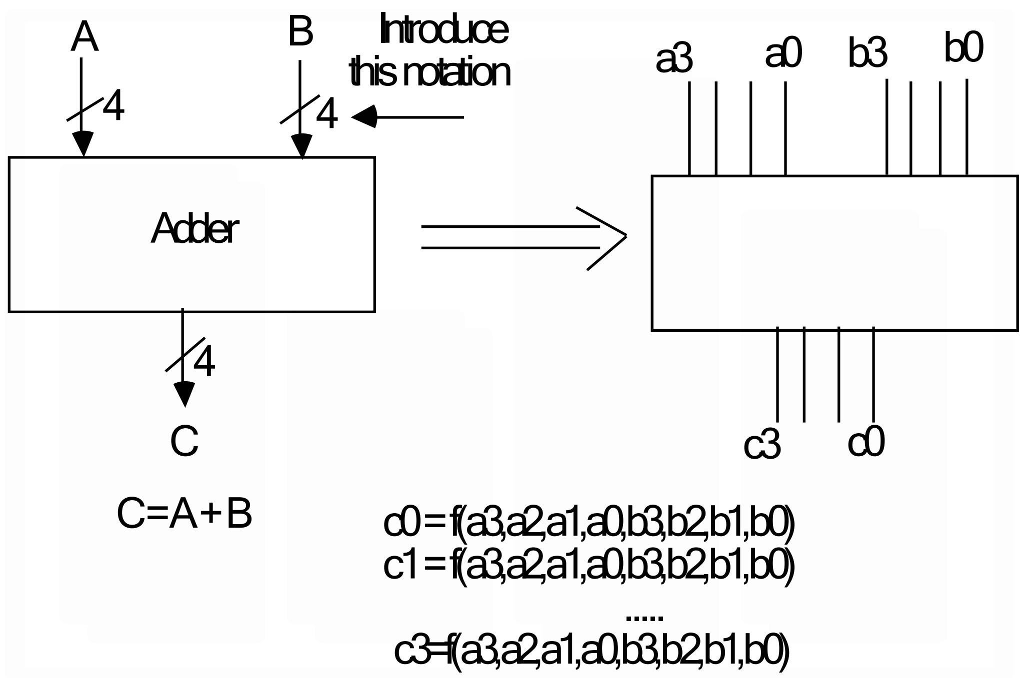
Carry out

ALU Inputs and Outputs



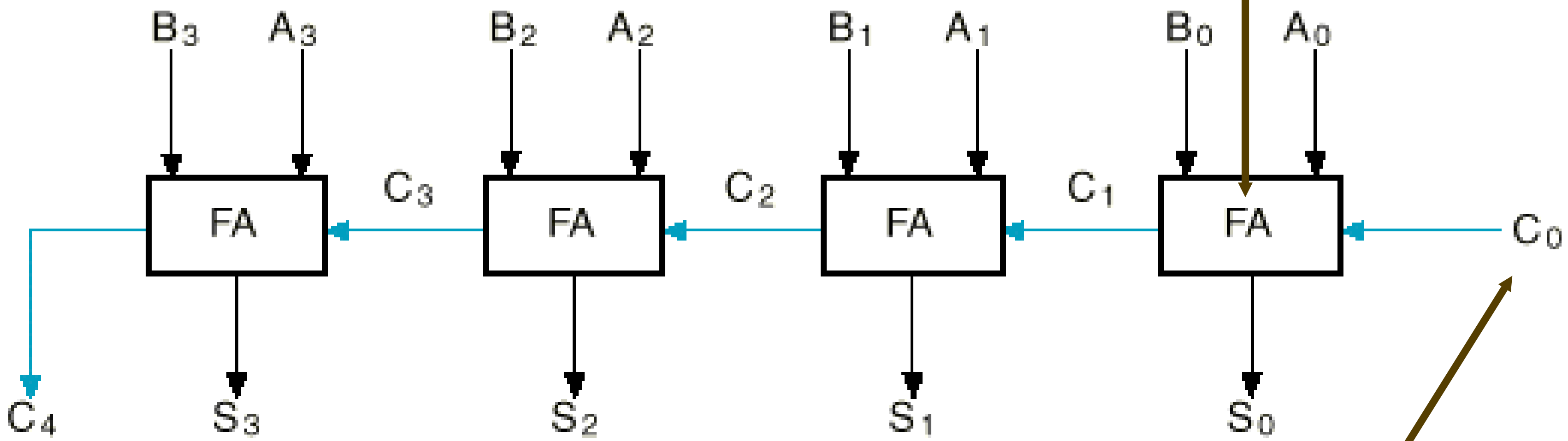
ALU - Addition

Could try this as an 8 input, 4 output combinational logic problem



Instead - Consider Stages

FA - Full Adder



Depends on 1’s or 2’s comp arithmetic

Full Adder

Truth Table

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

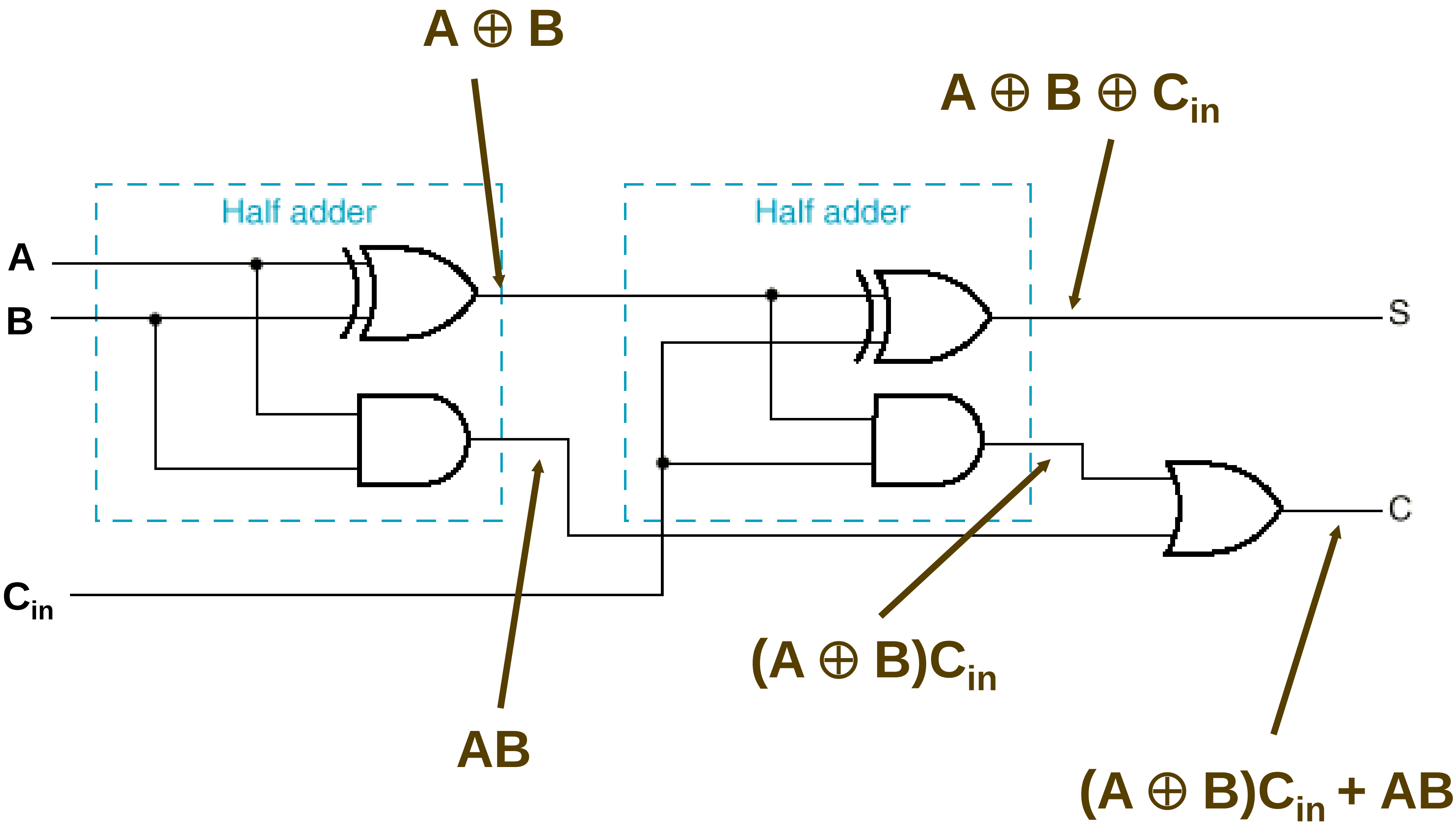
$S = A'B'C_{in} + A'BC_{in}' + AB'C_{in}' + ABC_{in}$

$= A \oplus B \oplus C_{in}$

$C = A'BC_{in} + AB'C_{in} + ABC_{in}' + ABC_{in}$

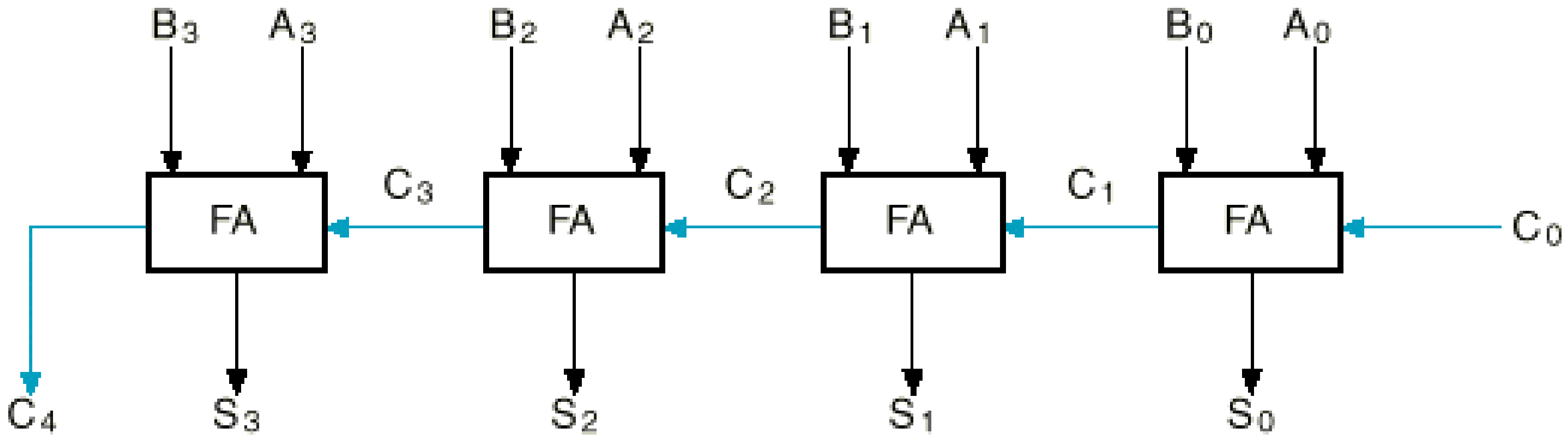
$= (A \oplus B)C_{in} + AB$

Full Adder



Graph from: Logic and Computer Design Fundamentals, Mano & Kime, Prentice Hall

4 Bit Ripple Carry 2's Complement Adder

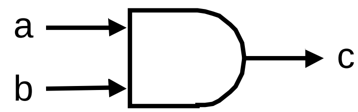


Constructing an Arithmetic Logic Unit

Start with a 1-Bit ALU

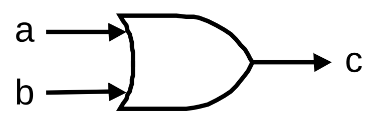
Simple Logical Operations

1. AND gate (c = a . b)



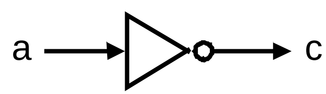
a	b	c = a . b
0	0	0
0	1	0
1	0	0
1	1	1

2. OR gate (c = a + b)



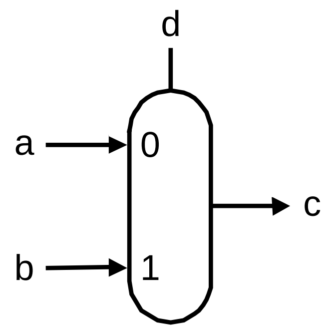
a	b	c = a + b
0	0	0
0	1	1
1	0	1
1	1	1

3. Inverter (c = a)



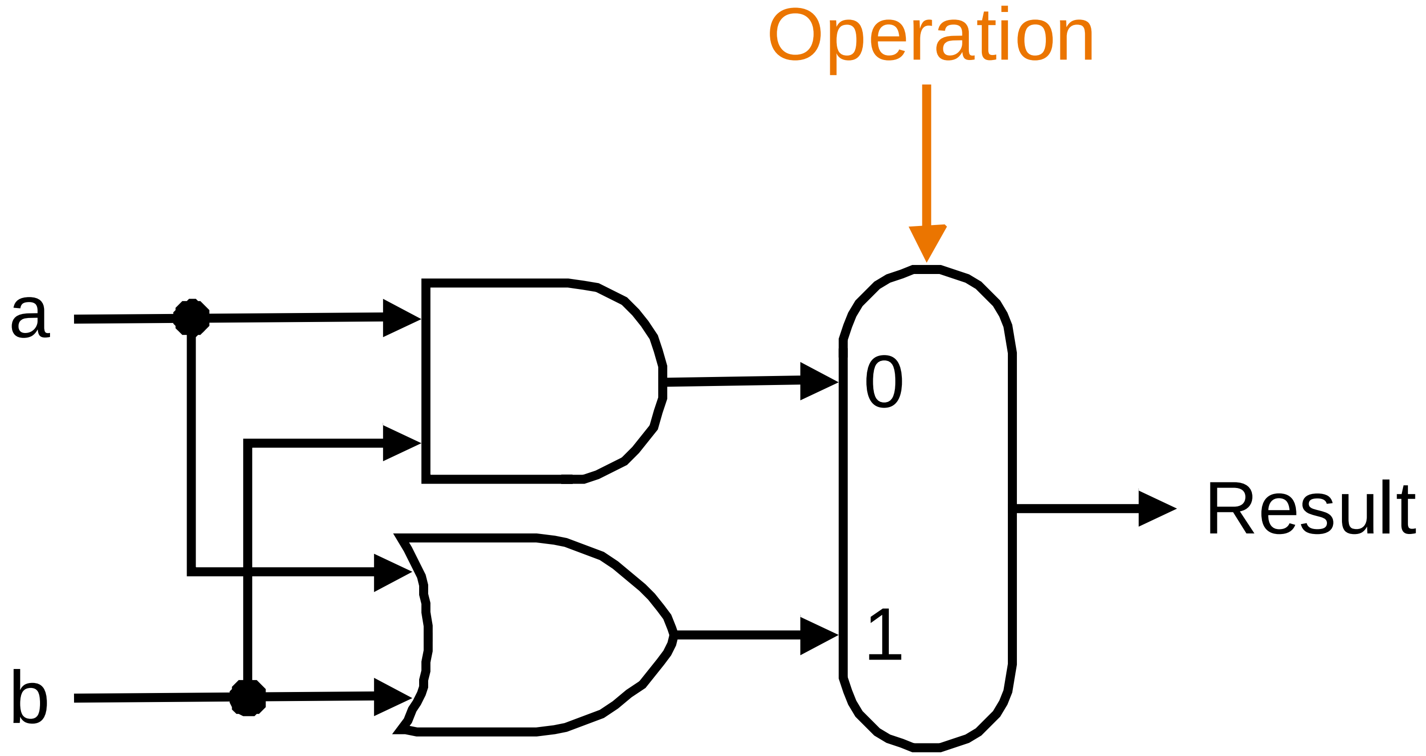
a	$c = \bar{a}$
0	1
1	0

4. Multiplexor
   (if d = = 0, c = a;
   else c = b)



d	c
0	a
1	b

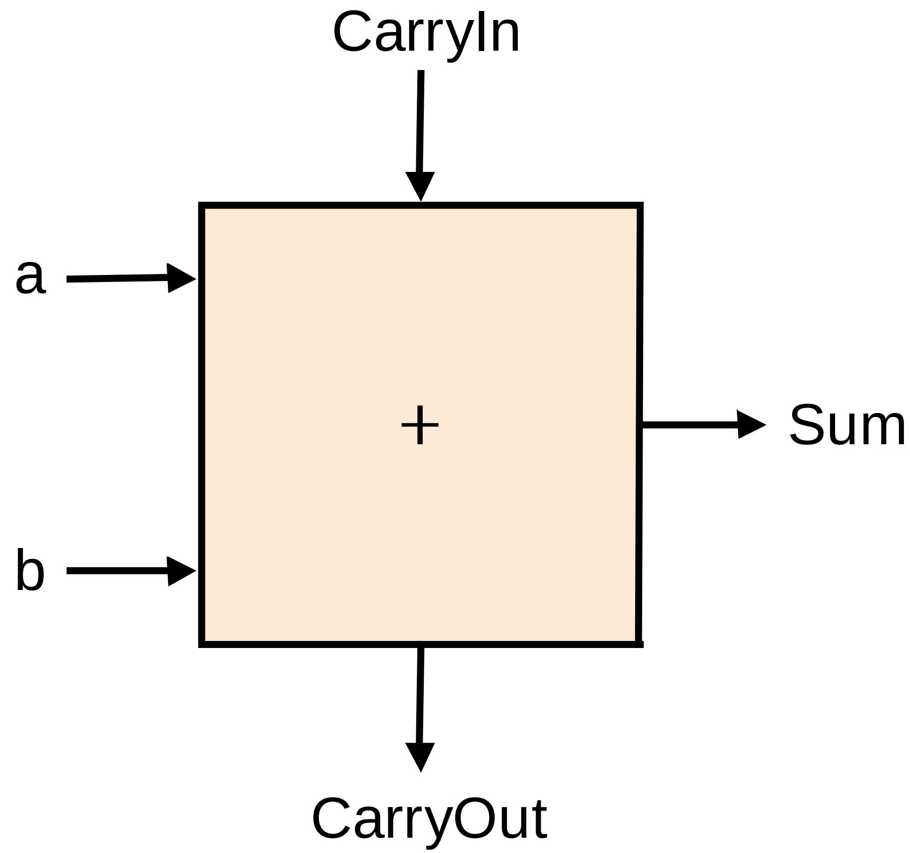
Starting from “AND” and “OR”



If Operation is 0, then Result = a AND b

If Operation is 1, then Result = a OR b

Consider a 1 bit Full Adder

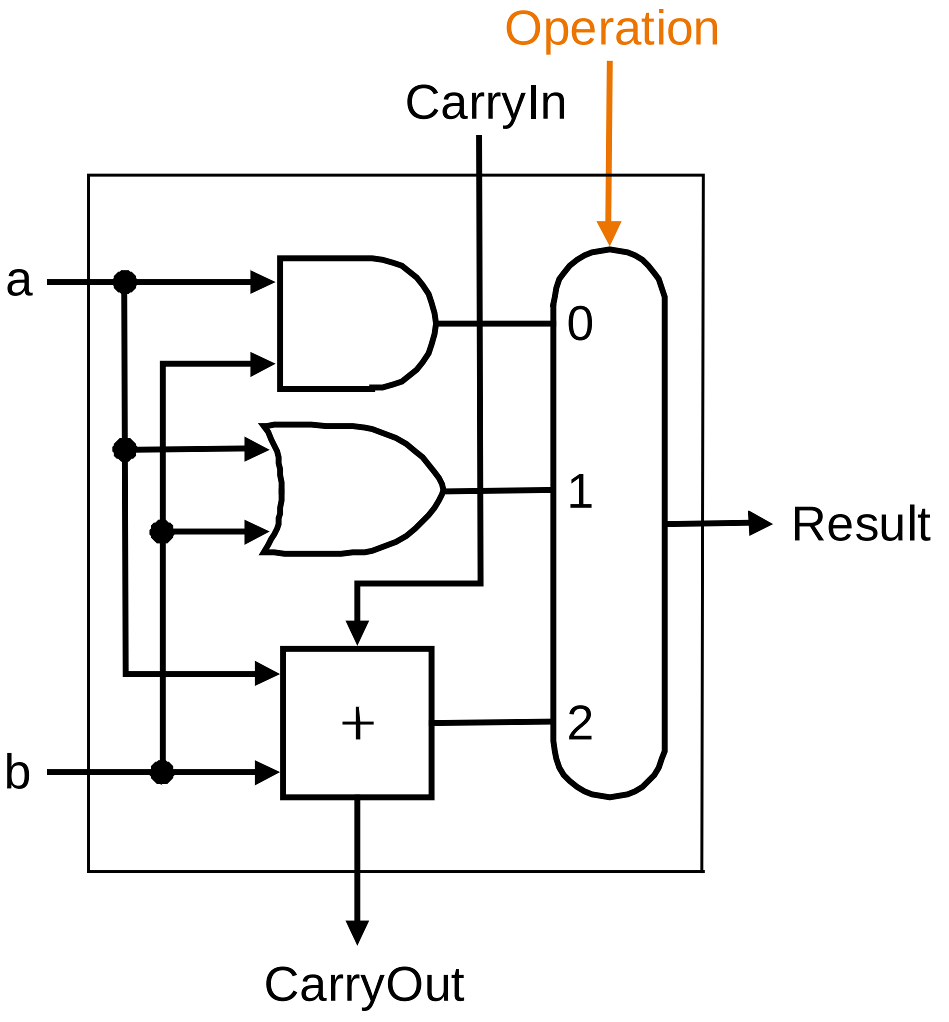


With “add”

If Op is 0, then Result = a AND b

If Op is 1, then Result = a OR b

If Op is 2, then Result = sum of (a + b)

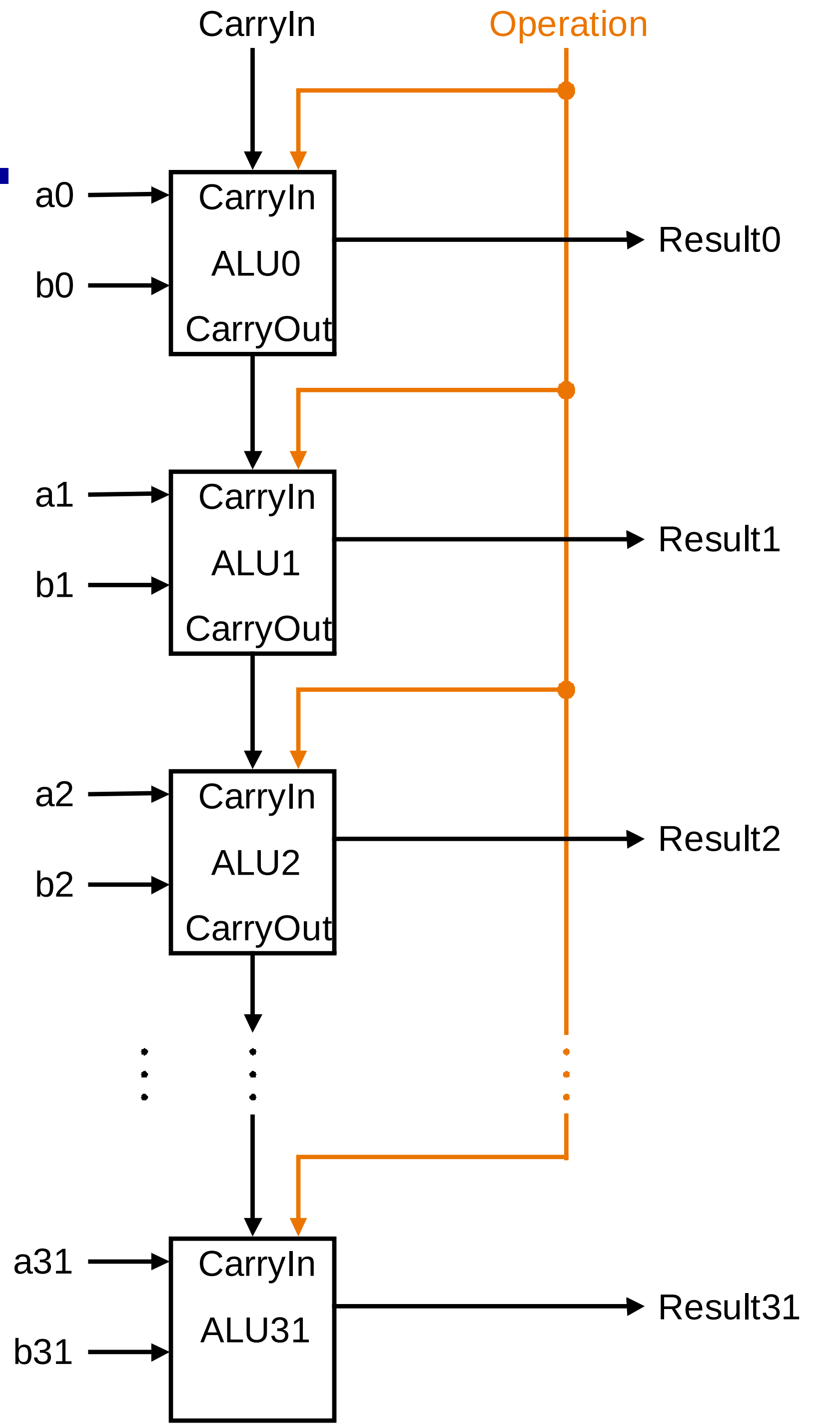


If we repeat the 1-Bit ALU 32 times

If Op is 0, then $Res_i = a_i$ AND b_i

If Op is 1, then $Res_i = a_i$ OR b_i

If Op is 2, then Res_i = sum of $(a_i + b_i)$



With Subtraction

If Op is 0, then Res = a AND b

If Op is 1, then Res = a OR b

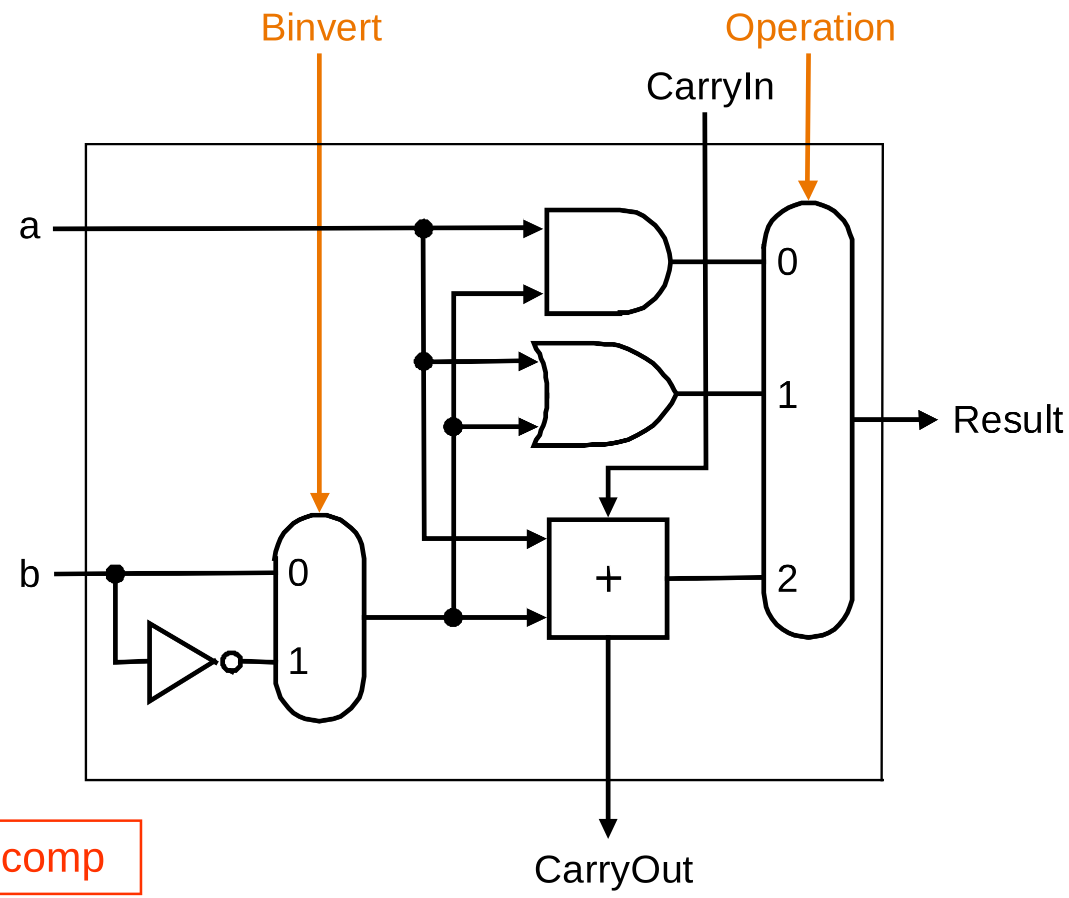
If Op is 2,

and if Binvert is 0,

then Res = sum (a + b)

if Binvert is 1,

then Res = sum (a + (-b))

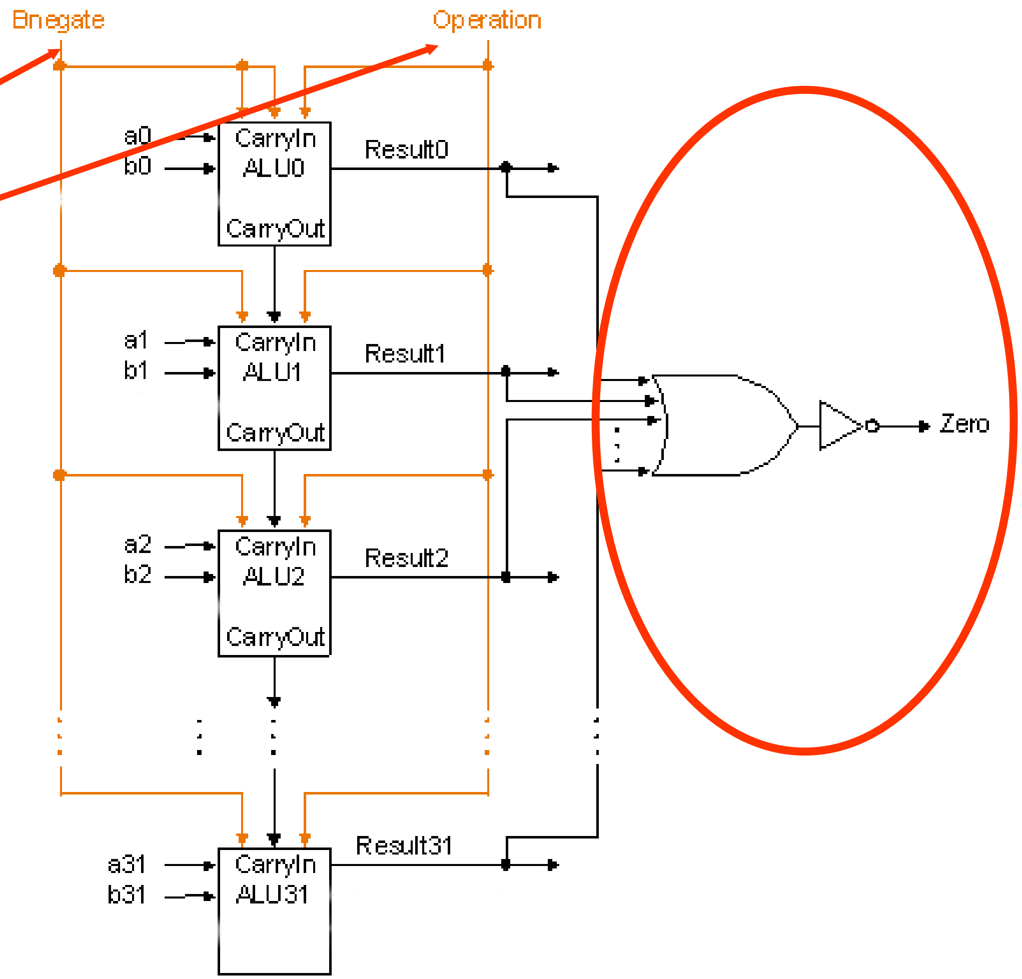


Note that (- b) is 1's comp

Add a 1 into CarryIn_0 to get 2's comp

ALU with Zero Detection — for comparing a and b

Control Lines	Function
000	and
001	or
010	add
110	sub



Overflow

- **Result too large for finite computer word:**
  - **e.g., adding two n-bit numbers does not yield an n-bit number**

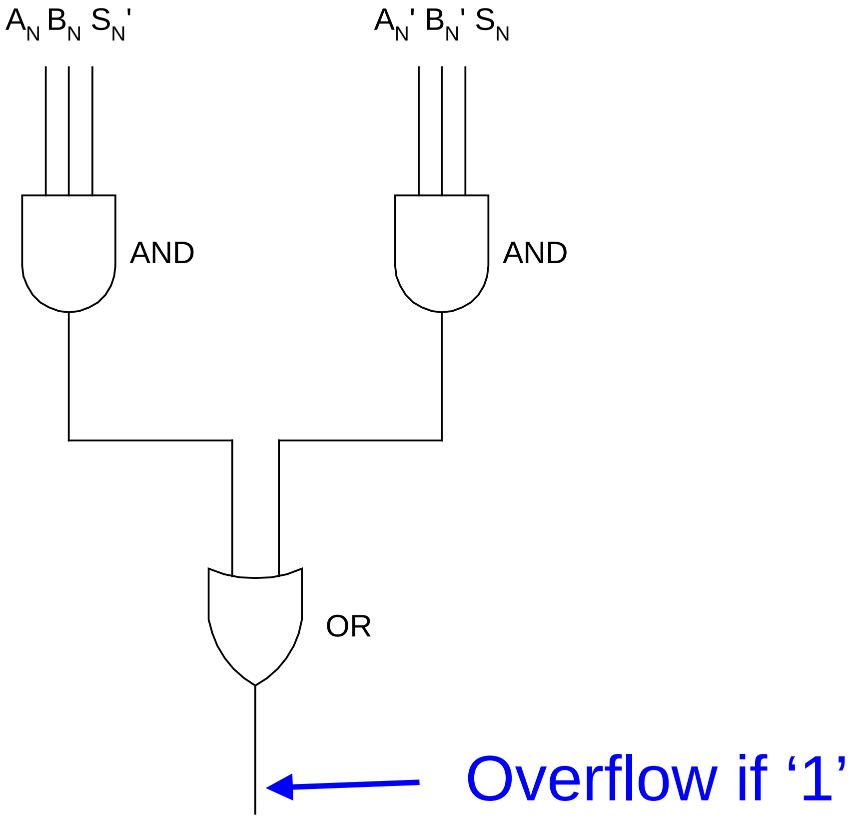
```
  0111
+ 0001
------
  1000
```

note that overflow term is somewhat misleading, it does not mean a carry "overflowed"

Detecting Overflow

- No overflow when adding a positive and a negative number
- No overflow when signs are the same for subtraction
- Overflow occurs when the value affects the sign:
  - overflow when adding two positives yields a negative
  - or, adding two negatives gives a positive
  - or, subtract a negative from a positive and get a negative
  - or, subtract a positive from a negative and get a positive
- Consider the operations A + B, and A – B
  - Can overflow occur if B is 0 ?
  - Can overflow occur if A is 0 ? Yes

Example Overflow Logic

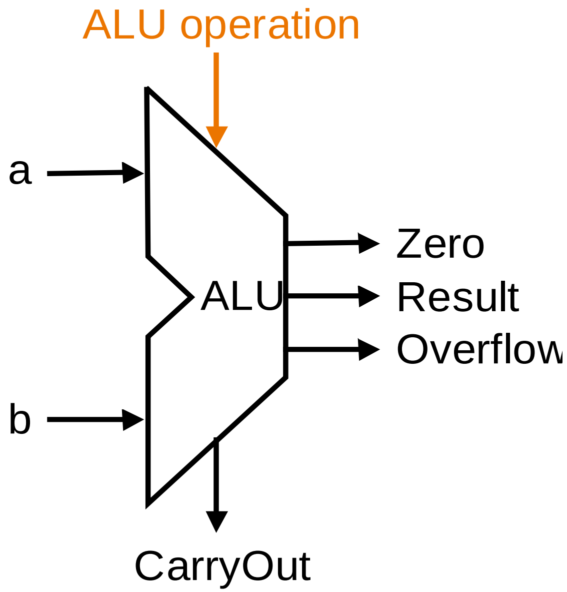


How is this derived? – Homework!

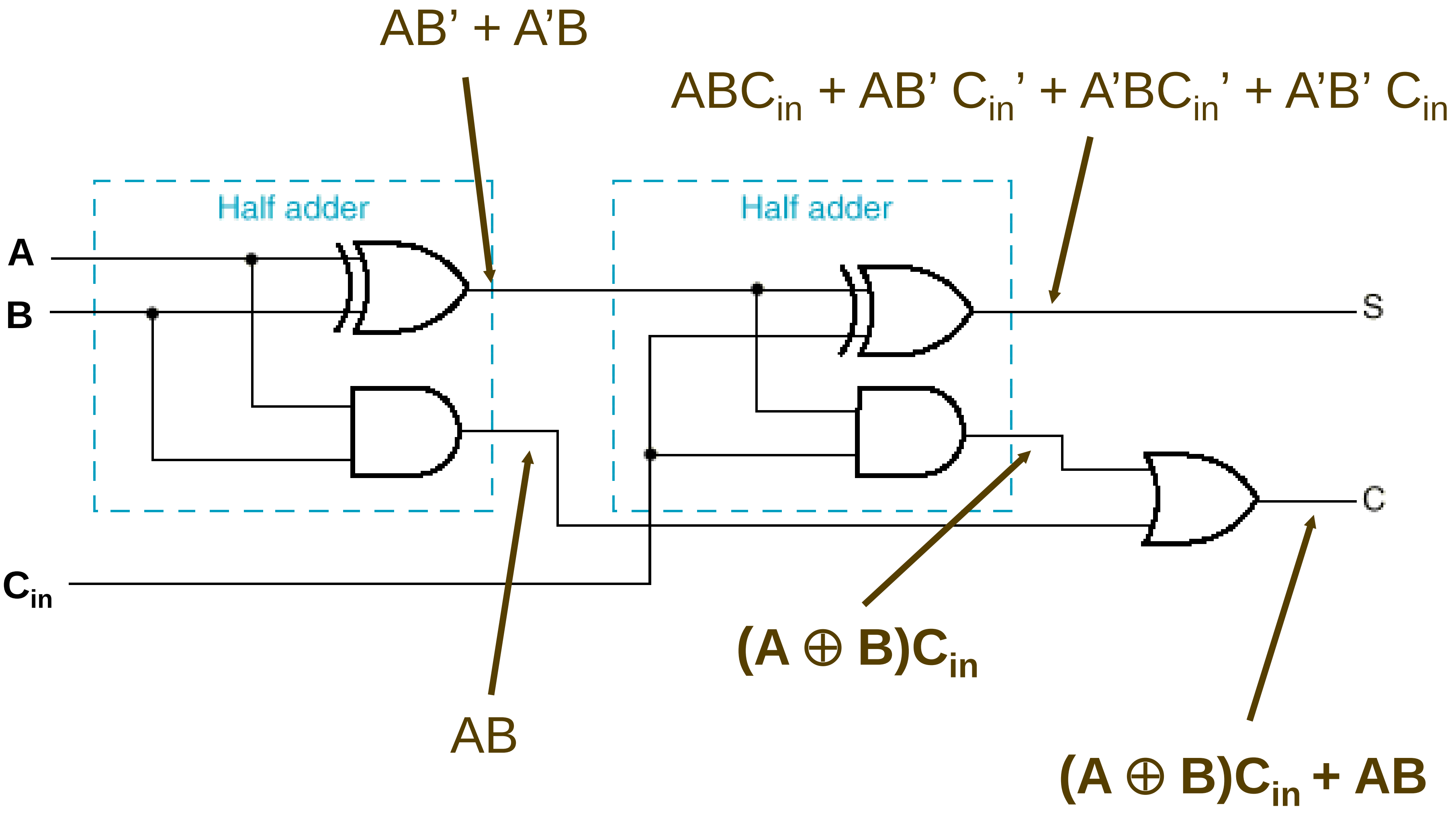
Effects of Overflow

- An exception (interrupt) occurs
  - Control jumps to predefined address for exception
  - Interrupted address is saved for possible resumption
- Details based on software system / language
  - example: flight control vs. homework assignment
- Don't always want to detect overflow
  - MIPS instructions: `addu, addiu, subu`
  - More later

Common Symbol for ALU

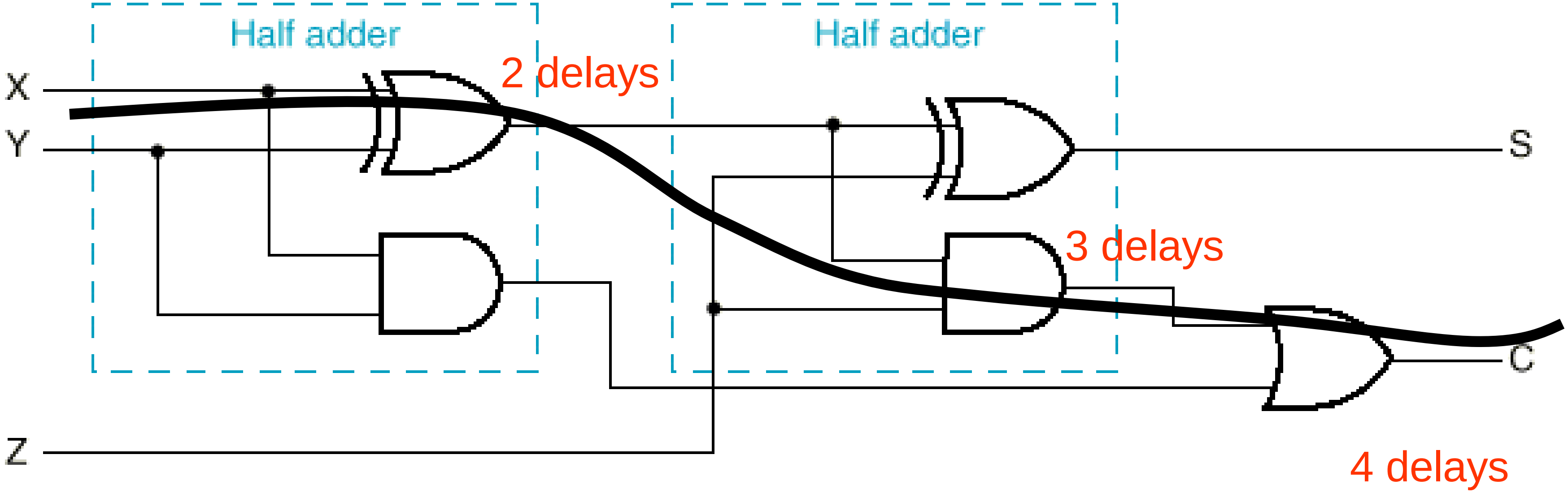


Recall Full Adder



Graph from: Logic and Computer Design Fundamentals, Mano & Kime, Prentice Hall

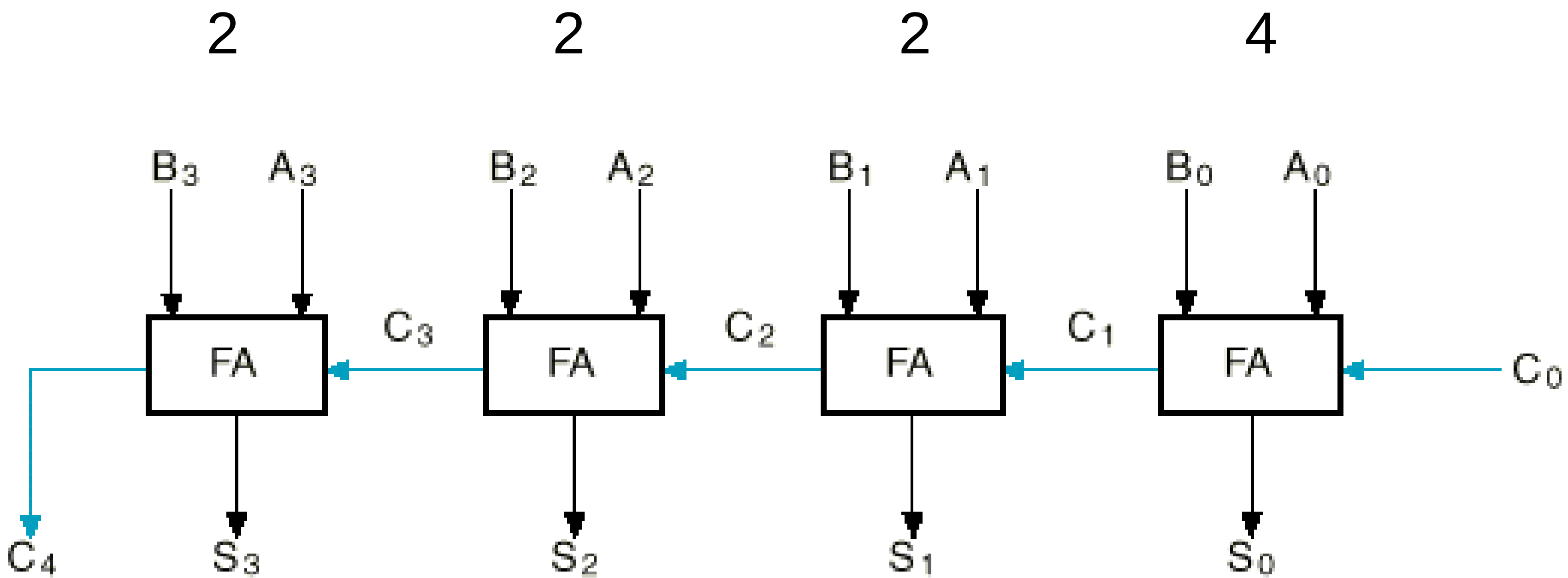
Full Adder - Half Adders



From Z to C is 2 delays for each subsequent stage or 2N + 2

Graphics from: Logic and Computer Design Fundamentals, Mano & Kime, Prentice Hall

4 Bit Ripple Carry Adder



2n+2 gate delays (10) for 2’s complement

Carry Lookahead Equations

Let $g_i = a_i b_i$ generating carry

$p_i = a_i + b_i$ propagating carry

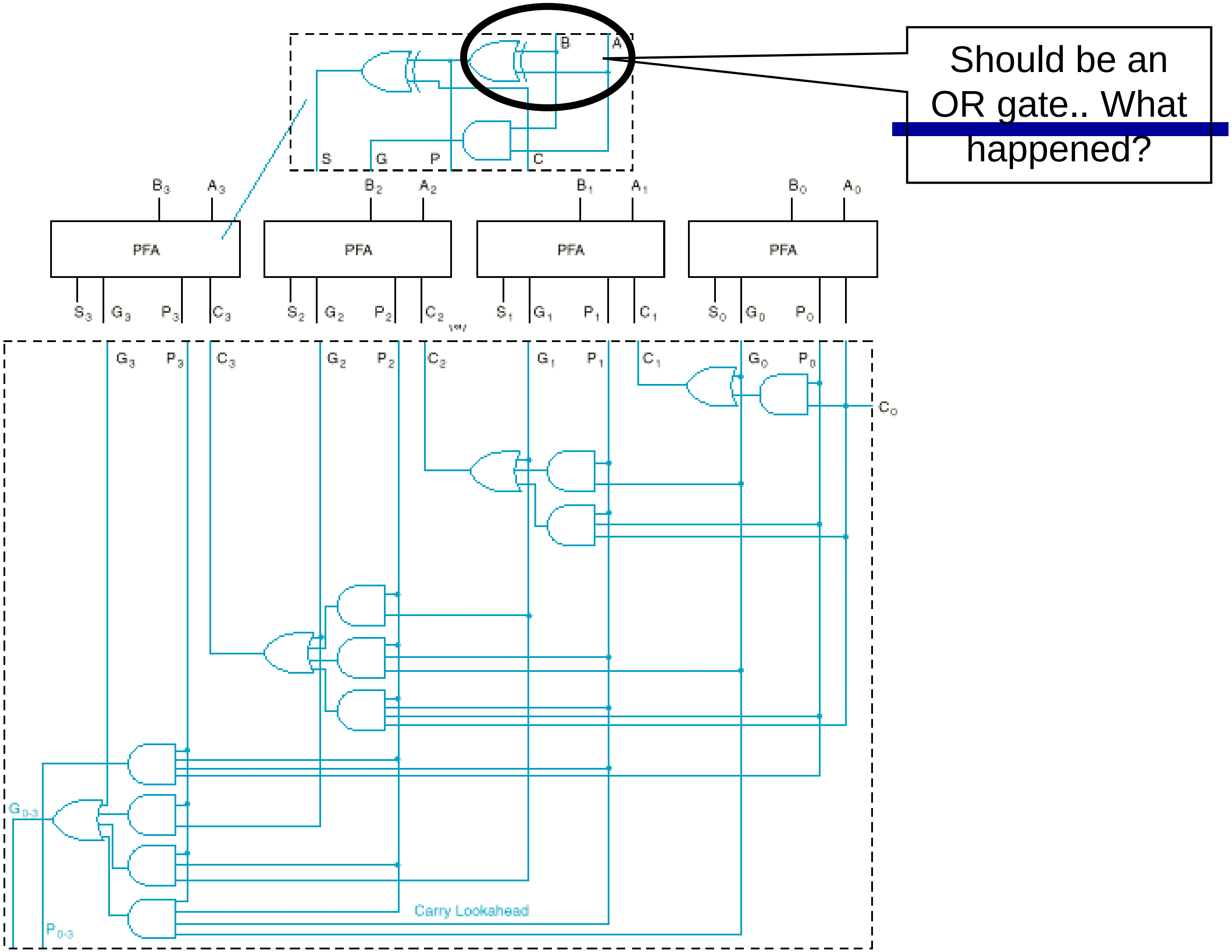
$c_1 = b_0c_0 + a_0c_0 + a_0b_0$ $c_1 = g_0 + p_0c_0$

$c_2 = b_1c_1 + a_1c_1 + a_1b_1$ $c_2 = g_1 + (p_1g_0) + (p_1p_0c_0)$

$c_3 = b_2c_2 + a_2c_2 + a_2b_2$ $c_3 = g_2 + p_2g_1 + (p_2p_1g_0) + (p_2p_1p_0c_0)$

$c_4 = b_3c_3 + a_3c_3 + a_3b_3$

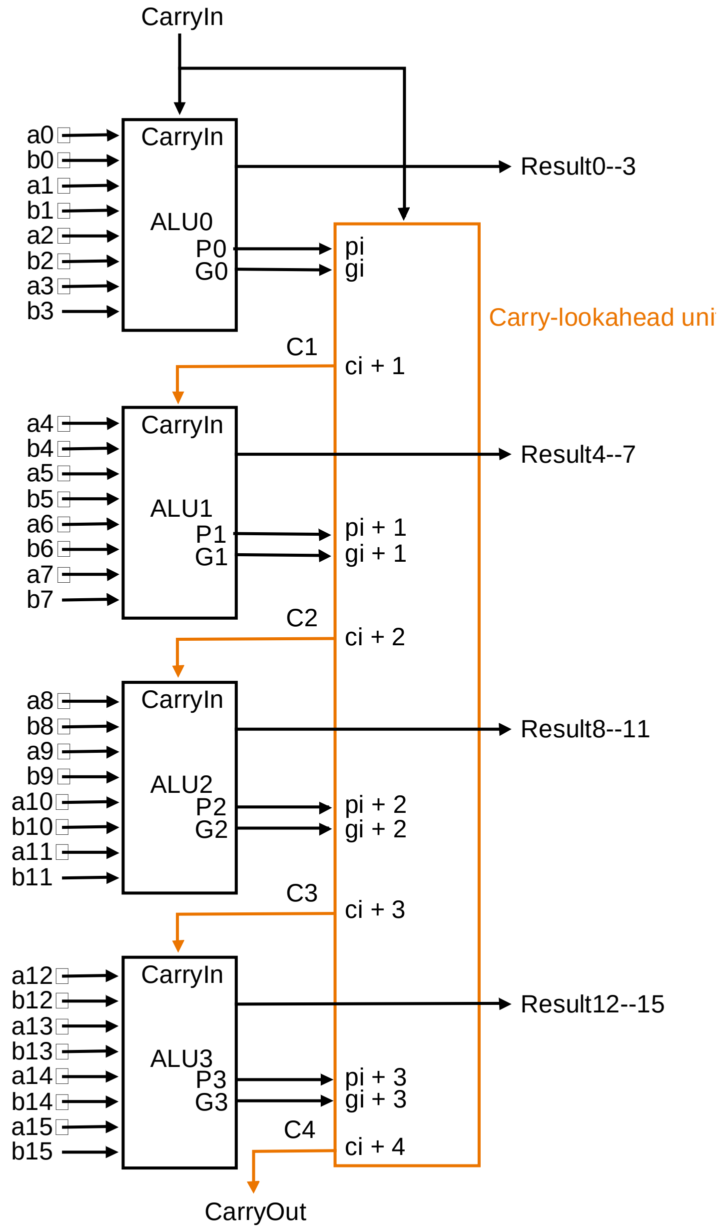
$$c_4 = \underbrace{g_3 + p_3g_2 + p_3p_2g_1 + (p_3p_2p_1g_0)}_{G_{0-3}} + (\underbrace{p_3p_2p_1p_0}_{P_{0-3}}c_0)$$



Reduces delay to 6 gate delays (from input to S)

4 gate delays from input to C

Carry Lookahead – Second Level



Carry Propagation

- 2’s complement best
- 1’s complement twice as long
- Significant delay reduction using Carry Look Ahead concept