Building an ALU

Jinyang

What we've learnt so far

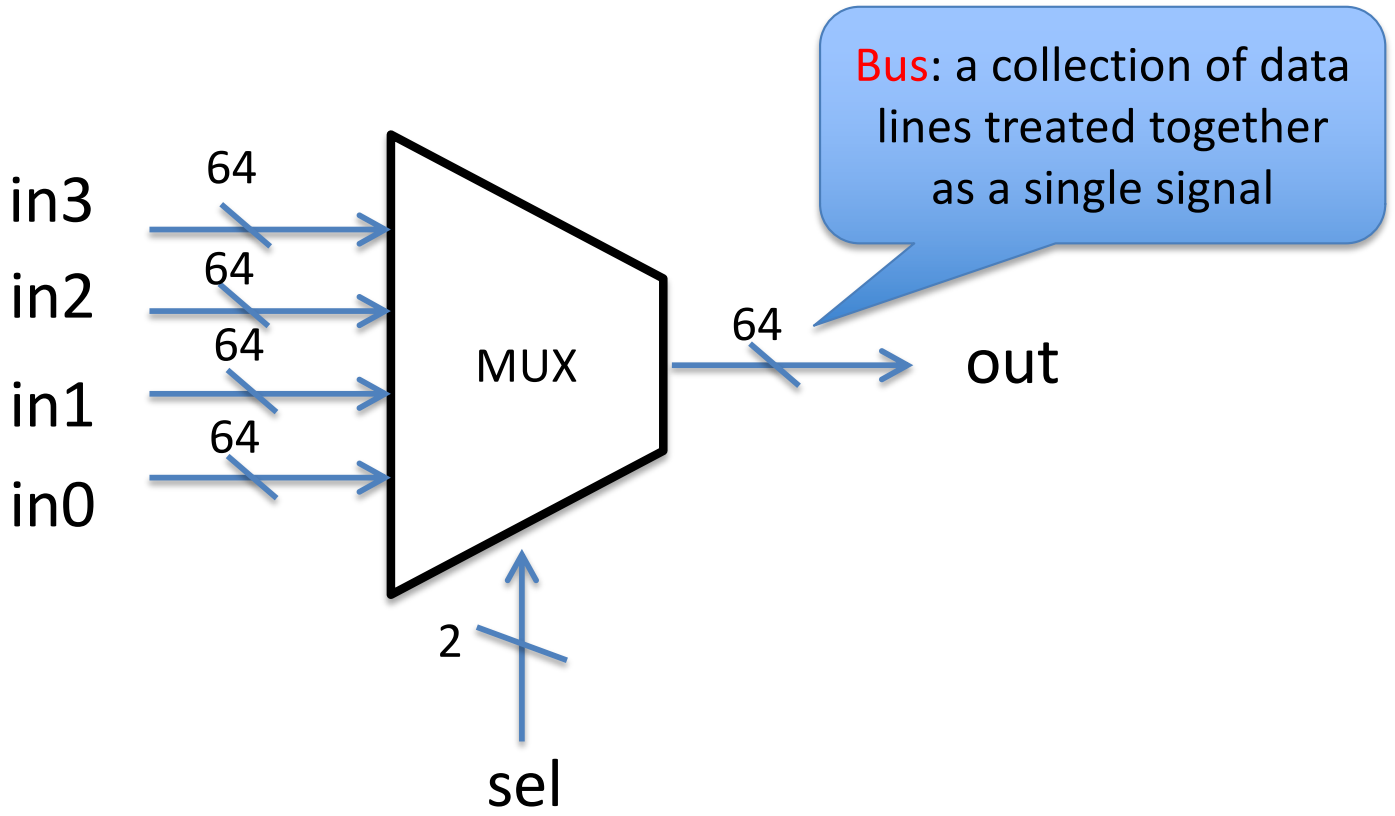
- Basic logic design
  - Logic circuits == Boolean expressions
- How to build a combinatorial logic circuit
  - Specify the truth table
  - Output is the sum of products
- Common CL
  - Decoder
  - Multiplexer

Lesson plan

- 1-bit ALU
- Logical ops: AND/OR/XOR/shift
- Arithmetic ops: addition, subtraction, multiplication, division

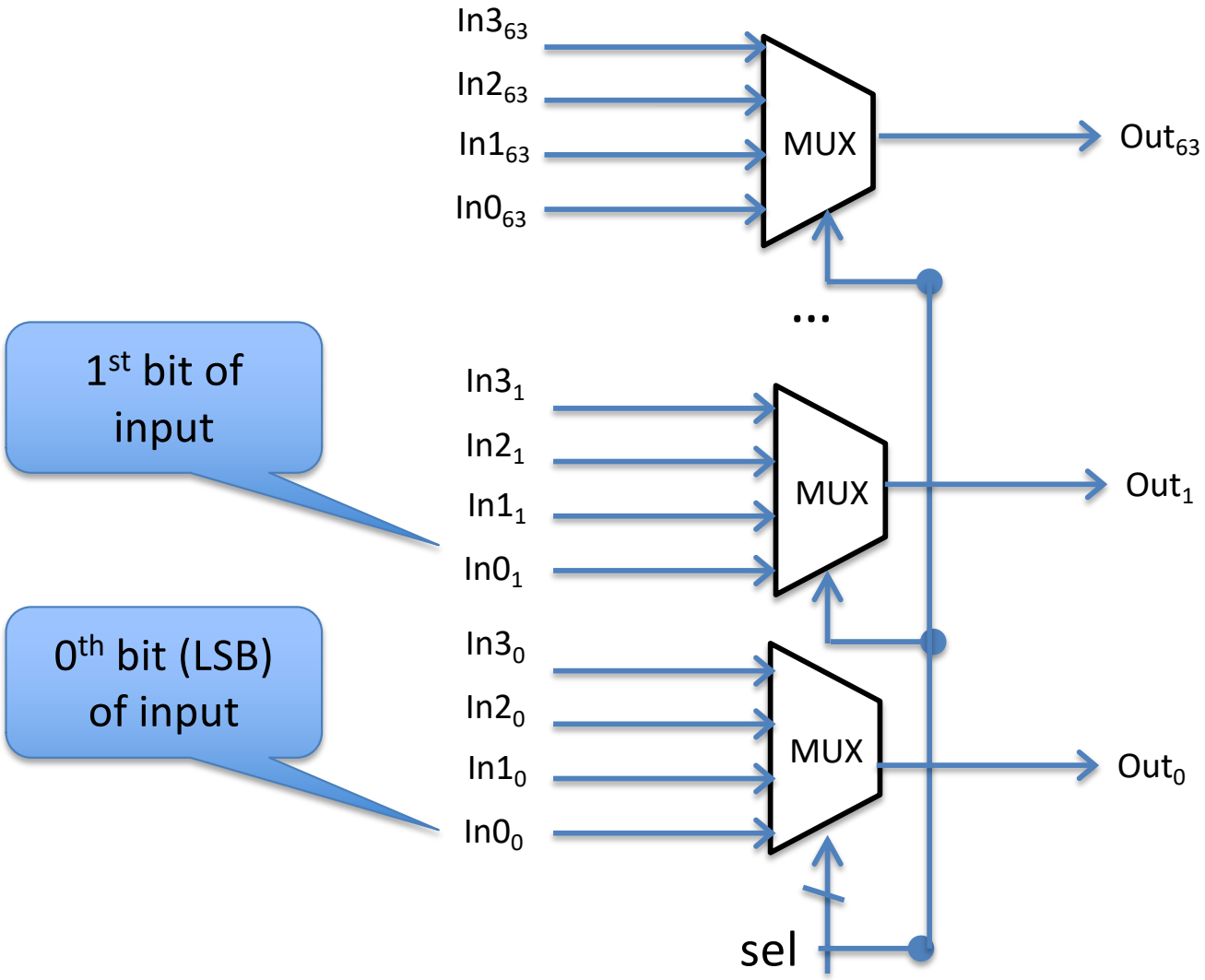
Array of logic elements

- So far, our circuits work on 1-bit inputs/outputs
- How to build circuits with n-bit inputs/outputs?

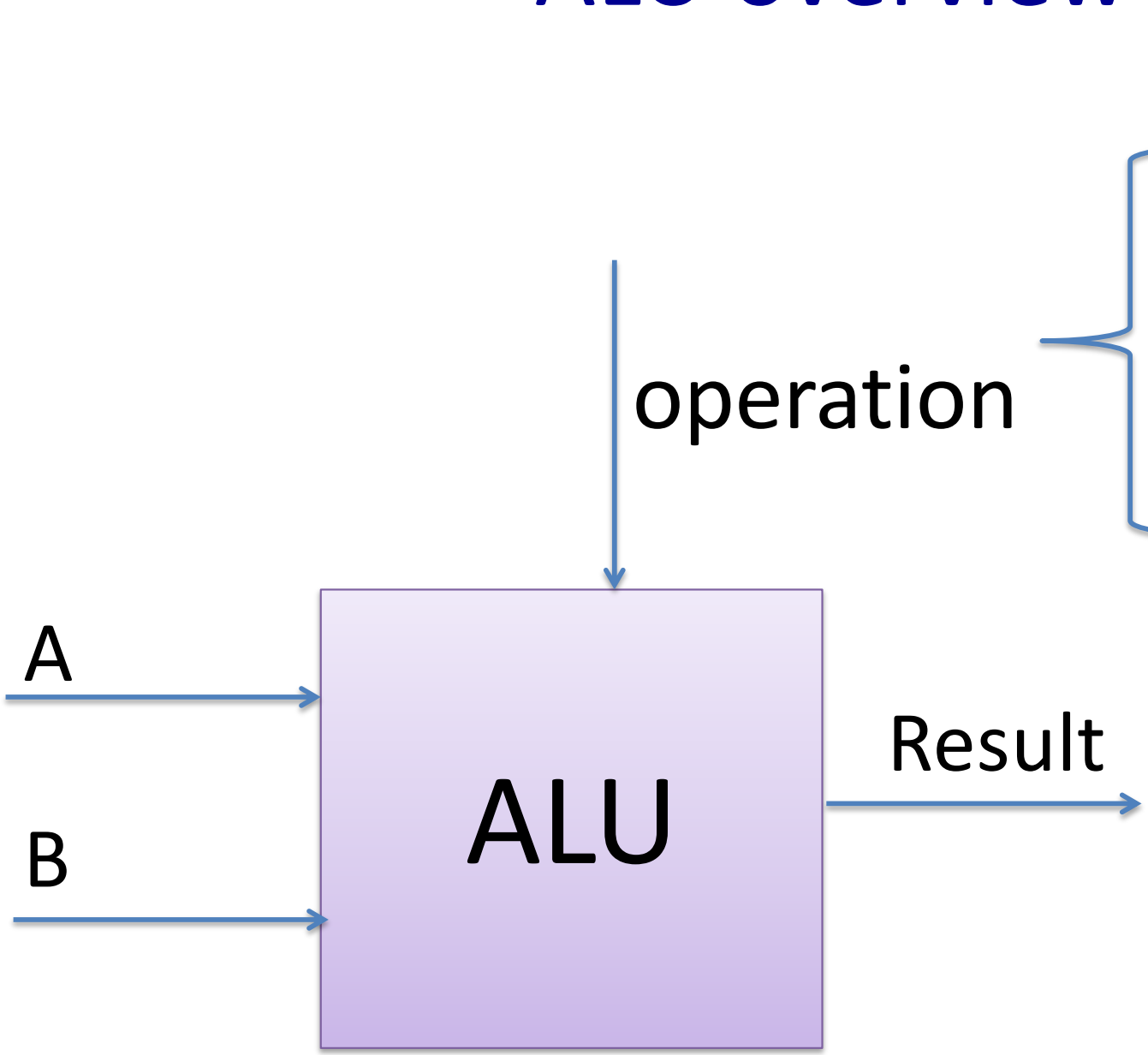


Array of logic elements

- 64-bit multiplexor: an array of 64 1-bit multiplexors



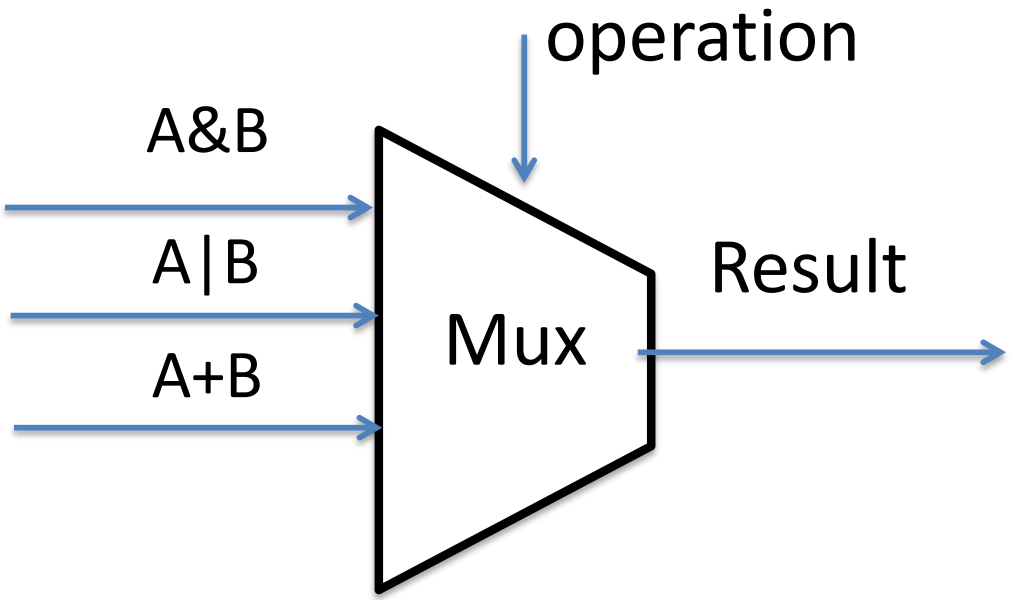
ALU overview



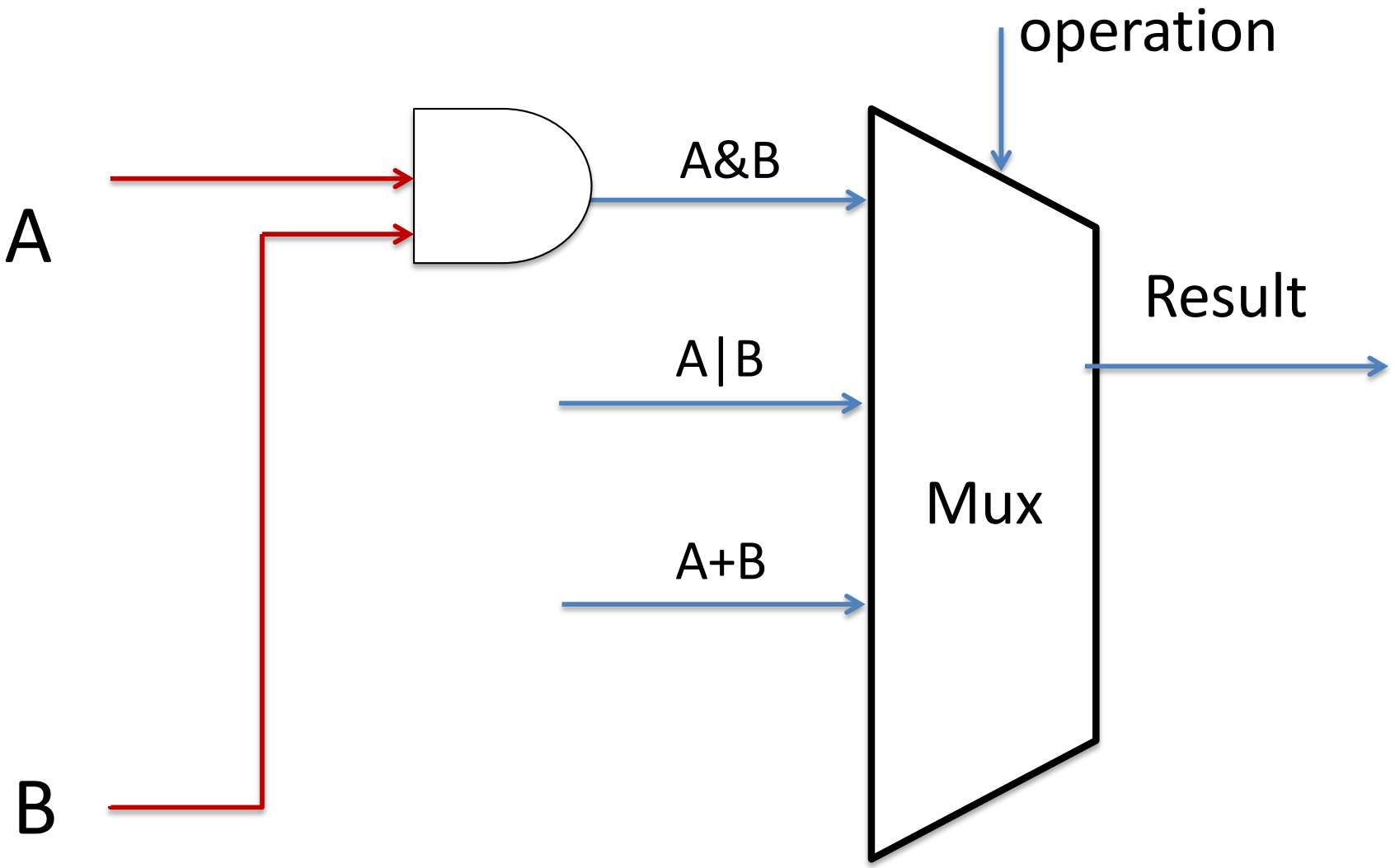
Example

Op	
00	A & B
01	A | B
10	A + B

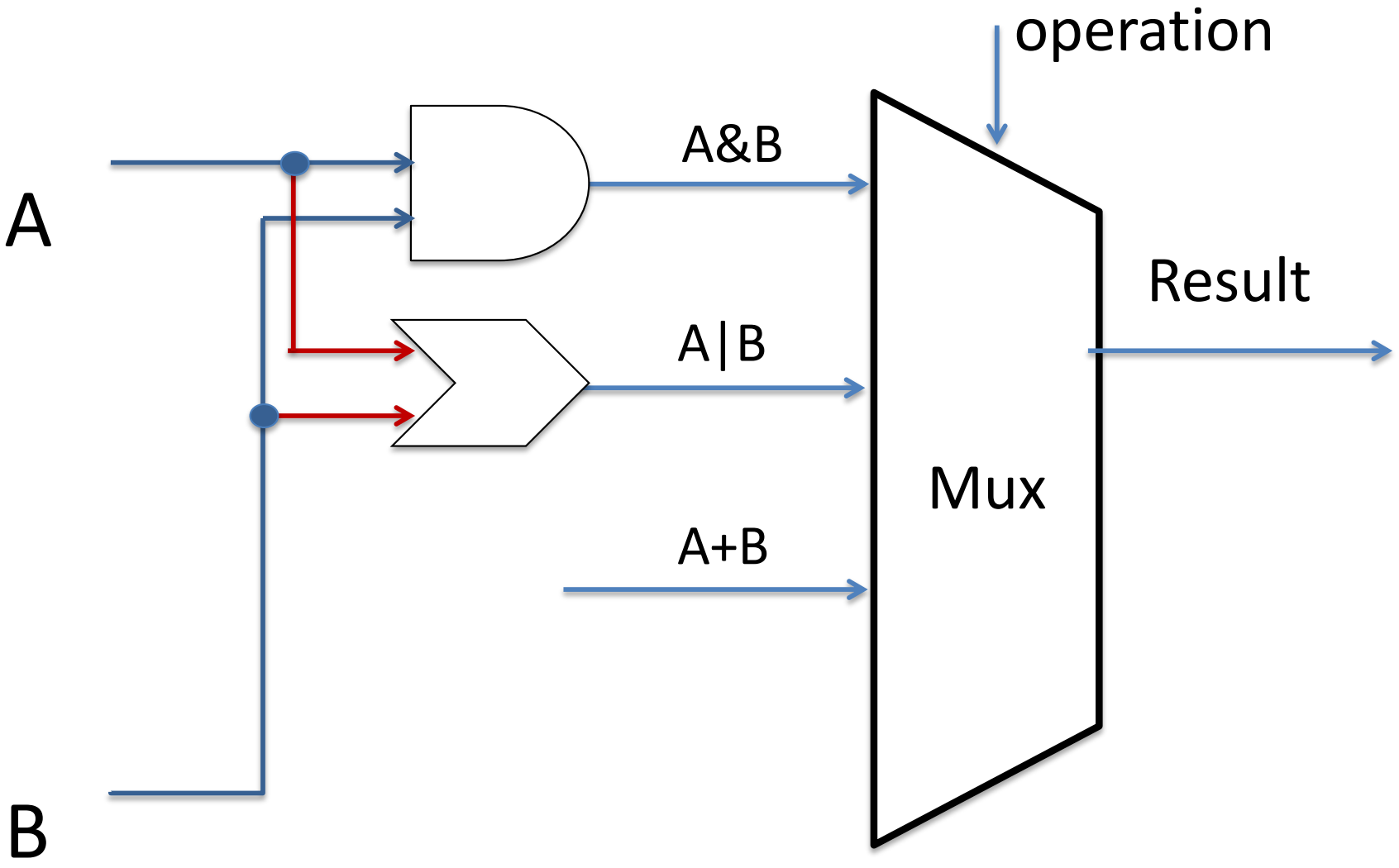
Implementing ALU: AND



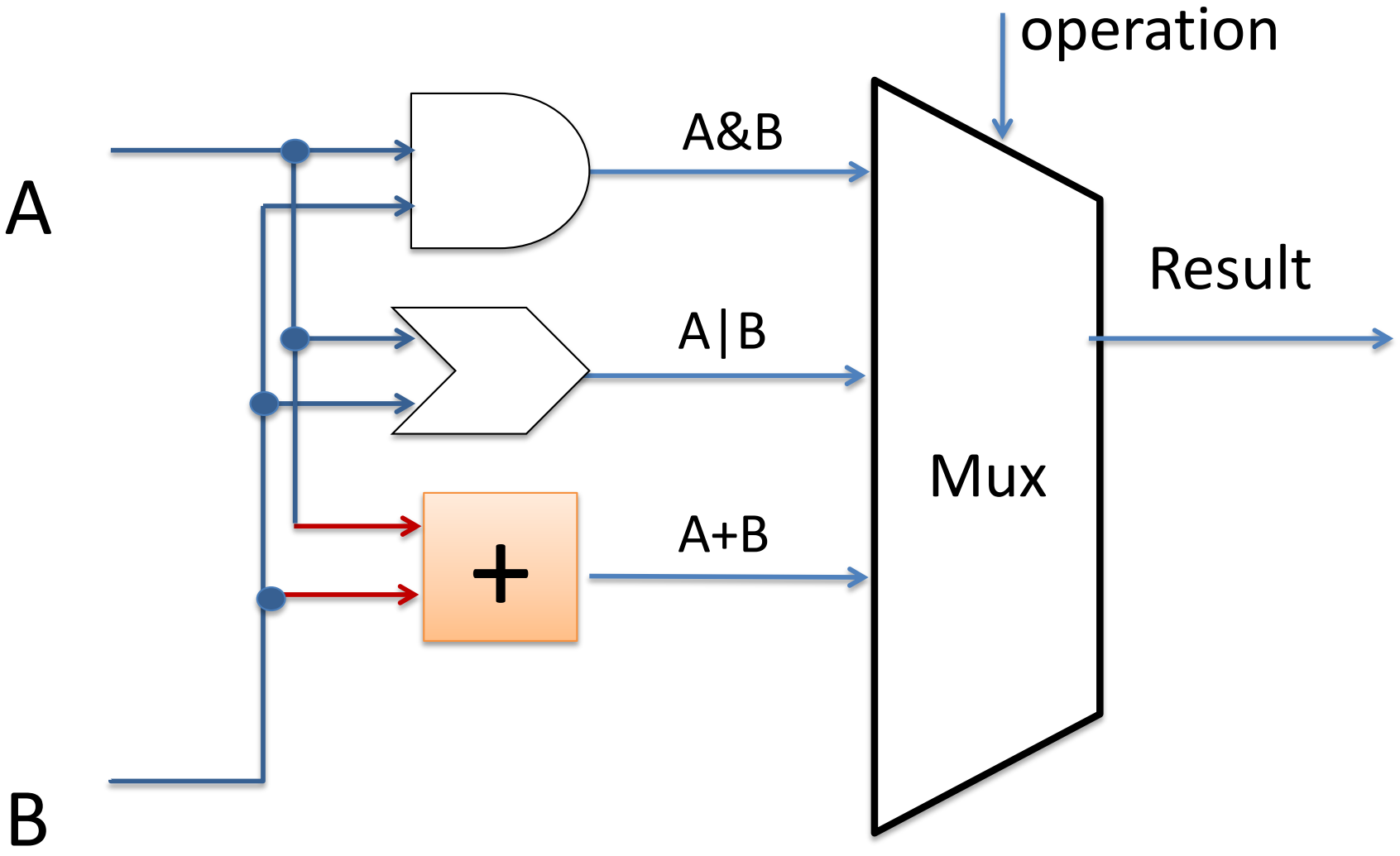
Implementing ALU: AND



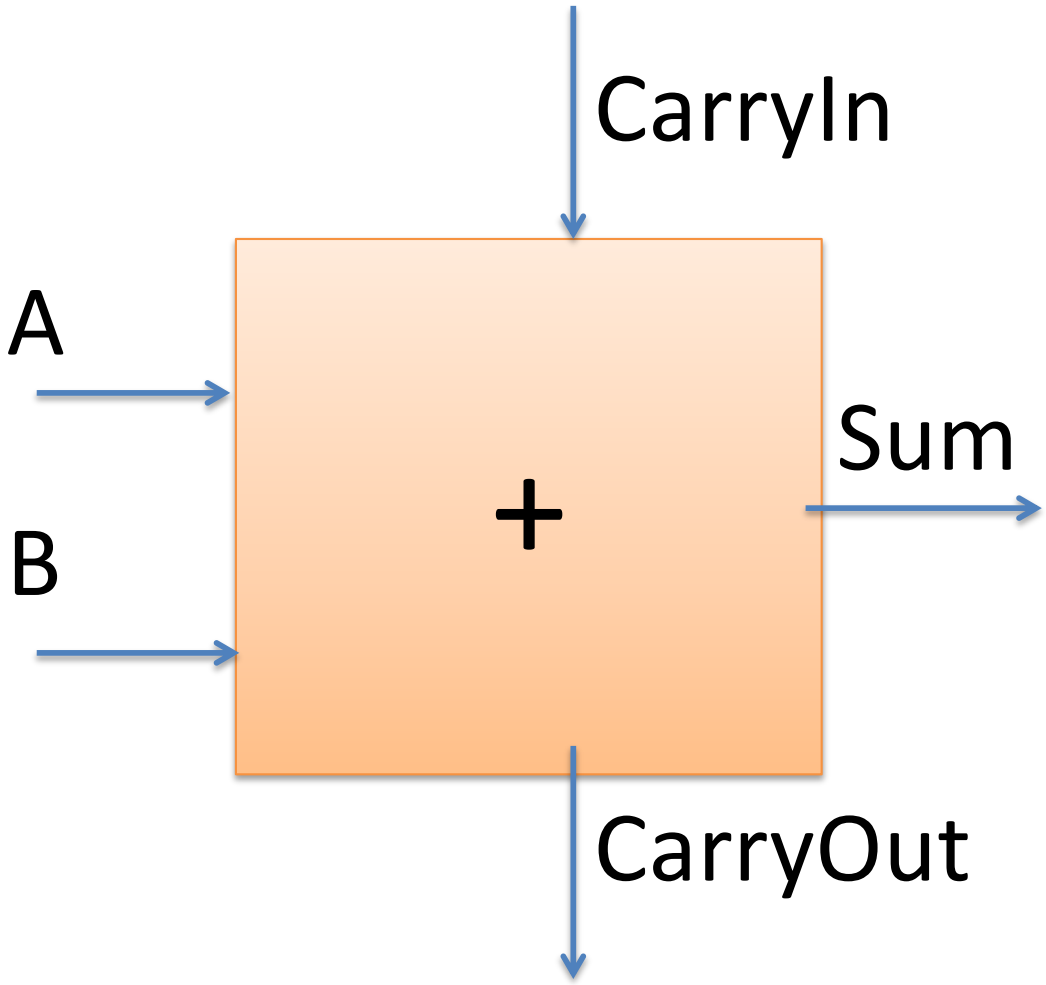
Implementing ALU: OR

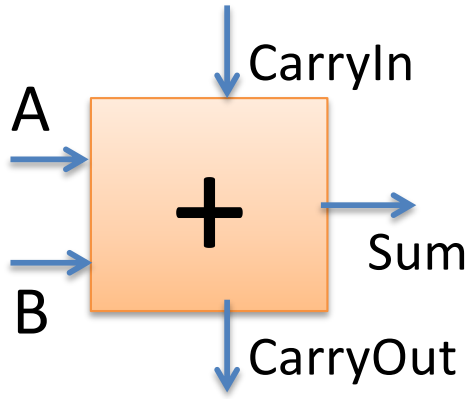


Implementing ALU: adder



1-bit adder





1-bit adder

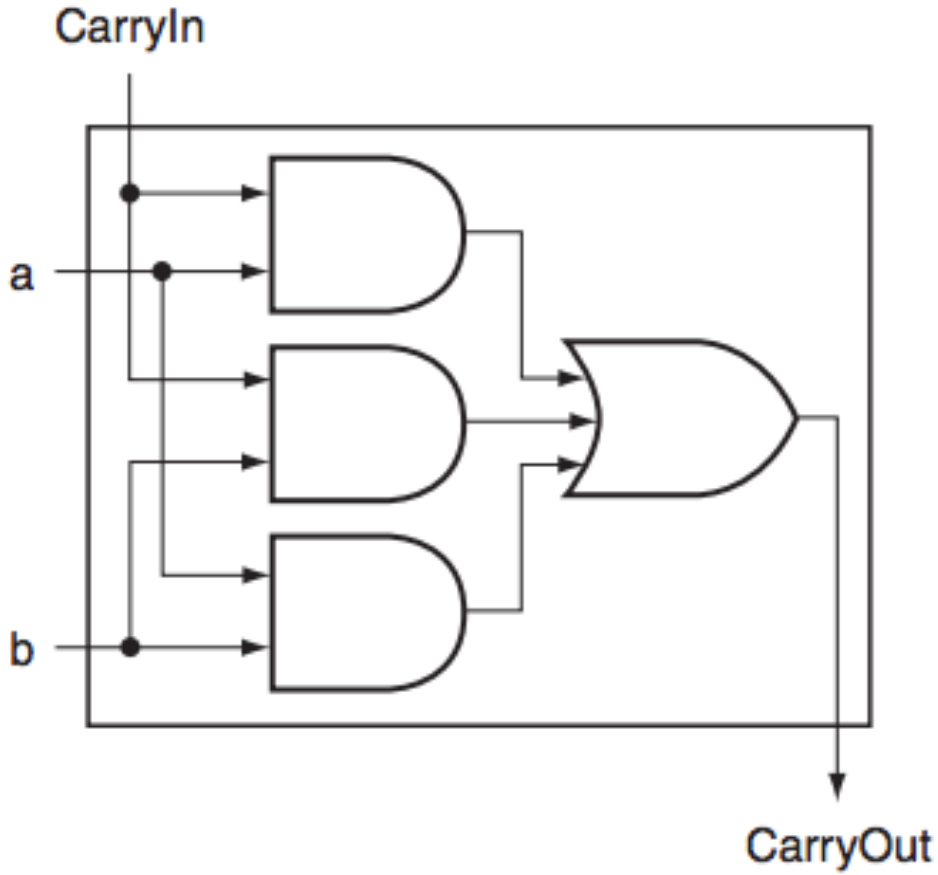
Inputs			Outputs	
a	b	CarryIn	CarryOut	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Brute force PLA

1-bit adder

- We can do better than PLA for CarryOut

$$\text{CarryOut} = (b \cdot \text{CarryIn}) + (a \cdot \text{CarryIn}) + (a \cdot b)$$



1-bit adder

$$\text{Sum} = (a \cdot \bar{b} \cdot \overline{\text{CarryIn}}) + (\bar{a} \cdot b \cdot \overline{\text{CarryIn}}) + (\bar{a} \cdot \bar{b} \cdot \text{CarryIn}) + (a \cdot b \cdot \text{CarryIn})$$

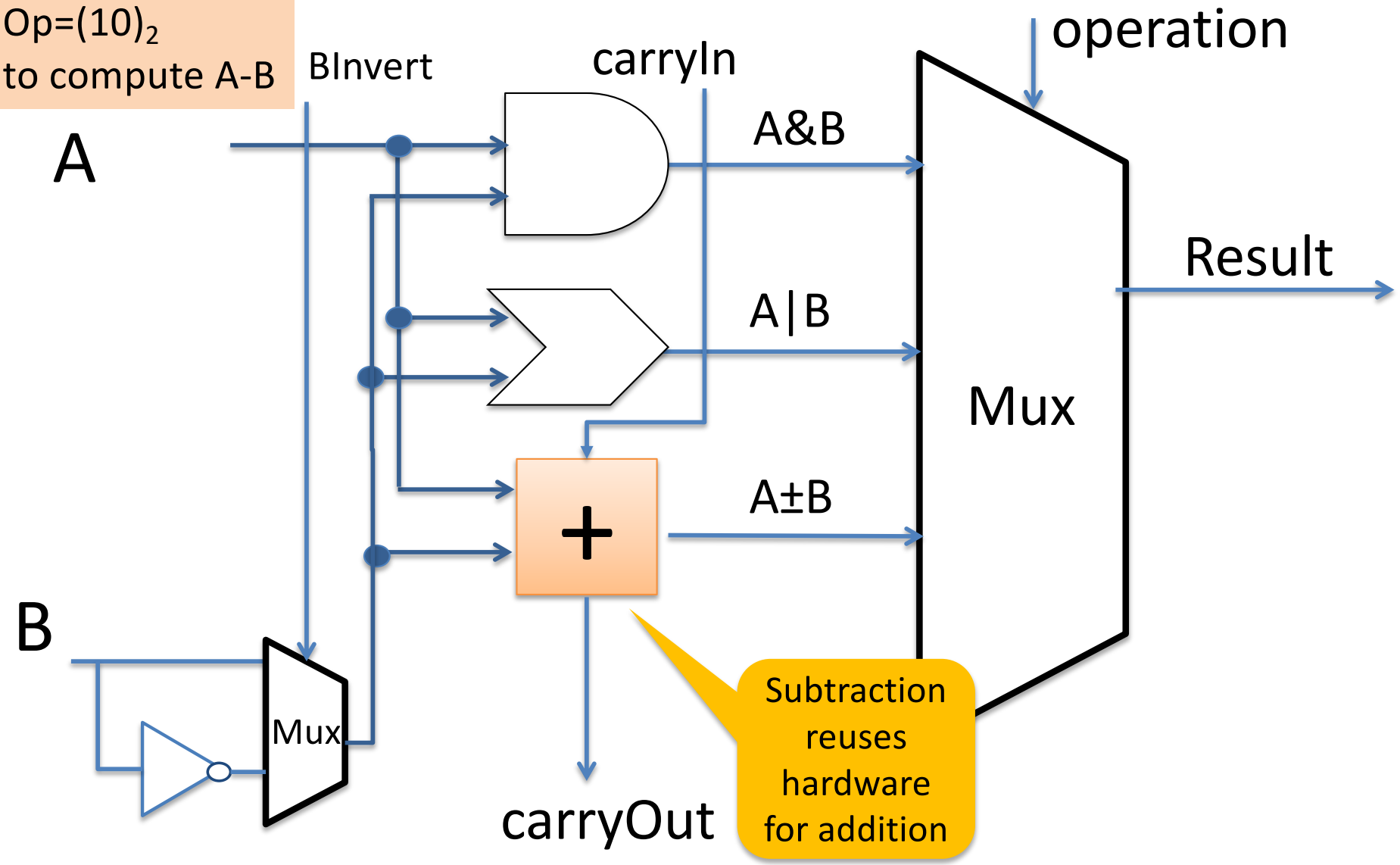
Inputs			Outputs	
a	b	CarryIn	CarryOut	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Subtraction

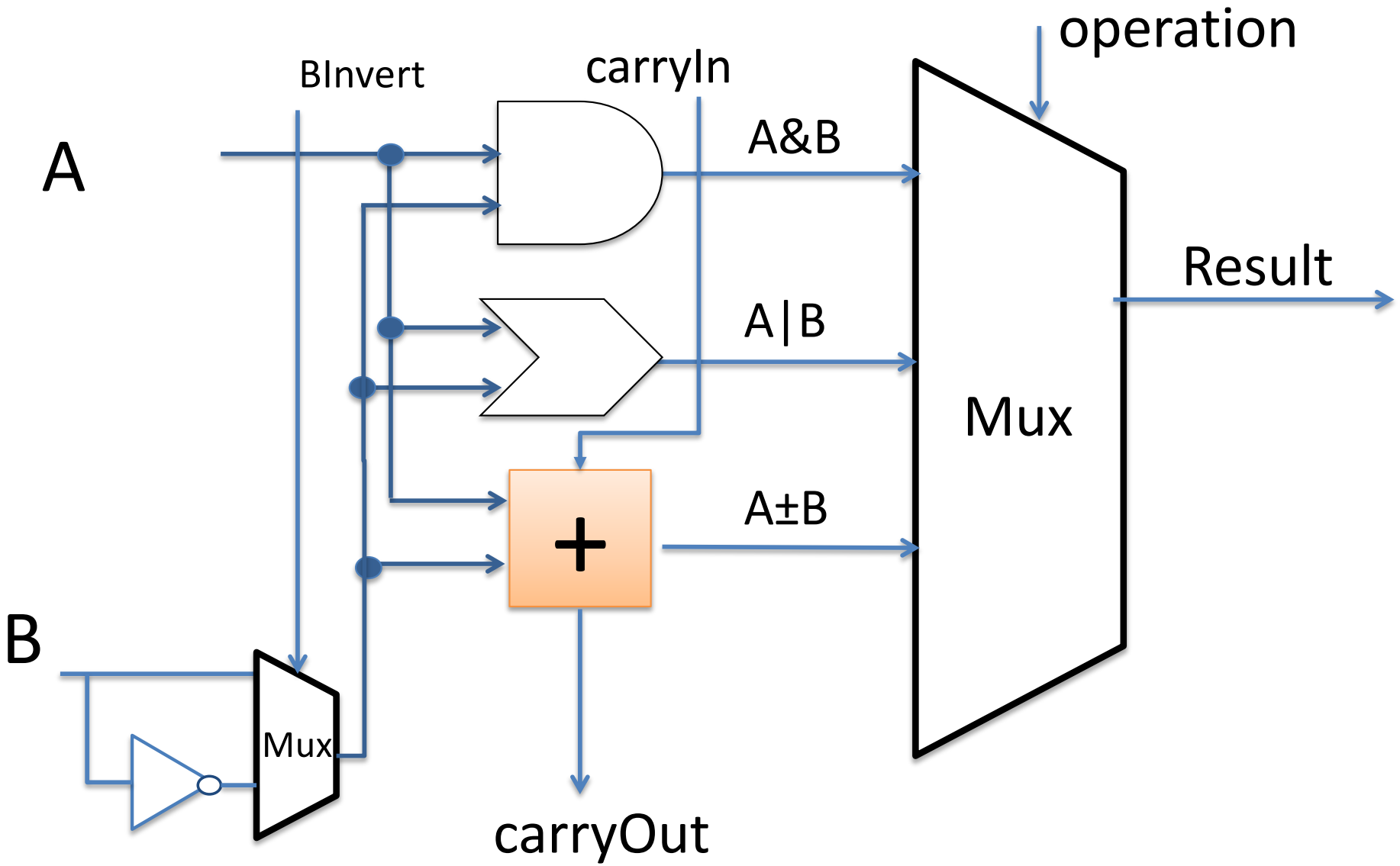
- Idea: a – b = a + (-b)
- How to calculate 2's complement?

Subtraction in1-bit ALU

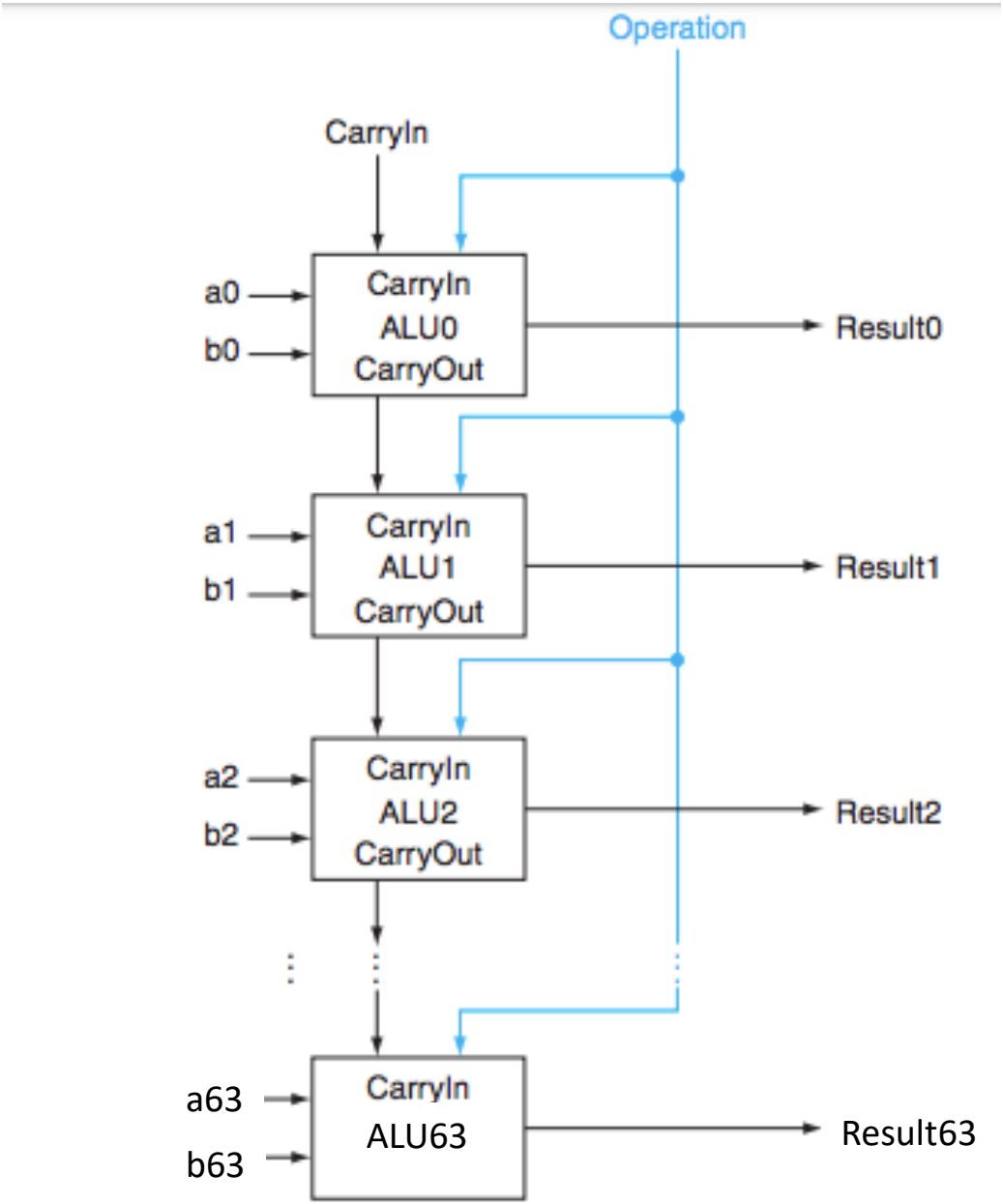
Set Binvert=1
carryIn=1
Op=$(10)_2$
to compute A-B



Extend 1-bit ALU to 64-bit

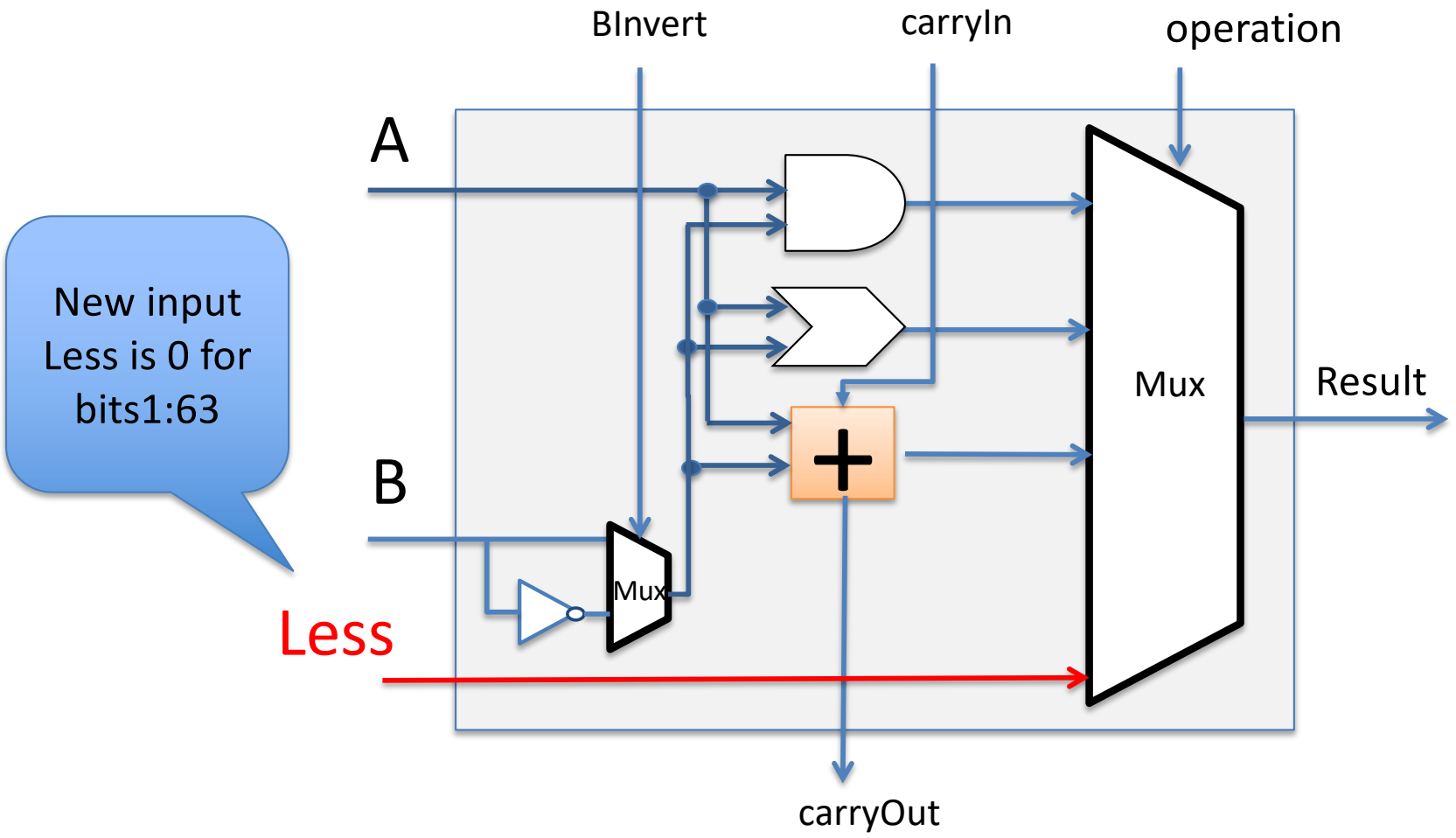


Simple 64-bit adder: ripple carry



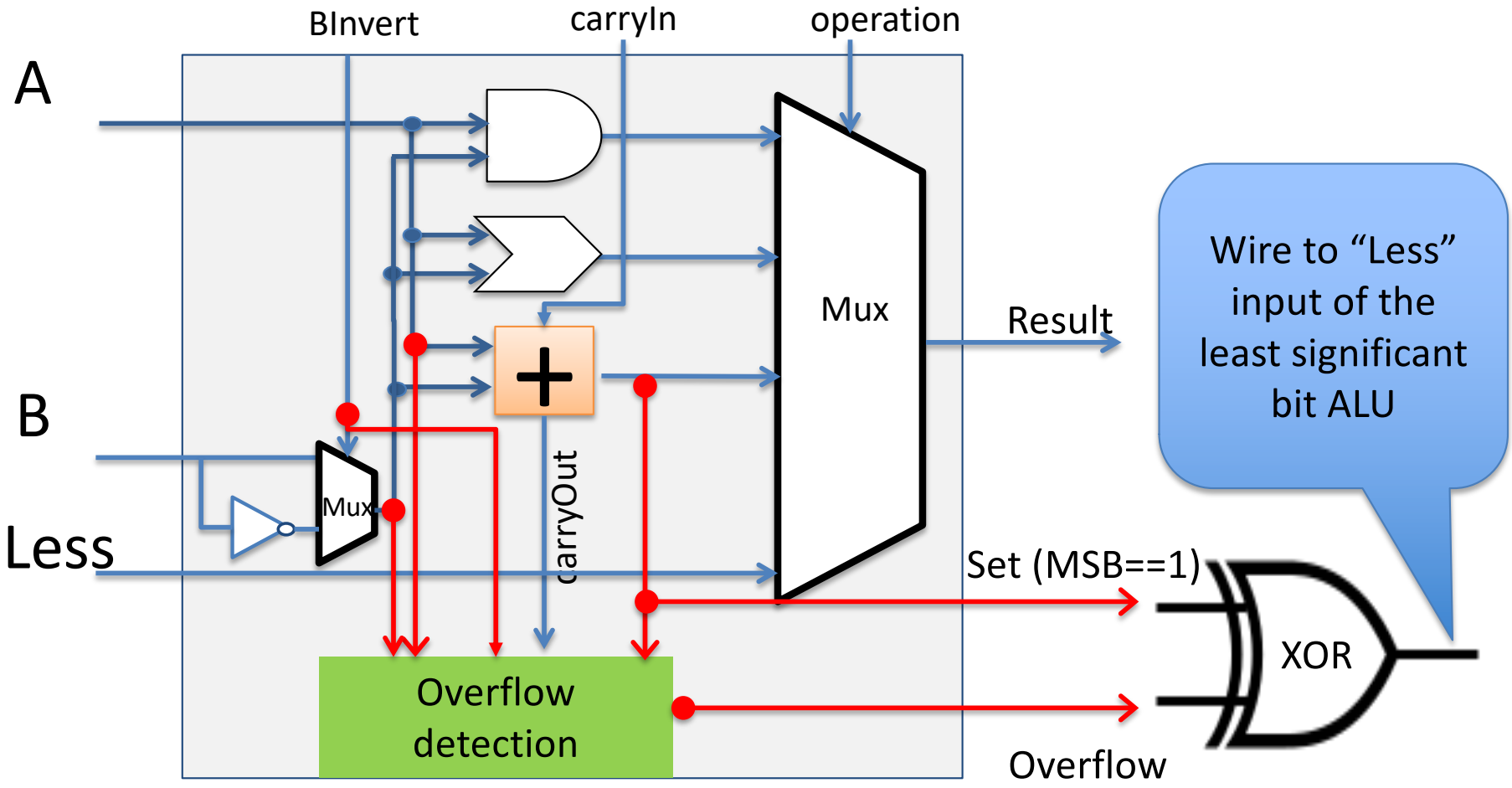
Extend ALU to include `slt`

- RISC-V slt instruction: Result = (A < B) ? 1 : 0 Signed
  - X86 equivalent: cmpq %rbx,%rax setl %rcx

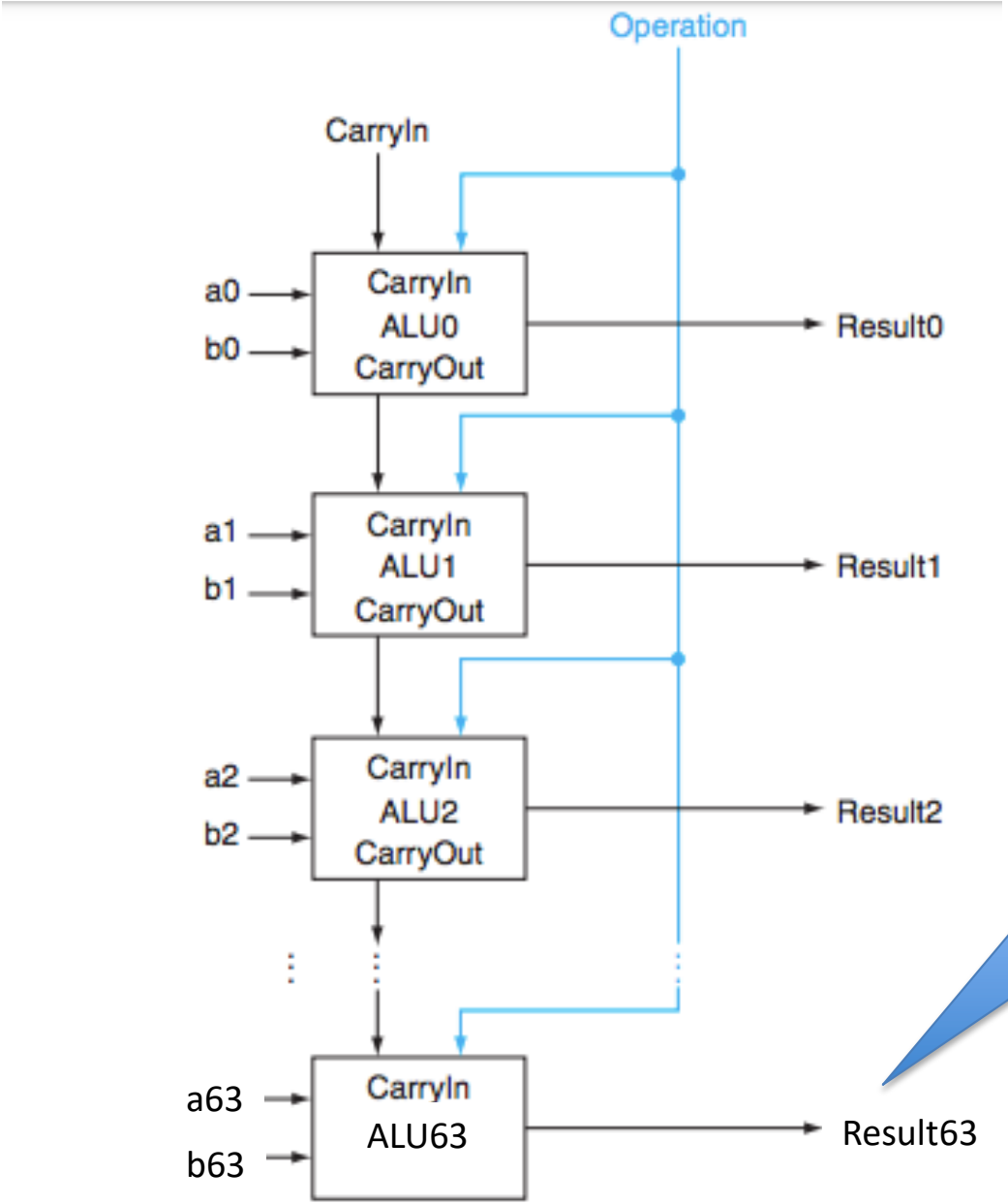


Extend ALU to include `slt`

- A < B iff:
  - (A-B) is negative (MSB is 1)
  - (A-B) overflowed
  - But not both



Downside of ripple carry?



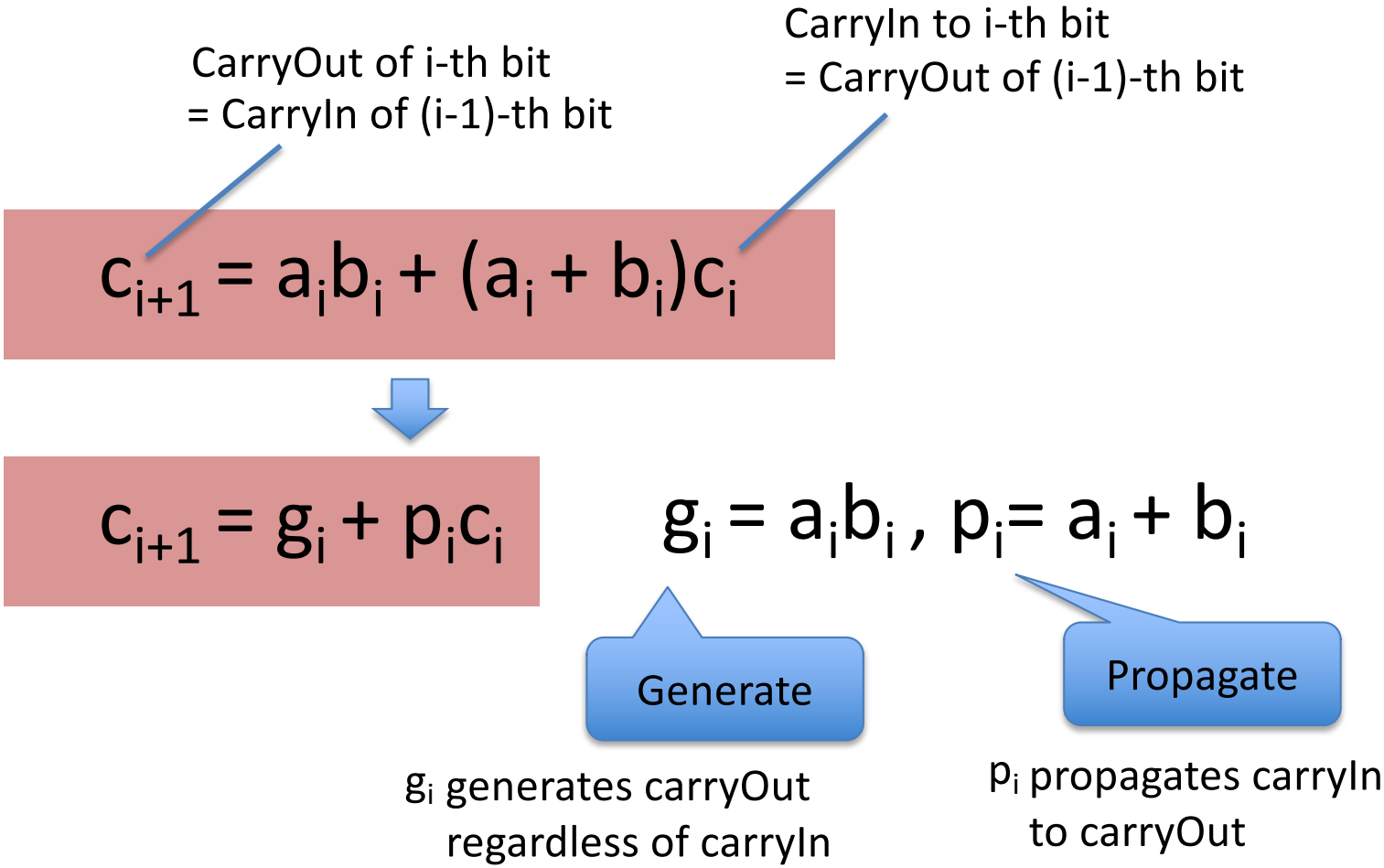
Must wait for sequential evaluation of all 64 1-bit adders

In search of faster adder

- Ripple carry:
  - Delay: 64, Gate count: 64*c
- Brute-force (truth table->PLA)
  - Delay: 2, Gate count: $O(2^{64+64})$
- Clever designs in between?
- Idea #1: (Carry lookahead) compute multiple carry-bits at a time

Faster adder: carry lookahead

- Idea #1: (Carry lookahead) compute multiple carry-bits at a time



Faster adder: carry lookahead

- Idea #1: (Carry lookahead) compute multiple carry-bits at a time

  Computing all carry-bits of a 4-bit adder:

$$c1 = g0 + (p0 \cdot c0)$$

$$c2 = g1 + (p1 \cdot g0) + (p1 \cdot p0 \cdot c0)$$

$$c3 = g2 + (p2 \cdot g1) + (p2 \cdot p1 \cdot g0) + (p2 \cdot p1 \cdot p0 \cdot c0)$$

$$c4 = g3 + (p3 \cdot g2) + (p3 \cdot p2 \cdot g1) + (p3 \cdot p2 \cdot p1 \cdot g0) + (p3 \cdot p2 \cdot p1 \cdot p0 \cdot c0)$$

  Delay? 3

  4-bit ripple carry
  delay: 2 * 4

Faster adder: carry lookahead

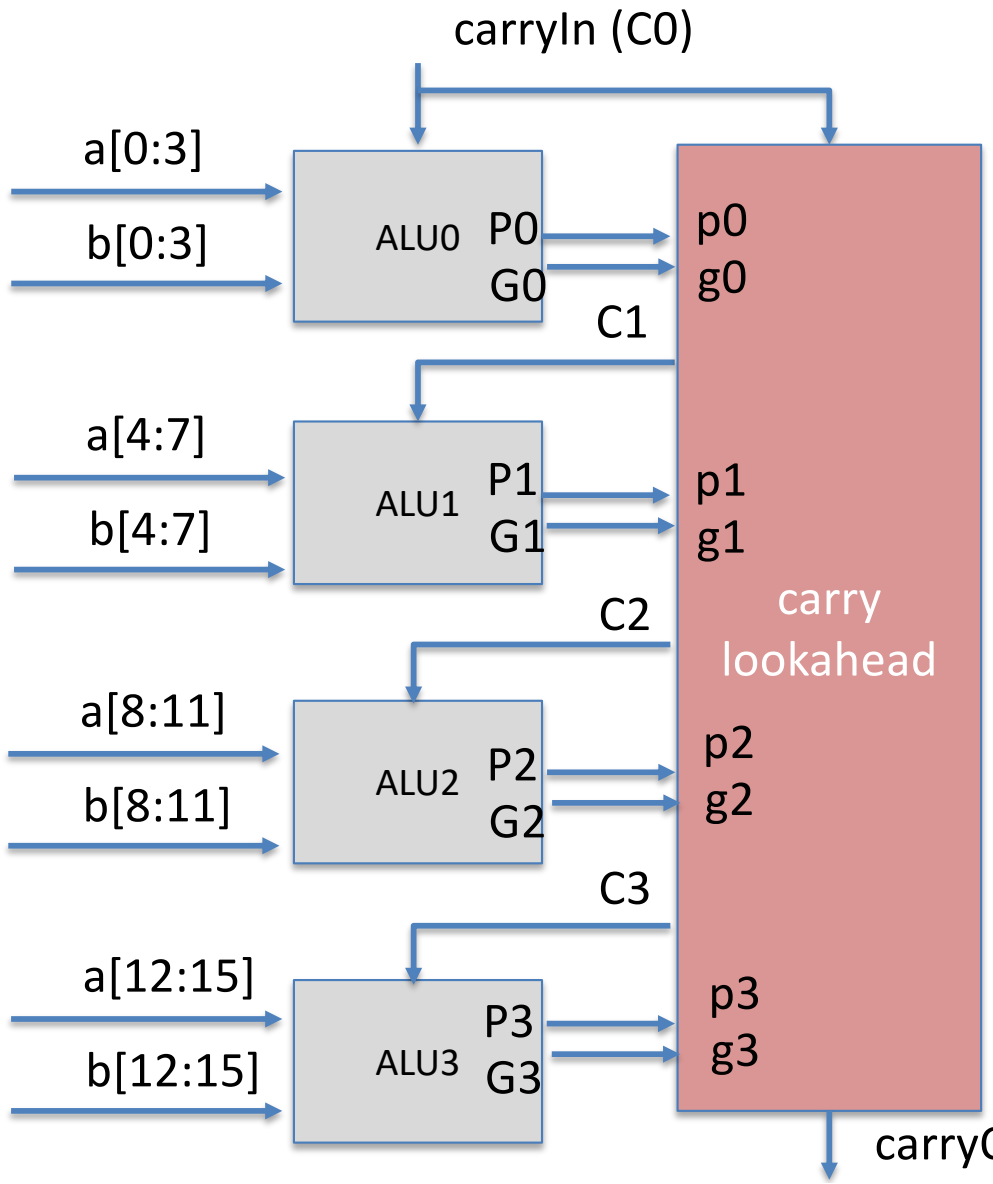
- Idea #1: (Carry lookahead) compute multiple carry-bits at a time

  Computing all result bits in a 4-bit adder:

$$s_i = \overline{c_i} \cdot a_i \cdot b_i + c_i \cdot \overline{a_i} \cdot b_i + c_i \cdot a_i \cdot \overline{b_i} \ , \ i = 0, .., 3$$

Faster adder: carry lookahead

- Build a 16-bit adder with carry-ahead 4-bit adders



P0 = p3· p2· p1· p0

P1 = p7· p6· p5· p4

P2 = p11· p10· p9· p8

P3 = p15· p14· p13· p12

G0 = g3+ p3·g2 +
p3· p2·g1+ p3· p2·p1 ·g0

G1 = g7+ p7·g6 +
p7· p6·g5+ p7· p6·p5 ·g4

G2 = g11+ p11·g10 +
p11· p10·g9+ p11· p10·p9·g8

G3 = g15+ p15·g14 +
p15·p14·g13+ p15·p14·p13·g12