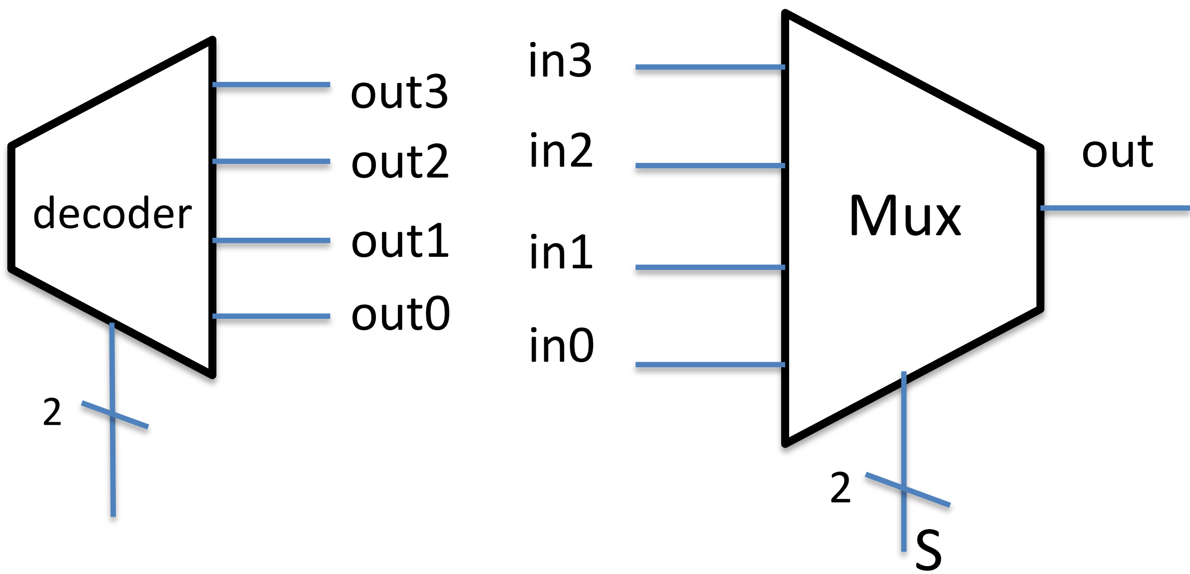
Building an ALU

Jinyang Li

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 (implemented in PLA, programmable logic array)
- Common CL
  - Decoder
  - Multiplexer

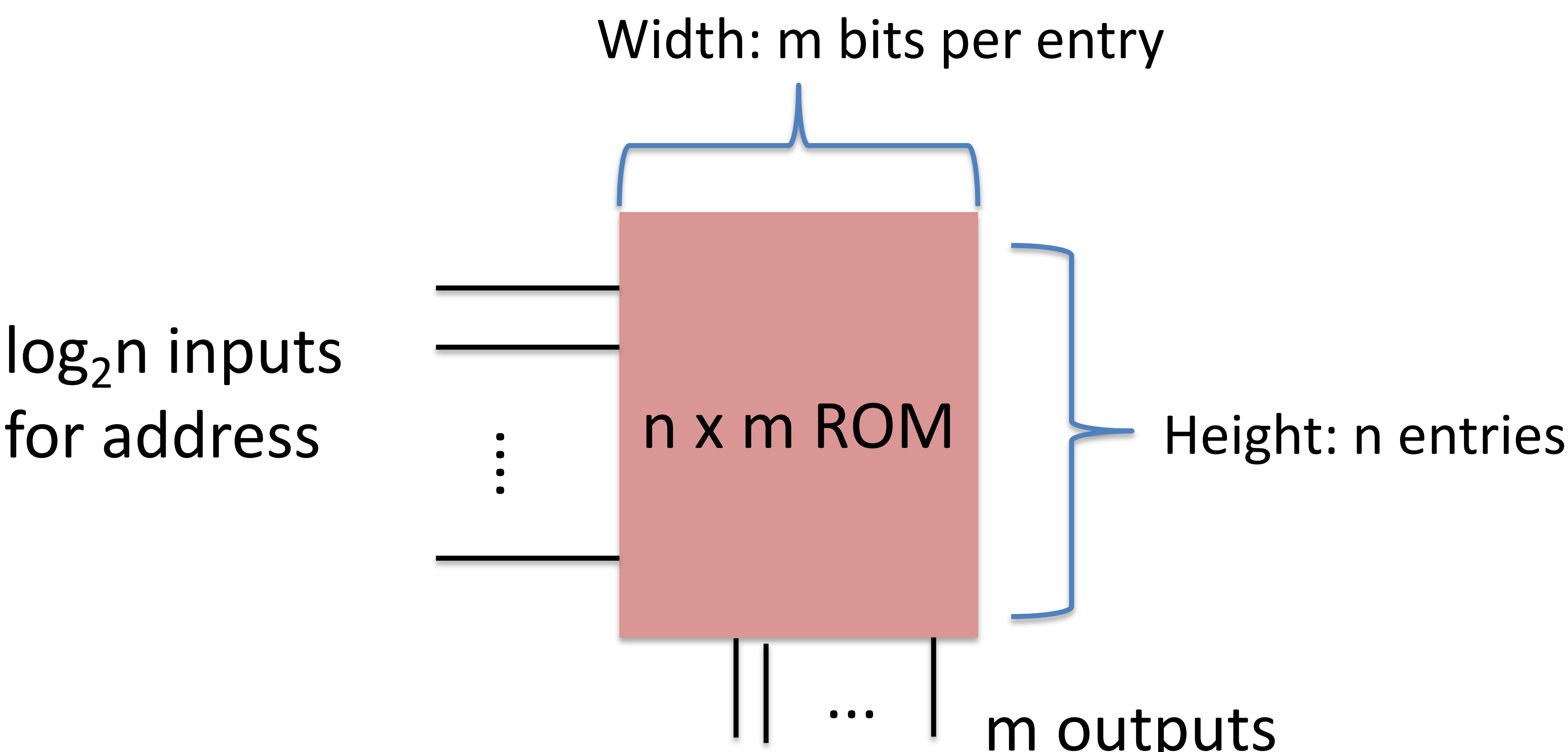


Lesson plan

- ROM (another way to implement CL)
- ALU
  - Logical ops: AND/OR
  - Arithmetic ops: addition, subtraction…

ROM (read-only memory)

- A combinatorial component for storing (fixed) data
- Programmed in the factory or field



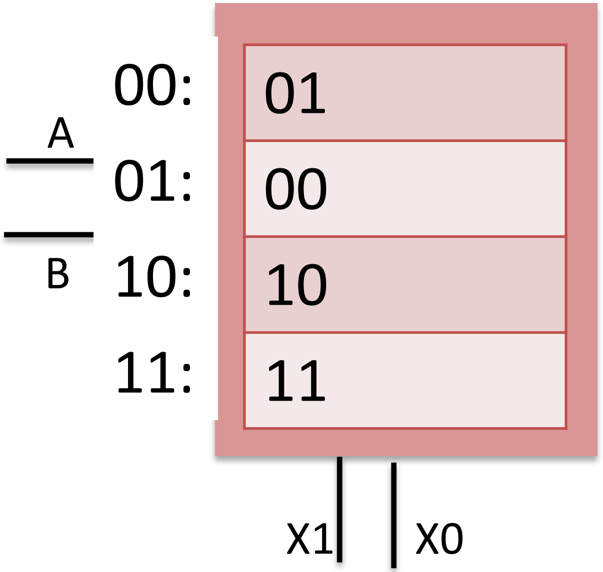
ROM (read-only memory)

- A n x m ROM can store the truth table for m functions defined on $\log_2 n$ variables.

$$X_1 = A$$

$$X_0 = \bar{A} \bullet \bar{B} + A \bullet B$$

A	B	X1	X0
0	0	0	1
0	1	0	0
1	0	1	0
1	1	1	1

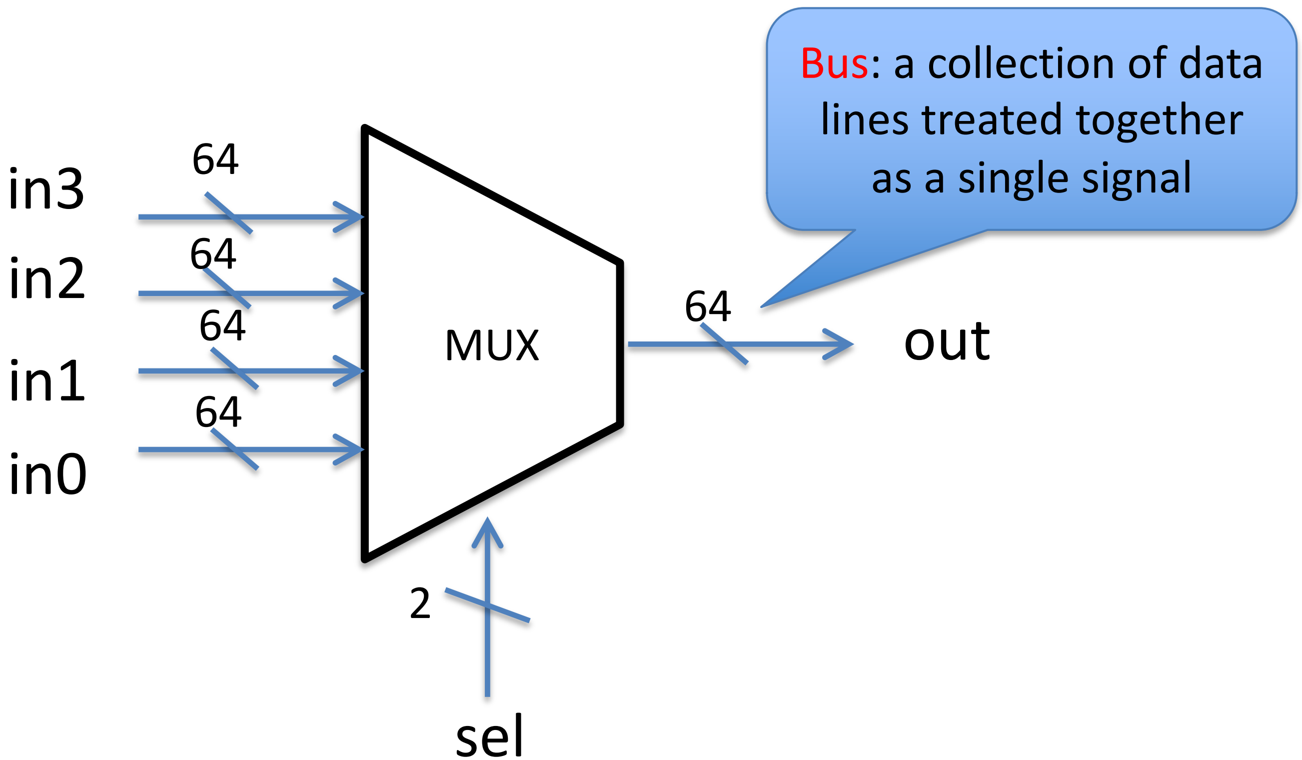


ROM (read-only memory)

- Both ROM and PLA can impl. boolean functions
- ROM is not as efficient for sparse functions
  - # of entries grows exponentially with inputs
- ROM is easier to change if function changes

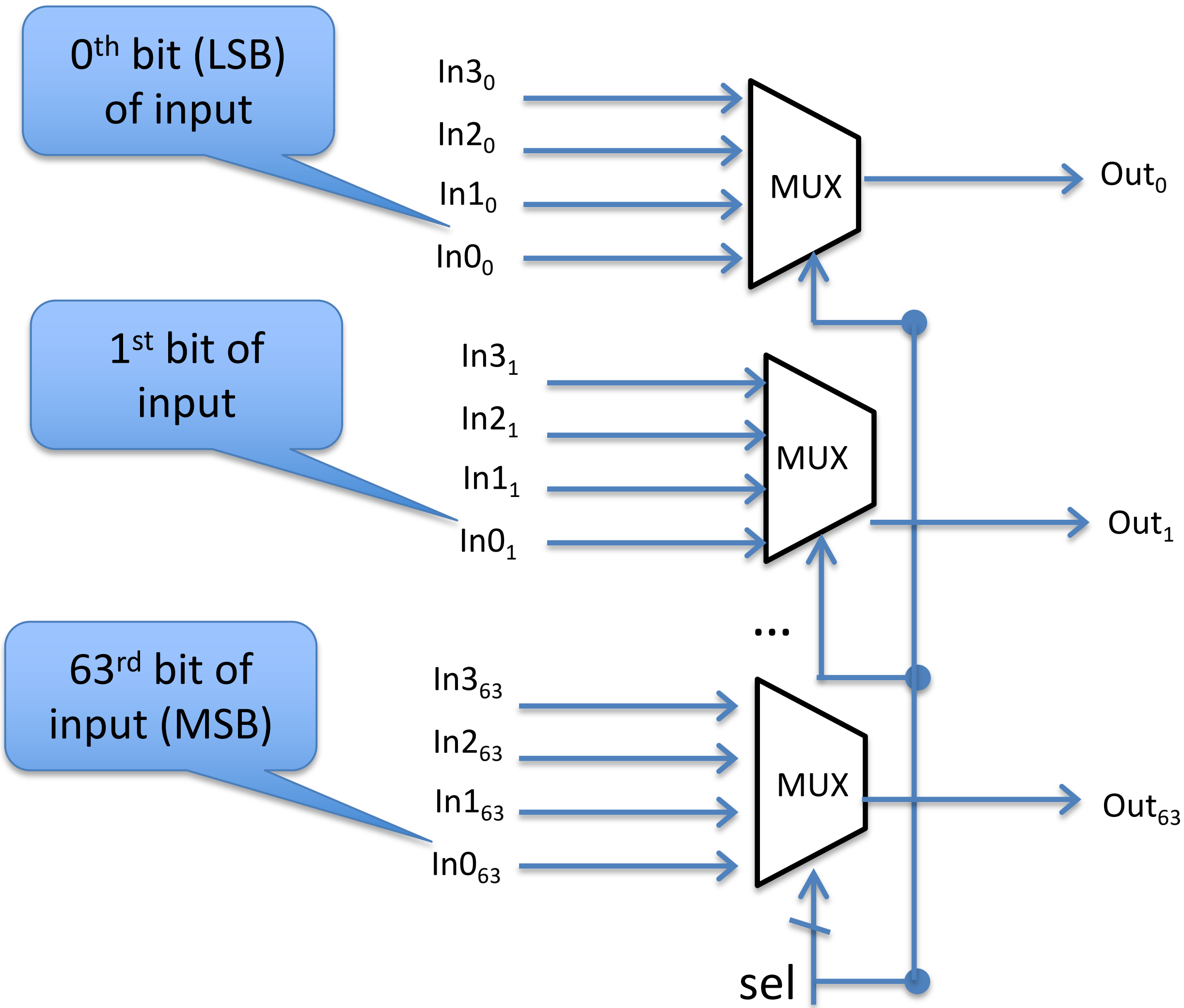
Array of logic elements

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

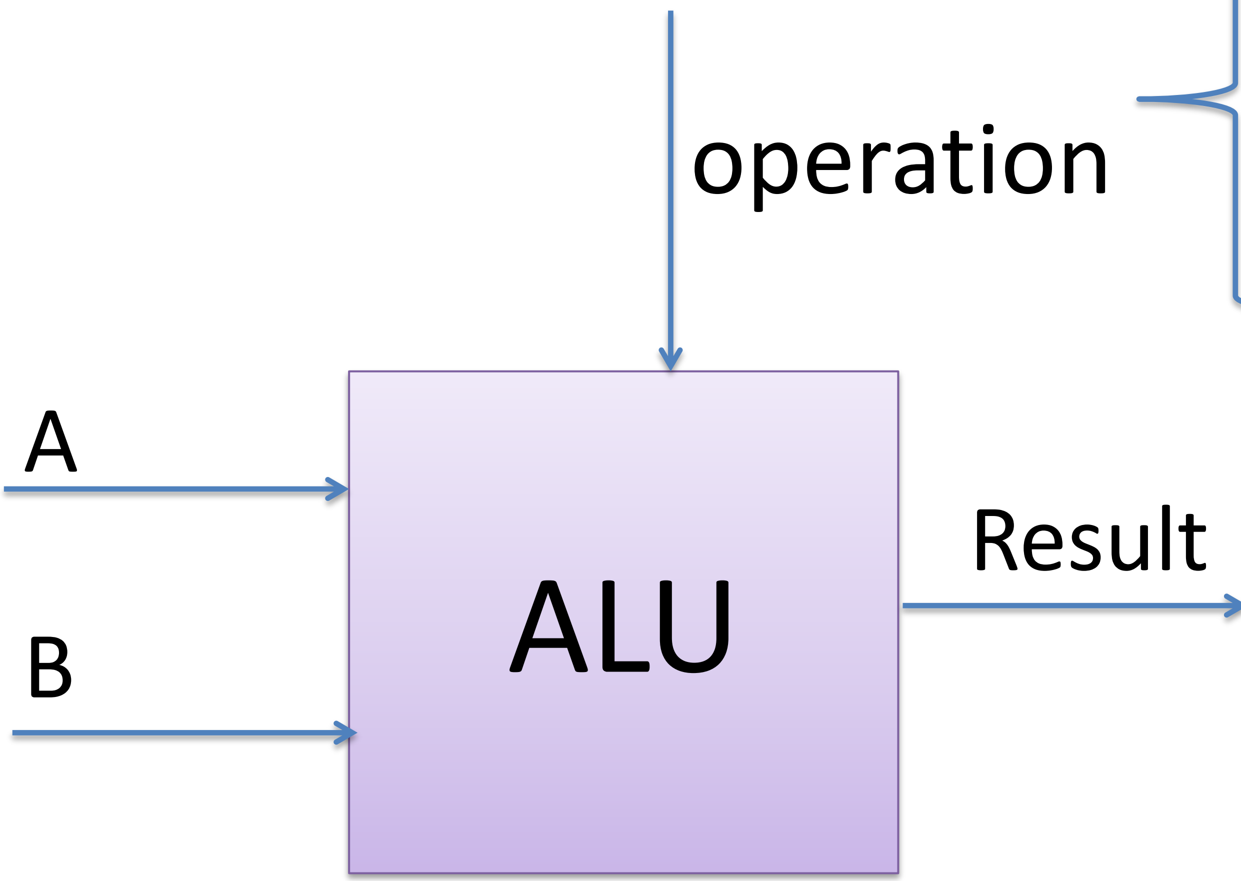


Array of logic elements

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



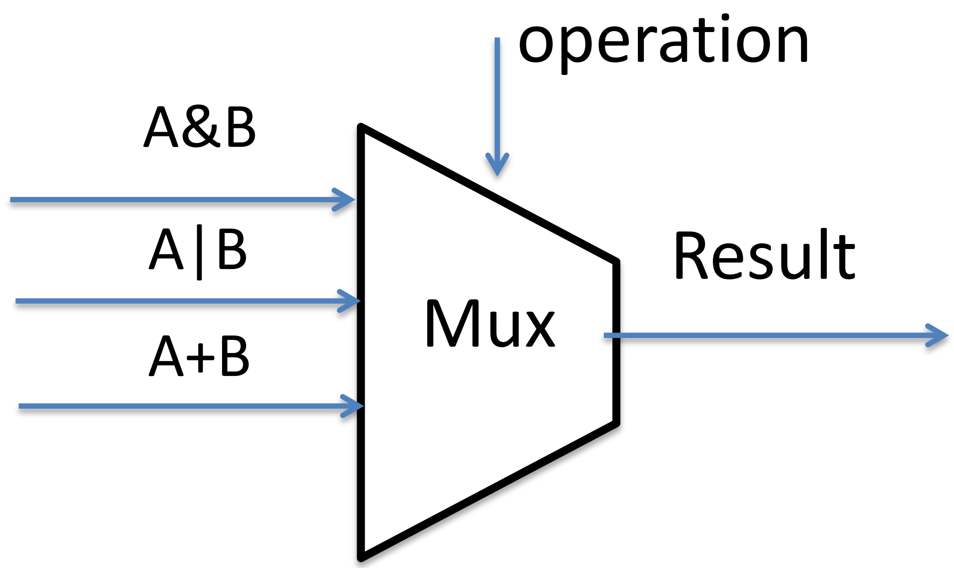
ALU overview



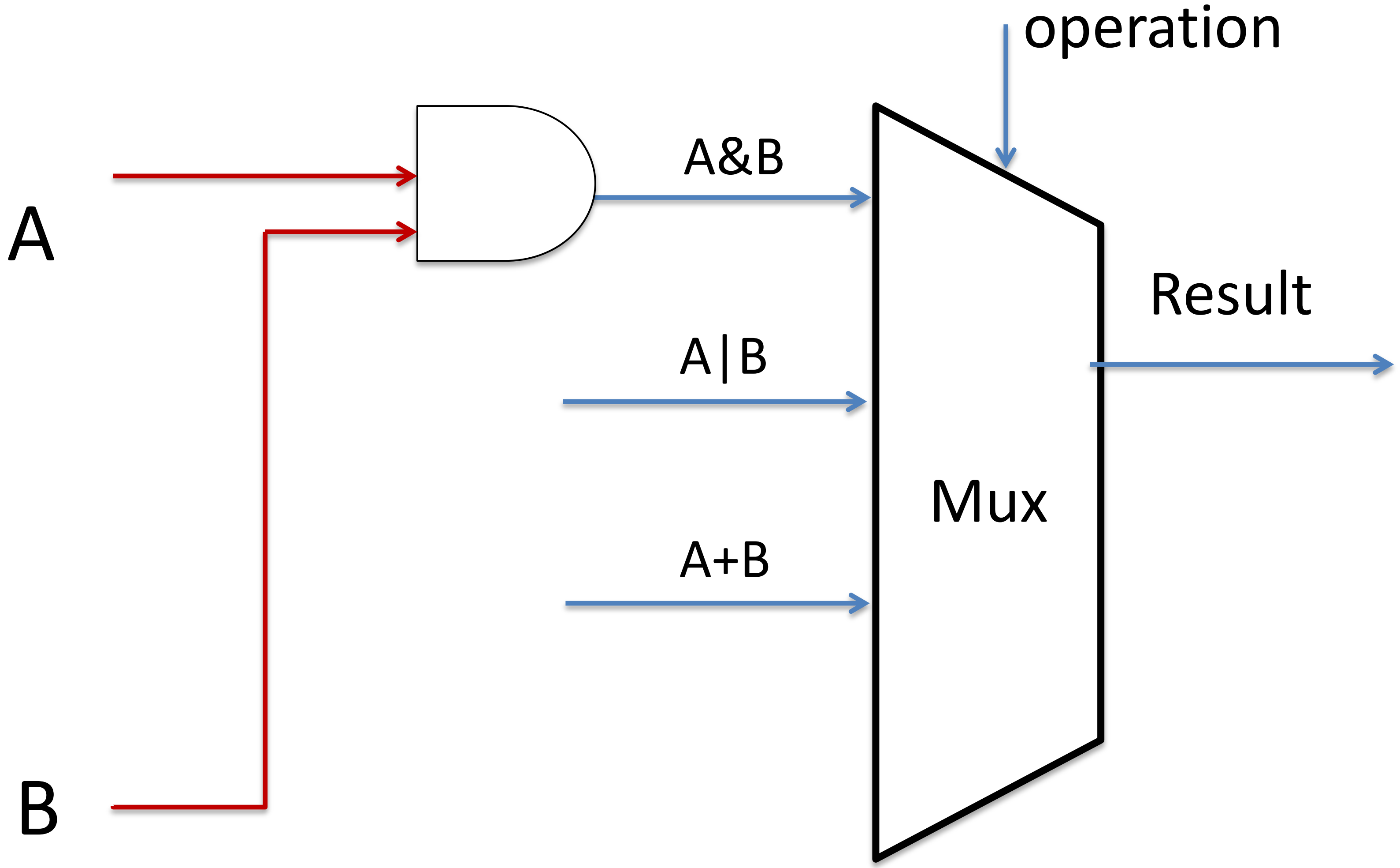
Example

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

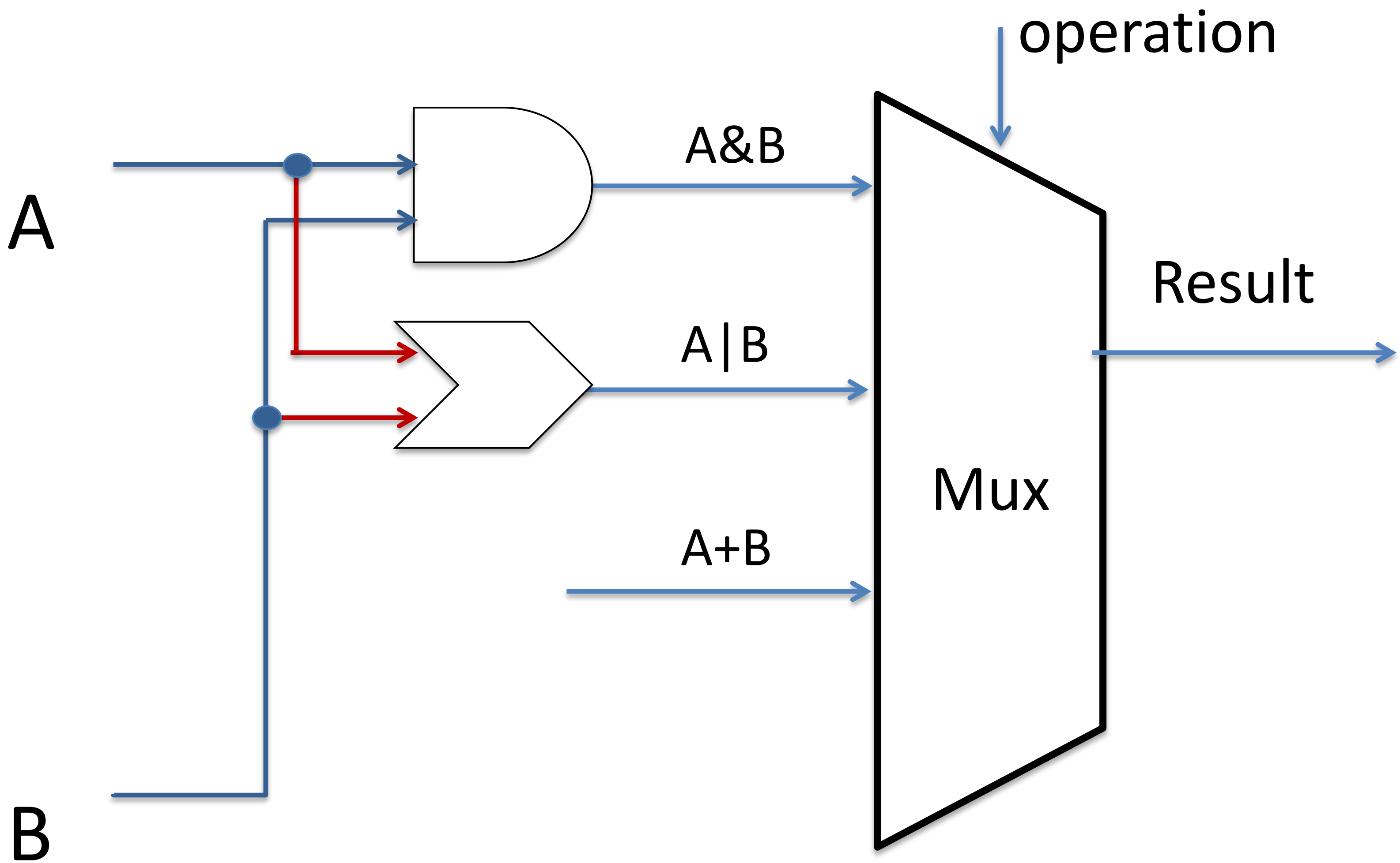
Implementing ALU: AND



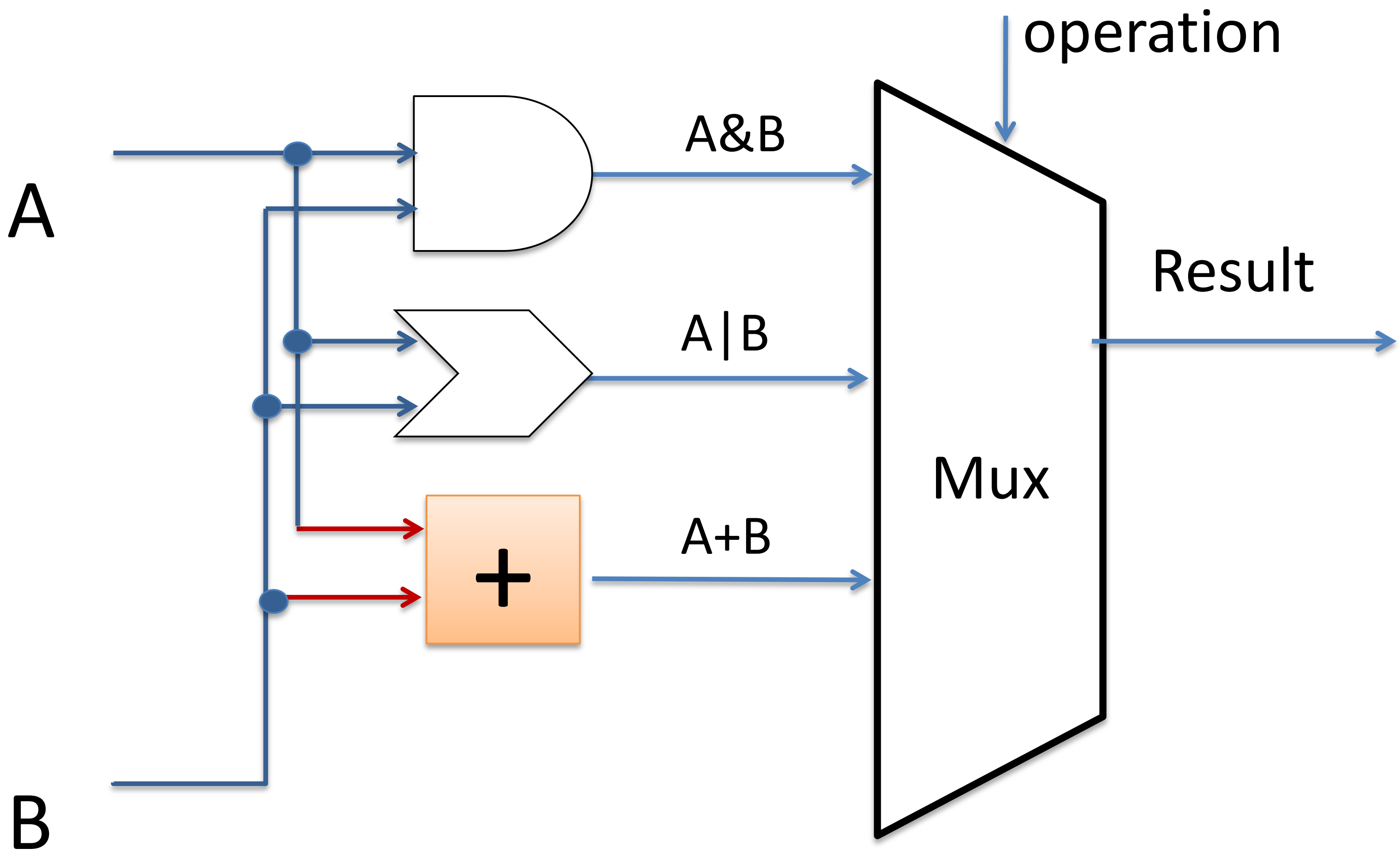
Implementing ALU: AND



Implementing ALU: OR



Implementing ALU: adder



Implementing the adder: 1-bit adder

- Recall how base-2 addition works

carryOut at pos3

carryOut at pos2 = carryIn at pos3

carryOut at pos1 = carryIn at pos2

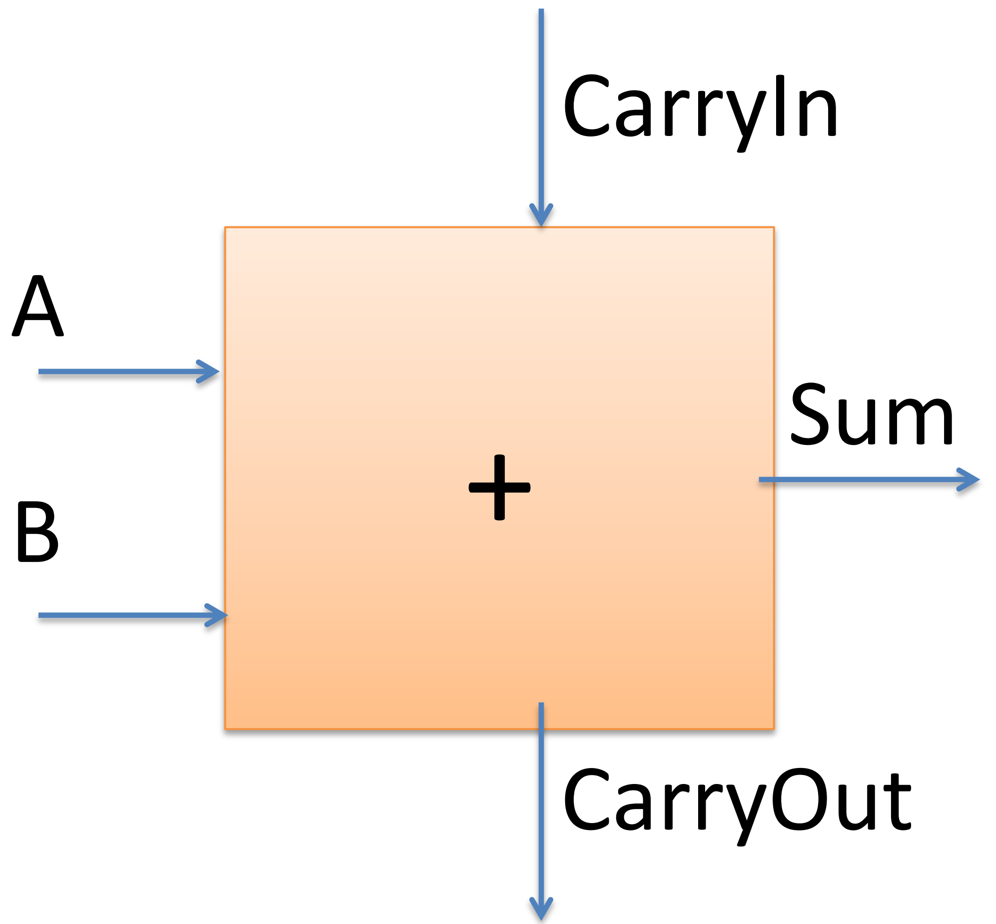
carryOut at pos0 = carryIn at pos1

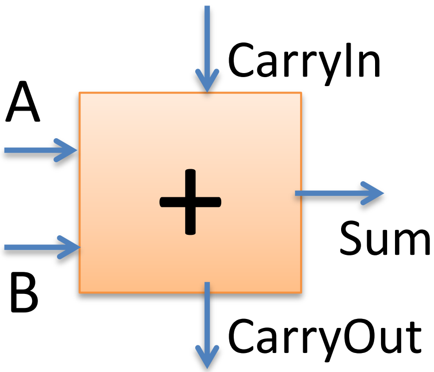
carryIn at pos0

$$
\begin{array}{cccccc}
 & 0 & 0 & 1 & 1 & 0 \\
 & & 0 & 0 & 1 & 1 \\
+ & & 1 & 0 & 0 & 1 \\
\hline
 & & 1 & 1 & 0 & 0
\end{array}
\qquad
\begin{array}{cccccc}
 & c_4 & c_3 & c_2 & c_1 & c_0 \\
 & & a_3 & a_2 & a_1 & a_0 \\
+ & & b_3 & b_2 & b_1 & b_0 \\
\hline
 & & s_3 & s_2 & s_1 & s_0
\end{array}
$$

At each bit position (e.g. pos1), take as inputs carryIn(c1), a1, b1, and compute sum (s1), carryout(c2). Feed c2 to the next bit position as carryIn.

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

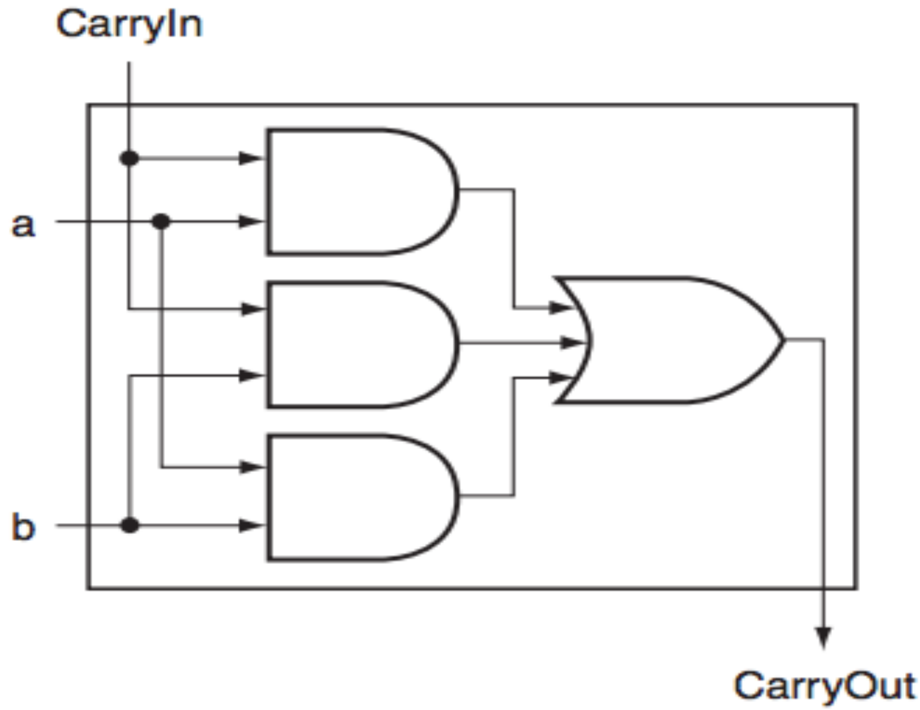
1-bit adder: computing CarryOut

a	b	CarryIn
0	1	1
1	0	1
1	1	0
1	1	1

Rows where CarryOut is 1

1-bit adder: computing CarryOut

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



1-bit adder: computing Sum

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

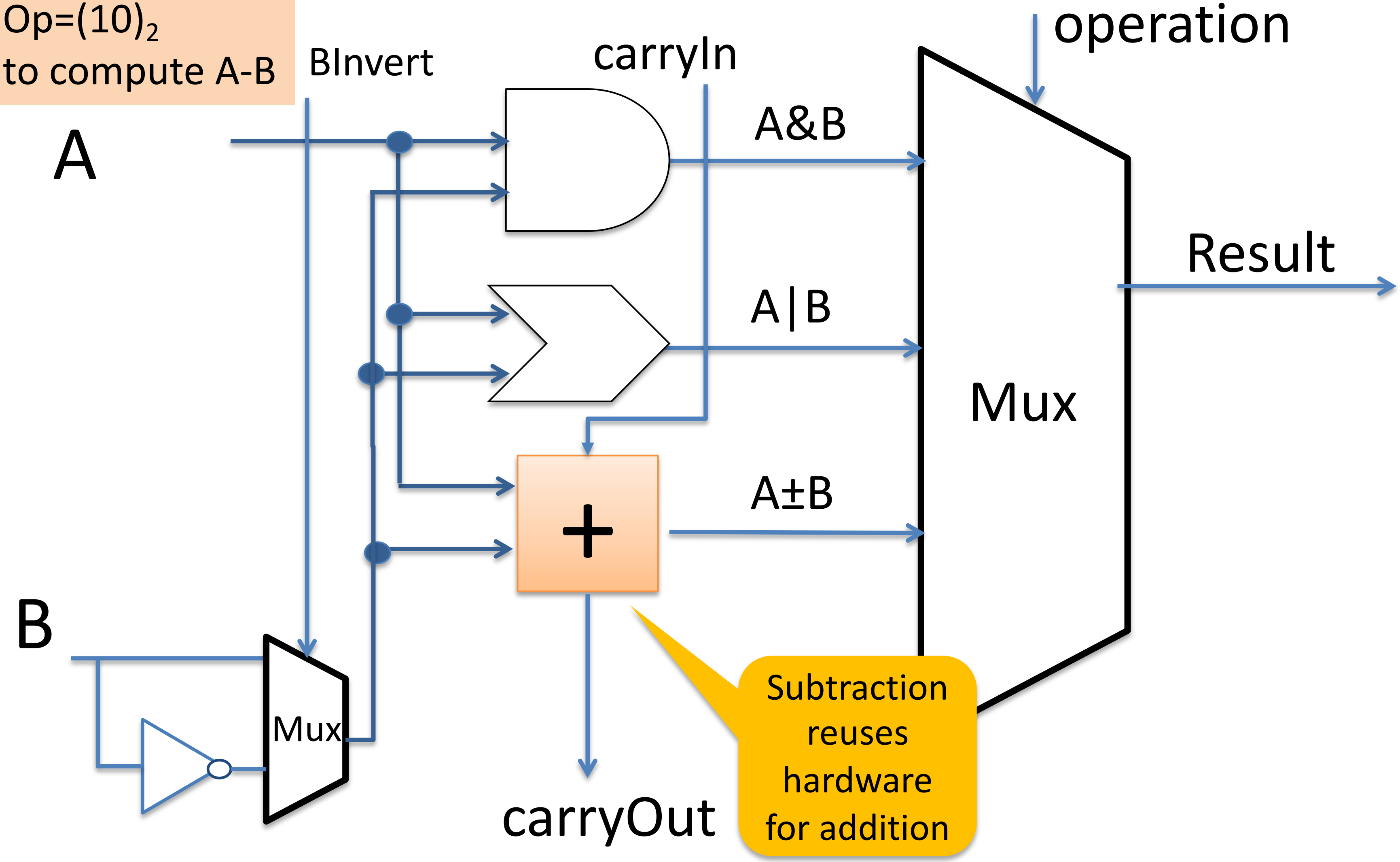
Inputs			Outputs	
a	b	CarryIn	CarryOut	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Subtraction

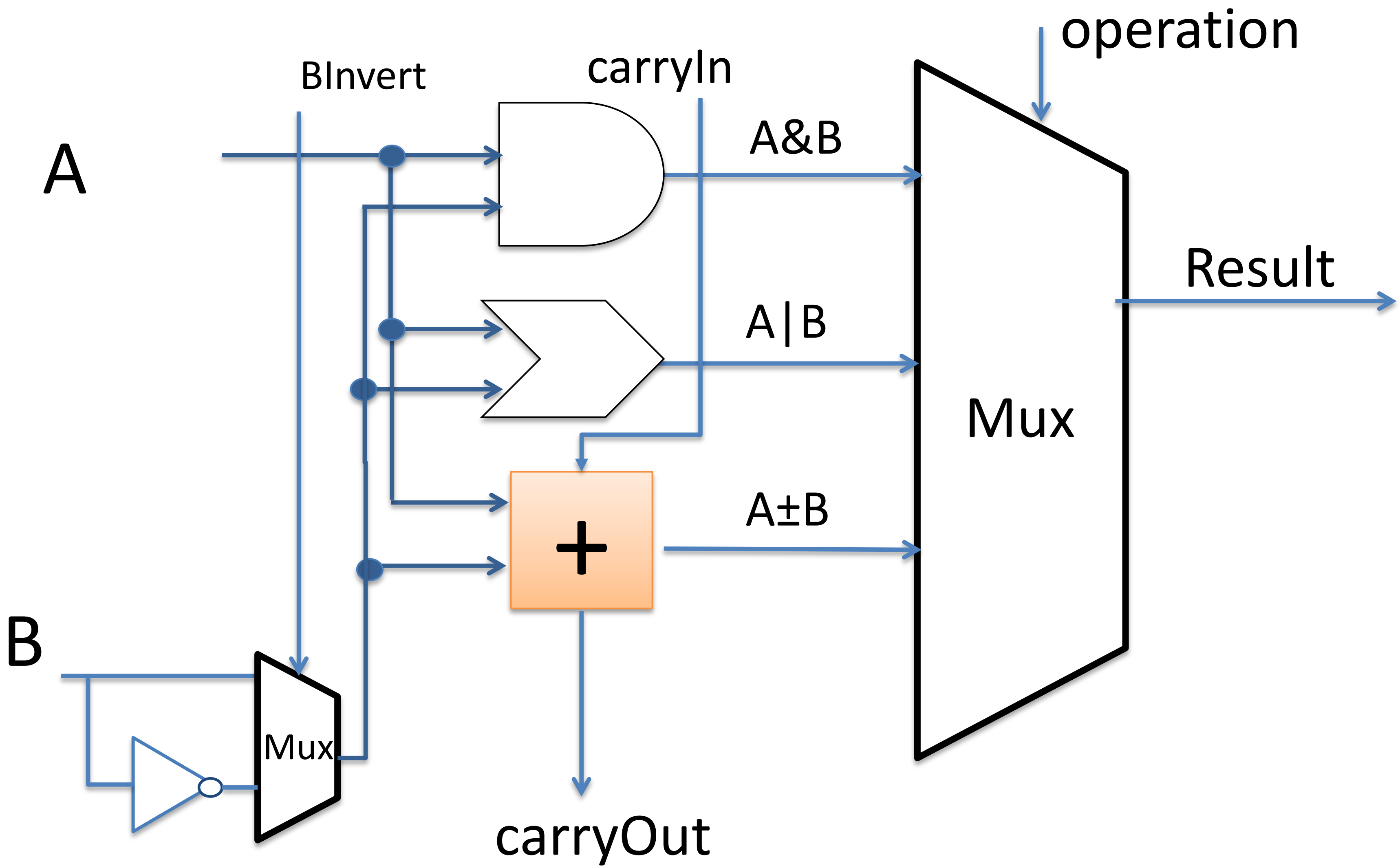
- Idea: a – b = a + (-b)
- How to calculate 2’s complement?

Subtraction in 1-bit ALU

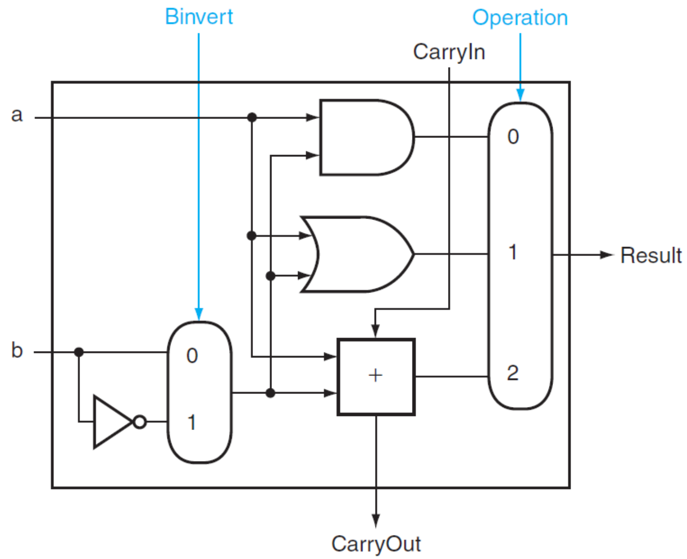
Set Binvert=1
carryIn=1
Op=$(10)_2$
to compute A-B

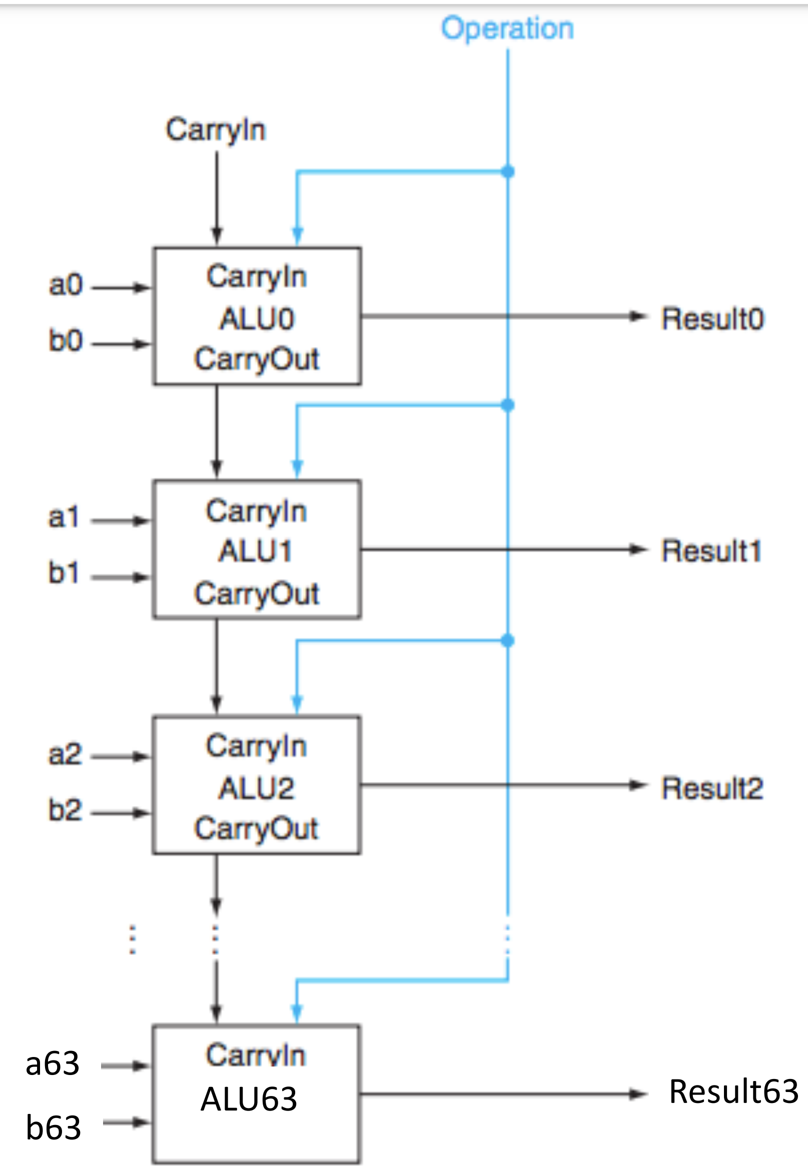


Extend 1-bit ALU to 64-bit



Extend 1-bit ALU to 64-bit ALU

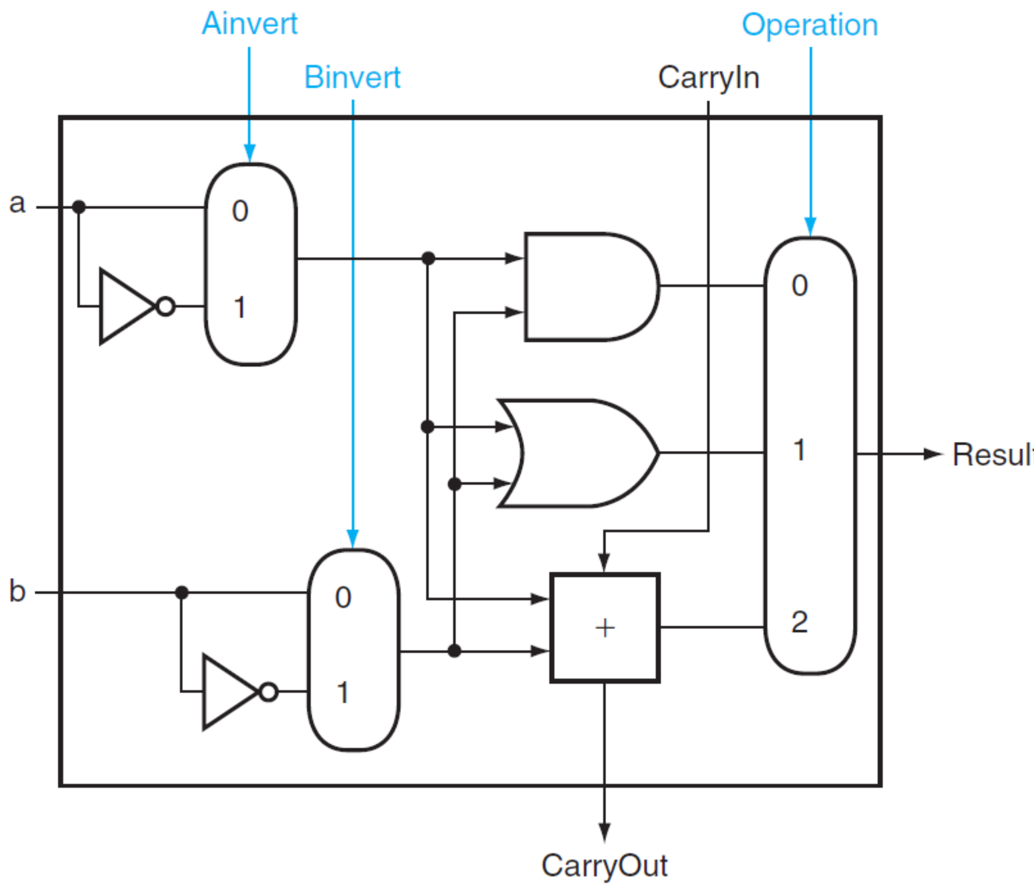




Extend ALU to include NOR

$$\overline{A+B} = \bar{A} \cdot \bar{B}$$

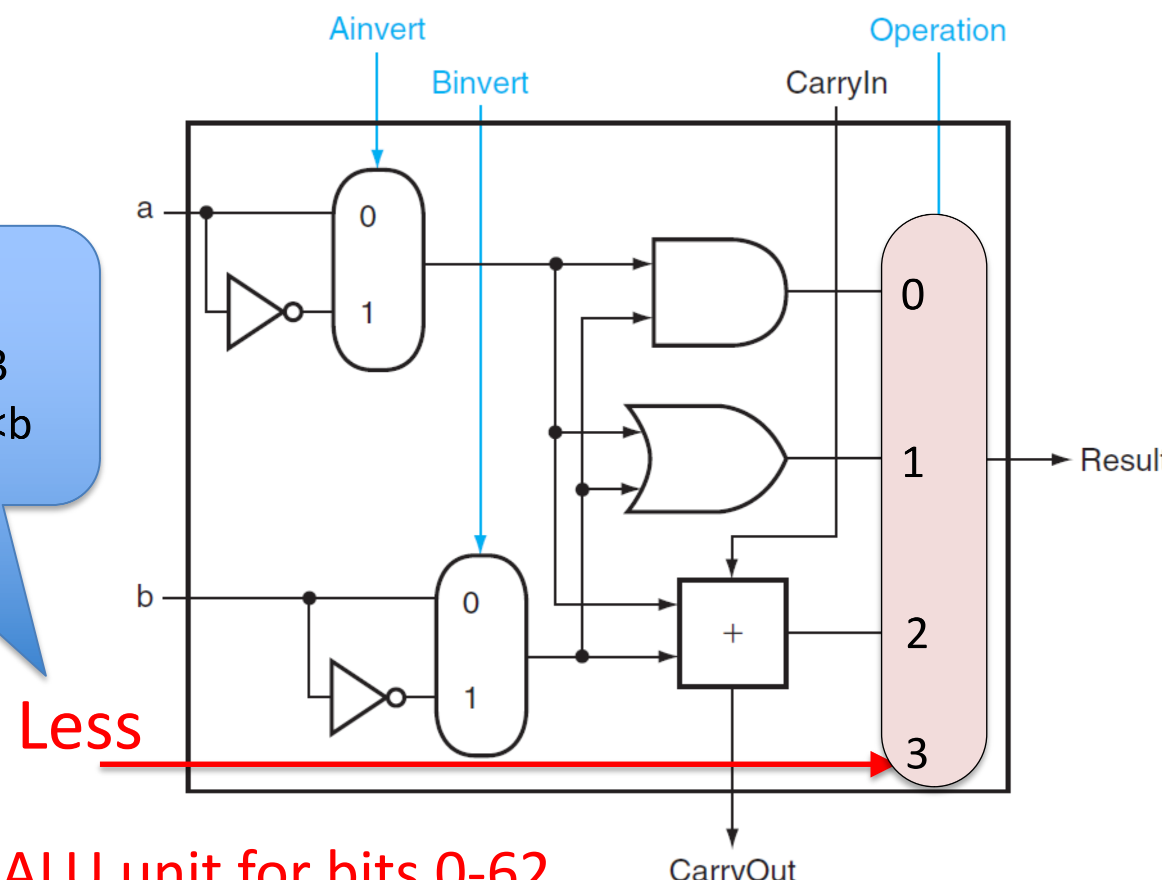
Set Binvert=1, Ainvert=1
Op=$(00)_2$
to compute a NOR b



Extend ALU to include `slt`

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

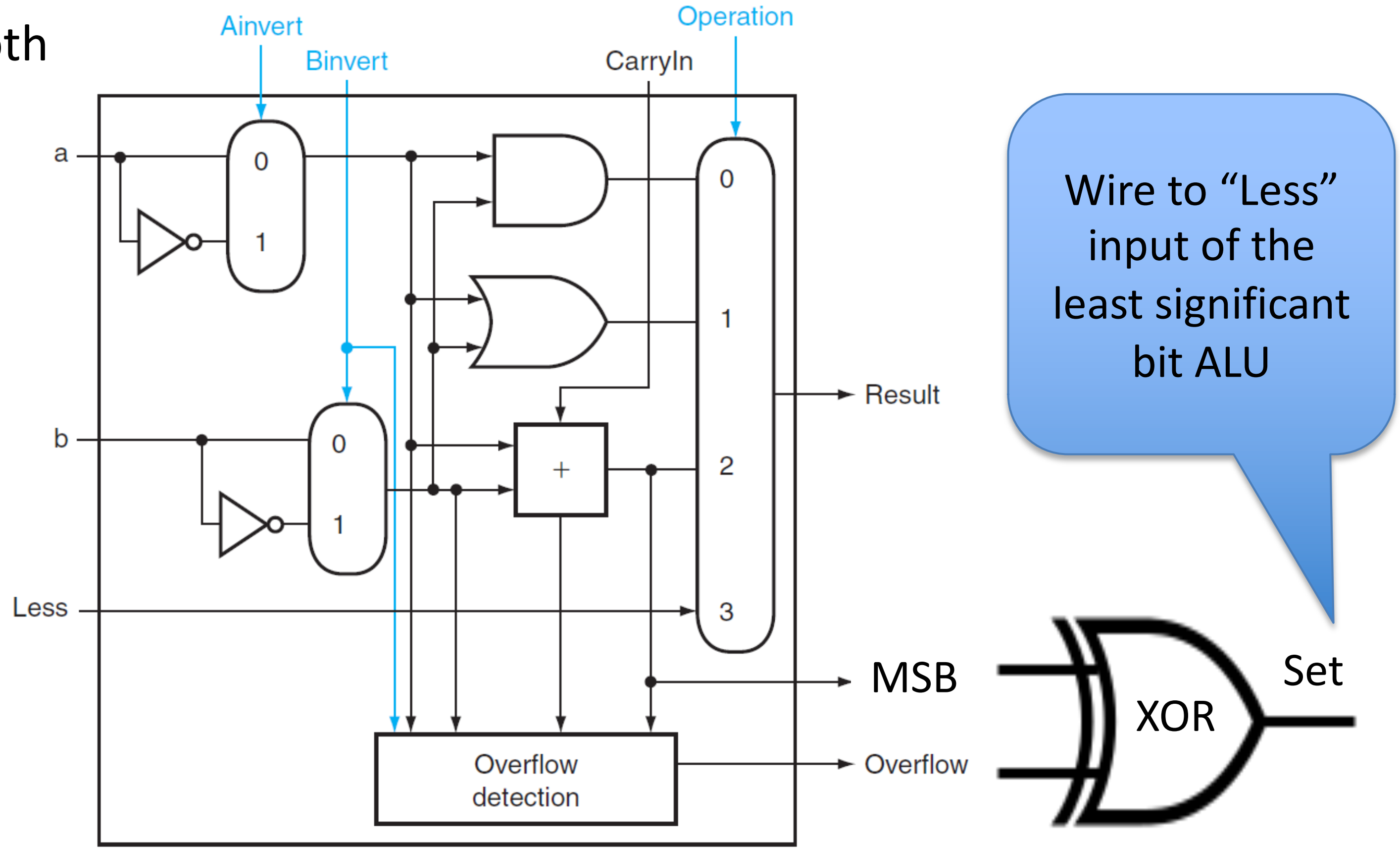
New input “Less”:
- always 0 for bits1:63
- Set to 1 for bit0 if a<b



ALU unit for bits 0-62

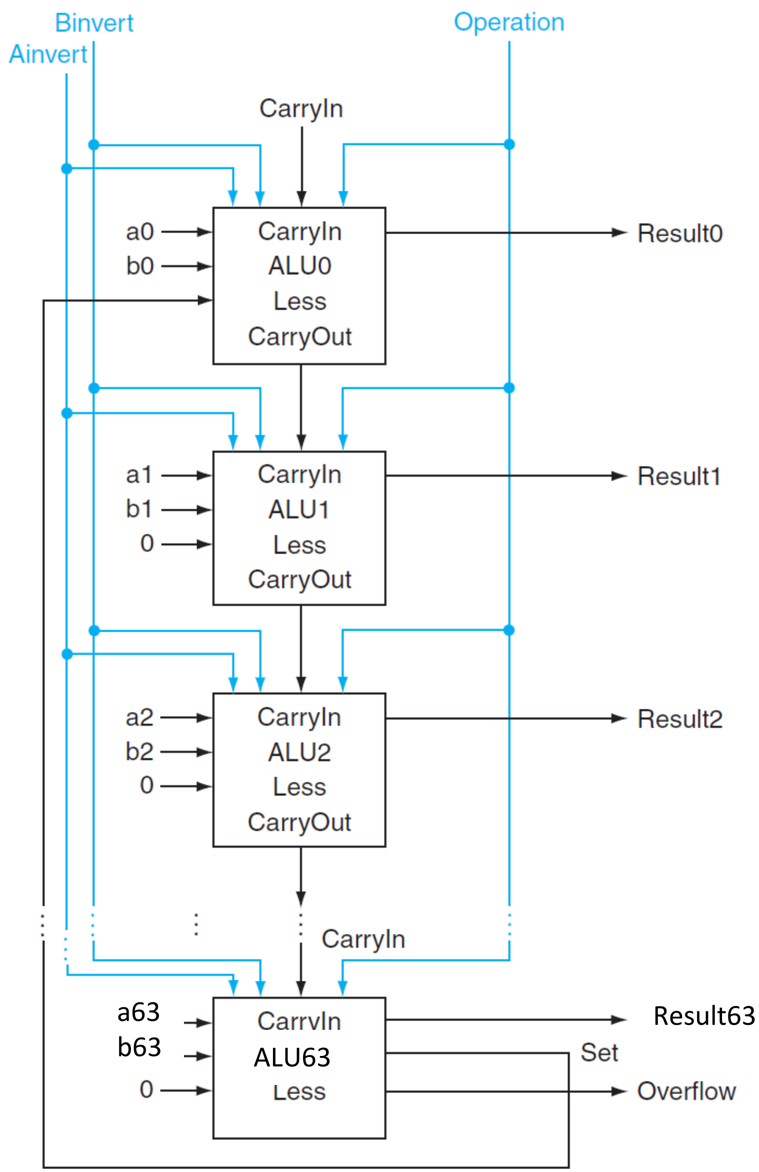
Extend ALU to include `slt`

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

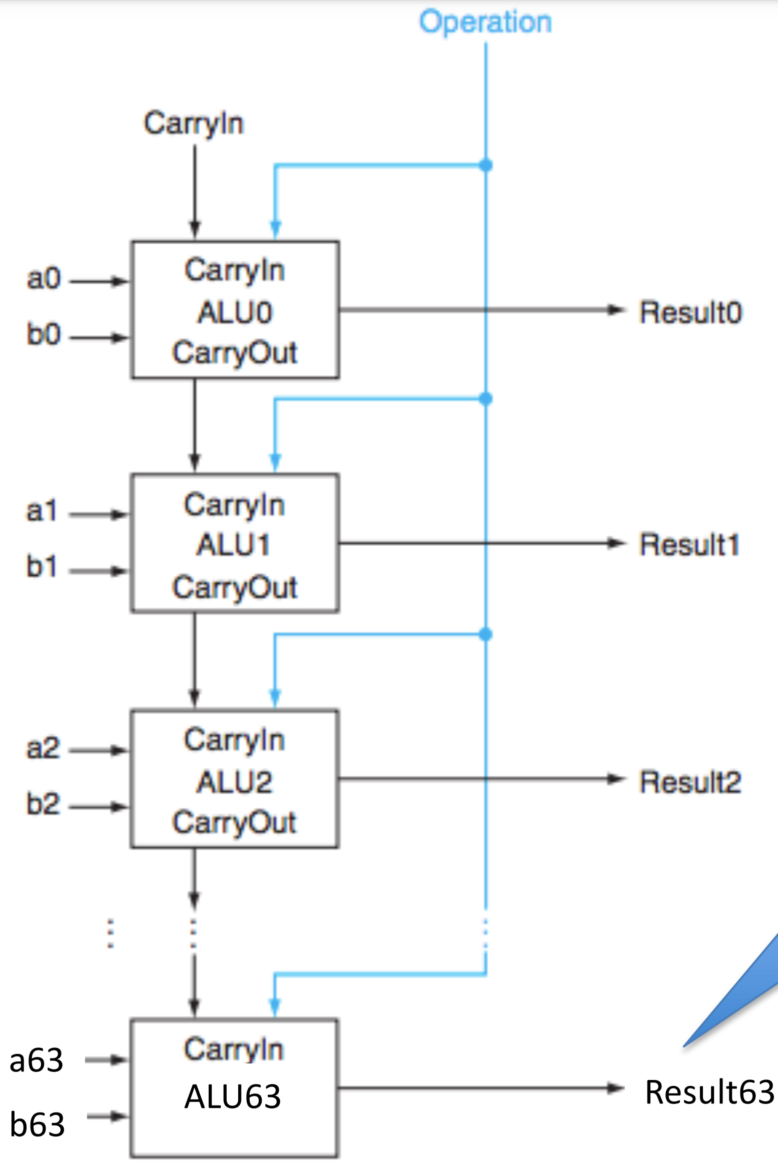


ALU unit for MSB(bit63)

Extend ALU to include `slt`



Downside of ripple carry?



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

In search of a 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

CarryOut of i-th bit
= CarryIn of (i+1)-th bit

CarryIn to i-th bit
= CarryOut of (i-1)-th bit

$$c_{i+1} = a_i b_i + (a_i + b_i)c_i$$

$$c_{i+1} = g_i + p_i c_i$$

$$g_i = a_i b_i \,,\ p_i = a_i + b_i$$

Generate

Propagate

g_i generates carryOut regardless of carryIn

p_i propagates carryIn to carryOut

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

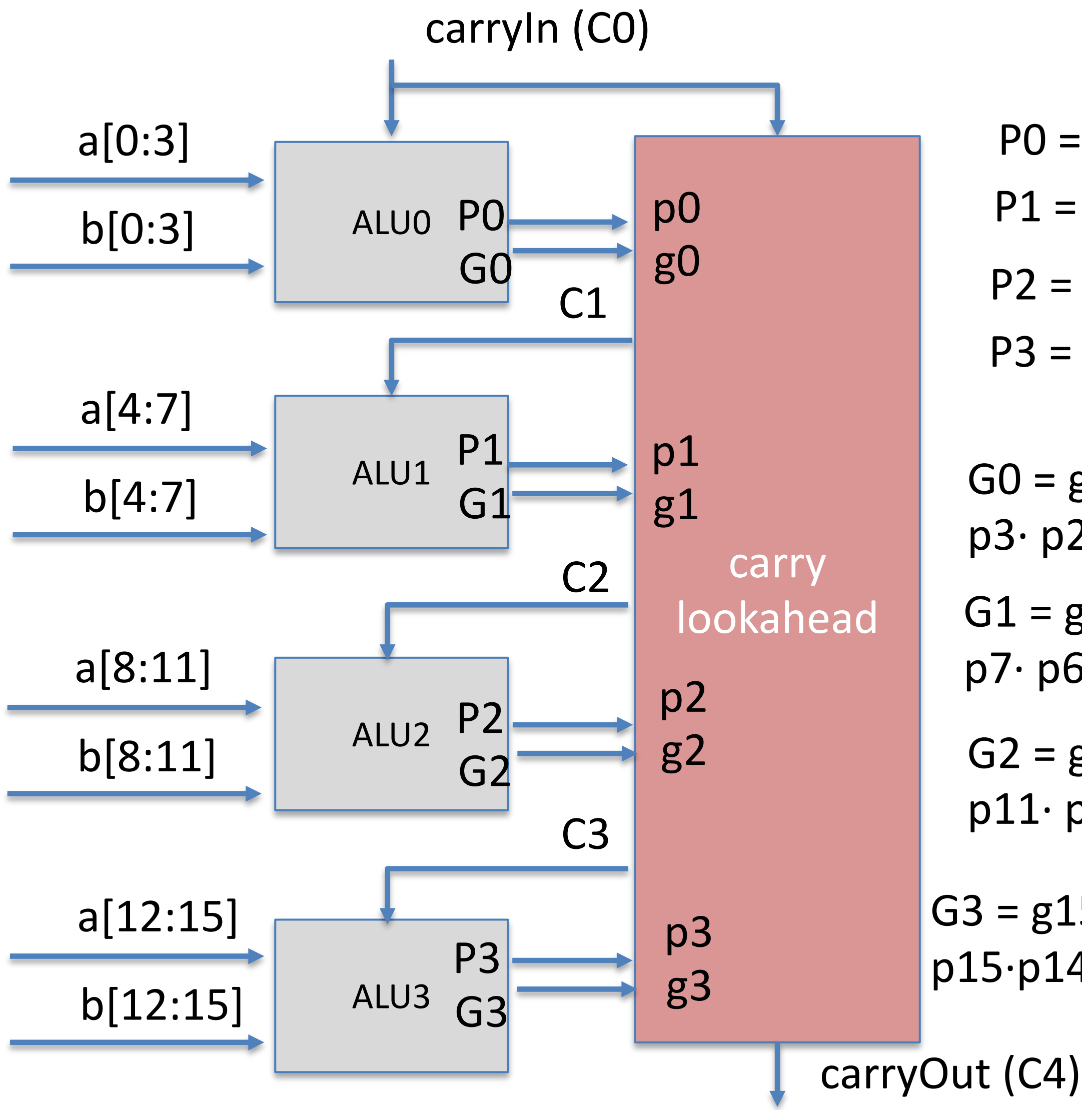
  Computing all result bits in a 4-bit adder:

$$s_i = a_i \cdot \bar{b}_i \cdot \bar{c}_i + \bar{a}_i \cdot b \, \bar{c}_i + \bar{a}_i \cdot \bar{b}_i \, c_i + a_i \cdot b_i \cdot c_i$$

for $i = 0, 1, 2, 3$

Faster adder: carry lookahead

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



$P0 = p3 \cdot p2 \cdot p1 \cdot p0$

$P1 = p7 \cdot p6 \cdot p5 \cdot p4$

$P2 = p11 \cdot p10 \cdot p9 \cdot p8$

$P3 = p15 \cdot p14 \cdot p13 \cdot p12$

$G0 = g3 + p3 \cdot g2 + p3 \cdot p2 \cdot g1 + p3 \cdot p2 \cdot p1 \cdot g0$

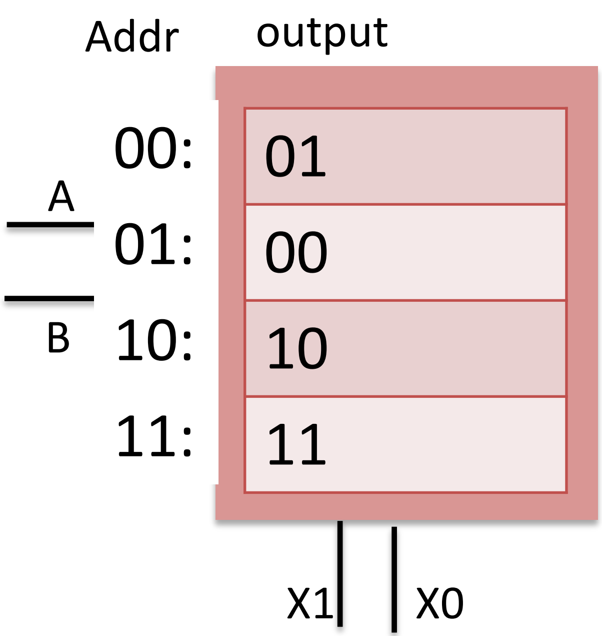
$G1 = g7 + p7 \cdot g6 + p7 \cdot p6 \cdot g5 + p7 \cdot p6 \cdot p5 \cdot g4$

$G2 = g11 + p11 \cdot g10 + p11 \cdot p10 \cdot g9 + p11 \cdot p10 \cdot p9 \cdot g8$

$G3 = g15 + p15 \cdot g14 + p15 \cdot p14 \cdot g13 + p15 \cdot p14 \cdot p13 \cdot g12$

Summary

- ROM



- ALU