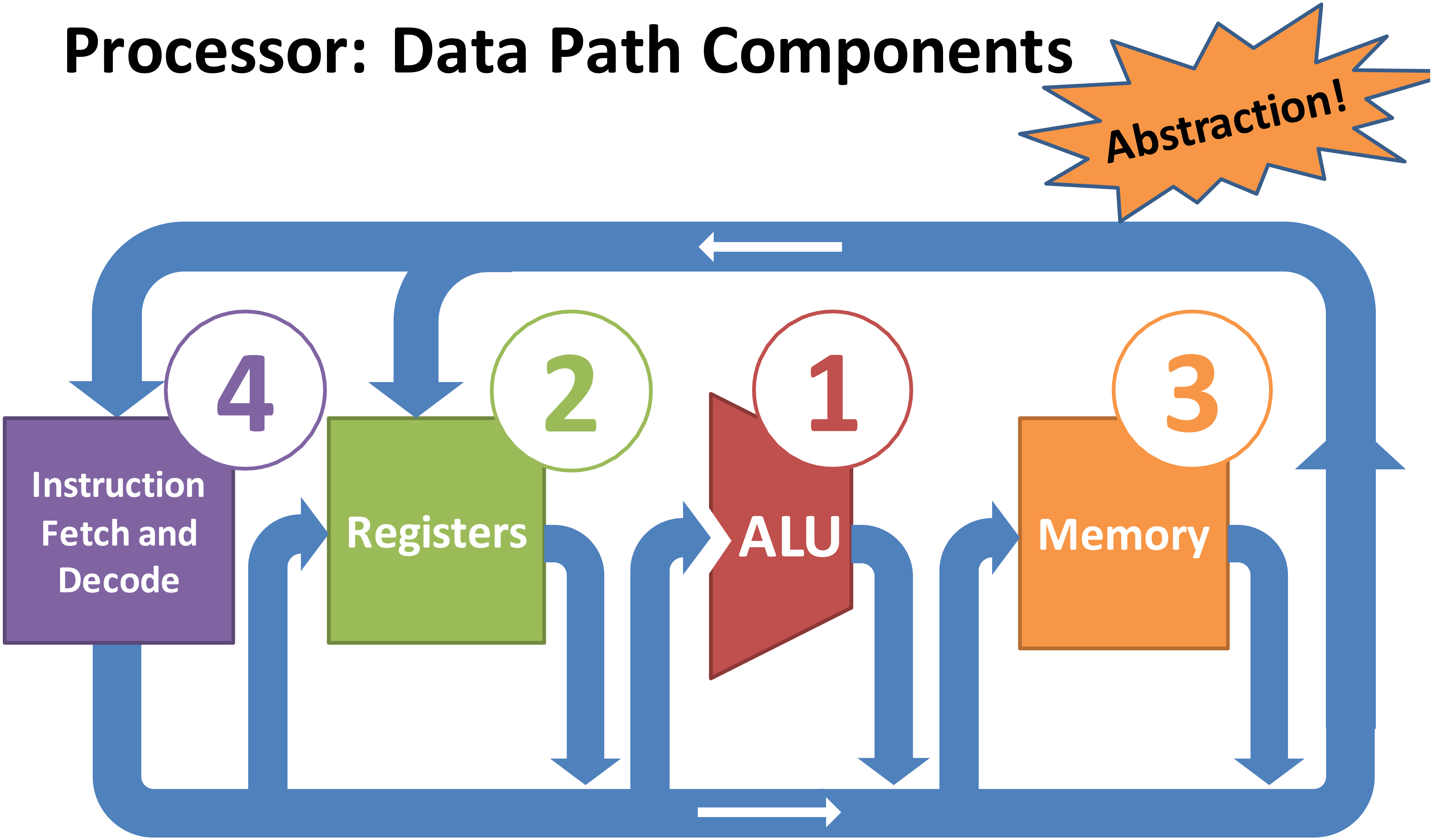
Processor: Data Path Components



Building Blocks

Abstraction!

Microarchitecture

Processor datapath

Instruction Decoder
Arithmetic Logic Unit

Memory

Digital Logic

Adders
Multiplexers
Demultiplexers
Encoders
Decoders

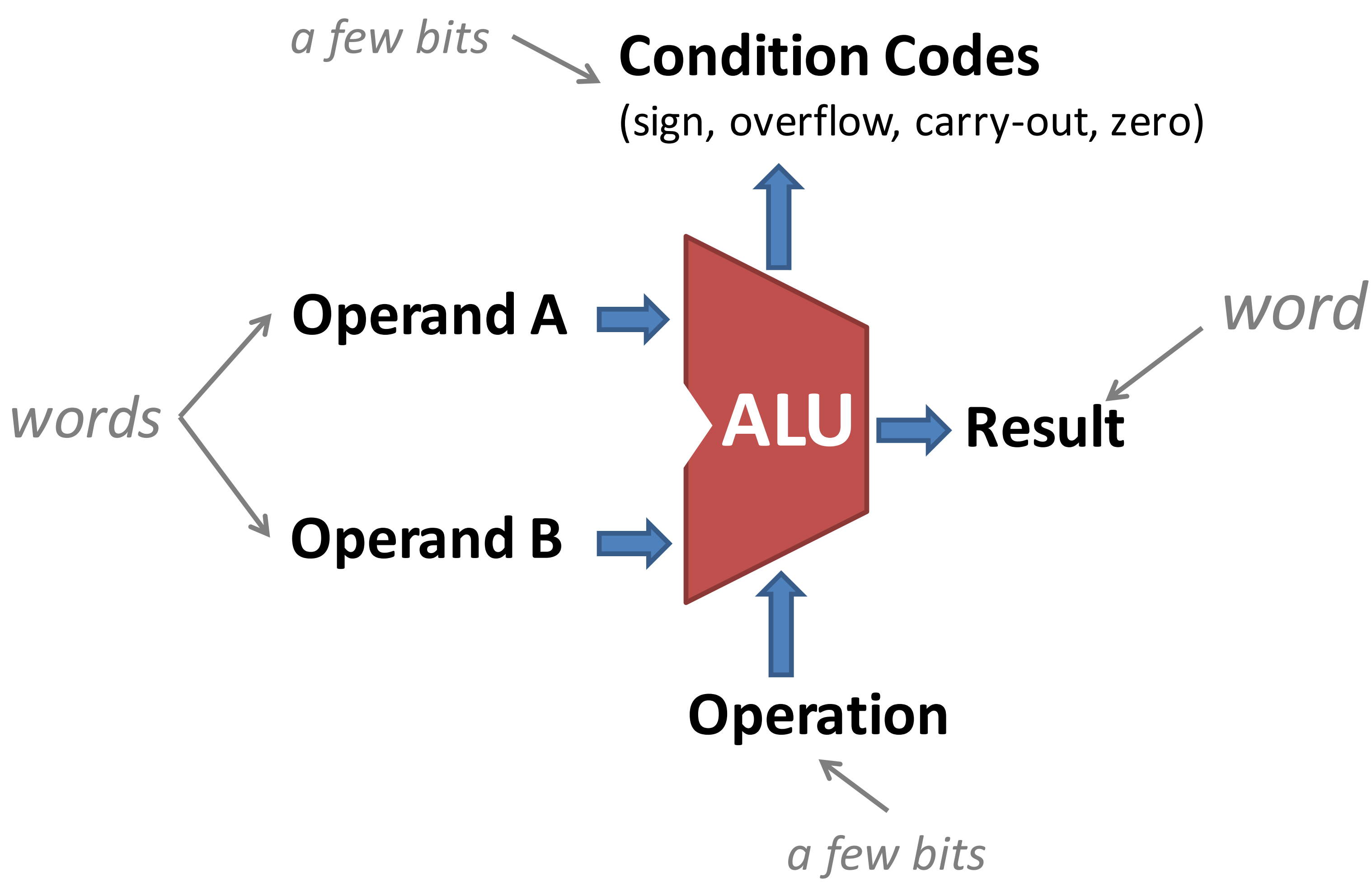
Registers

Flip-Flops
Latches

Gates

Devices (transistors, etc.)

Arithmetic Logic Unit (ALU)



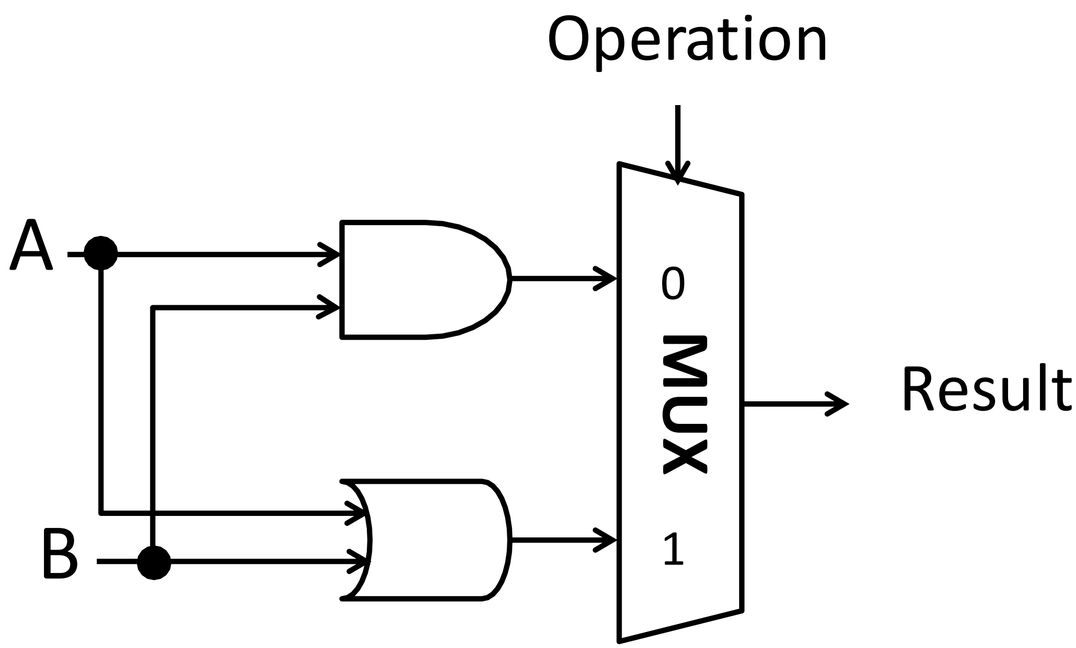
Hardware unit for arithmetic and bitwise operations.

1-bit ALU for bitwise operations

ex

Build an n-bit ALU from n 1-bit ALUs.

Each bit *i* in the result is computed from the corresponding bit *i* in the two inputs.



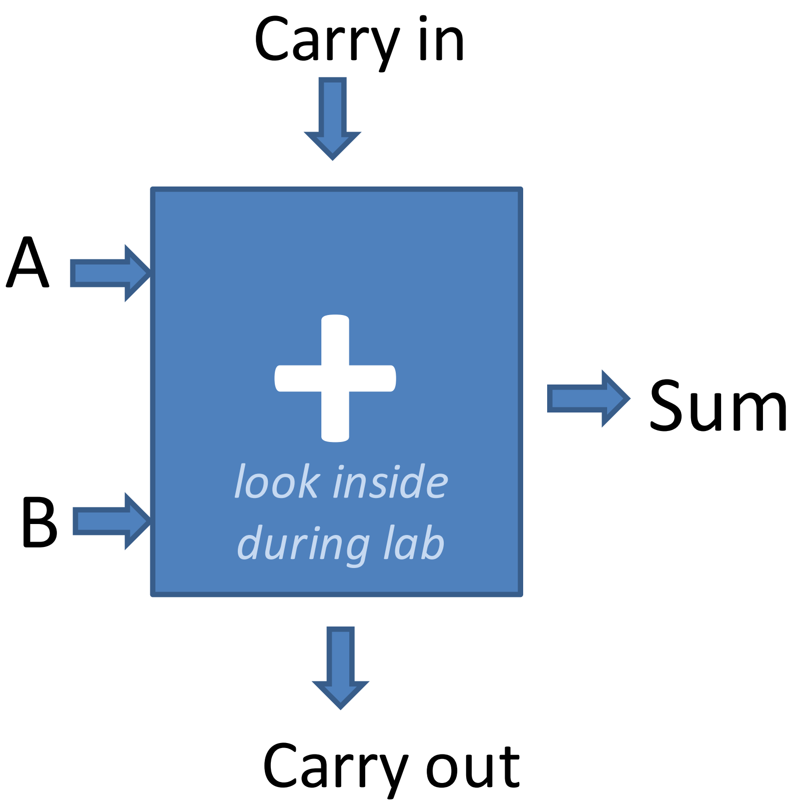
Op	A	B	Result
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

1-bit adder

ex

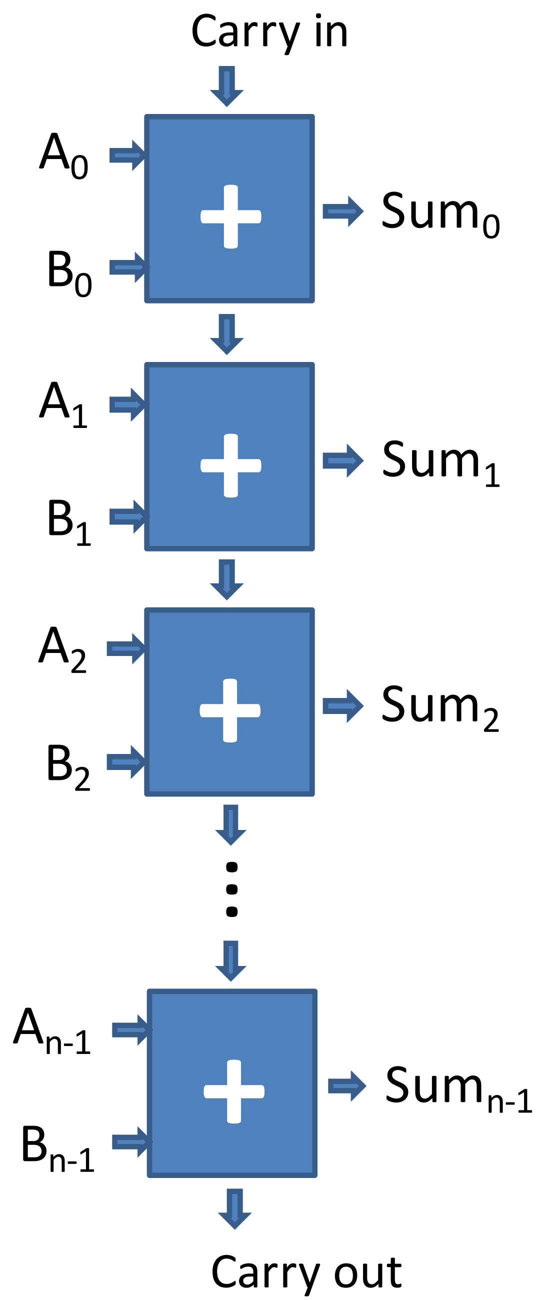
Build an n-bit adder from n 1-bit adders.

Each bit *i* in the result is computed from the corresponding bit *i* in the two inputs **and the carry out of bit *i-1***.



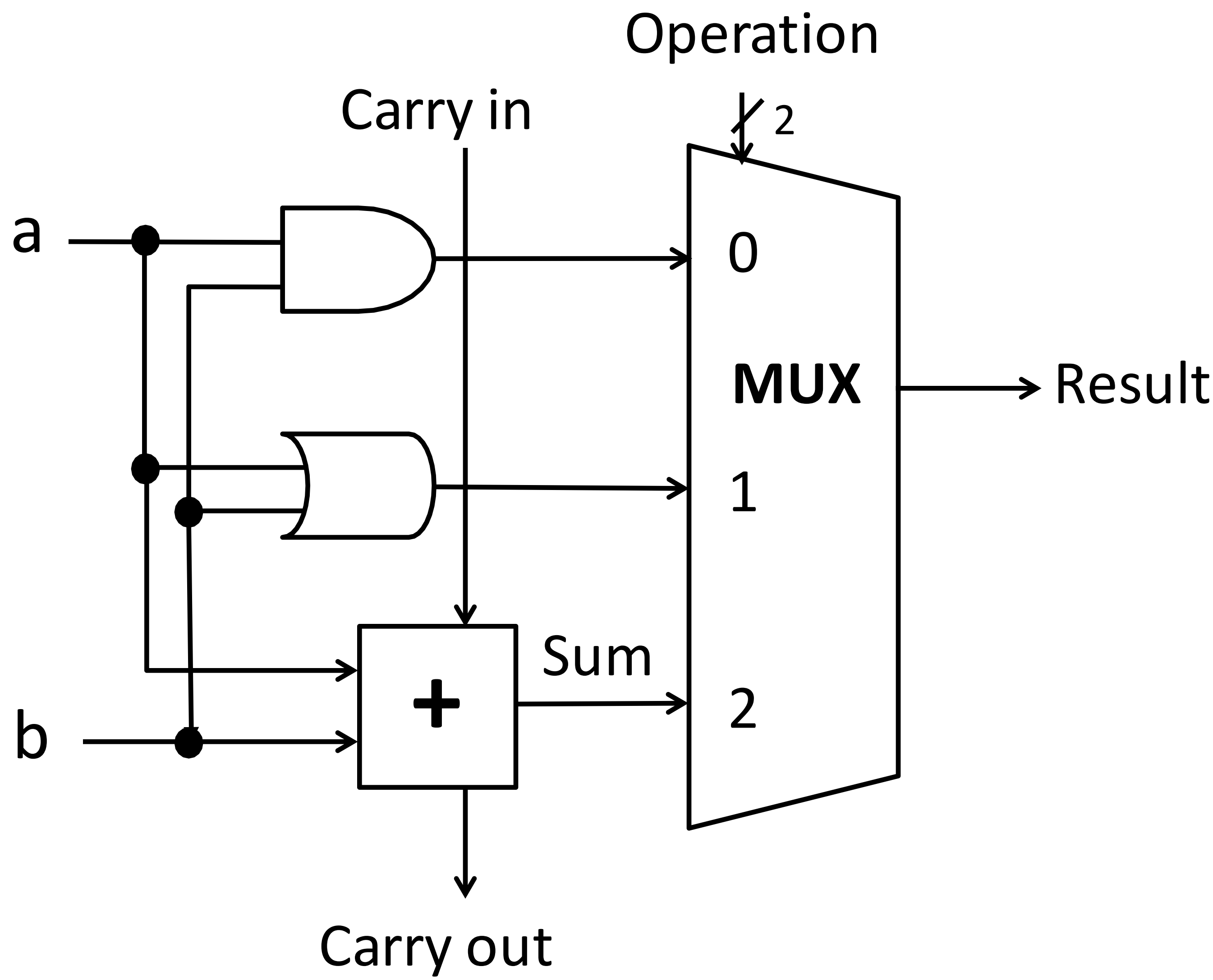
A	B	Carry in	Carry out	Sum
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

n-bit ripple-carry adder



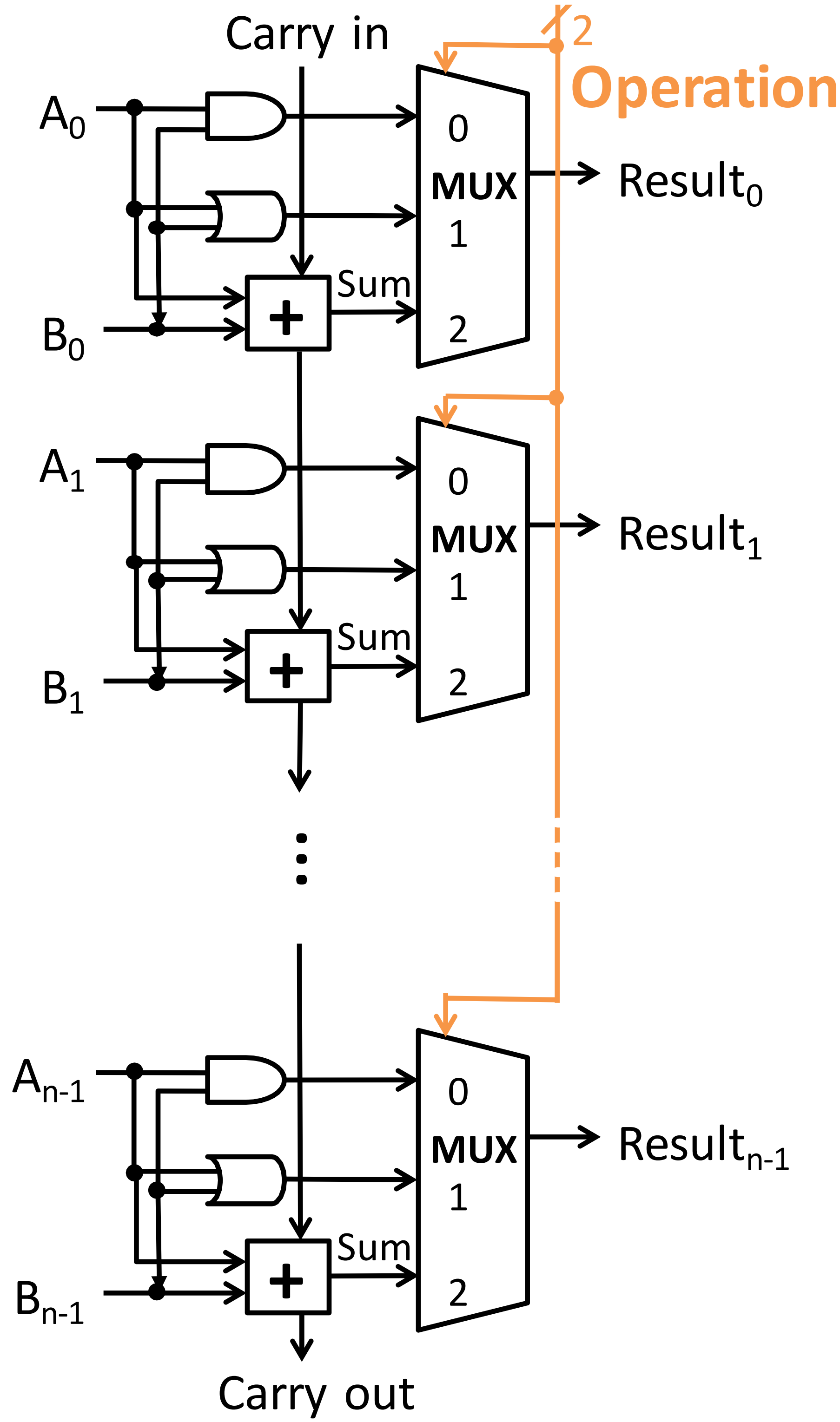
There are faster, more complicated ways too…

1-bit ALU

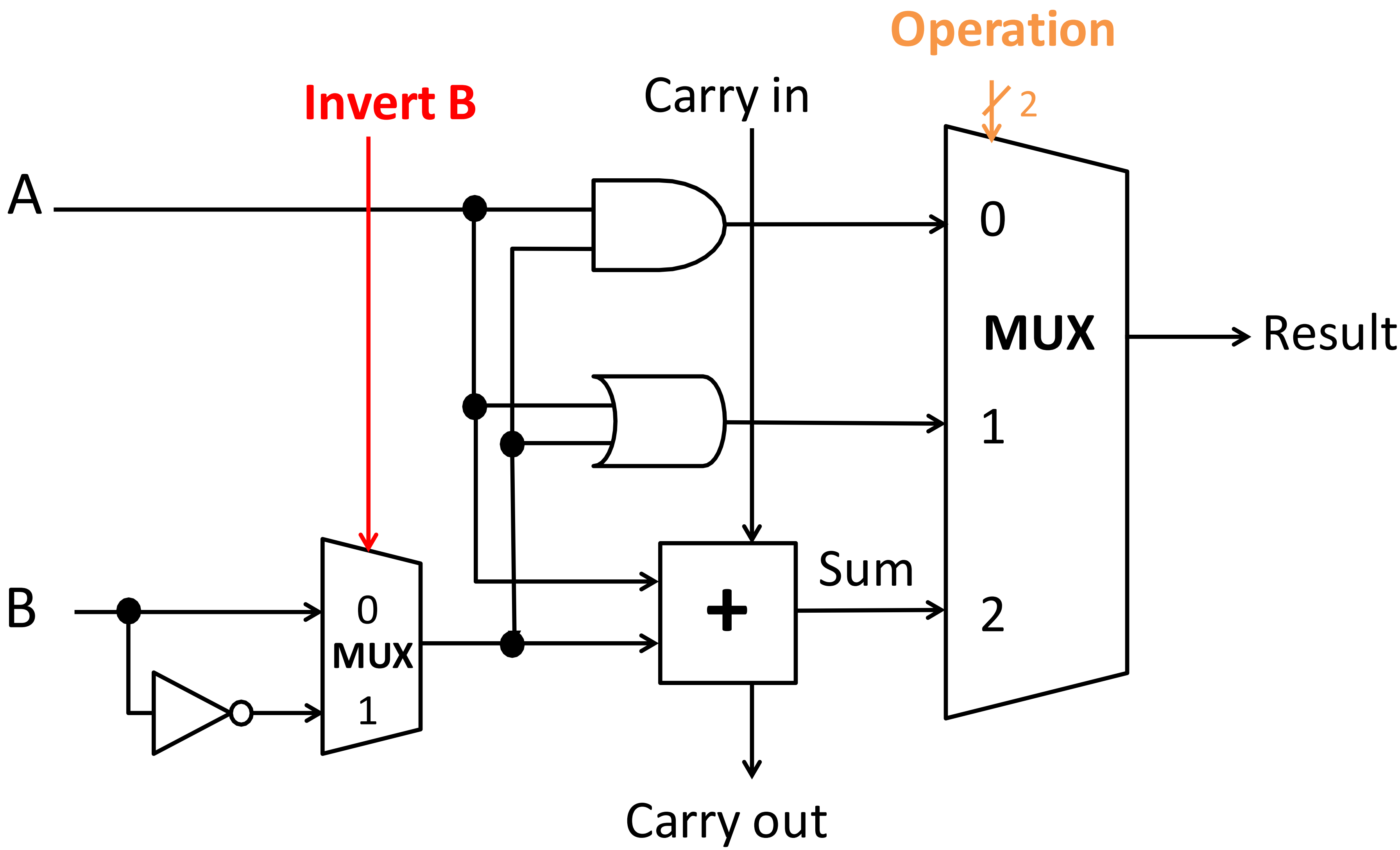


n-bit ALU

with ripple carry



Adding subtraction



Different than in SCO book.

ALU Condition Codes (x86)

Extra ALU outputs describing properties of result.

Zero Flag: 1 if result is 00...0 else 0

Sign Flag: sign bit of result

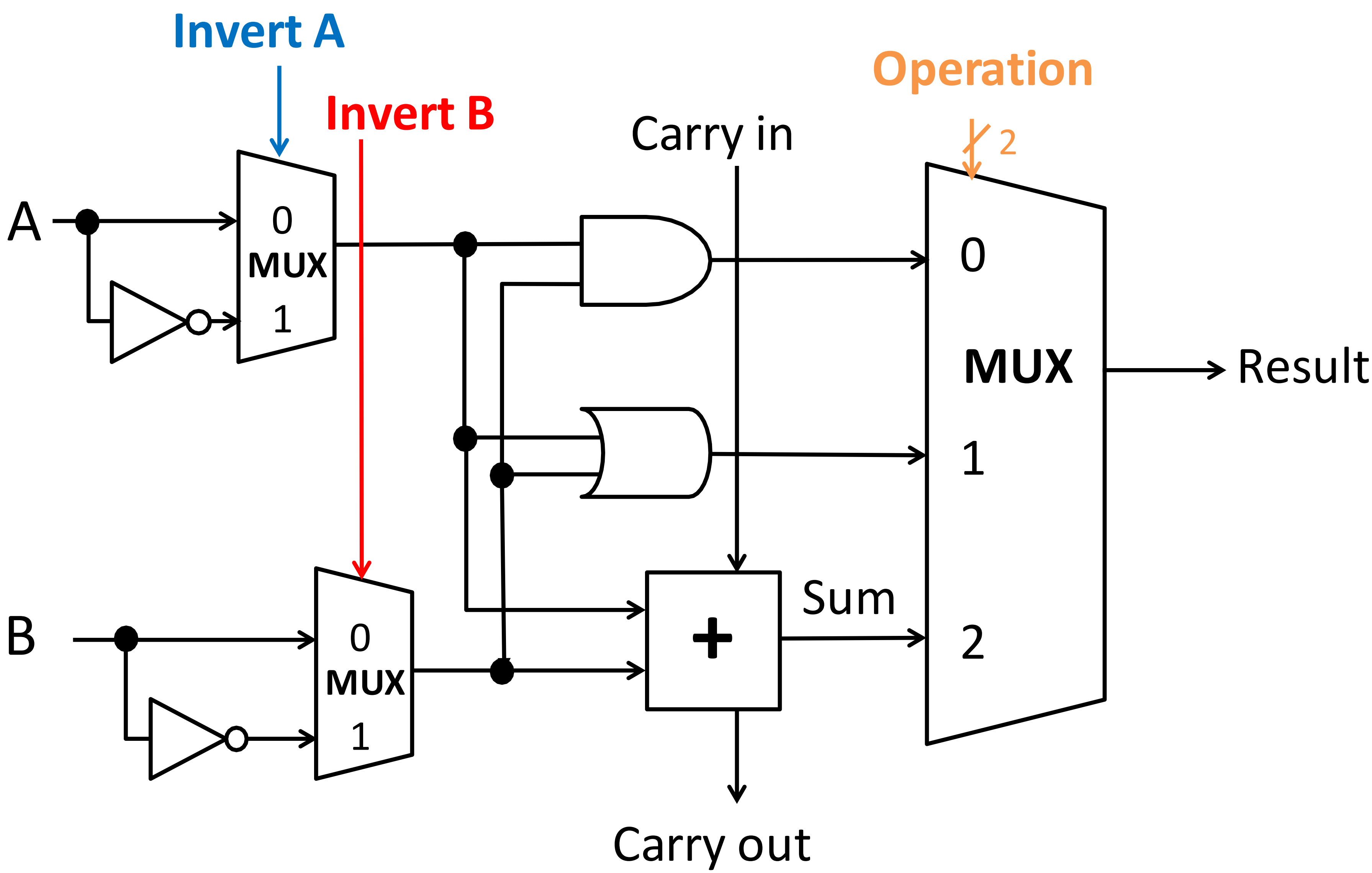
Carry Flag: 1 if unsigned overflow else 0
carry-out bit of result

Overflow Flag: 1 if signed overflow else 0

Compute NAND, NOR, NOT A,

Set inputs as needed.

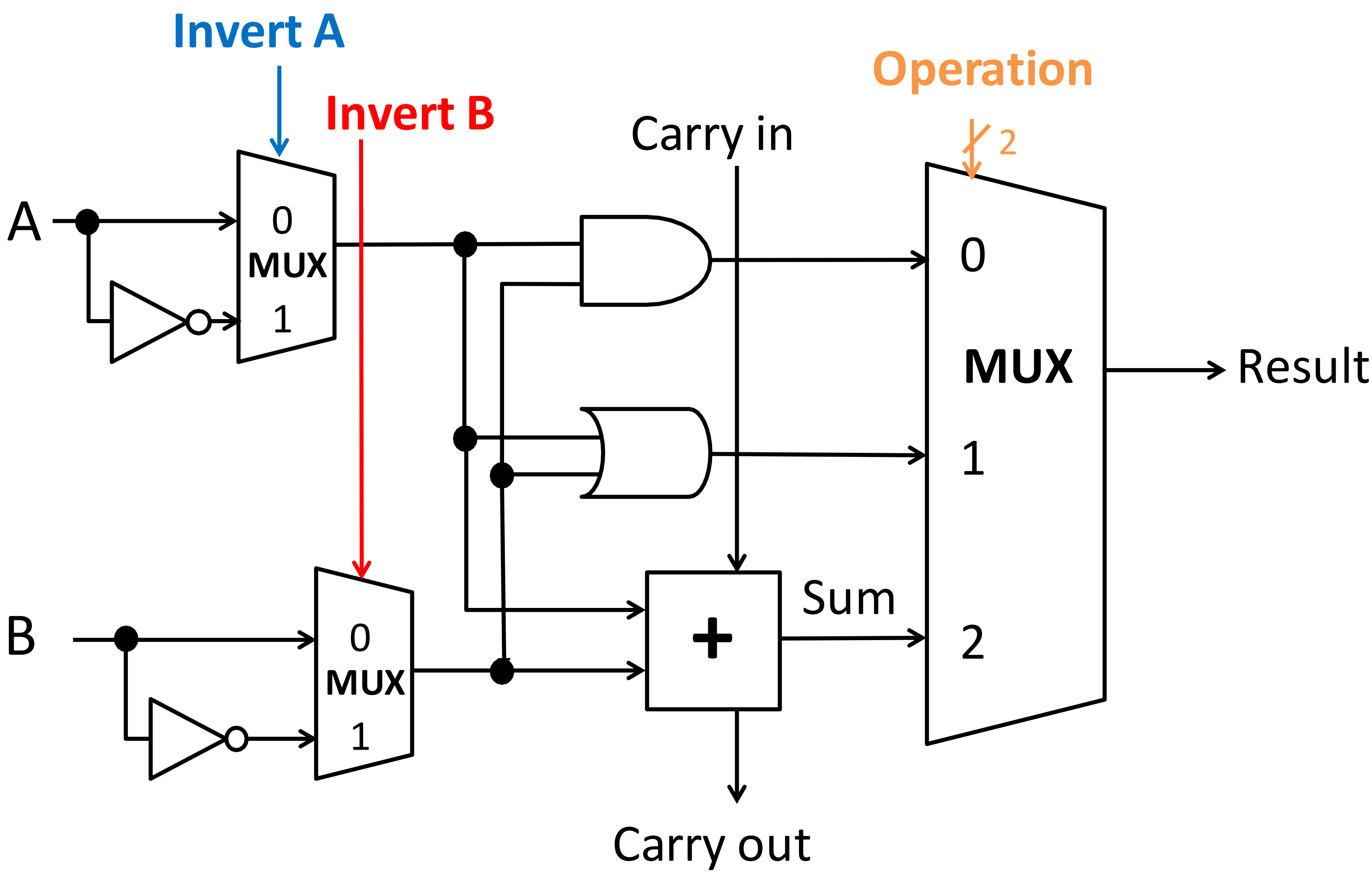
ex



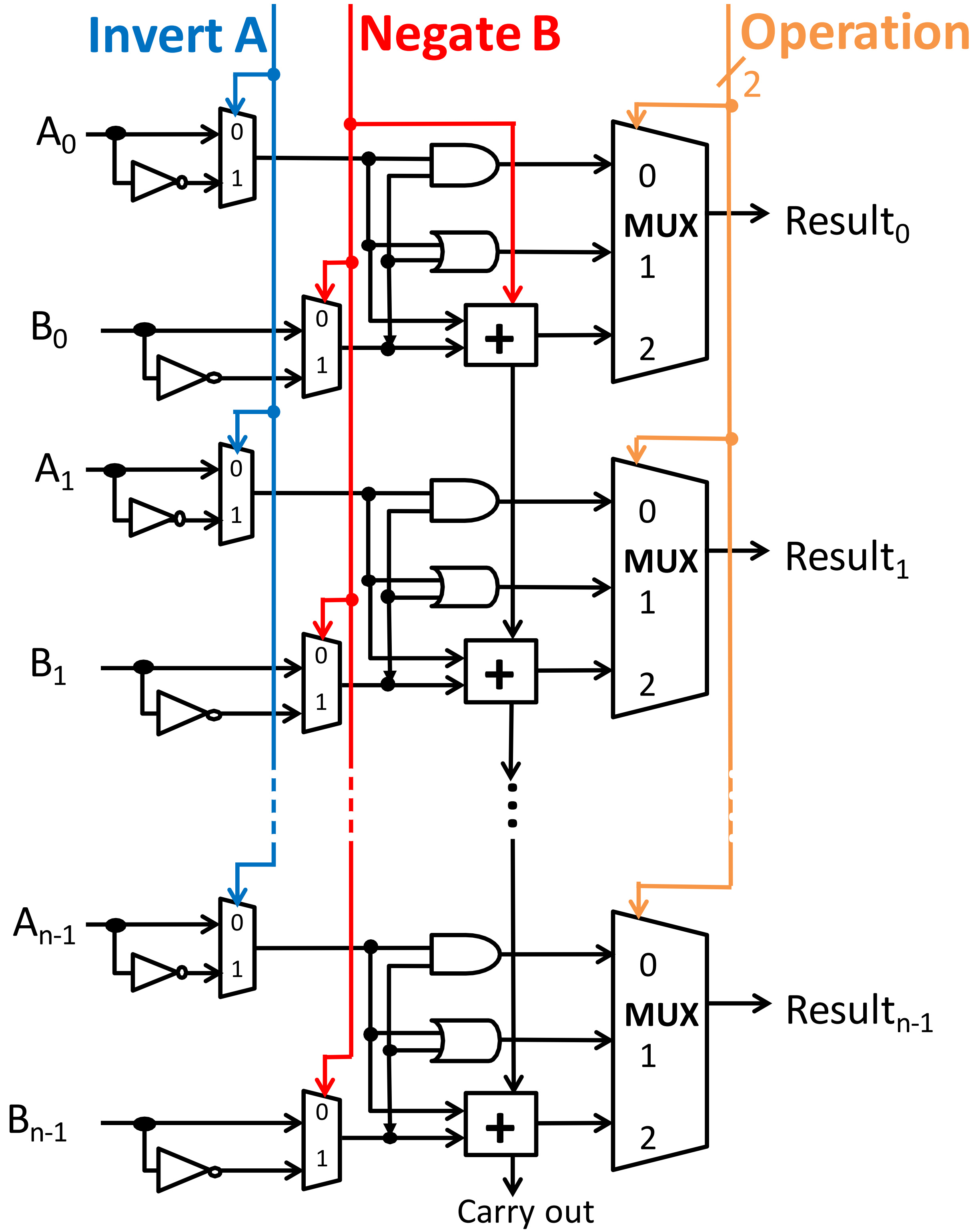
Compute <, ==? Detect overflow?

Set inputs as needed, add minimal logic for overflow.

ex



n-bit ALU



Invert A
Negate B
Operation
2
A_0
B_0
A_1
B_1
A_{n-1}
B_{n-1}
0
1
MUX
2
$Result_0$
$Result_1$
$Result_{n-1}$
Carry out

Controlling the ALU

Abstraction!

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
1100	NOR

Invert A

Negate B

Operation ID