Arithmetic/Logical Unit (ALU)

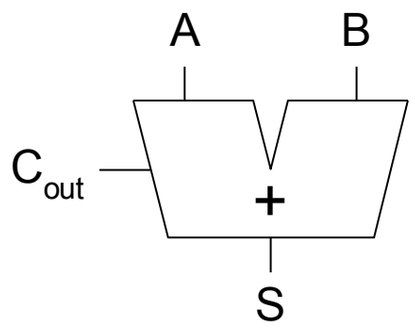
Digital Computer Design

Arithmetic Circuits

- Arithmetic circuits are the **central building blocks** of computers.
- Computers and digital logic perform many arithmetic functions:
  - addition, subtraction, comparisons, shifts, multiplication and division

1-Bit Adders

Half Adder

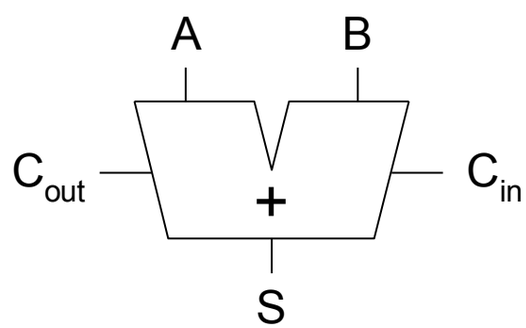


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = A \oplus B$

$C_{out} = AB$

Full Adder



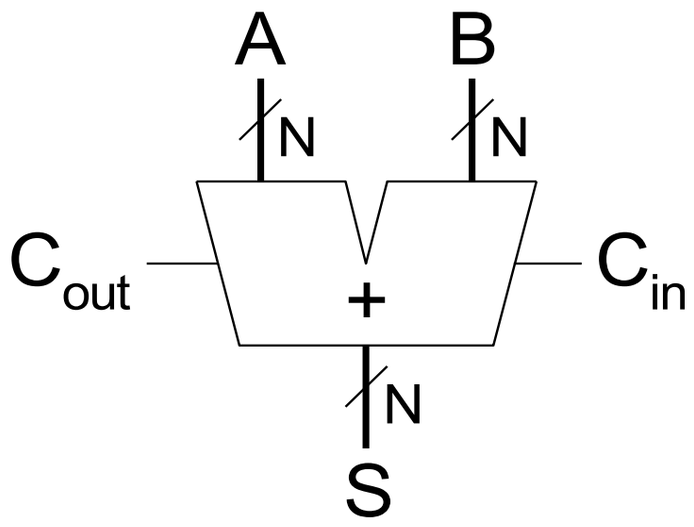
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

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

$C_{out} = AB + AC_{in} + BC_{in}$

Multibit Adders - Carry Propagate Adders (CPAs)

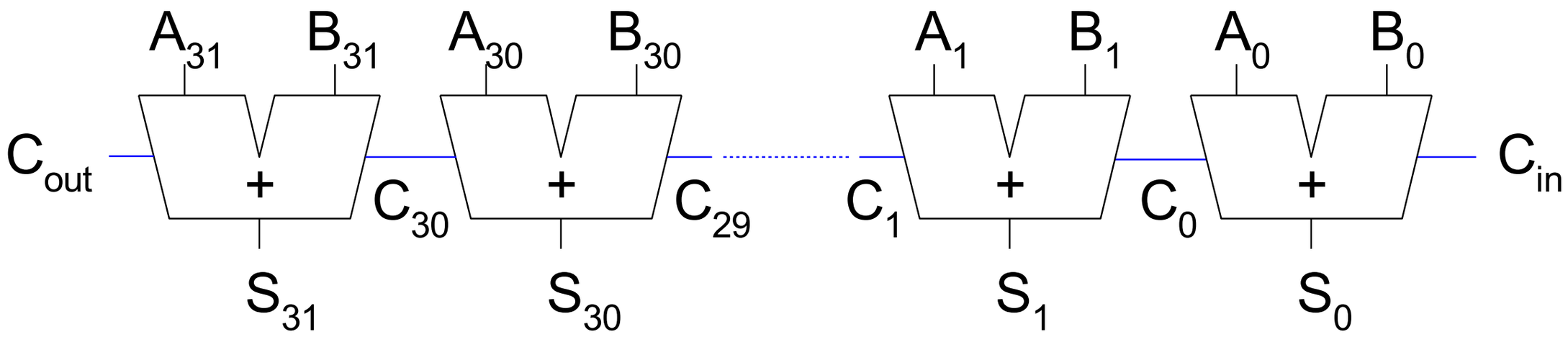
- An **N-bit adder** sums two *N-bit* inputs A and B, and a carry in C_{in} to produce an *N-bit* result S and a carry out C_{out}.



- Multibit adder is commonly called a **carry propagate adder** (CPA)
  - because the carry out of one bit propagates into the next bit.

CPAs: Ripple-Carry Adder

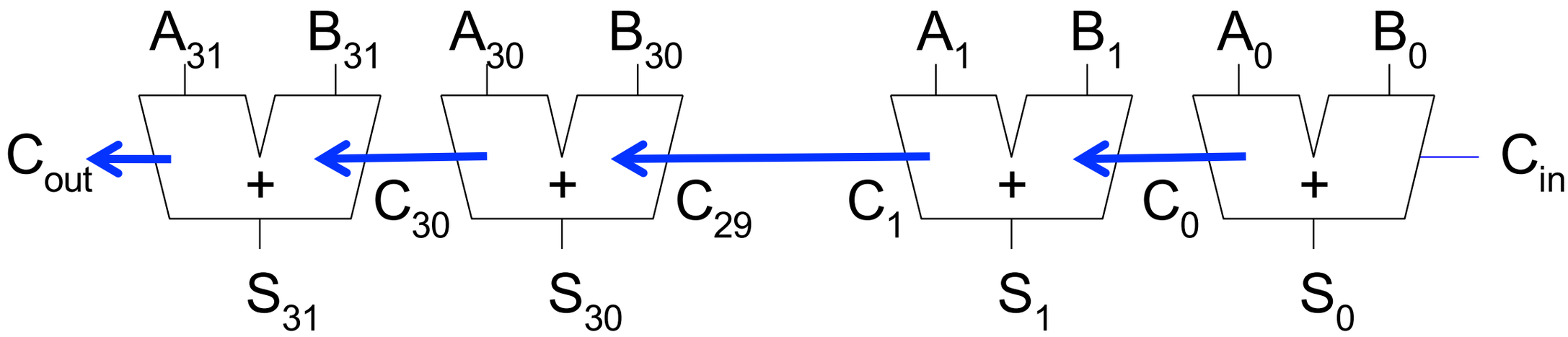
- Chain together N full adders.
  - The C_{out} of one stage acts as the C_{in} of the next stage



- The ripple-carry adder is **slow** when N is large.
  - For an 32-bit adder, S_{31} depends on C_{30}, which depends on C_{29}, which depends on C_{28}, and so forth all the way back to C_{in}.

CPAs: Ripple-Carry Adder

- The fundamental reason that large ripple-carry adders are **slow** is that *the carry signals must propagate* through **every bit** in the adder.



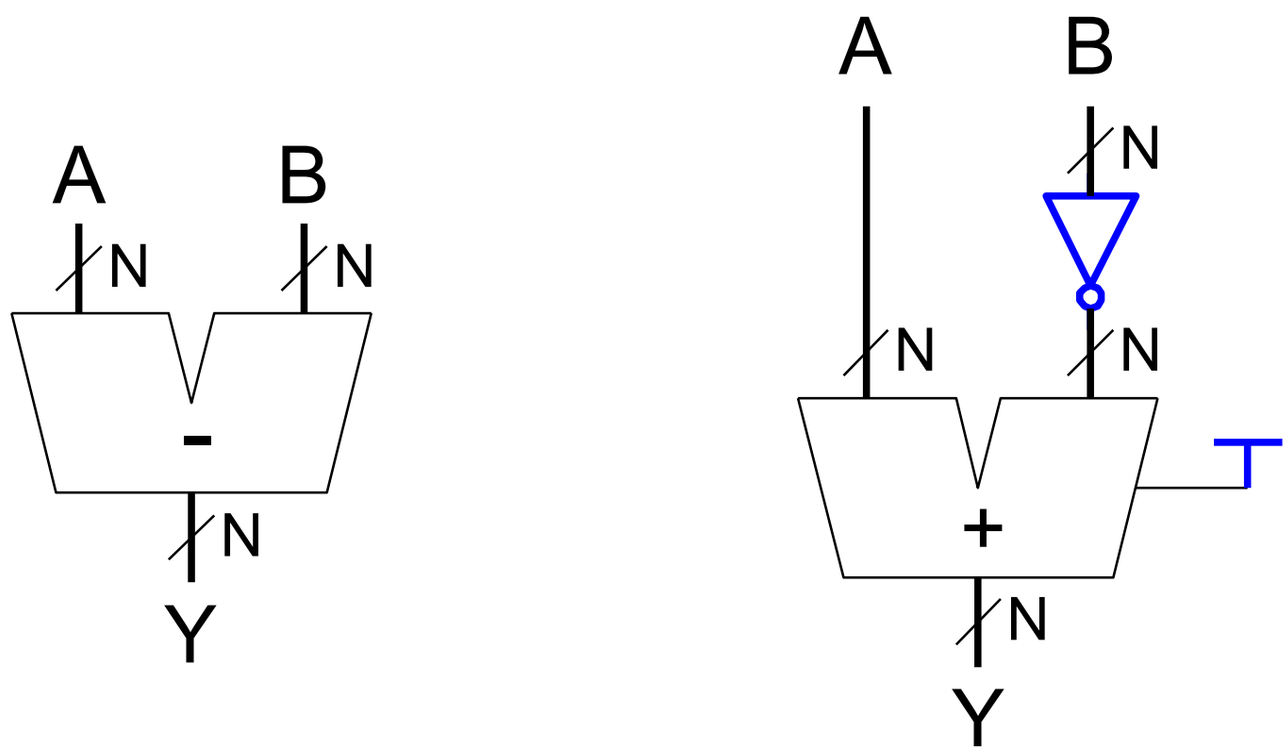
tripple = Nt_{FA}

where t_{FA} is the delay of a 1-bit full adder

Subtracter

- Subtraction is almost as easy:
  - flip the sign of the second number, then add.

Symbol **Implementation**

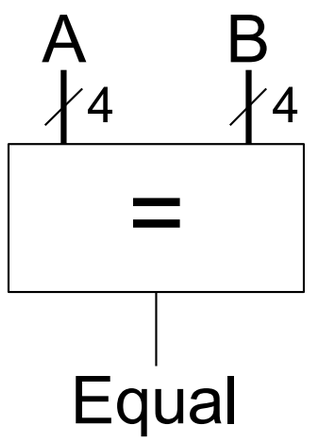


- Flipping the sign of **a two' s complement** number is done by inverting the bits and adding 1.

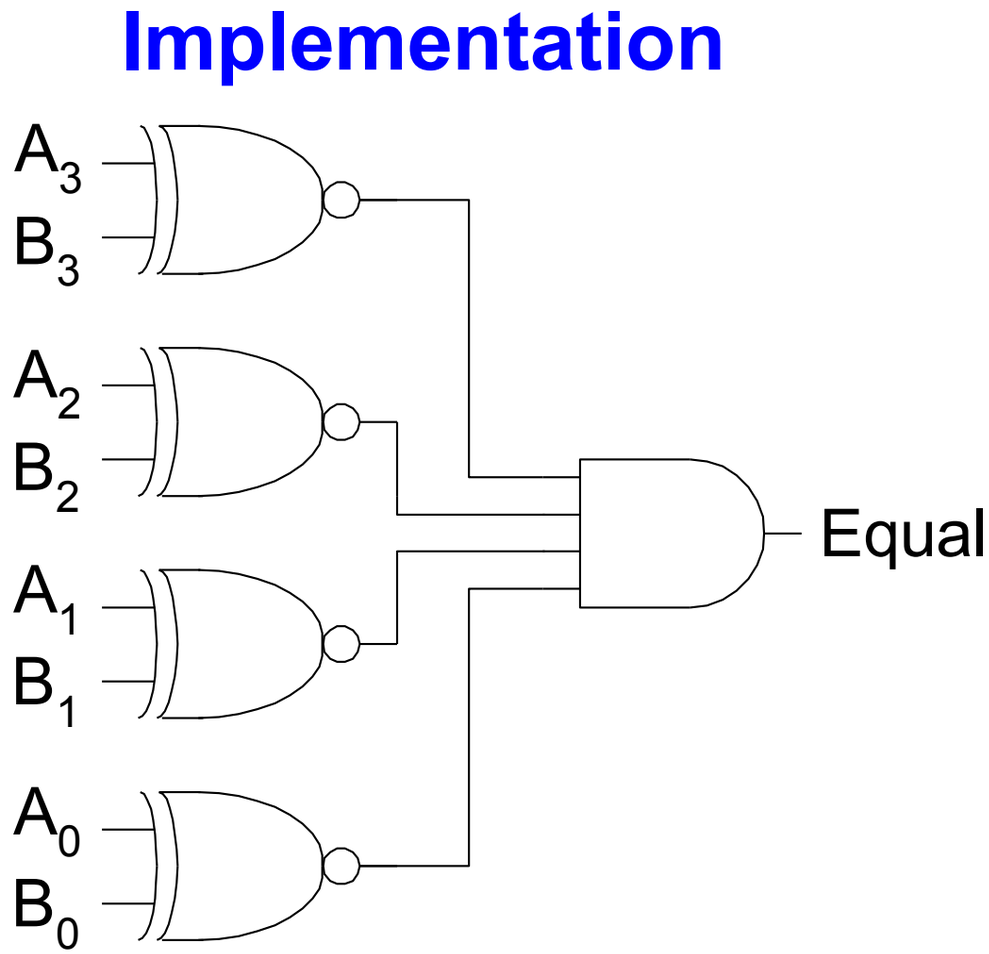
Comparator: Equality

- A comparator determines whether two binary numbers are **equal** or if one **is greater or less than** the other.

Symbol

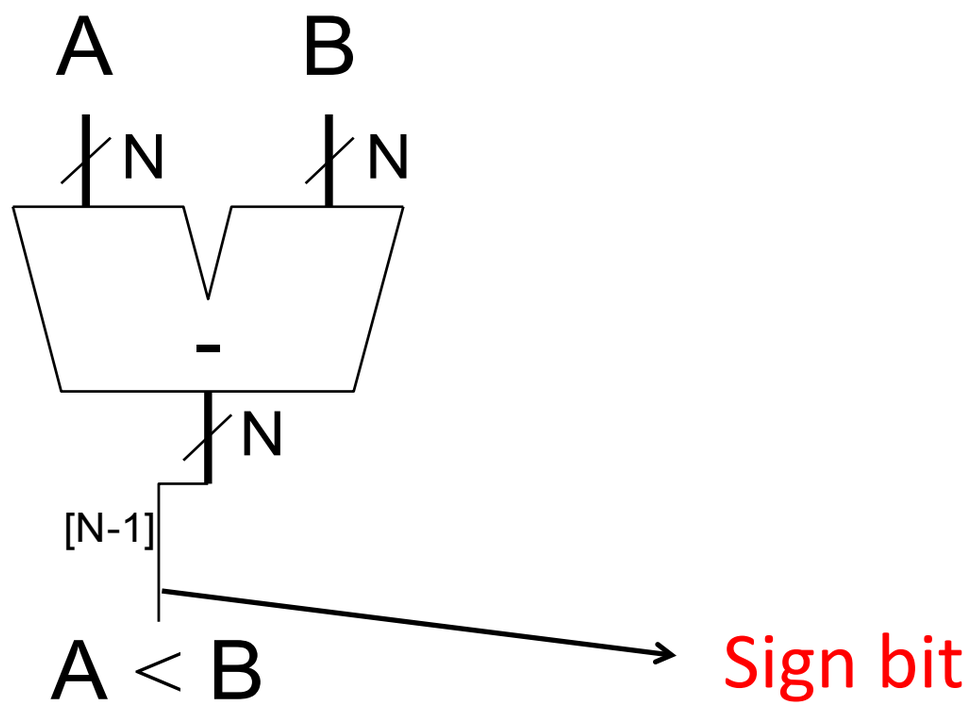


Implementation



Comparator: Less Than

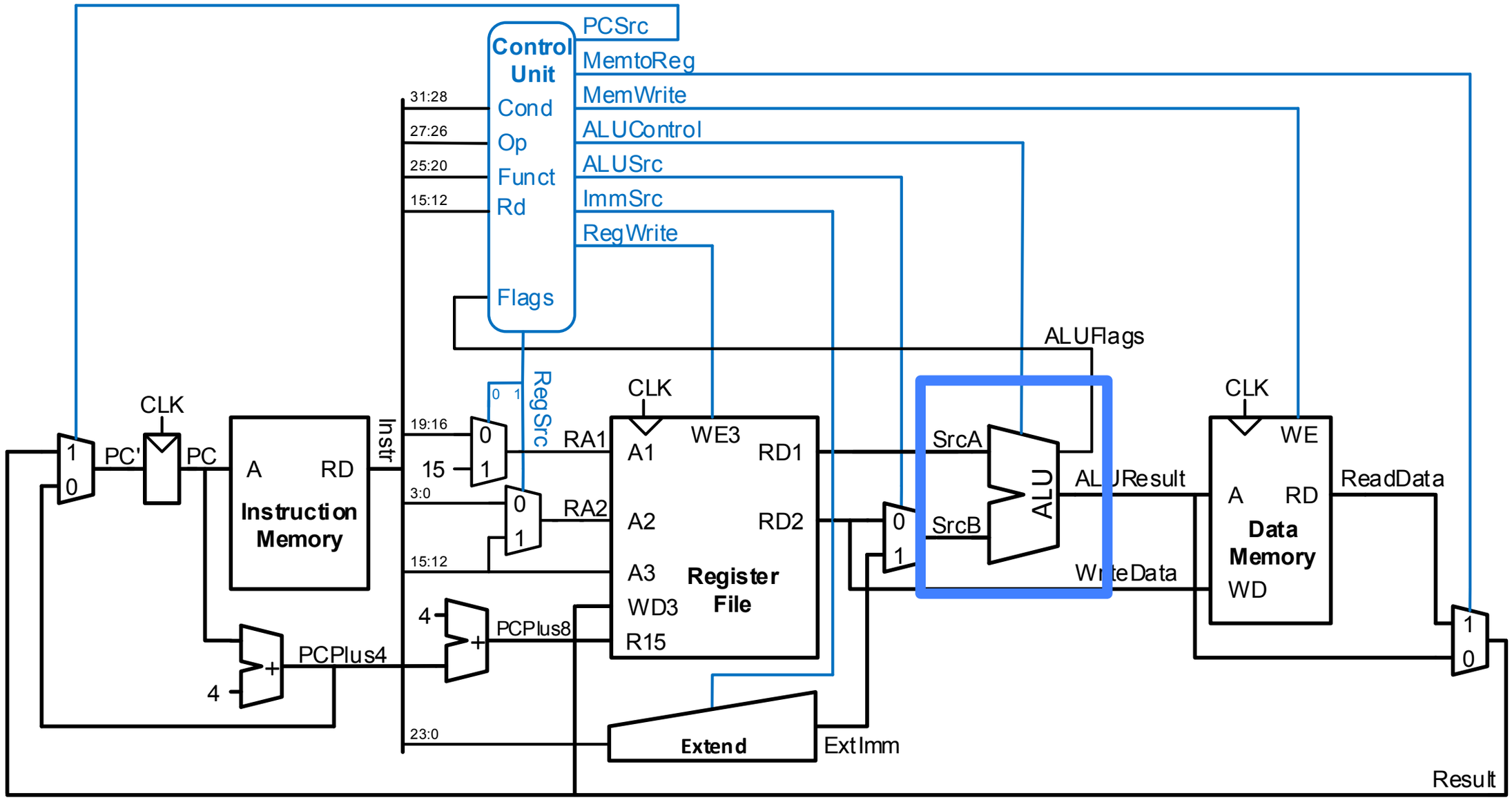
- Compute A − B and looking at **the sign** (most significant bit) of the result.
  - If the result is negative (i.e., the sign bit is 1), then A is less than B. Otherwise A is greater than or equal to B.



- This comparator, however, functions **incorrectly** upon **overflow**.

Arithmetic/Logical Unit

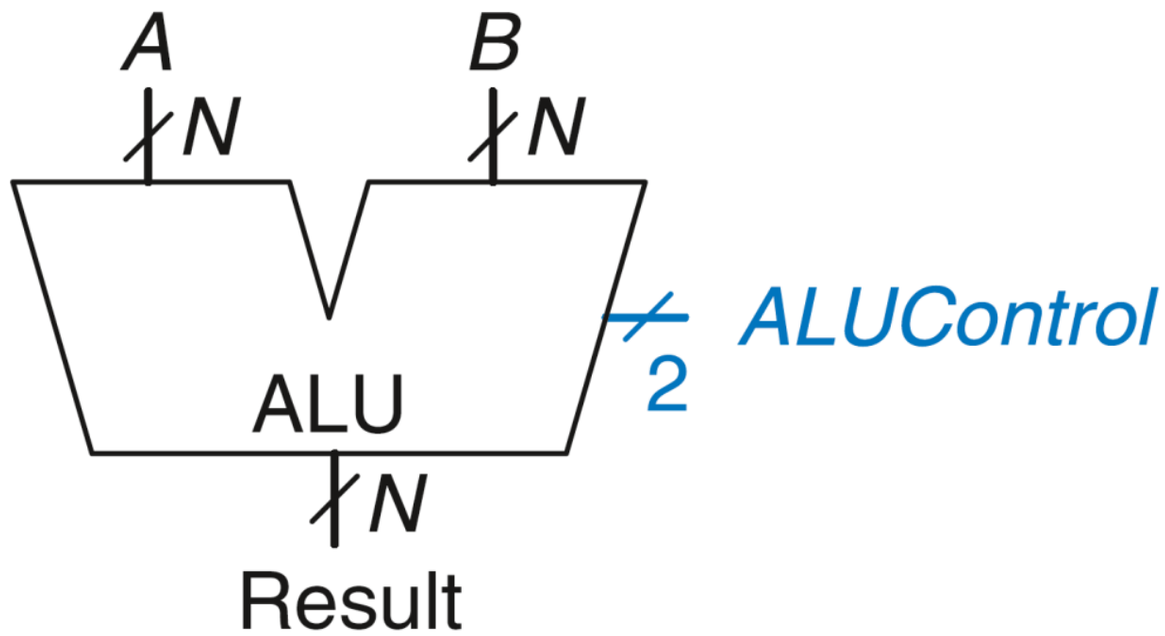
- ALU is one of the main components in the microprocessor.



ALU: Arithmetic/Logical Unit

- An Arithmetic/Logical Unit (ALU) combines a variety of **mathematical** and **logical** operations into a single unit.
- The ALU forms the heart of most computer systems.

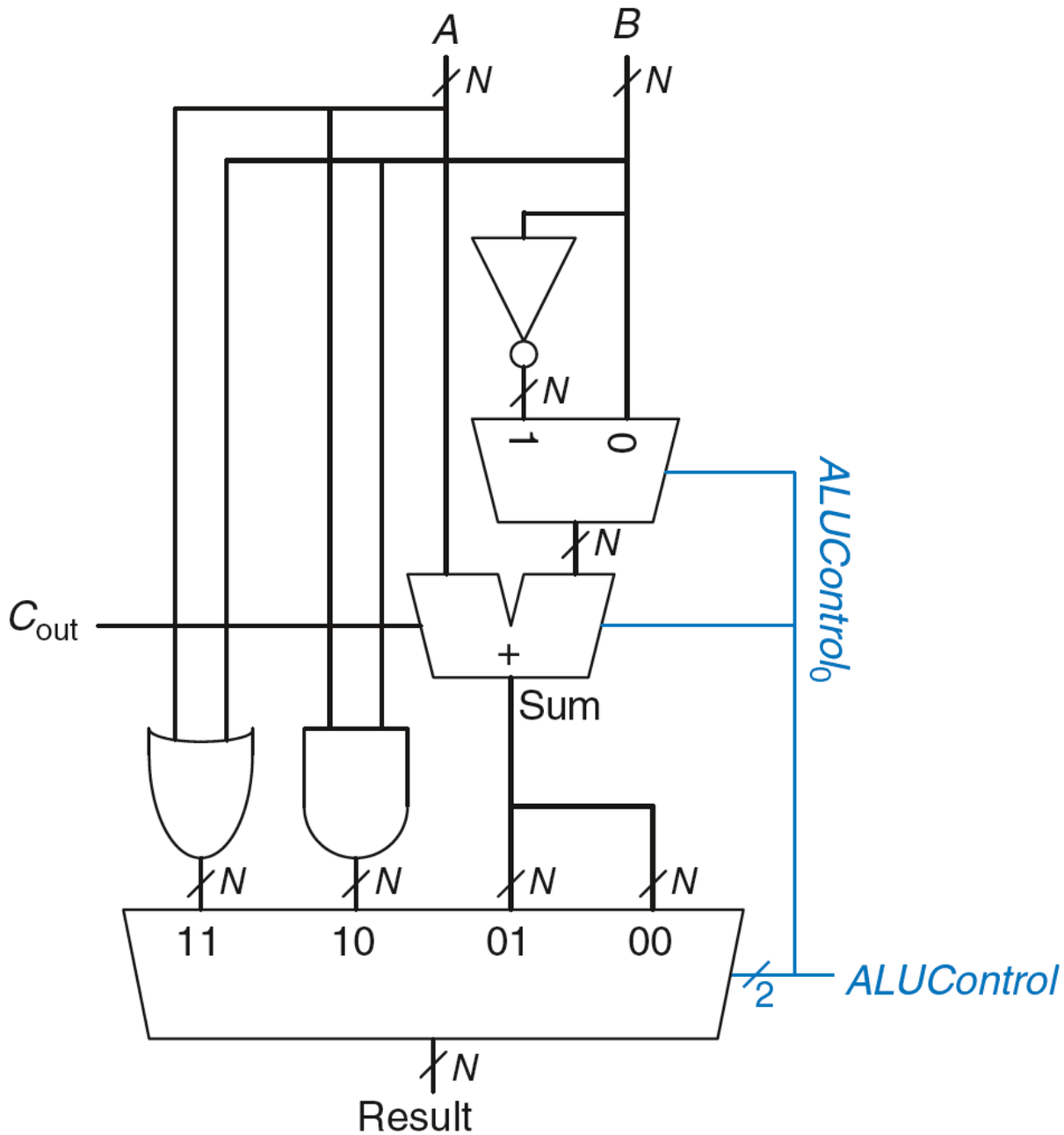
$ALUControl_{1:0}$	Function
00	**Add**
01	**Subtract**
10	**AND**
11	**OR**



The ALU receives a 2-bit **control signal** *ALUControl* that specifies which function to perform.

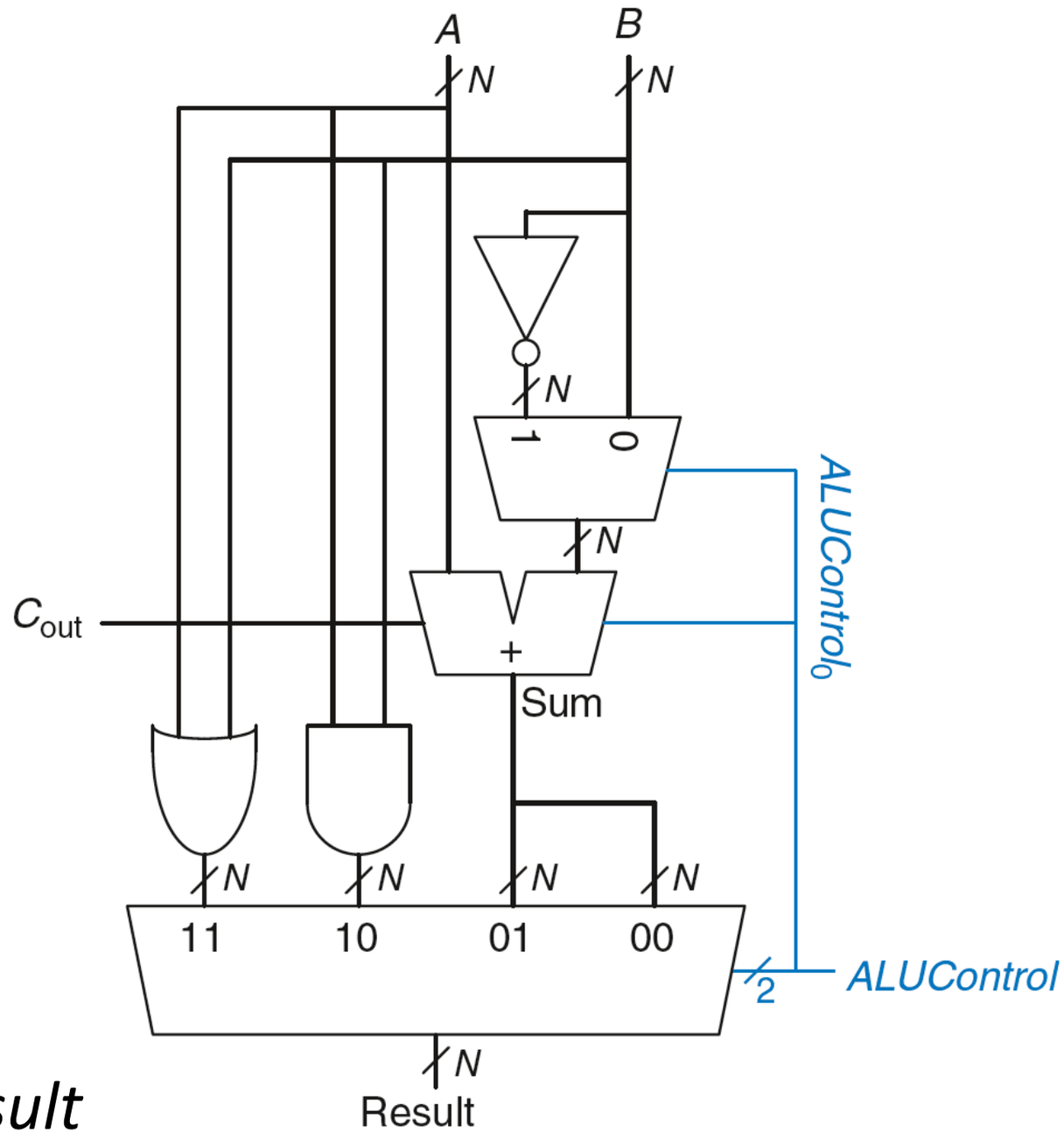
ALU Implementation

$ALUControl_{1:0}$	Function
00	Add
01	Subtract
10	AND
11	OR



ALU Implementation

$ALUControl_{1:0}$	Function
00	Add
01	Subtract
10	AND
11	OR



Example: Perform ***A*** **OR** ***B***

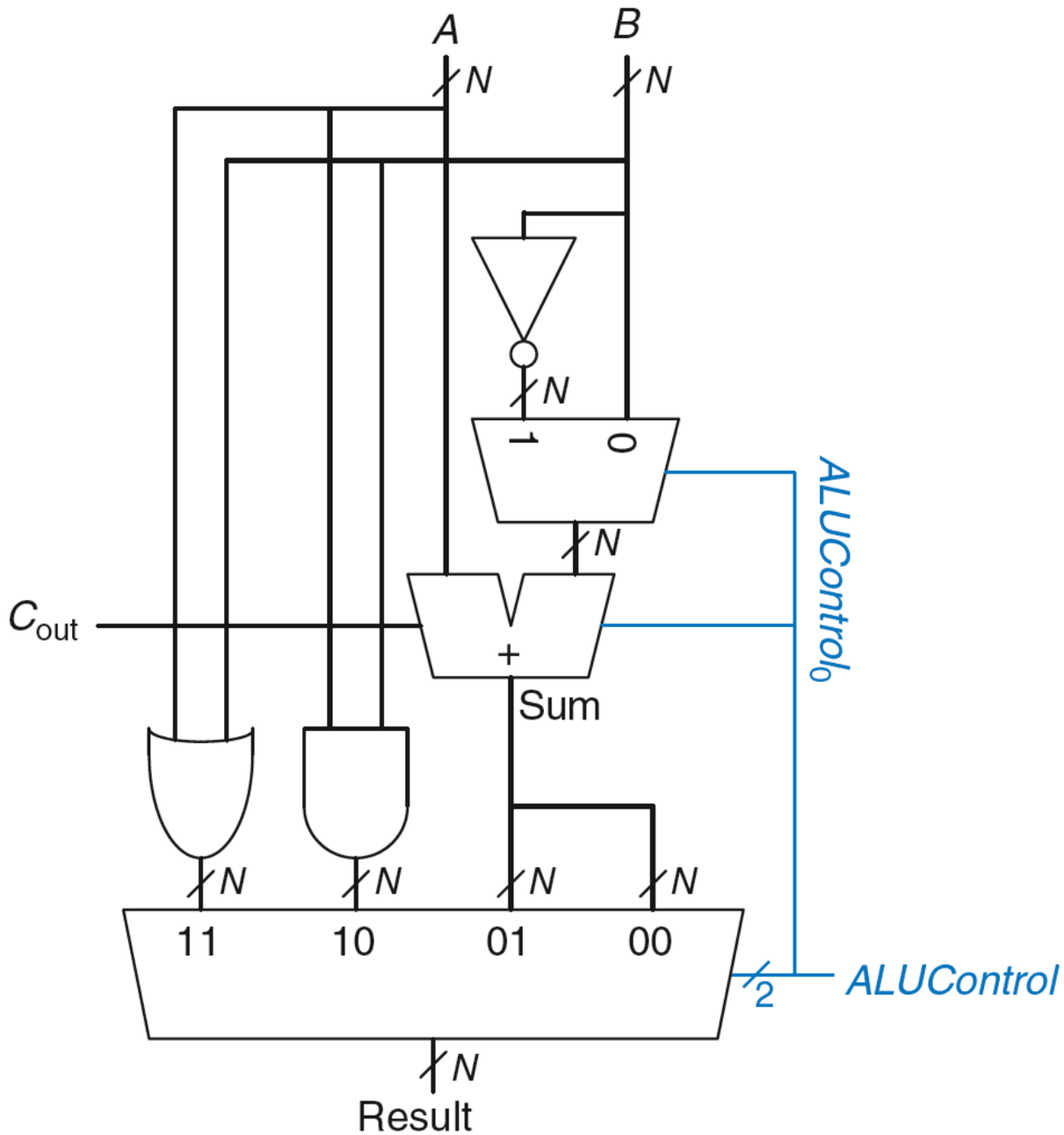
$ALUControl_{1:0}$ = 11

Mux selects output of OR gate as *Result*

Result **=** ***A*** **OR** ***B***

ALU Implementation

$ALUControl_{1:0}$	Function
00	Add
01	Subtract
10	AND
11	OR



Example: Perform *A* + *B*

$ALUControl_{1:0}$ = 00

C_{in} to adder = 0

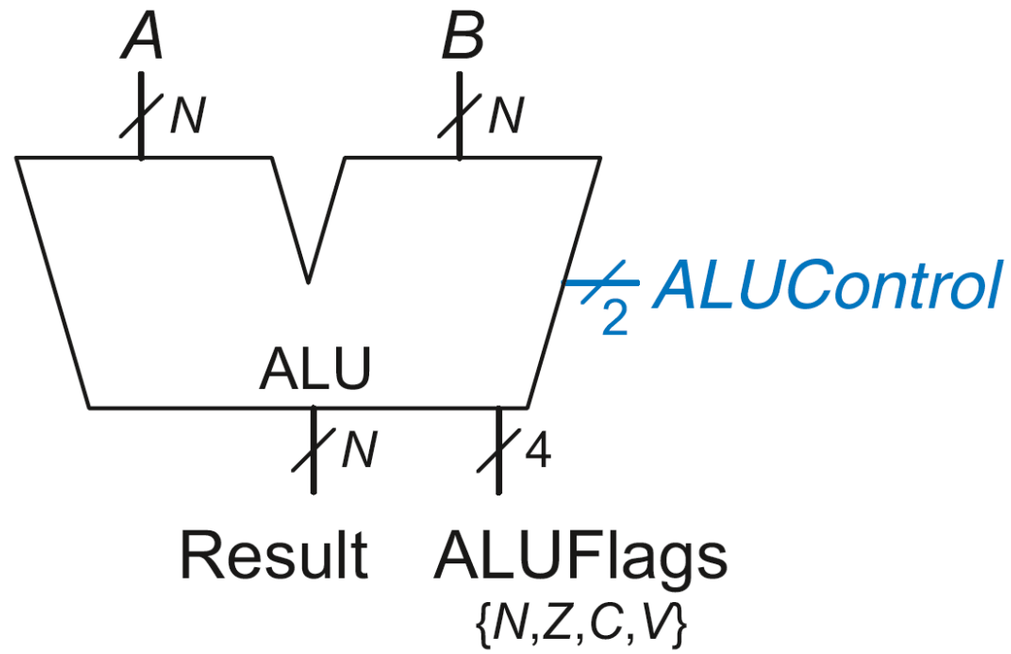
2nd input to adder is *B*

Mux selects *Sum* as *Result*

Result* = *A* + *B

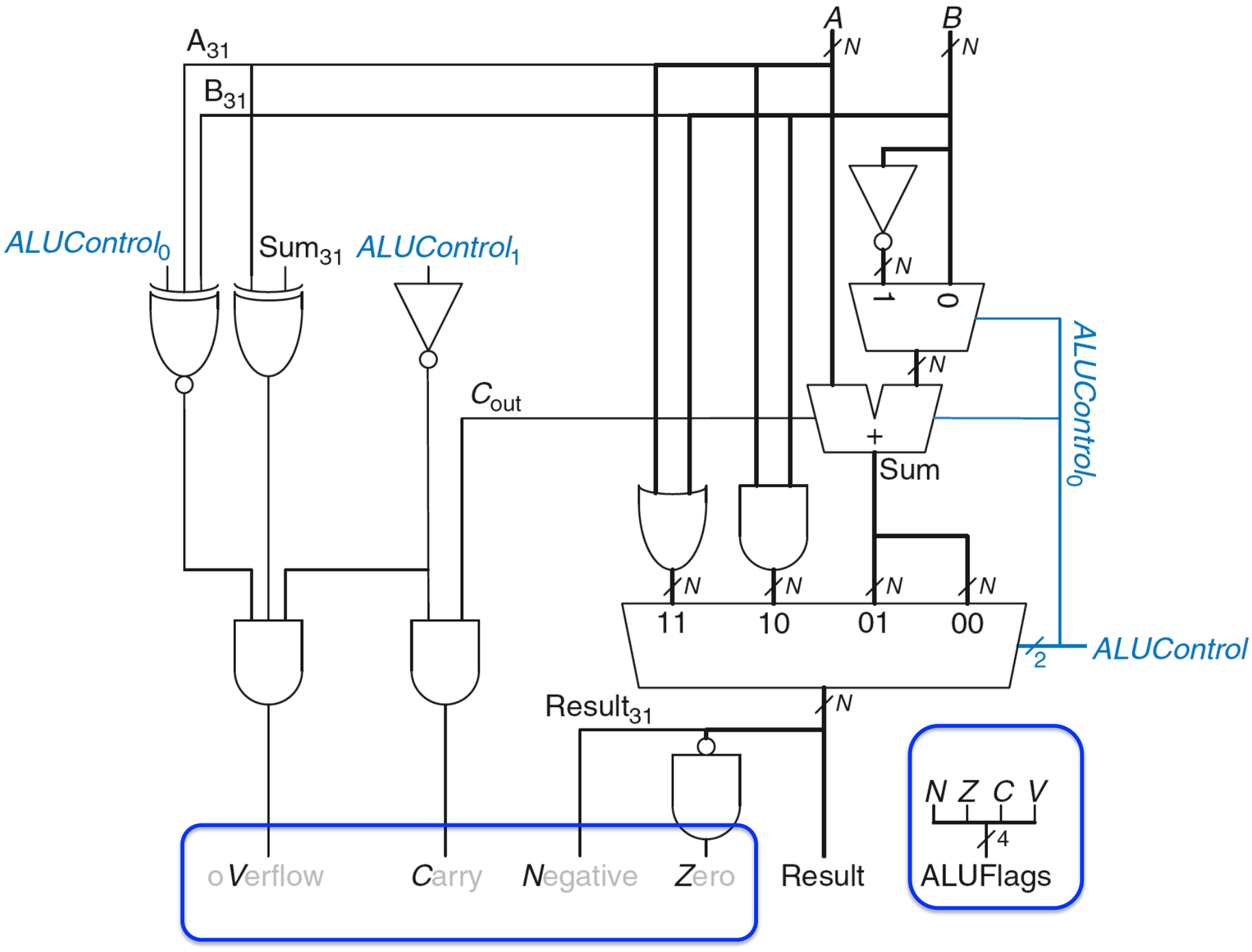
ALU with Status Flags

Flag	Description
N	*Result* is **N**egative
Z	*Result* is **Z**ero
C	Adder produces **C**arry out
V	Adder o**V**erflowed

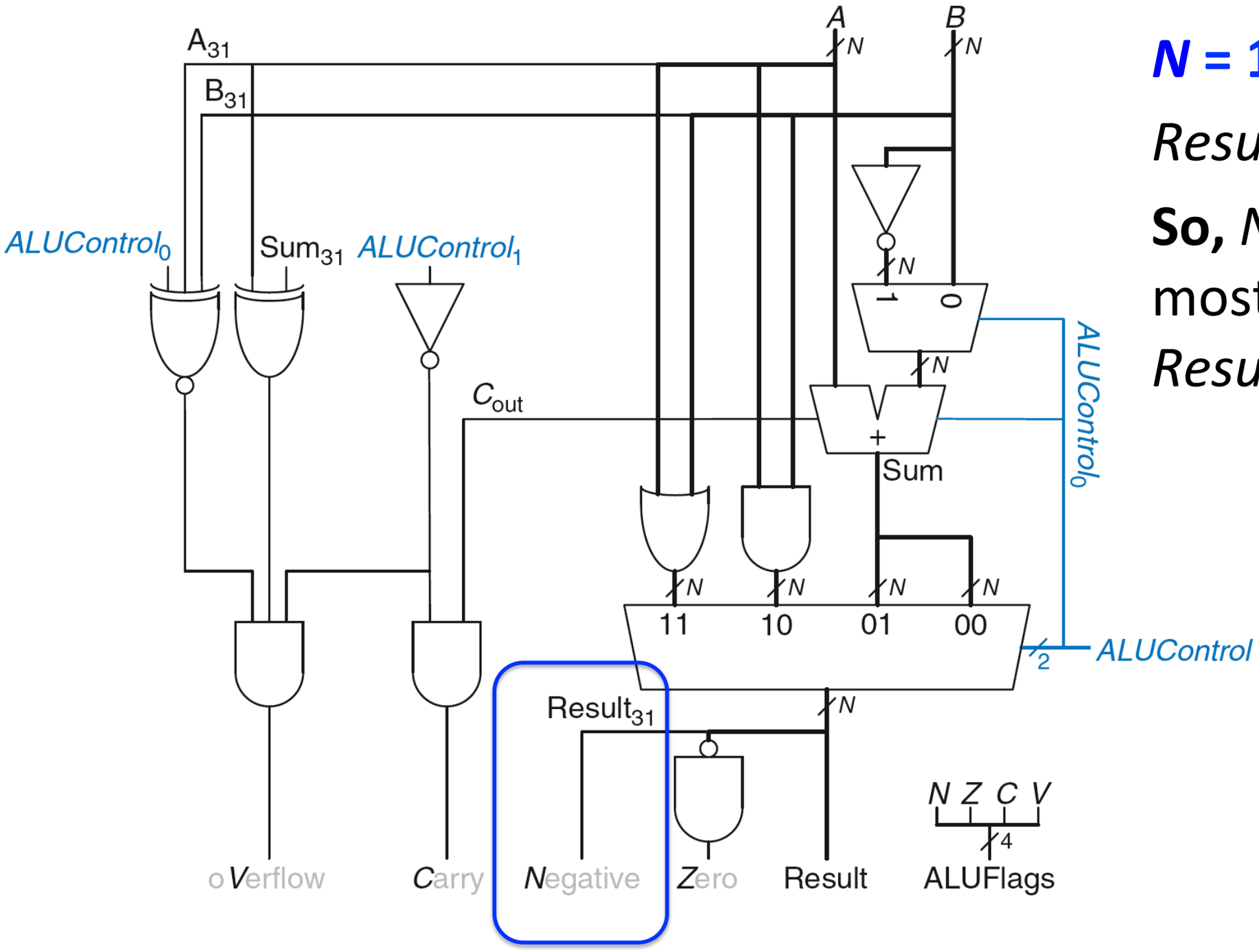


Some ALUs produce extra outputs, called **flags**, that indicate **information** about the ALU output.

ALU with Status Flags



ALU with Status Flags: Negative

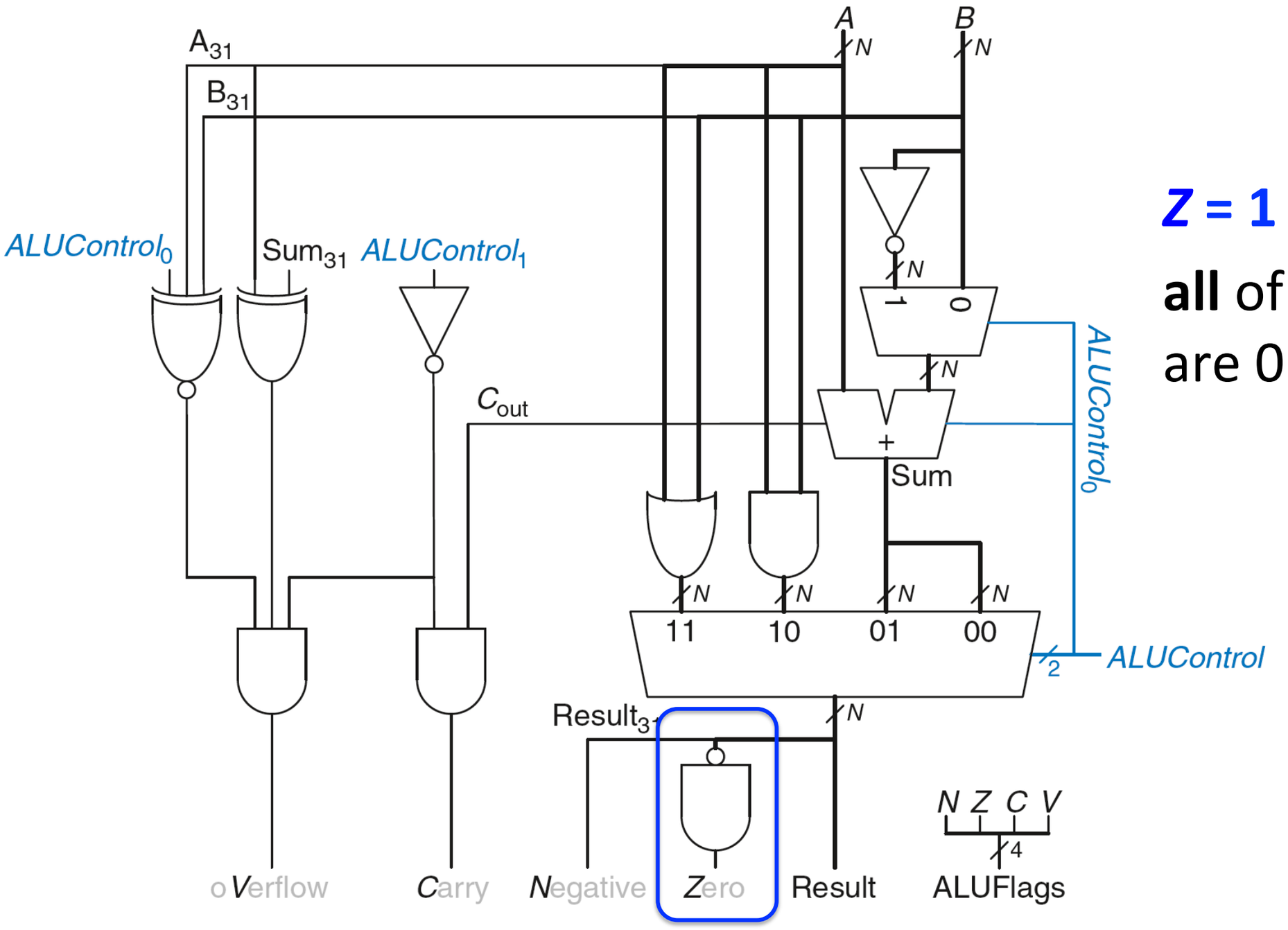


***N* = 1** if:

Result is **negative**

So, *N* is connected to most significant bit of *Result*

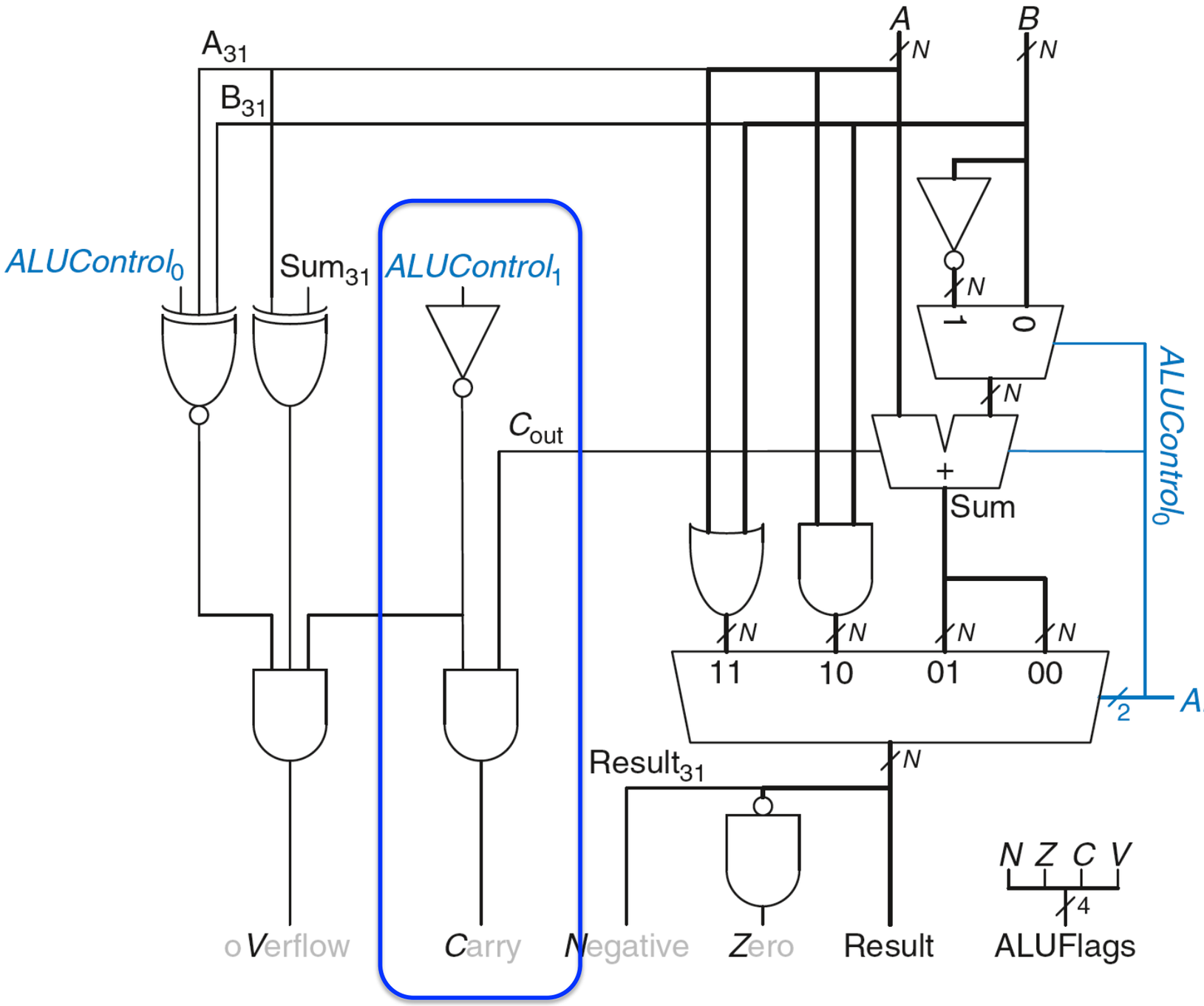
ALU with Status Flags: Zero



Z = 1 if:

all of the bits of *Result* are 0

ALU with Status Flags: Carry



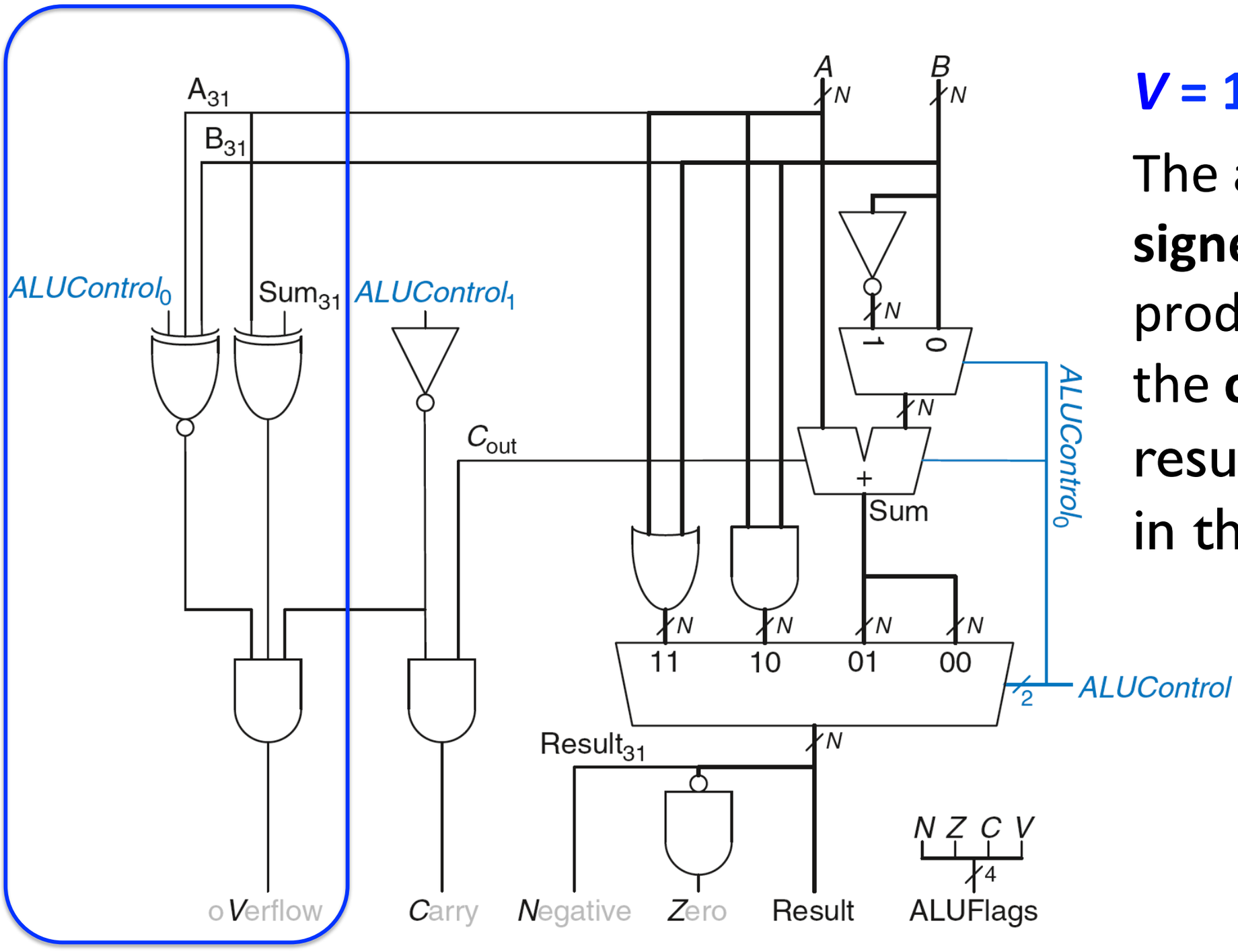
***C* = 1** if:

C_{out} of Adder is 1

AND

ALU is adding or subtracting (ALUControl is 00 or 01)

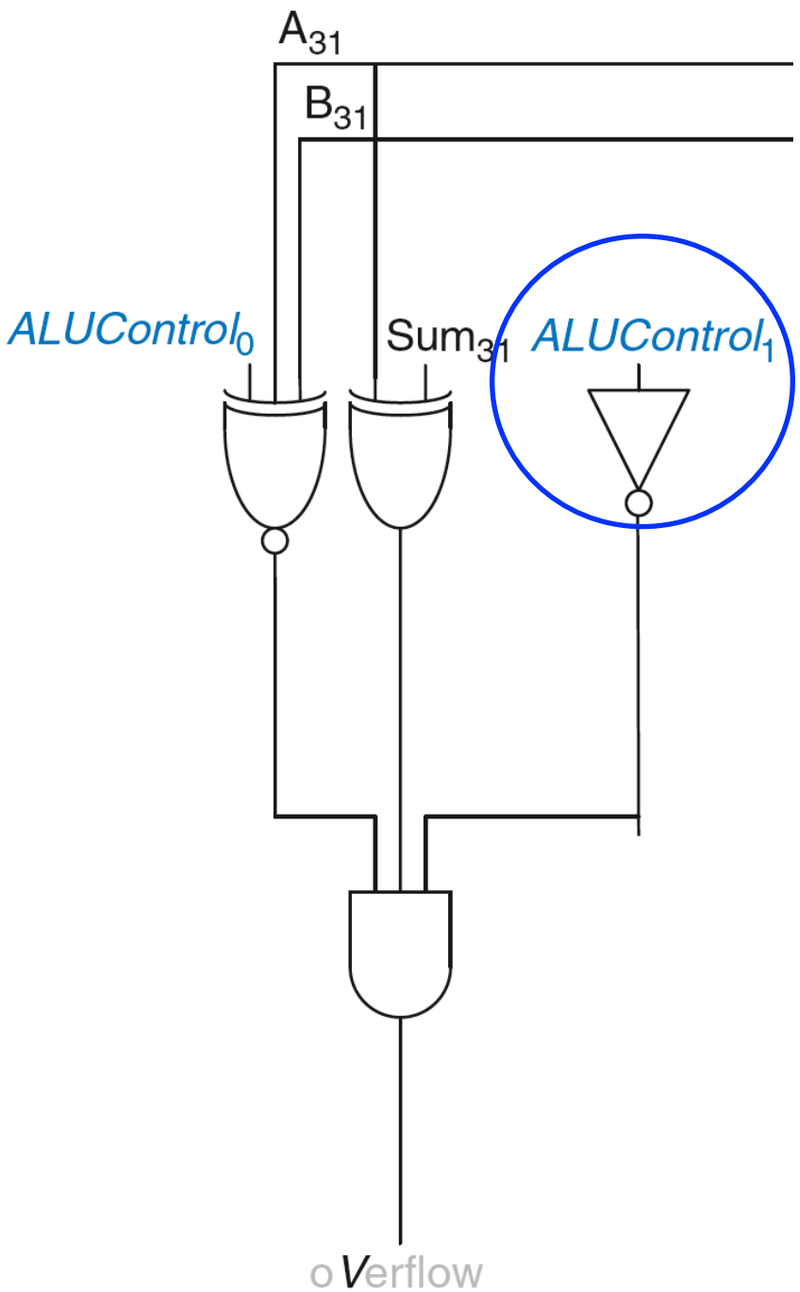
ALU with Status Flags: oVerflow



***V* = 1** if:

The addition of 2 **same-signed numbers** produces a result with the **opposite sign**. (the result is too big to fit in the available digits.)

ALU with Status Flags: o**V**erflow

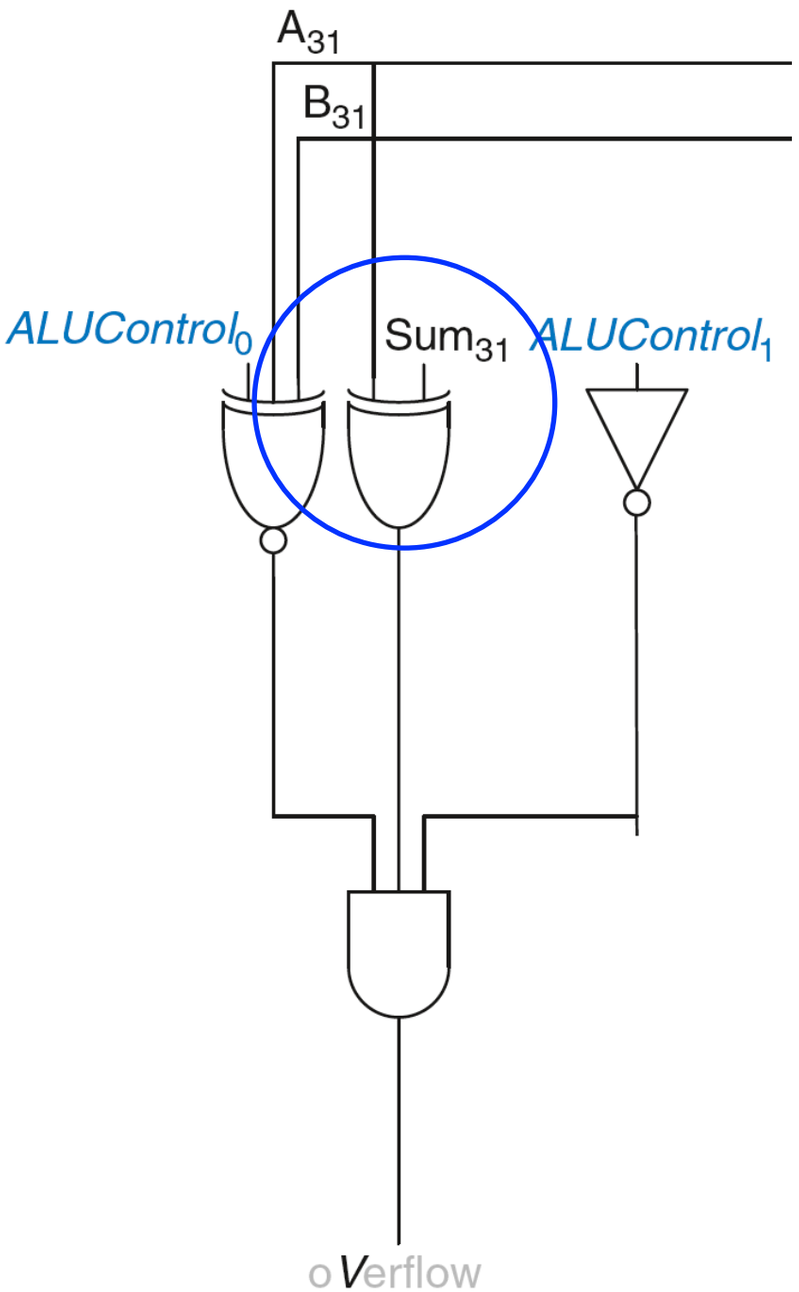


***V* = 1** if:

ALU is performing addition or subtraction

($ALUControl_1 = 0$)

ALU with Status Flags: o**V**erflow



***V* = 1** if:

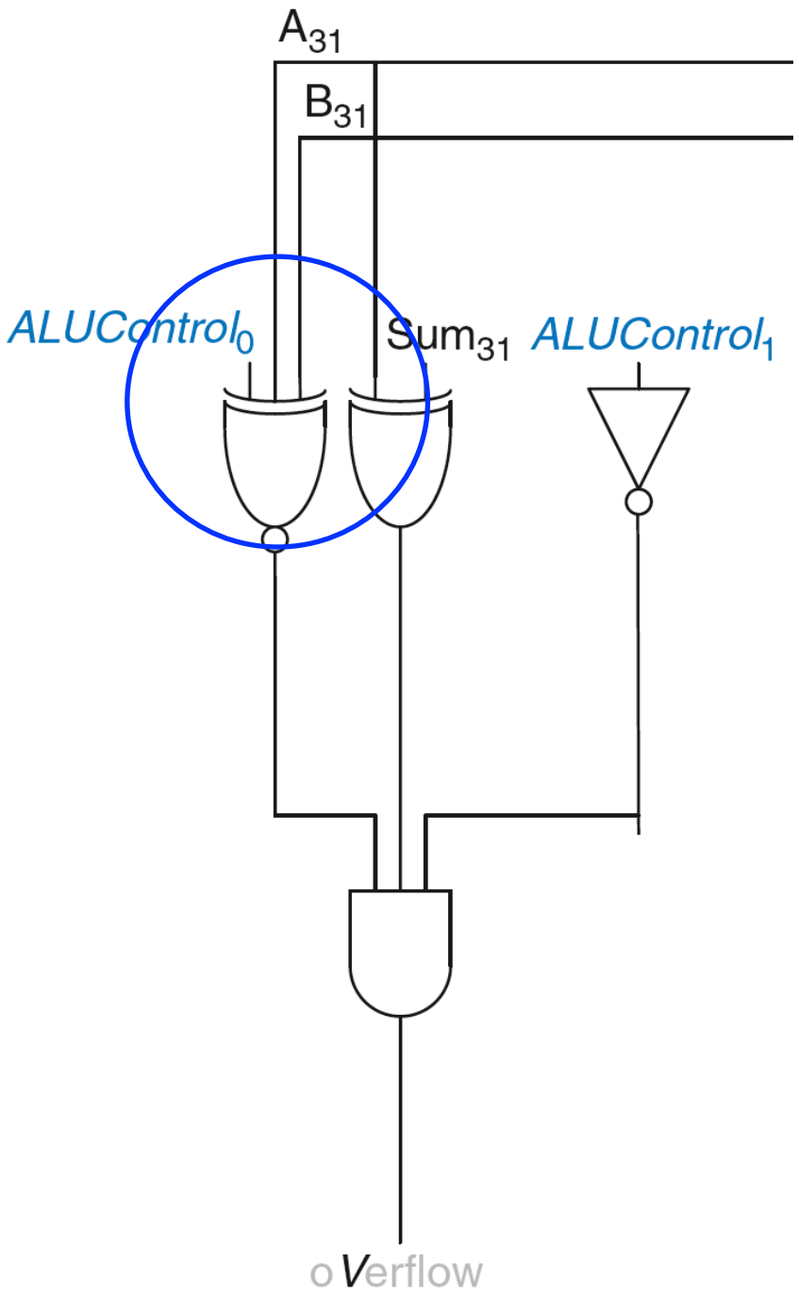
ALU is performing addition or subtraction

(*ALUControl*$_1$ = 0)

AND

A and Sum have opposite signs

ALU with Status Flags: o**V**erflow



***V* = 1** if:

ALU is performing addition or subtraction

(*ALUControl*$_1$ = 0)

AND

A and Sum have opposite signs

AND

A and B have same signs upon addition

(*ALUControl*$_0$ = 0)

OR

A and B have different signs upon subtraction

(*ALUControl*$_0$ = 1)

Shifters/Rotators

- Shifters and rotators move bits and multiply or divide by powers of 2.
- As the name implies, a shifter **shifts** a binary number left or right by a specified number of positions.

Logical shifter: shifts value to left or right and fills empty spaces with 0's

- Ex: 11001 >> 2 = 00110
- Ex: 11001 << 2 = 00100

Shifters/Rotators

Arithmetic shifter: right shift, fills empty spaces with the old most significant bit (msb)

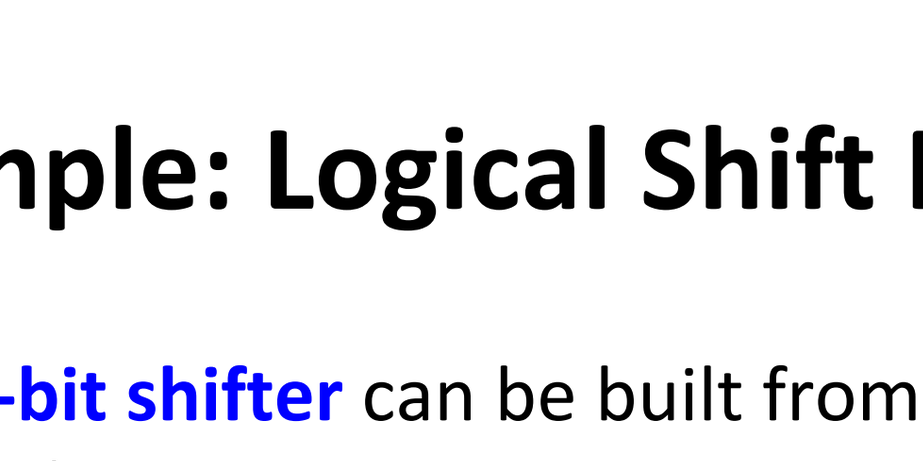
- Ex: 11001 >>> 2 = 11110
- Ex: 11001 <<< 2 = 00100

Rotator: rotates bits in a circle, such that bits shifted off one end are shifted into the other end

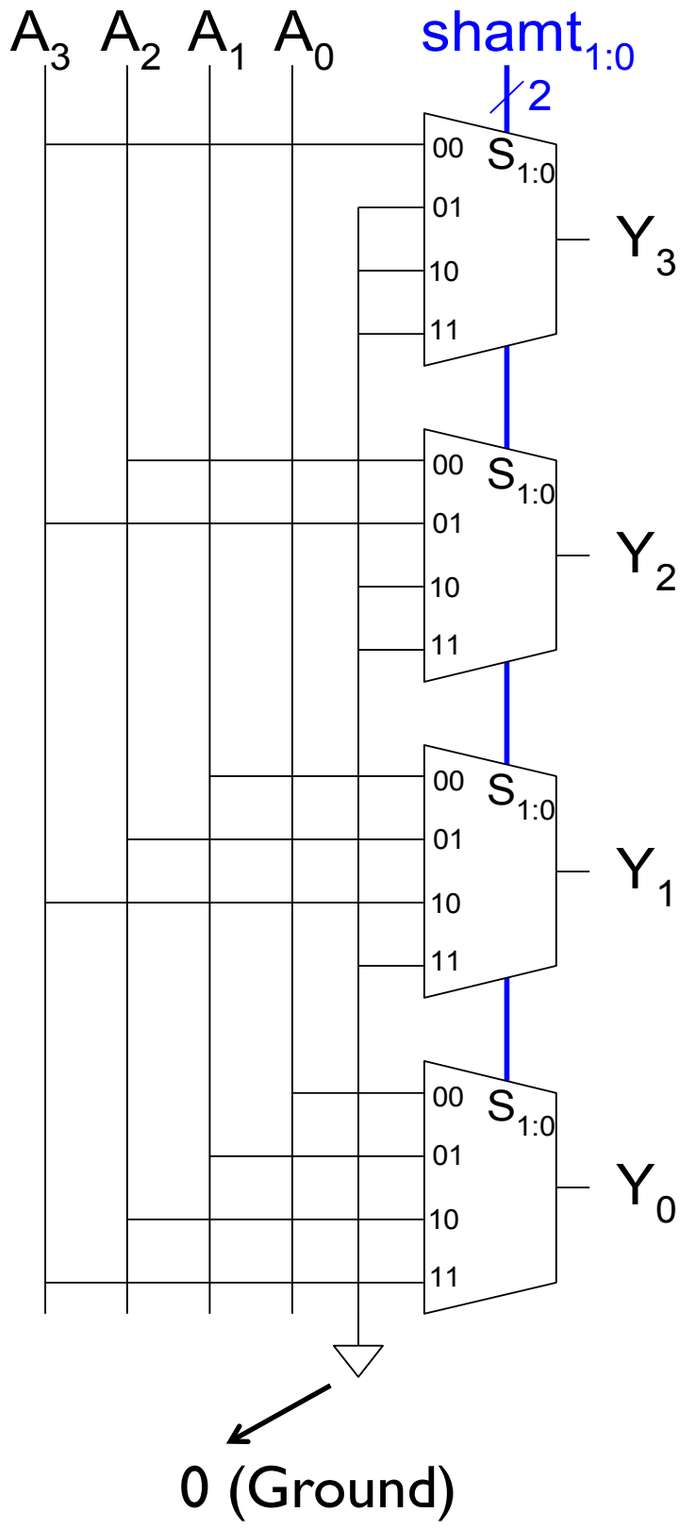
- Ex: 11001 ROR 2 = 01110
- Ex: 11001 ROL 2 = 00111

Example: Logical Shift Right Implementation

- An **N-bit shifter** can be built from N N:1 multiplexers.
- The input is shifted by 0 to N−1 bits, depending on the value of the $\log_2$N-bit select lines.



Depending on the value of the 2-bit shift amount $shamt_{1:0}$, the output Y receives the input A shifted by 0 to 3 bits.



Further Reading

- You can read **Chapter 5** of your book
  - **From Section 5.1 to 5.2.5**