Gates and Circuits

9/13/16

You're going to want scratch paper today … borrow some if needed.

The system stack

This class

software

c program

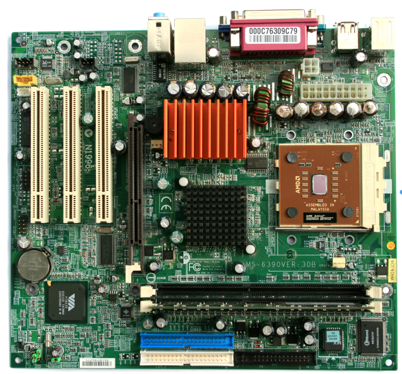
compiler

shell

operating system

Starting this week

hardware



memory

CPU

circuits

gates

electrical engineering

transistors

wires

How a Computer Runs a Program

C Program	*How C program is run on System:*
Binary Program	How instructions & data are encoded
Operating System	OS Abstractions, Resource management
Computer Hardware	How underlying HW organized & works

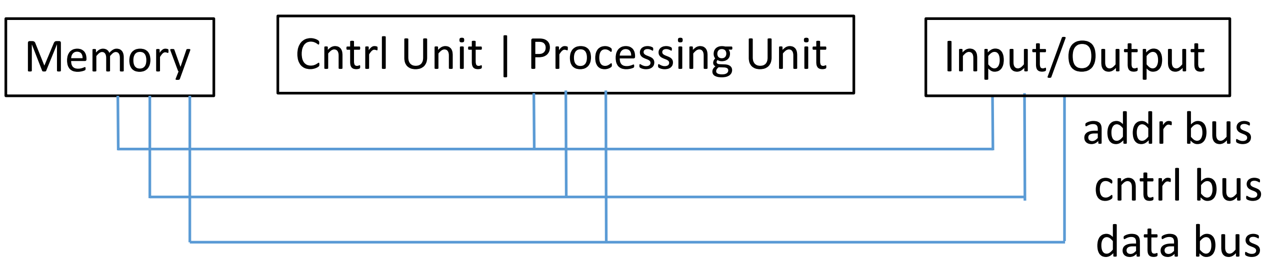
What we know so far:

- Much of the C programming language
  - types, operators, arrays, parameter passing, some structs
- Binary encodings & sizes for different C types
  - char, unsigned char, int, unsigned int, …
- How to perform binary operations (Add, Sub)

Von Neumann Architecture

- A computer is a generic computing machine:
  - Based on Alan Turing's Universal Turing Machine
  - Stored program model: computer stores program rather than encoding it (feed in data and instructions)
  - <u>No distinction between data and instructions memory</u>

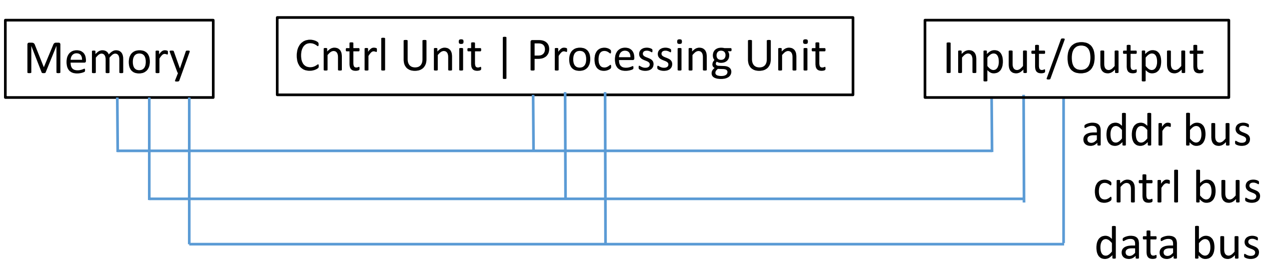
- 5 parts connected by buses (wires):
  - Memory, Control, Processing, Input, Output



Memory

- Stores instructions and data.
- Addressable, like array indices.
  - addr 0, 1, 2, …

- Memory Address Register: address to read/write
- Memory Data Register: value to read/write

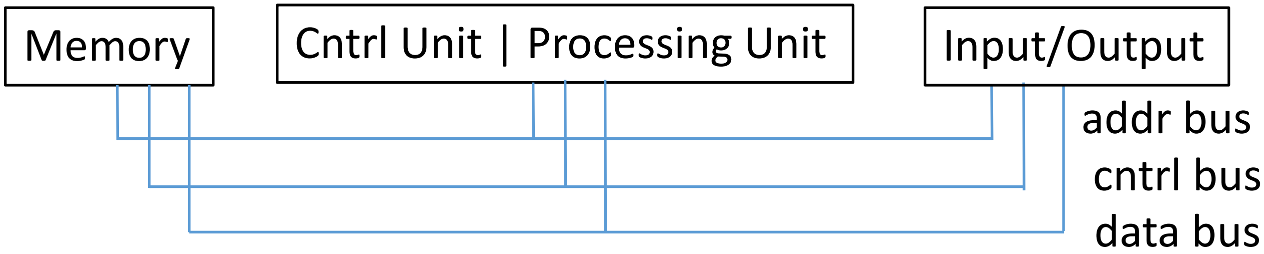


Central Processing Unit (CPU)

- Processing Unit: executes instructions selected by the control unit
  - ALU (arithmetic logic unit): simple functional units: ADD, SUB, AND…
  - Registers: temporary storage directly accessible by instructions
- Control unit: determines the order in which instructions execute
  - PC: program counter: address of next instruction
  - IR: instruction register: holds current instruction
  - clock-based control: clock signal+IR trigger state changes

Input/Output

- Keyboard
- Files on the hard drive
- Network communication



First Goal: Build a model of the CPU

Three main classifications of HW circuits:

1. ALU: implement arithmetic & logic functionality
   (ex) adder to add two values together

2. Storage: to store binary values
   (ex) Register File: set of CPU registers, Also: main memory (RAM)

3. Control: support/coordinate instruction execution
   (ex) fetch the next instruction to execute

Abstraction

User / Programmer
Wants low complexity

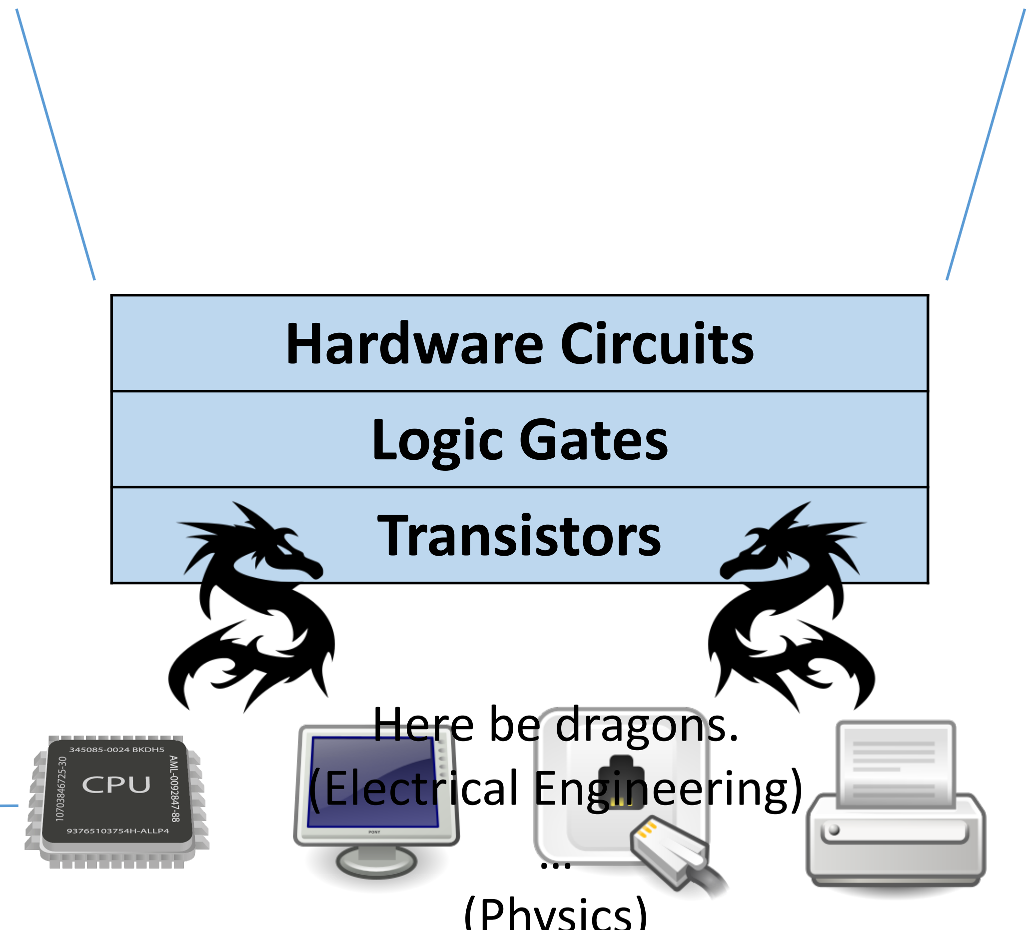
Applications
Specific functionality

Software library
Reusable functionality

Operating system
Manage resources

Complex devices
Compute & I/O

Abstraction

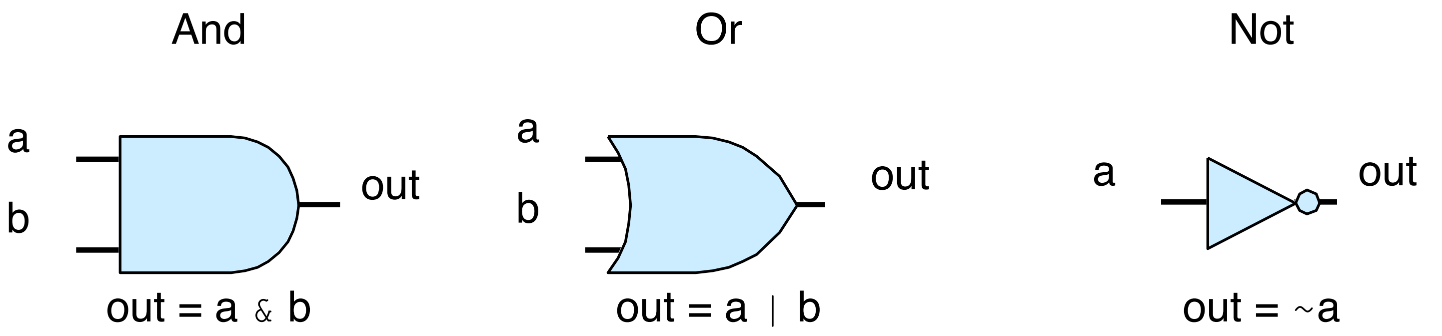


Complex devices
Compute & I/O

Logic Gates

Input: Boolean value(s) (high and low voltages for 1 and 0)

Output: Boolean value, the result of a Boolean function

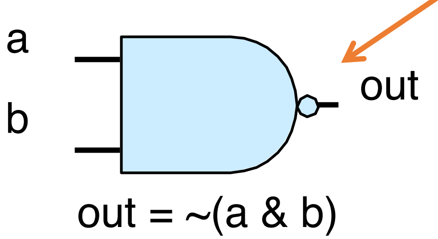


A	B	A & B	A | B	~A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

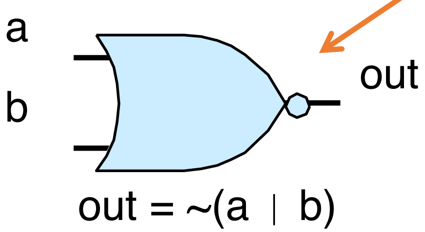
More Logic Gates

Note the circle on the output.
This means “negate it.”

NAND



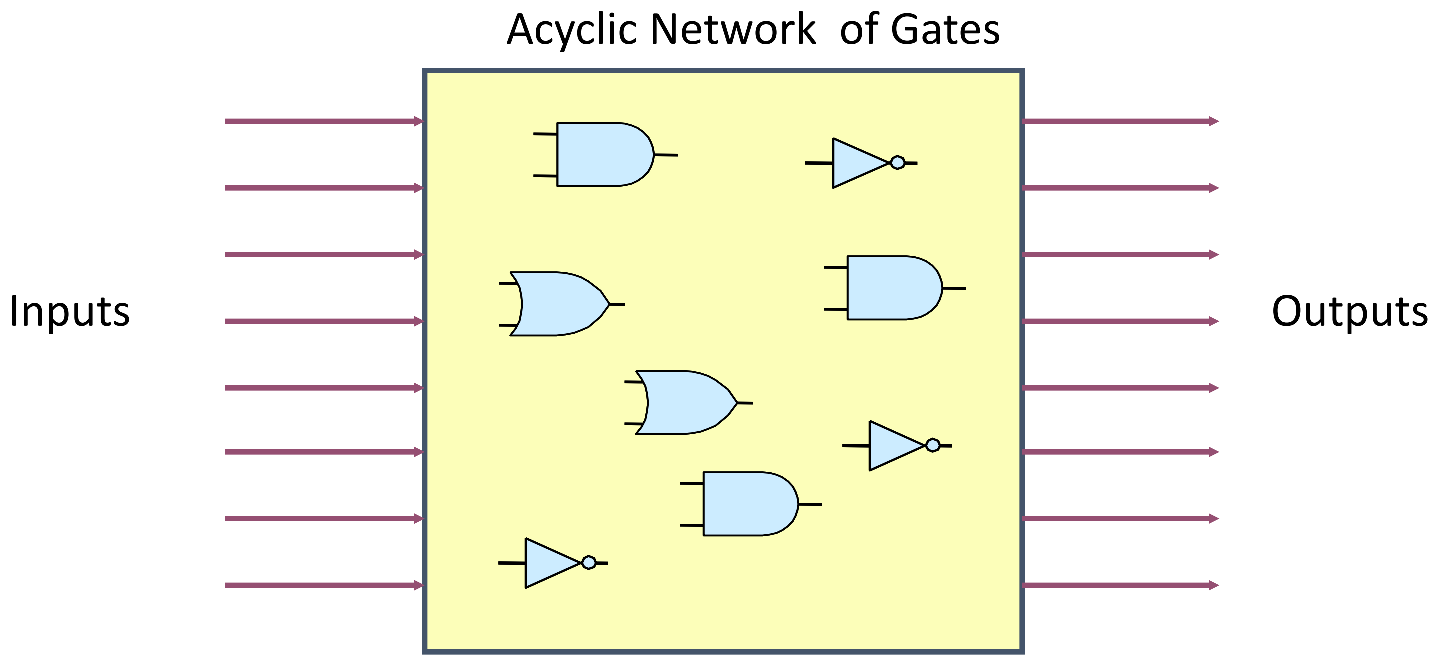
NOR



A	B	A NAND B	A NOR B
0	0	1	1
0	1	1	0
1	0	1	0
1	1	0	0

Combinational Logic Circuits

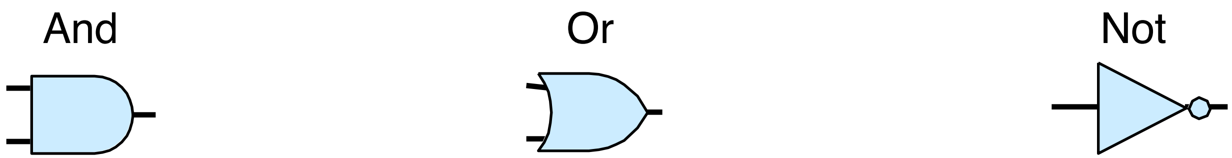
- Build up higher level processor functionality from basic gates.

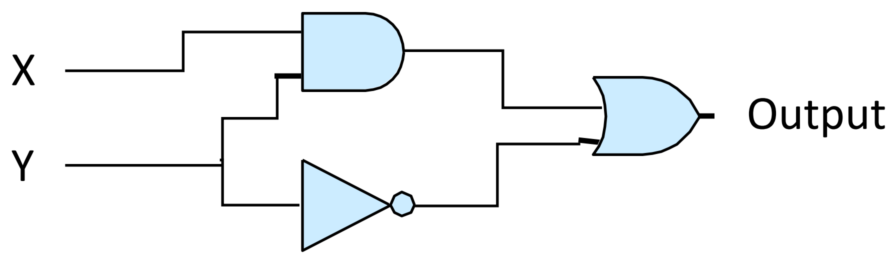


Outputs are Boolean functions of inputs.

Outputs continuously respond to changes to inputs.

What does this circuit output?





Clicker Choices

X	Y	Out_A	Out_B	Out_C	Out_D	Out_E
0	0	0	1	0	1	0
0	1	0	1	0	0	1
1	0	1	0	1	1	1
1	1	0	0	1	1	0

Build new gates

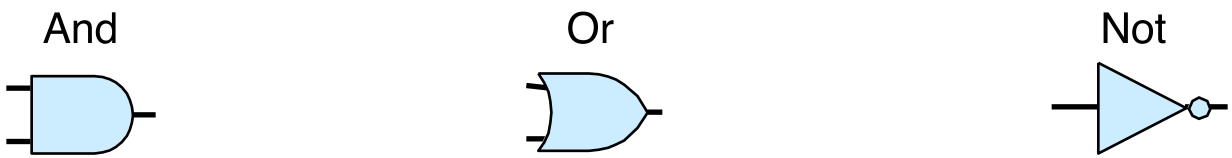
- Build-up XOR from basic gates (AND, OR, NOT)

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Q: When is A^B ==1?

```
A ^ B  == (~A & B) | (A & ~B)
```

Which of these is an XOR circuit?

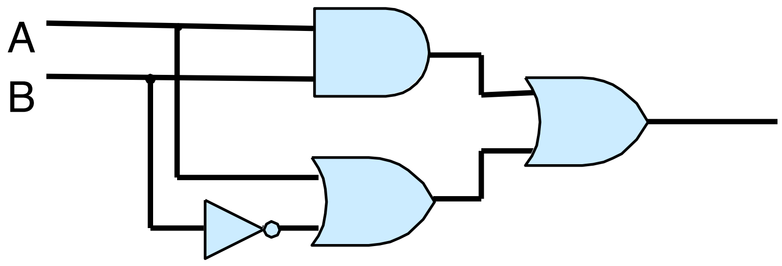


Draw an XOR circuit using AND, OR, and NOT gates.

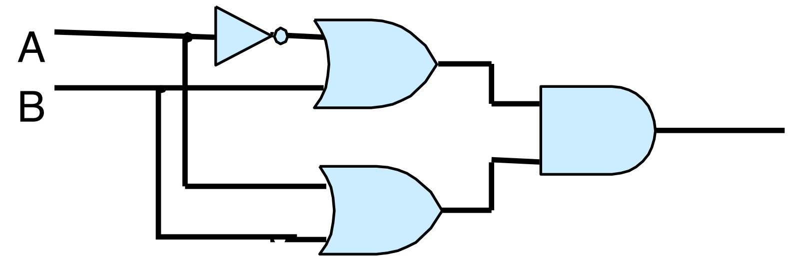
I'll show you the clicker options after you've had some time.

Which of these is an XOR circuit?

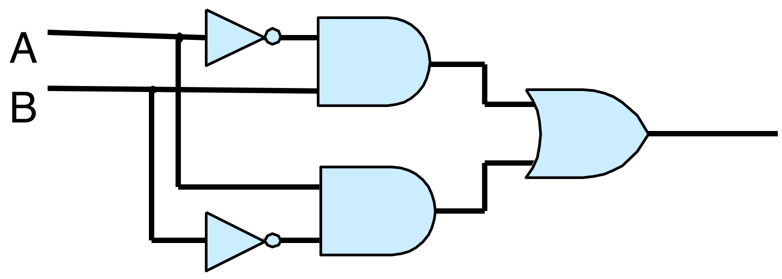
A:



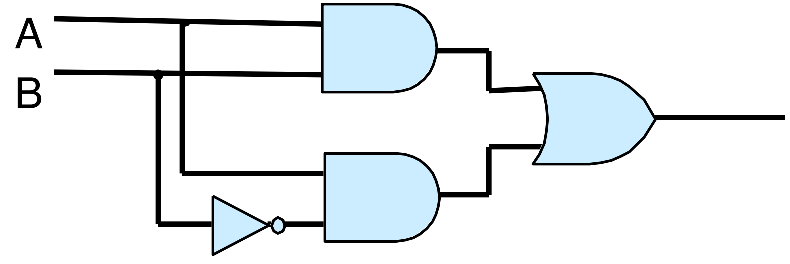
B:



C:



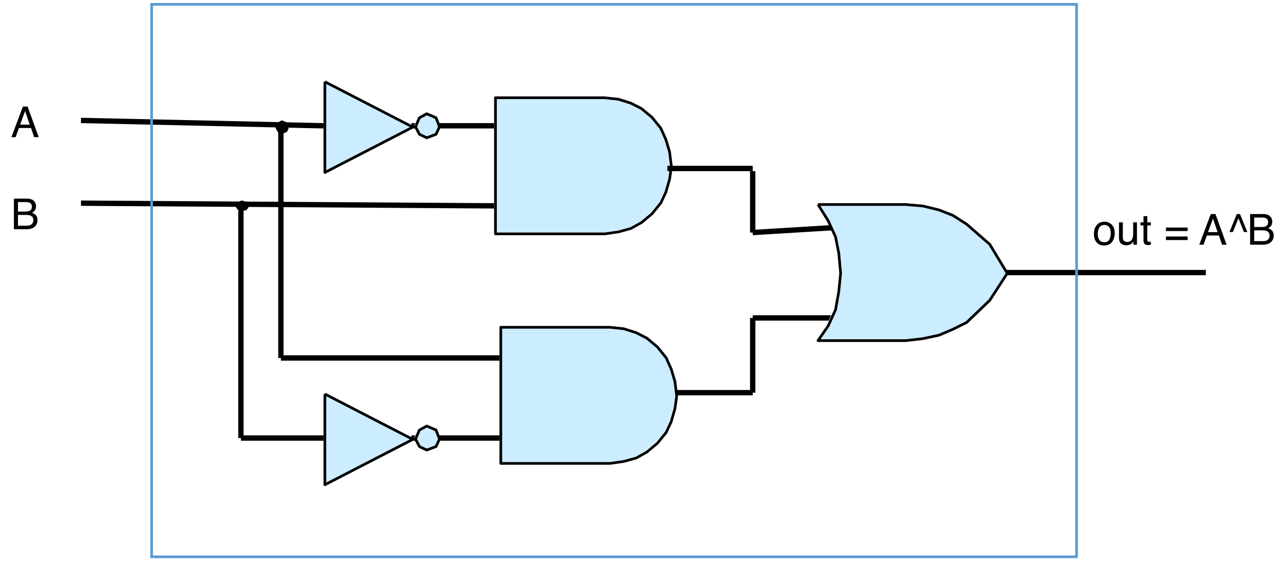
D:



E: None of these is an XOR.

Checking the XOR circuit

`A^B  ==  (~A & B) | (A & ~B)`



A:0 B:0 A^B: 0

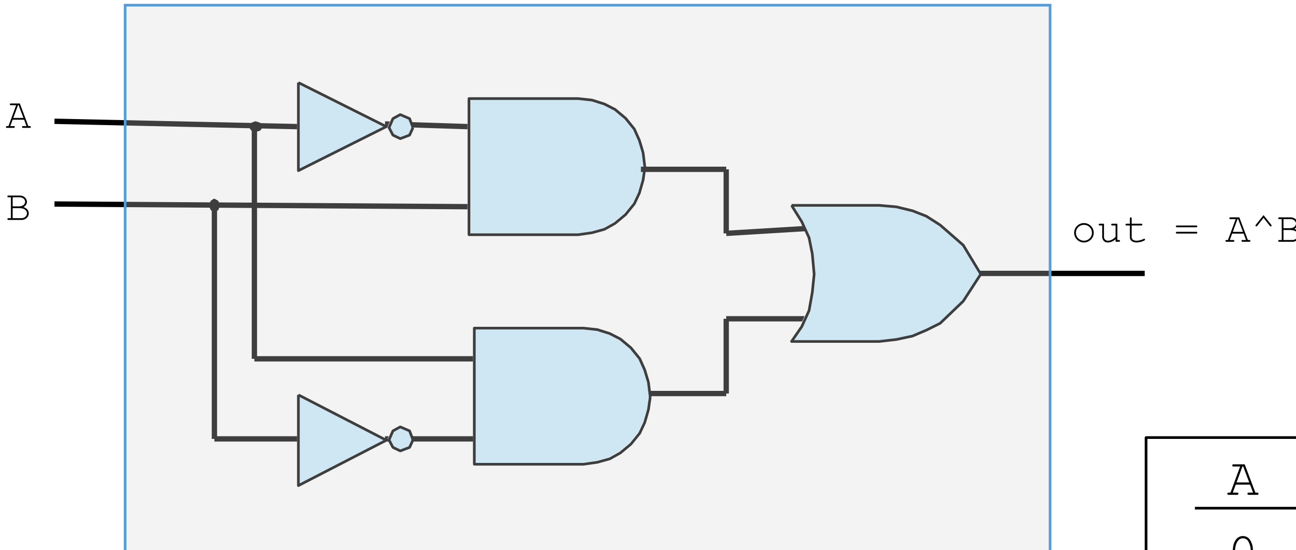
A:0 B:1 A^B: 1

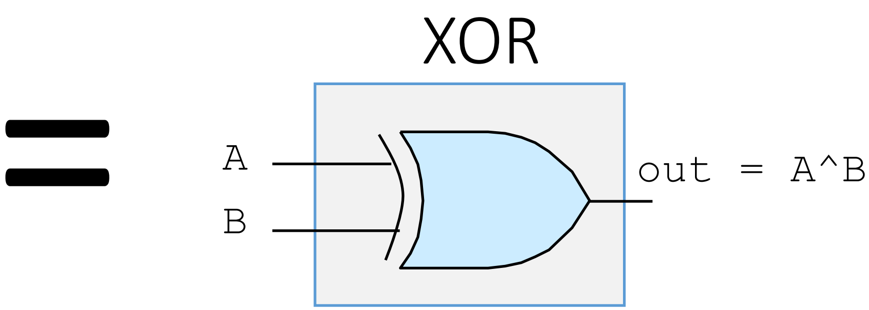
A:1 B:0 A^B: 1

A:1 B:1 A^B: 0

Abstracting the XOR circuit

```
A^B  ==  (~A & B) | (A & ~B)
```



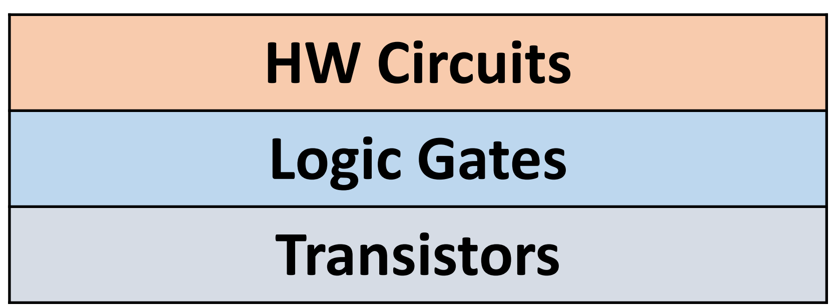


A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

First Goal: Build a model of the CPU

<u>Three main classifications of HW circuits:</u>

1. ALU: implement arithmetic & logic functionality
   (ex) adder to add two values together
2. Storage: to store binary values
   (ex) Register File: set of CPU registers
3. Control: support/coordinate instruction execution
   (ex) fetch the next instruction to execute



Building an ALU via abstraction

Step 1: zoom in

- Build circuits for each operation the ALU must perform
  - Arithmetic
    - Integer addition, subtraction, multiplication …
    - Floating point addition, subtraction, multiplication ...
  - Logic
    - Bitwise operations: AND, OR, …
    - Shifts: left, right, arithmetic

Step 2: zoom out

- Take each component circuit as given.
- Connect the components to memory and control circuits.

Addition Circuits via abstraction

- We want to build an N-bit (e.g. 32-bit) adder.
- Step 1: design a 1-bit adder.
- Step 2: string N 1-bit adders together.

1-bit adder

Inputs: A, B

Outputs: sum, cout

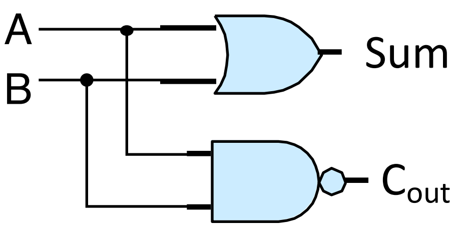
Let’s fill in the truth table.

A	B	Sum(A+B)	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

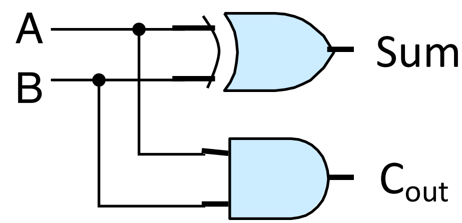
Which of these circuits is a one-bit adder?

A	B	Sum(A+B)	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

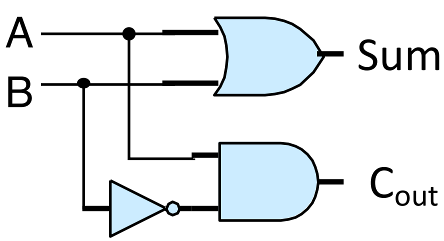
A:



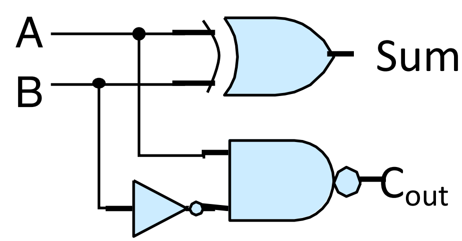
B:



C:



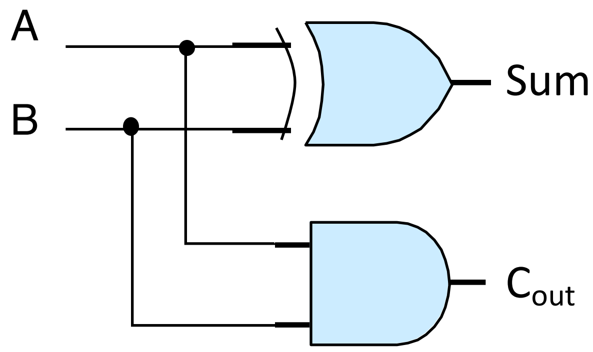
D:



E: None of these

What's missing?

- This circuit is called a half-adder.



A	B	Sum(A+B)	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- A one-bit full-adder takes a third input: cin.

```
    0011010
+   0001111
```

Which of these is a full-adder?

Hint: use abstraction. Start with two half-adders and connect them appropriately.

A	B	Sum	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

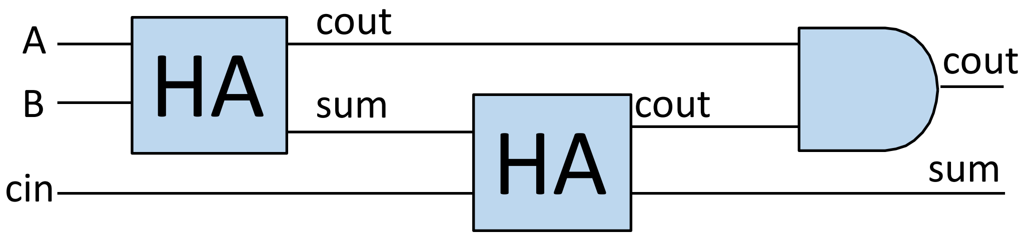
Half-Adder

Full-Adder

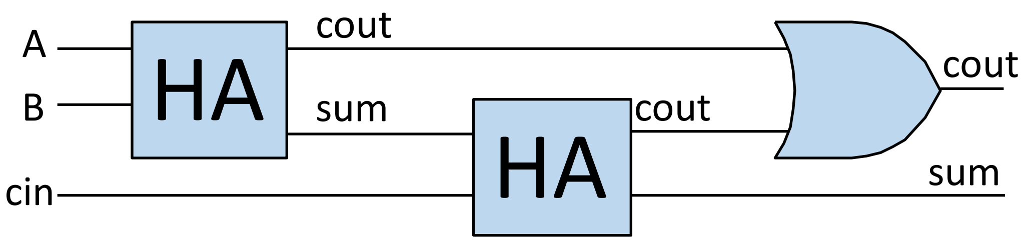
A	B	Cin	Sum	Cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Which of these is a full-adder?

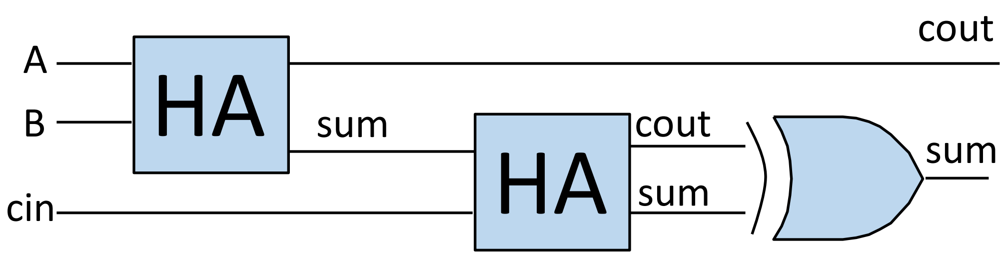
A:



B:

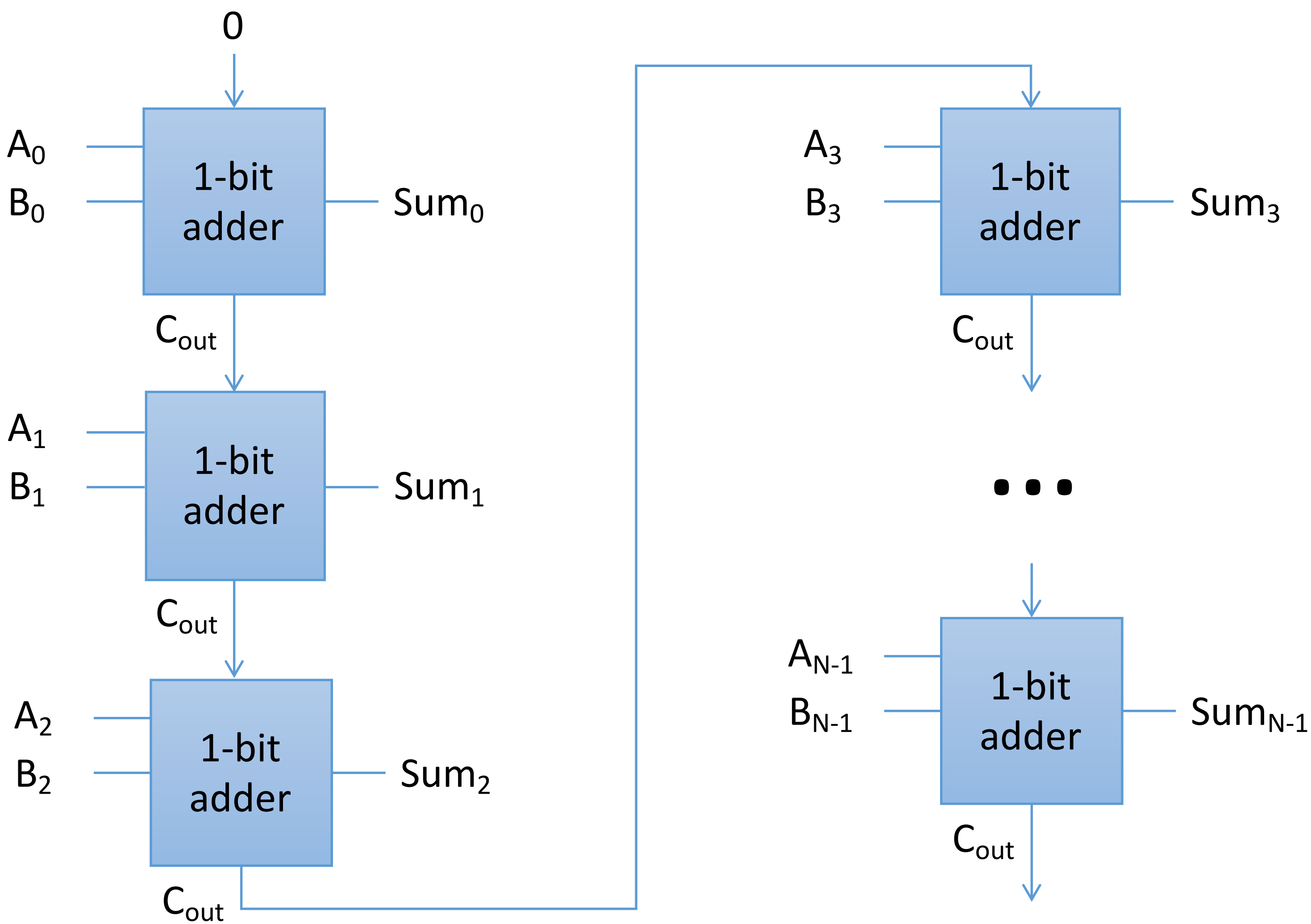


C:



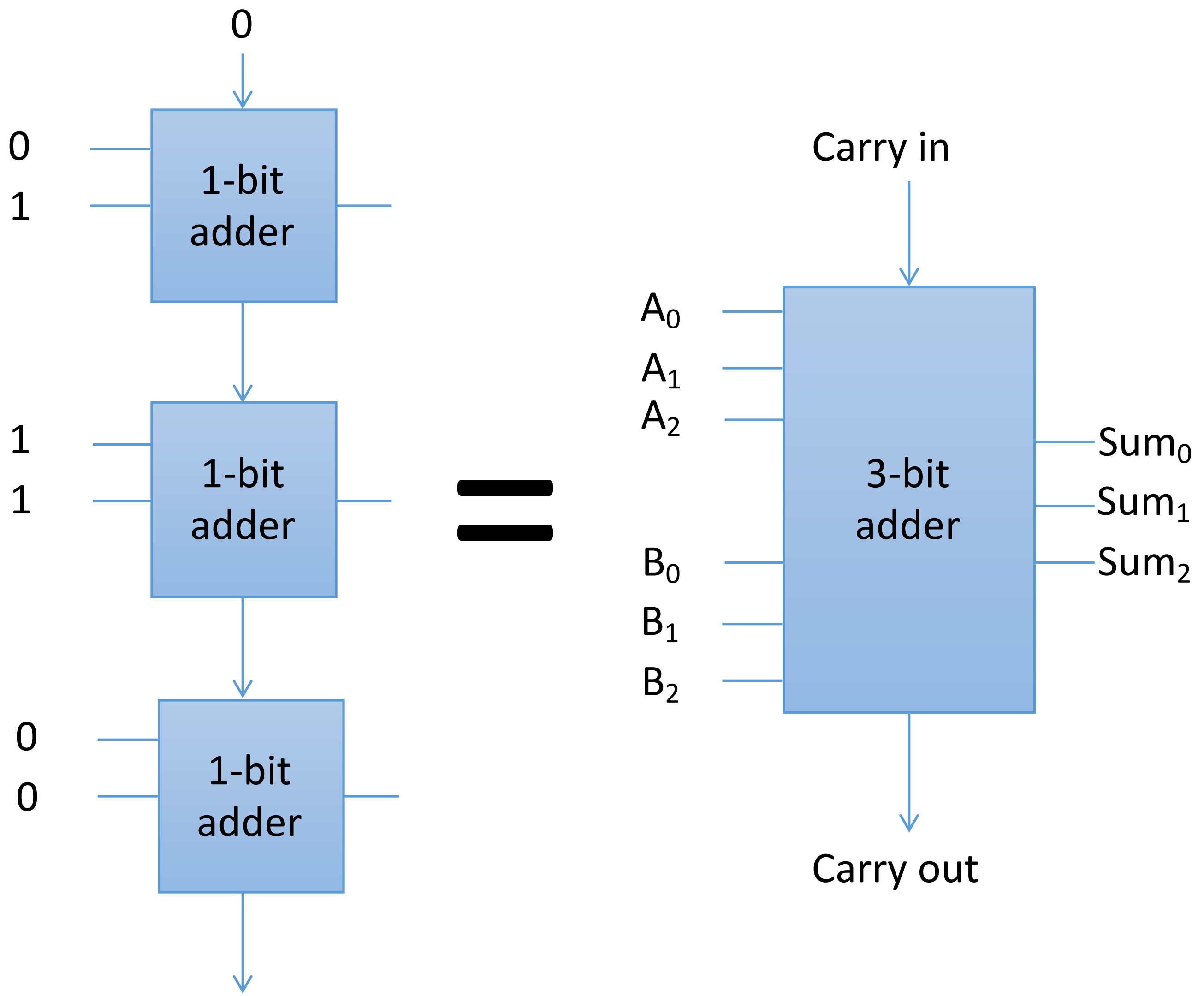
D: None of these.

N-bit adder (ripple-carry adder)



3-bit ripple-carry adder

$$\begin{array}{r} 010\ (2) \\ +\ \underline{011\ (3)} \end{array}$$



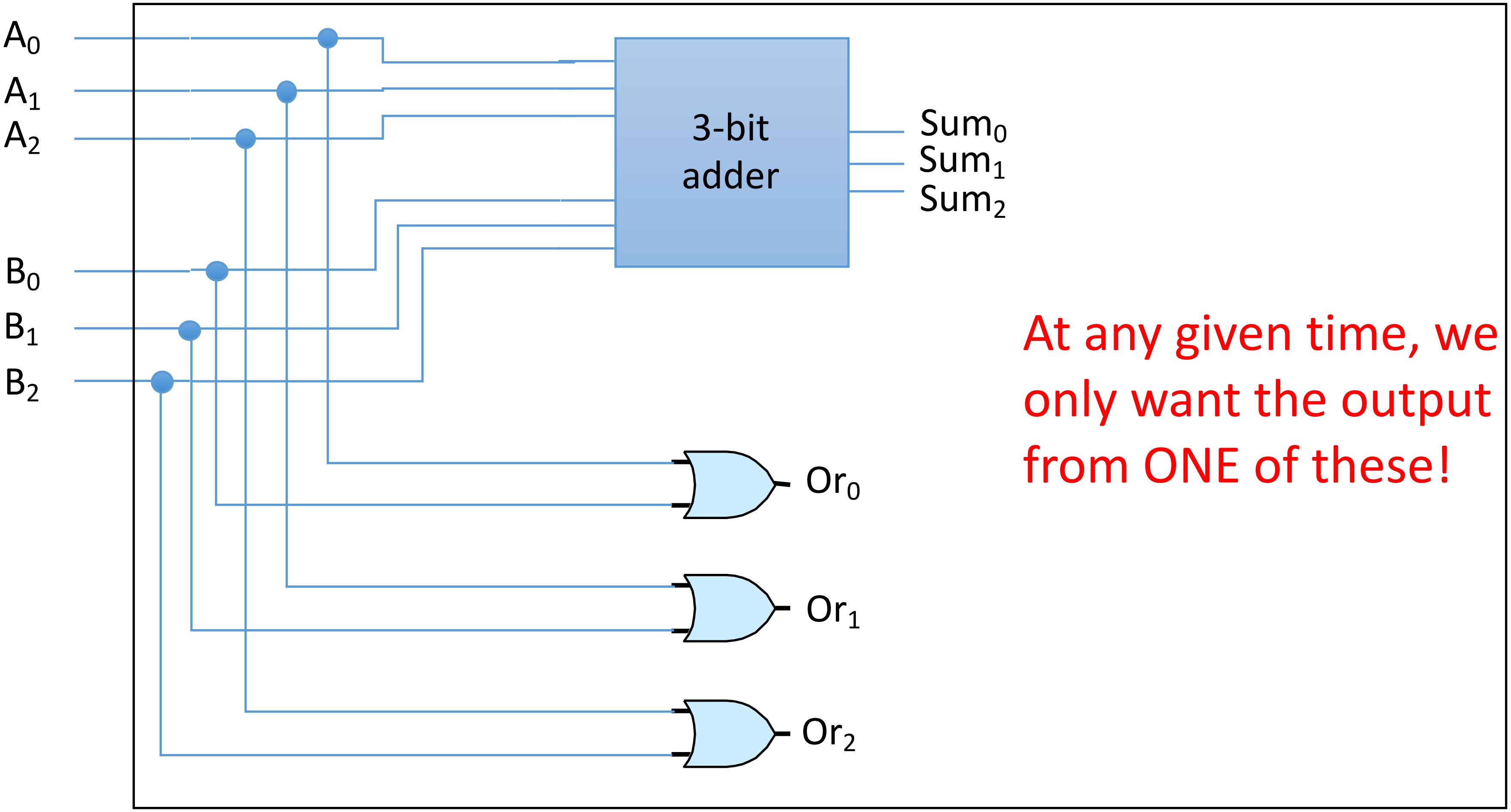
Arithmetic Logic Unit (ALU)

- One component that knows how to manipulate bits in multiple ways
  - Addition
  - Subtraction
  - Multiplication / Division
  - Bitwise AND, OR, NOT, etc.

- Built by combining components
  - Take advantage of sharing HW when possible (e.g., subtraction using adder)

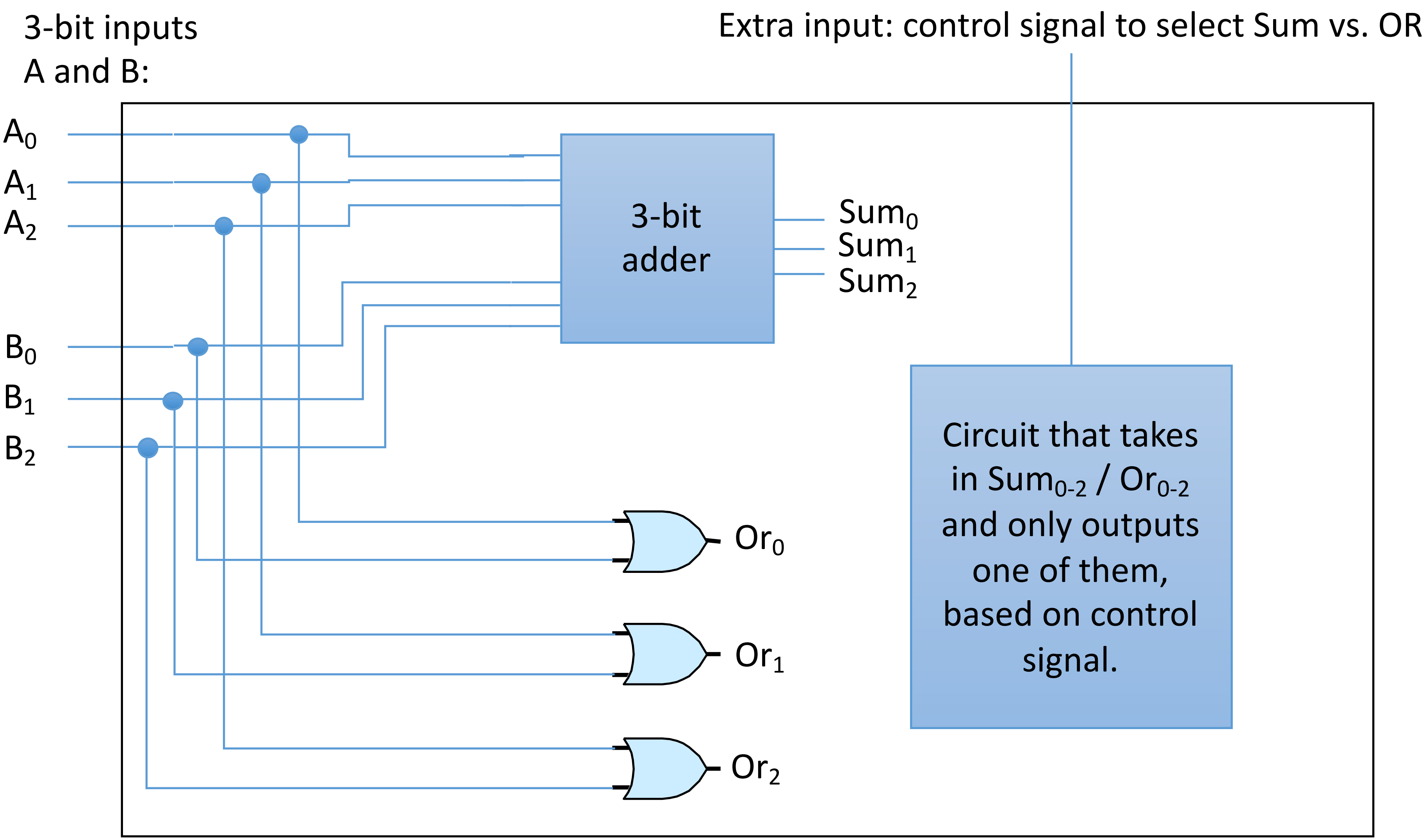
Simple 3-bit ALU: Add and bitwise OR

3-bit inputs
A and B:



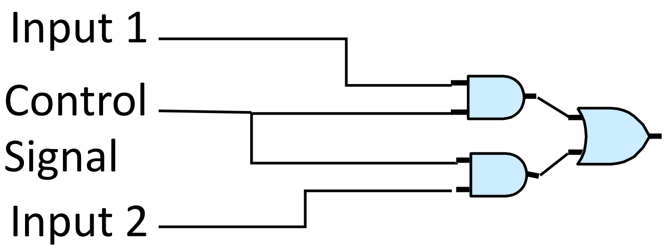
At any given time, we only want the output from ONE of these!

Simple 3-bit ALU: Add and bitwise OR

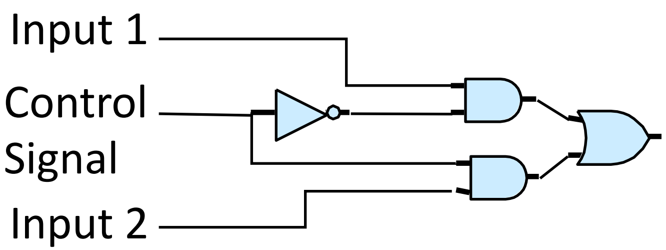


Which of these circuits lets us select between two inputs?

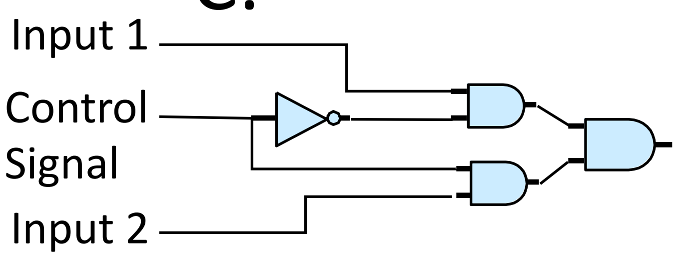
A:



B:



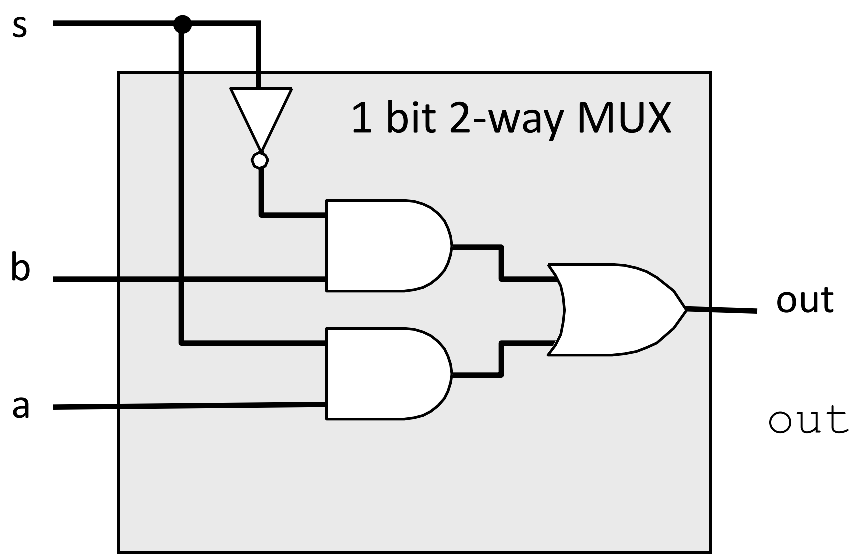
C:



Multiplexor: Chooses an input value

Inputs: 2^N data inputs, N signal bits

Output: is one of the 2^N input values

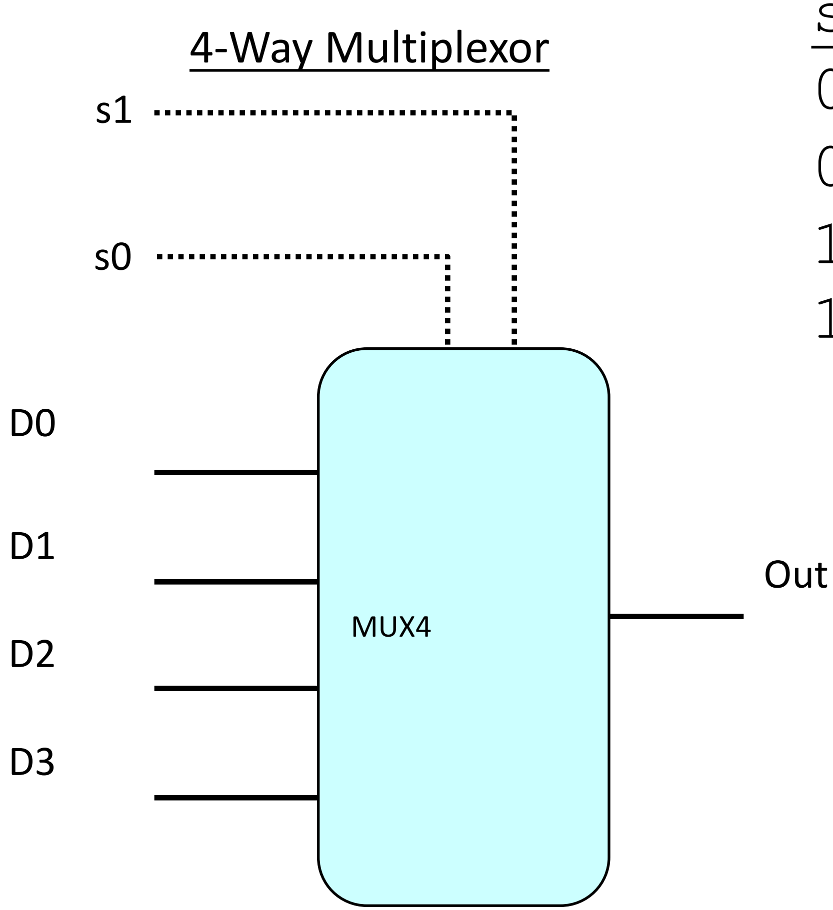


`out = (s & a)|(~s &b)`

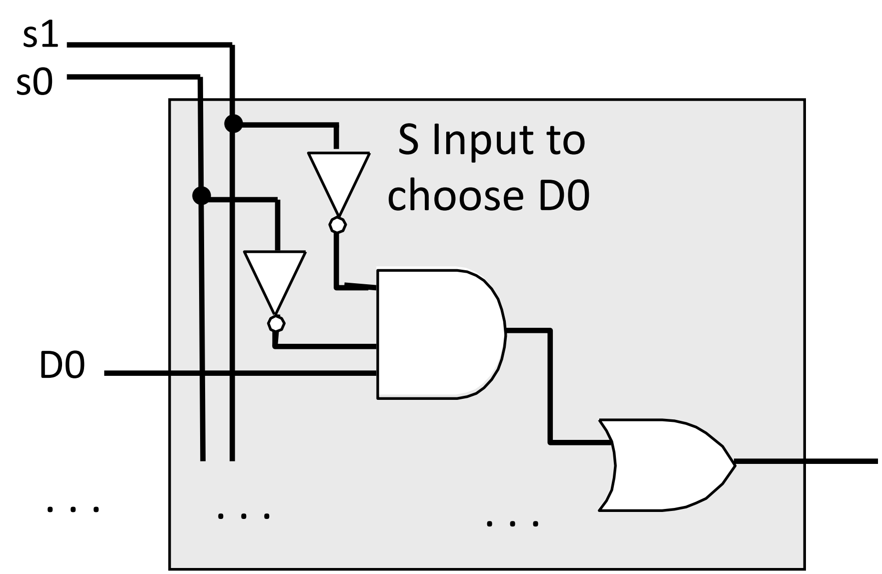
- Control signal s, chooses the input for output
  - When s is 1: choose a, when s is 0: choose b

N-Way Multiplexor

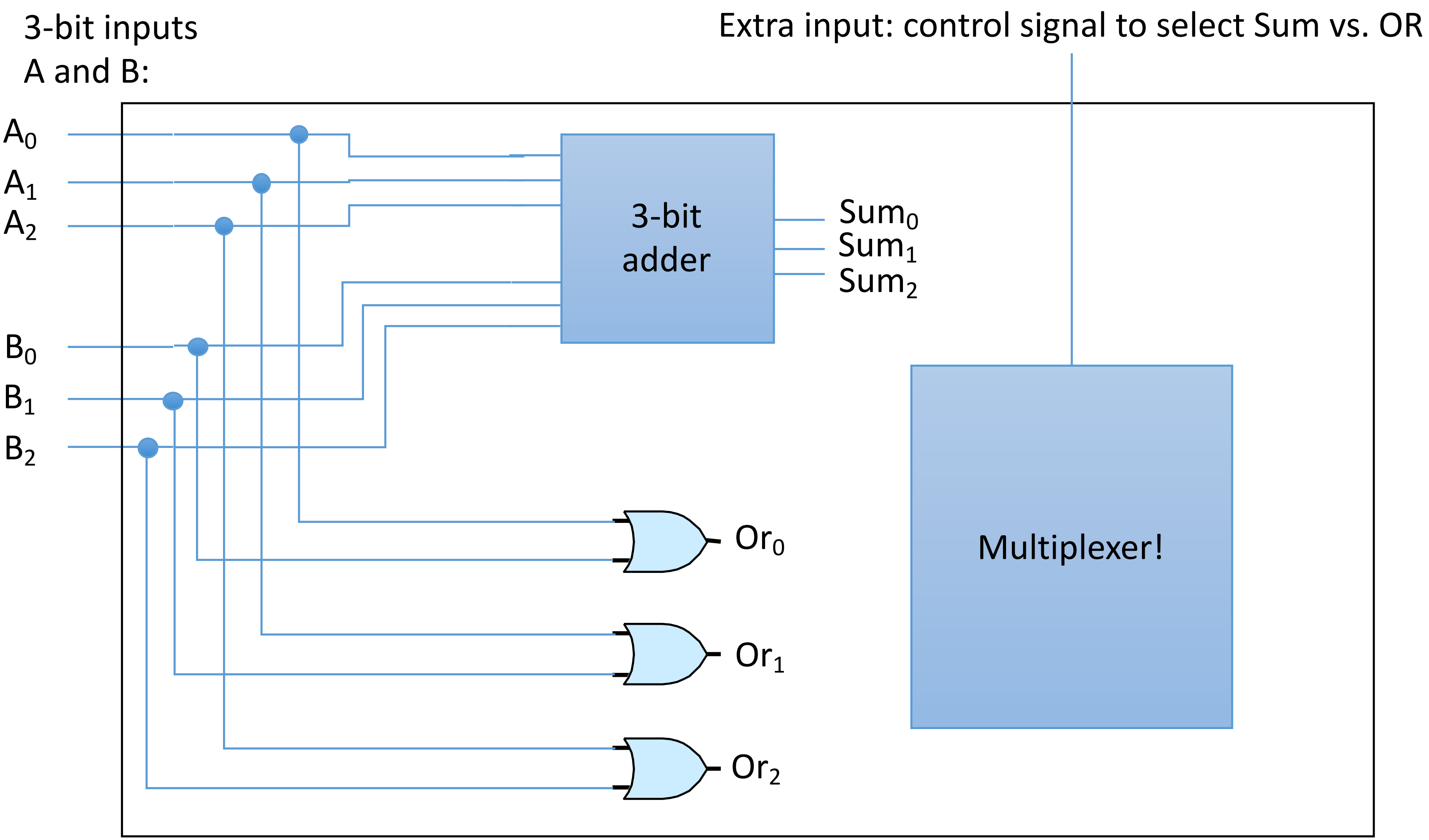
Choose one of N inputs, need $\log_2 N$ select bits



s1	s0	choose
0	0	D0
0	1	D1
1	0	D2
1	1	D3



Simple 3-bit ALU: Add and bitwise OR



1. Build a subtraction circuit

- Start with a 4-bit addition circuit.
- Create a 4-bit subtraction circuit.

2. Build an ALU that does + and -

- Use one 4-bit adder circuit.
- This adder should be used to perform addition and subtraction.
- Add control circuitry (a multiplexor) to determine which operation gets performed.