CS3102 Theory of Computation

www.cs.virginia.edu/~njb2b/cstheory/s2020

Warm up:

There are two “categories” of computing:

1) Hardware (e.g. CPU)
2) Software (e.g. Java)

Are they different? How are they the same

Differences

Hardware (CPU)

Software (Java)

Similarities

Logistics

- Exercise 0 was due last week
  - Didn't complete it? No problem (this time)! Just do it soon. Ask for an extension on the assignment page.
- First Quiz was due today
  - Didn't complete it? No problem (this time)! Ask for an extension on the assignment page.
- Exercise 1 is out.

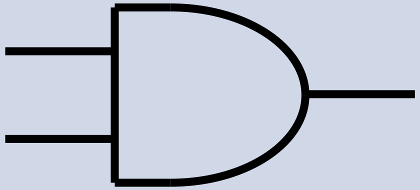
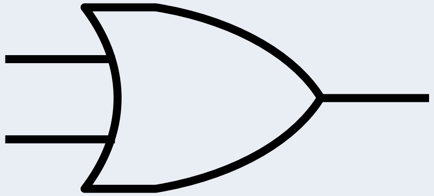
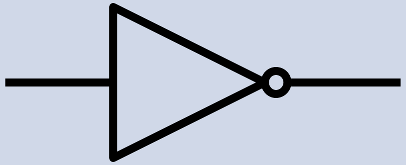
Today

- Finite computation
- A first model of computing!!
  - And a second!!

What do we need for a model?

- Define how to represent a computation
  - Programming languages: Syntax
- Define how to perform an execution
  - Programming languages: compiler

Boolean Logic

Operation	Symbol	Behavior	Gate
AND	$\wedge$ “\wedge”	$0 \wedge 0 = 0$ $0 \wedge 1 = 0$ $1 \wedge 0 = 0$ $1 \wedge 1 = 1$	
OR	$\vee$ “\vee”	$0 \vee 0 = 0$ $0 \vee 1 = 1$ $1 \vee 0 = 1$ $1 \vee 1 = 1$	
NOT	$\neg, -, \bar{b}$ “\neg”, “-”, “\overline{b}”	$\neg 0 = 1$ $\neg 1 = 0$	

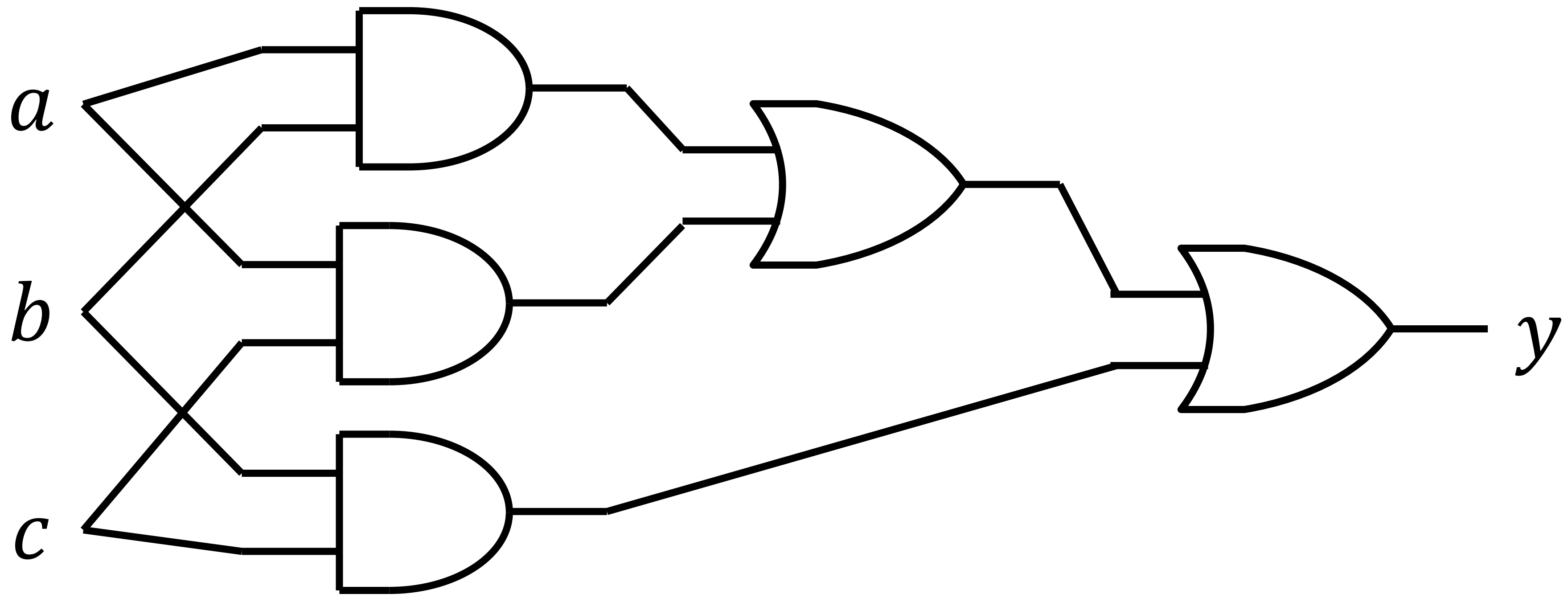
Example: Majority

- $MAJ: \{0,1\}^3 \rightarrow \{0,1\}$
- English:
  - The output is one if most of the inputs are one, and zero otherwise
- Math:
  - $MAJ(a, b, c) =$

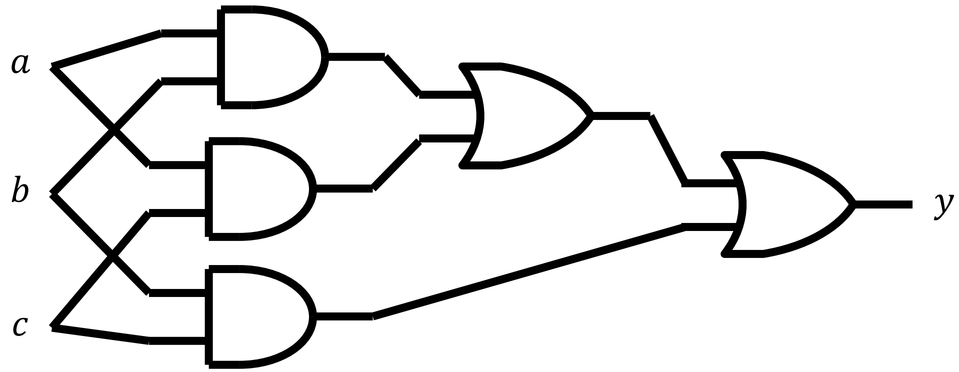
Input	Output
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

Majority as a circuit

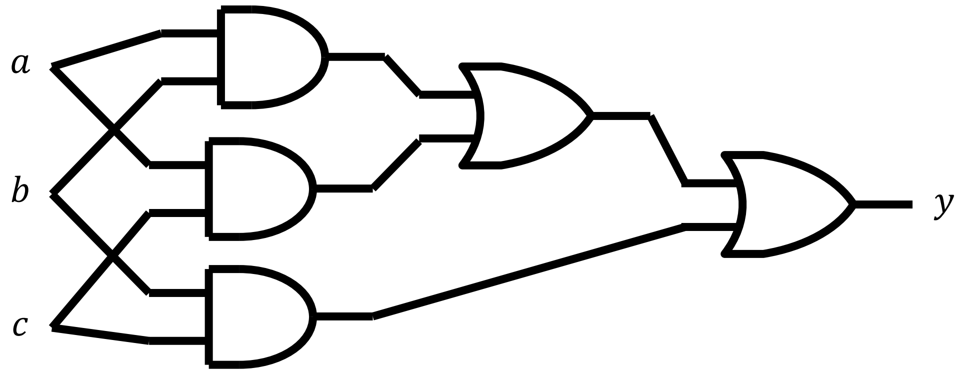
- Math:
  - $MAJ(a, b, c) = (a \land b) \lor (b \land c) \lor (a \land c)$



Components of a circuit



“Semiformal” Definition of a circuit



- $Circuit = (V, O, E)$
- $V = \{g_0, g_1, \dots, g_{n+s-1}\}$
  - Where each element of V has a “gate type” label
  - $label(g_i) \in \{INPUT, AND, OR, NOT, [others?]\}$
- $O = (y_0, \dots, y_{m-1})$ where $y_i \in V$ and $y_i = y_j \Rightarrow i = j$
- $E = \{(g_i, g_j) | g_i, g_j \in V \cup O, g_i \neq g_j\}$

What are we assuming?

- $Circuit = (V, O, E)$
- $V = \{g_0, g_1, \dots, g_{n+s-1}\}$
  - Where each element of V has a "gate type" label, $label(g_i) \in \{AND, OR, NOT, INPUT\}$
- $O = (y_0, \dots, y_{m-1})$ where $y_i \in V$ and $y_i = y_j \Rightarrow i = j$
- $E = \{(g_i, g_j) | g_i, g_j \in V \cup O, g_i \neq g_j\}$

1. All gates have 2 inputs
2. All gates have 1 output
3. All gates are total functions
4. All gates are commutative
5. Number of gates is finite

What are we assuming?

- $Circuit = (V, O, E)$
- $V = \{g_0, g_1, \dots, g_{n+s-1}\}$
  - Where each element of V has a "gate type" label, $label(g_i) \in \{AND, OR, NOT, INPUT\}$
- $O = (y_0, \dots, y_{m-1})$ where $y_i \in V$ and $y_i = y_j \Rightarrow i = j$
- $E = \{(g_i, g_j) | g_i, g_j \in V \cup O, g_i \neq g_j\}$

1. All gates have 2 inputs
2. All gates have 1 output
3. All gates are total functions
4. All gates are commutative
5. Number of gates is finite

What are we assuming?

- $Circuit = (V, O, E)$
- $V = \{g_0, g_1, \dots, g_{n+s-1}\}$
  - Where each element of V has a "gate type" label, $label(g_i) \in \{AND, OR, NOT, INPUT\}$
- $O = (y_0, \dots, y_{m-1})$ where $y_i \in V$ and $y_i = y_j \Rightarrow i = j$
- $E = \{(g_i, g_j) | g_i, g_j \in V \cup O, g_i \neq g_j\}$

1. All gates have 2 inputs
2. All gates have 1 output
3. All gates are total functions
4. All gates are commutative
5. Number of gates is finite

What are we assuming?

- $Circuit = (V, O, E)$
- $V = \{g_0, g_1, \dots, g_{n+s-1}\}$
  - Where each element of V has a "gate type" label, $label(g_i) \in \{AND, OR, NOT, INPUT\}$
- $O = (y_0, \dots, y_{m-1})$ where $y_i \in V$ and $y_i = y_j \Rightarrow i = j$
- $E = \{(g_i, g_j) | g_i, g_j \in V \cup O, g_i \neq g_j\}$

1. All gates have 2 inputs
2. All gates have 1 output
3. All gates are total functions
4. All gates are commutative
5. Number of gates is finite

What do we need for a model?

- Define how to represent a computation
  - Circuits: gates, edges, outputs
- Define how to perform an execution

Executing a circuit

- What we eventually want to know:
  - Values of the outputs
- What we start with:
  - Values of the inputs
- What do we do in between:
  - Find values of gates

We need to define "value of"

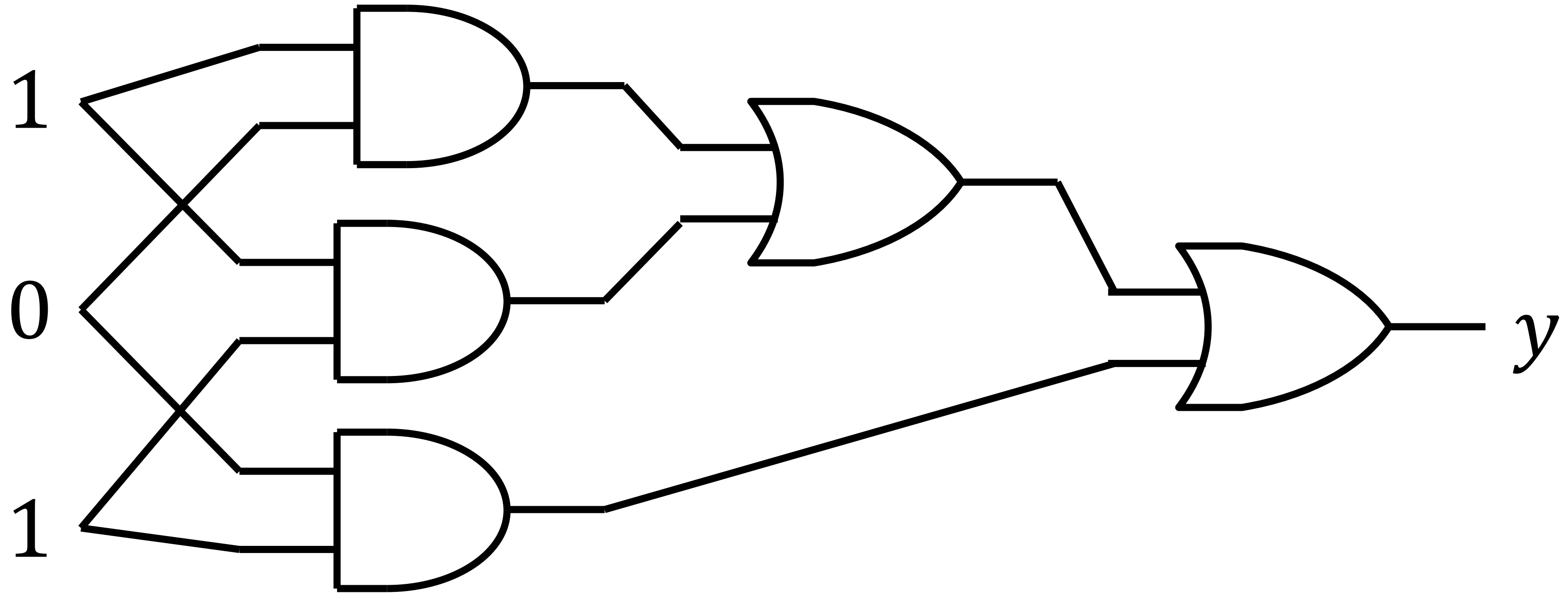
Value of

- Intuitively:
  - Wires carry a “signal”
  - The signal a wire carries comes from its gate
- $val: V \rightarrow \{0,1,\perp\}$
  - Gives the value of each gate/output
  - $\perp$ means “I don’t know”
- What should the starting values be?
  - Outputs:
  - Gates:
  - Inputs:

How to execute a Circuit

- As long as there's an output that's $\perp$:
  - Pick a gate/output whose value is $\perp$ and whose incoming edges all have a defined "source" (i.e. in $\{0,1\}$)
  - Change the value of that gate by executing the function labelled on its inputs

Example execution

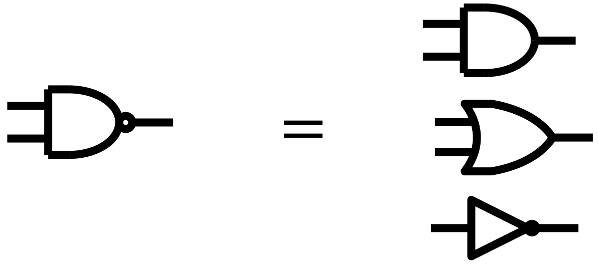


With your neighbor

- Build a circuit for $NAND$
  - $NAND(a, b) = \neg(a \land b)$

Input	Output
00	1
01	1
10	1
11	0

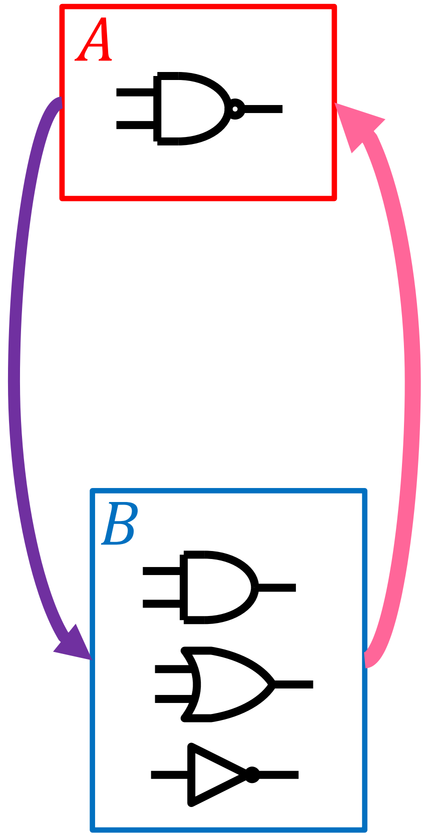
NAND Circuits



- The set of functions we can compute with $NAND$ gates only is the same as the set of functions we can compute with circuits AND, OR, NOT gates.
  - These computing models are “equivalent”
- How do we show this?

Equivalence of Computing Models

- Computing Model A and Computing Model B are "equivalent" if they compute the same set of functions
  - Any function that can be implemented with A can also be implemented with B, and vice-versa
- To show:
  - How to take an implementation of A and **convert it into** an implementation of B (which computes the same function)
  - How to take an implementation of B and **convert it into** an implementation of A (which computes the same function)



AND/OR/NOT using NAND

- AND

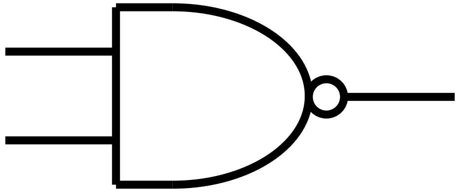
- OR

- NOT

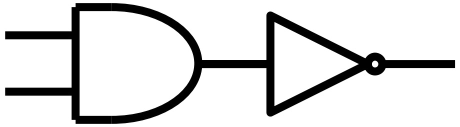
NAND = AON

NAND to AON

Everywhere you see:

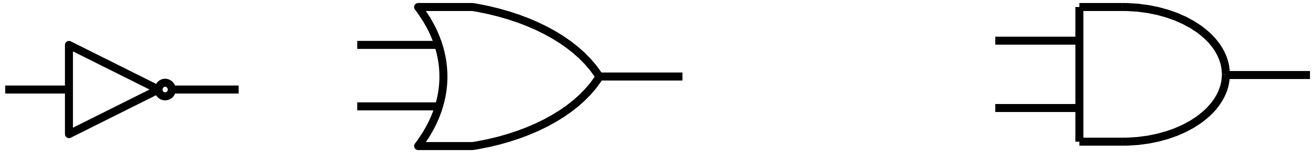


Instead put:

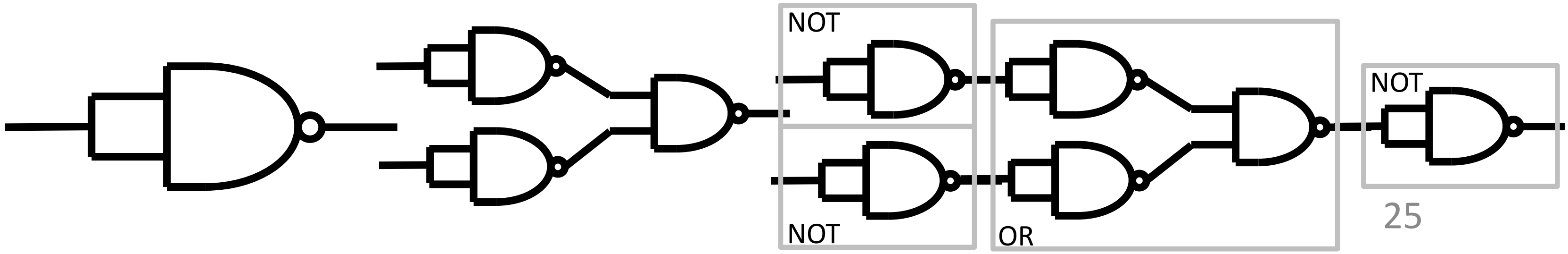


AON to NAND

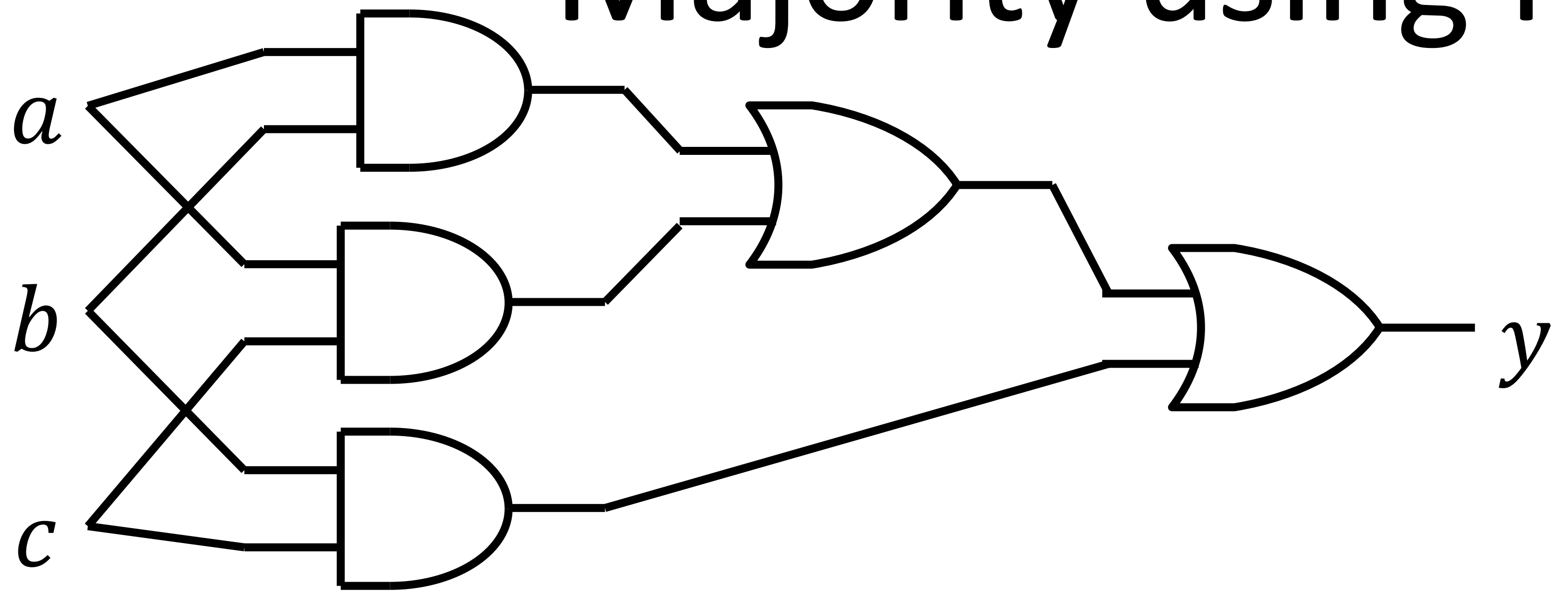
Everywhere you see:



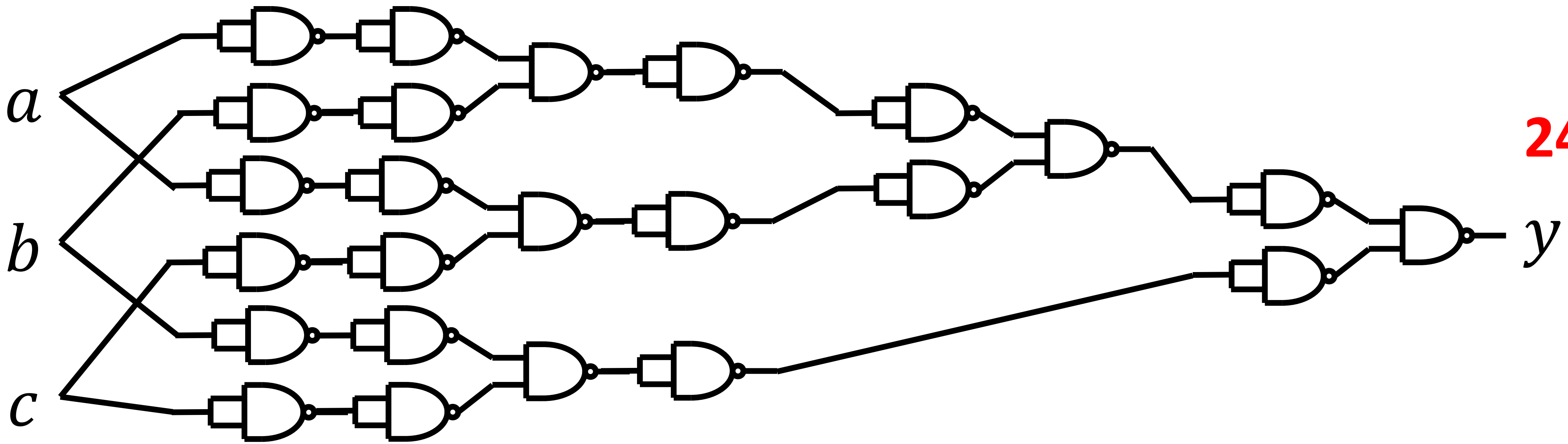
Instead put:



Majority using NAND



5 gates



24 gates

Takeaway

- We now have a hardware-based model of computing to work with
  - Actually two!
- Meant to be similar to how CPUs operate
- We’ve already made proofs about models of computation!
- While some models are equivalent in what they can compute, they may not be in how efficiently they can do it
- **Next time:** a software-like model of computing