

Logic (Gates) (Continued)

Reese Hatfield



0





Logic Gates

- When you design a system
 - The first thing:
 - Define the inputs and desired outputs
- Pretty universal concept
- Especially when designing circuits





Logic Gates

- Defining inputs and outputs
- Is just like defining a truth table
- We've gone from
 - Circuit to Truth Table
- Go backwards
 - More akin to real design





Logic Gates

- Let's start by doing
 - Truth Table → Boolean Expression
- Looking at an arbitrary table
 - Hard to come up with a nice and clean rule
 - Take a more algorithmic approach





Logic Gates

- “Sum of Products”
- OR a bunch of groups of AND
- Consider every value where our output is true (1)
- If any of those specific sets of variables are met
 - Output becomes true





A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Find any rows where the output is true



A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Find any rows where the output is true
- Look at the specific inputs where this is the case



A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Find any rows where the output is true
- Look at the specific inputs where this is the case
- For the first case
- Z is true if
 - \bar{A}, \bar{B}, C are all true
 - If they are 0, we negate, lets AND them



A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Find any rows where the output is true
- Look at the specific inputs where this is the case
- Let's OR that with the next

$$(\bar{A} \wedge \bar{B} \wedge C) \vee (\bar{A} \wedge B \wedge C)$$

- Hard-coded in these two cases



A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Find any rows where the output is true
- Look at the specific inputs where this is the case
- Do this for all the ones

$$\begin{aligned} & (\bar{A} \wedge \bar{B} \wedge C) \vee \\ & (\bar{A} \wedge B \wedge C) \vee \\ & (A \wedge B \wedge C) \end{aligned}$$



Logic Gates

- Now that you have a nice boolean expression
 - Easy to implement
 - Let's do that
-
- Note: wires can break off into multiple
 - They just keep the same value





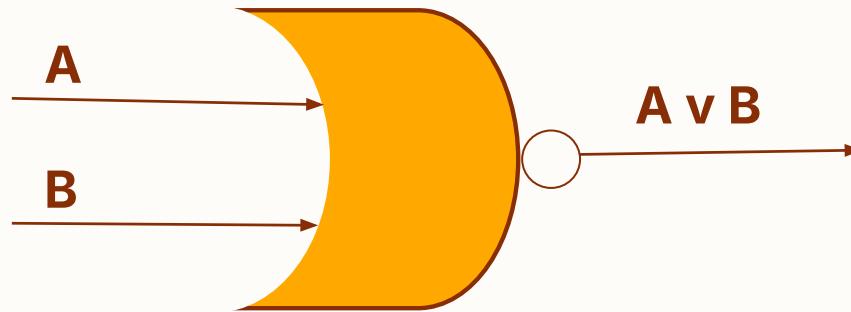
Logic Gates

- There are a few more gates to talk about
- You *could* accomplish everything with just AND, OR, NOT
- But there are some benefits to having a few others



NOR gates

- Negated OR
- Denote with this symbol
- $\nabla \Rightarrow$ Logical NOR
- Let's play around with this idea
- Logicly





NOR Gates

- Truth Table
- Denotes the output
- For all possible input

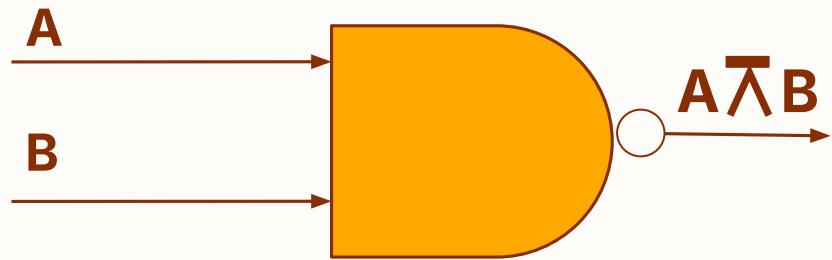
- A, B count up in binary

A	B	$A \nabla B$
0	0	1
0	1	0
1	0	0
1	1	0



NAND gates

- Negated AND
- Denote with this symbol
- $\overline{\wedge}$ \Rightarrow Logical NAND
- Let's play around with this idea
- Logicly



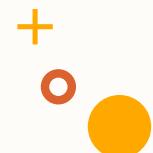


NAND Gates

- Truth Table
- Denotes the output
- For all possible input

- A, B count up in binary

A	B	$A \bar{A} B$
0	0	1
0	1	1
1	0	1
1	1	0

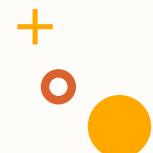




NAND Gates

- More efficient at a transistor level
- AND takes ~6 or more
- NAND takes ~4 transistors

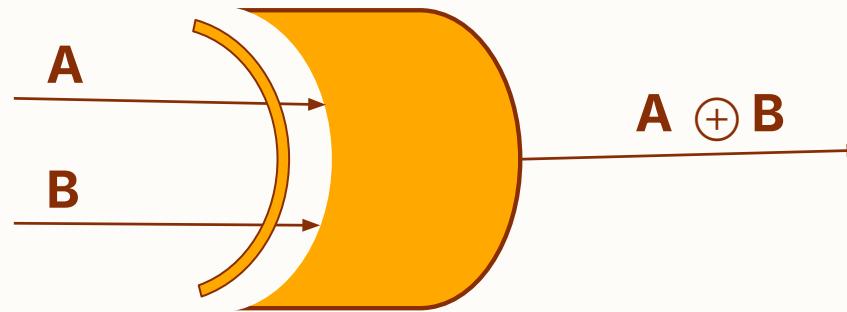
A	B	$A \bar{A} B$
0	0	1
0	1	1
1	0	1
1	1	0





XOR gates

- Exclusive OR
- Denote with this symbol
- $\oplus \Rightarrow$ Logical XOR
- Let's play around with this idea
- Logicly



A, or B is true, but NOT both





XOR Gates

- Truth Table
- Denotes the output
- For all possible input

- A, B count up in binary

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



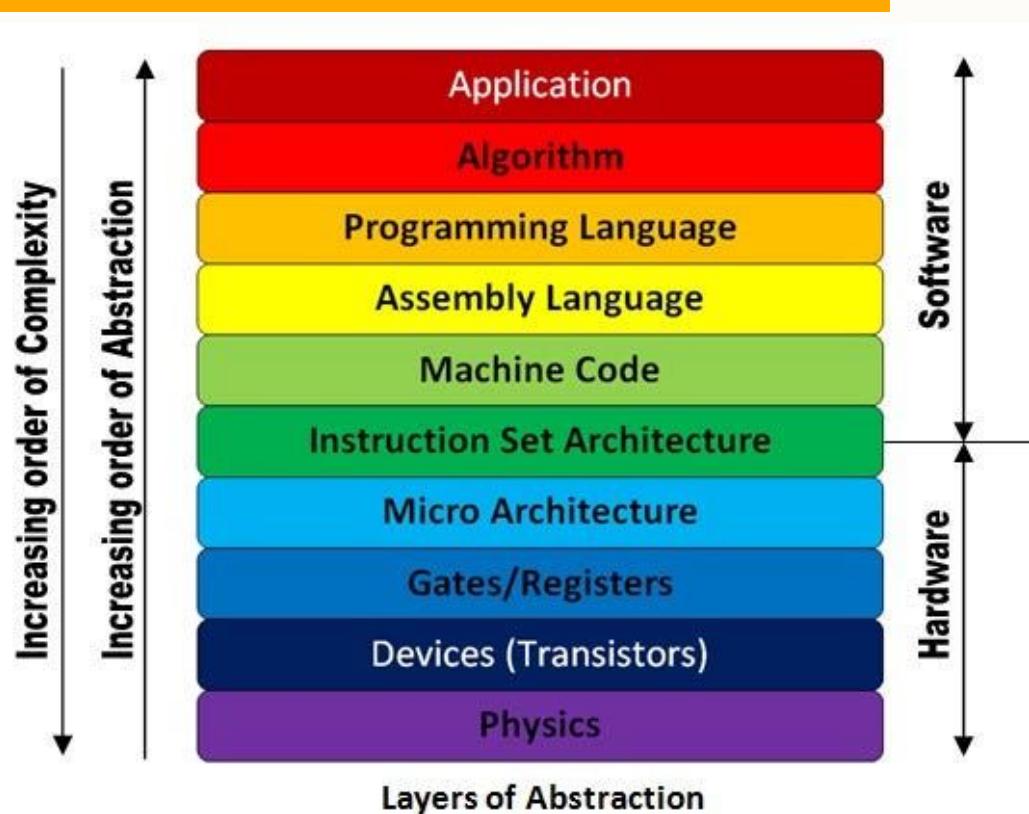


Abstraction

- Lots of gates
- Hide away the details
- As we scale our diagram
 - Systems get more complex
- Going to skip a lot
There are classes that cover this gap



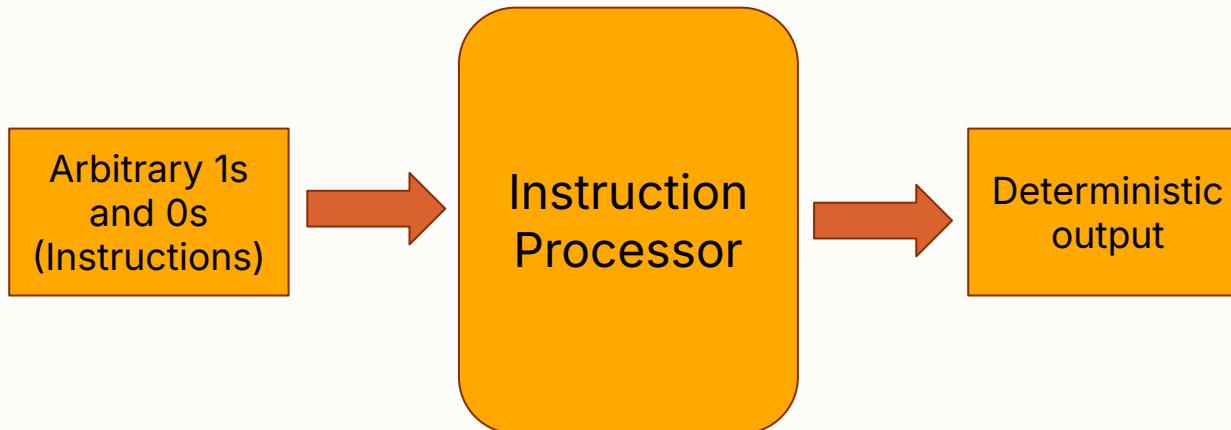
Abstraction

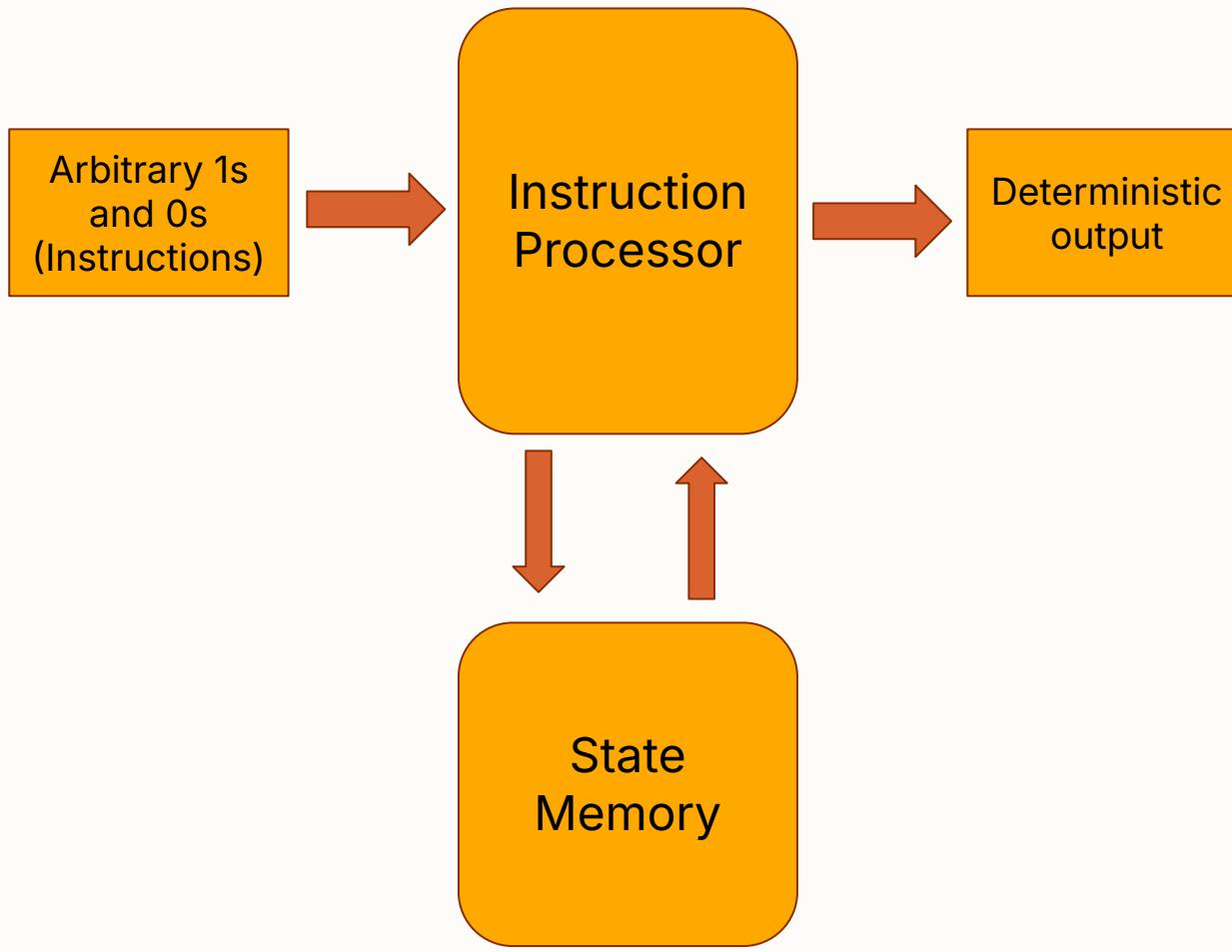




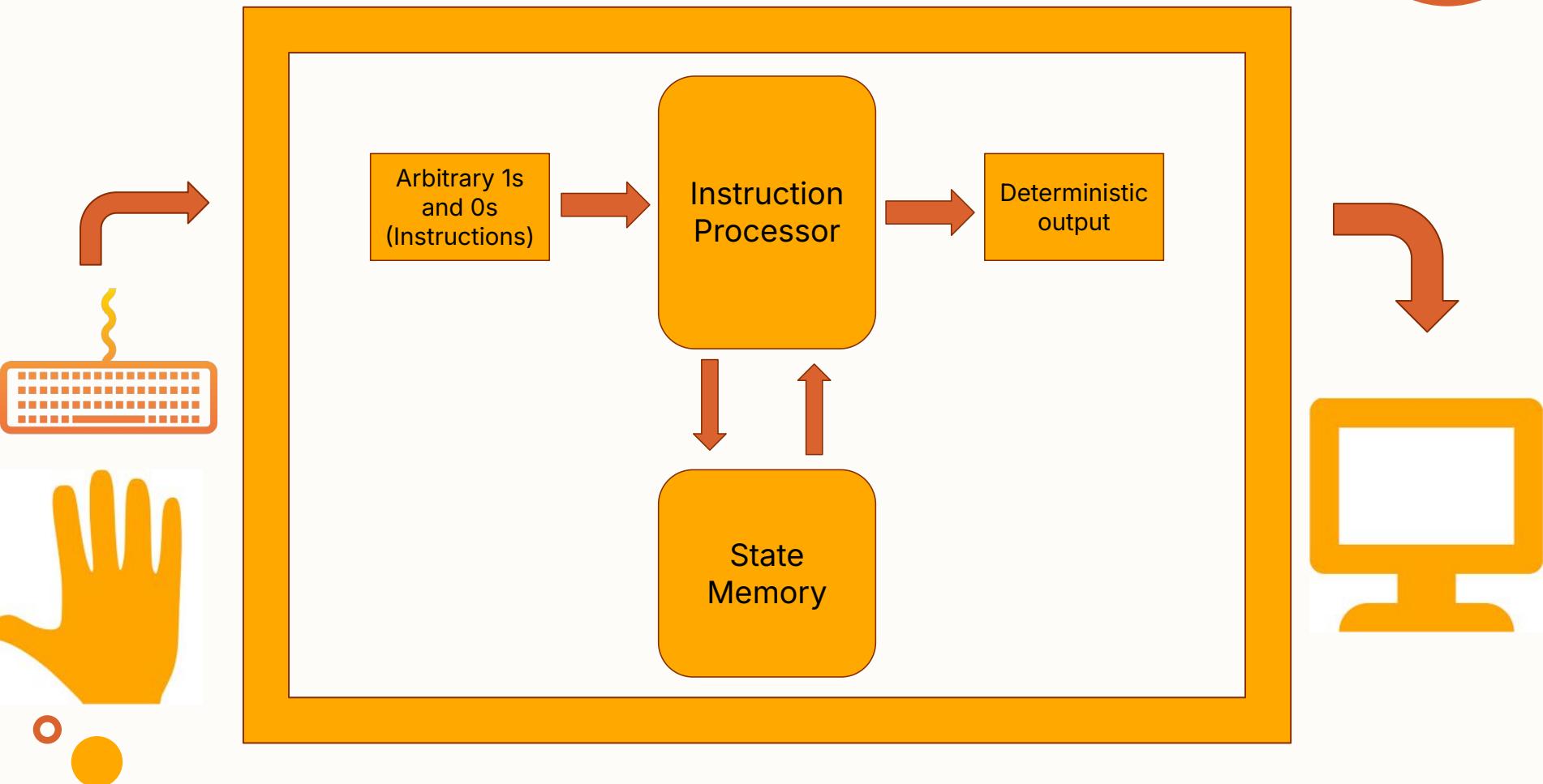
Abstraction

- Build hardware
- That take arbitrary instructions





Computer



data

- Since we can input arbitrary data
 - An important transition has happened
 - Transition from physical hardware
-
- Into purely digital software
 - “Digital Revolution”
 - Microprocessors

