

CS 211: Intro to Computer Architecture

11.1: Digital Logic and Boolean Algebra

Minesh Patel

Spring 2025 – Tuesday 8 April

Announcements

- Ongoing
 - **Extra Credit:** replaces WA6, due **Friday (April 11) @ 11:59 pm**
 - **WA7:** due **Wednesday (April 9) @ 11:59 pm**
 - **PA4:** due **Next Friday (April 18) @ 11:59 pm**
- Upcoming
 - **WA8:** TBA after WA7
 - **PA5:** TBA after PA4

Note on PA6

- PA6 is too ambitious
- **Tentative plan:** reweight PA1-PA5 to cover a missing PA6

- **Programming Assignments (53%):** We will have 6 programming assignments weighted as follows.

5% ~~• [3%] PA1: Intro to Linux~~

12% ~~• [10%] PA2: C Development + Integers~~

12% ~~• [10%] PA3: Pointers, Arrays, Strings, Structs and Explicit Memory Management~~

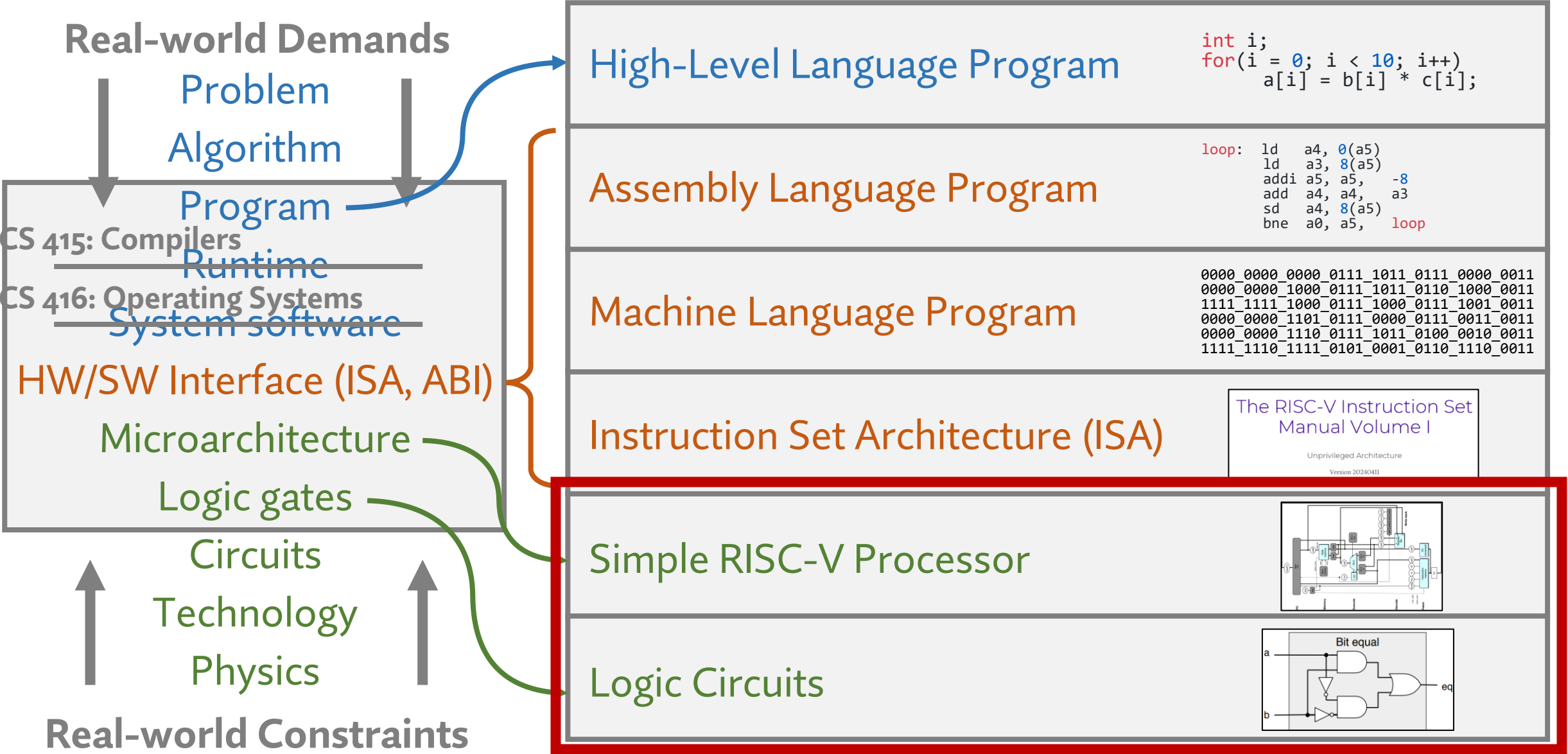
12% ~~• [10%] PA4: Assembly Programming~~

12% ~~• [10%] PA5: Single-cycle CPU Emulation~~

~~• [10%] PA6: Extension with Memory + Caches~~

- We will announce our final decision when we make it

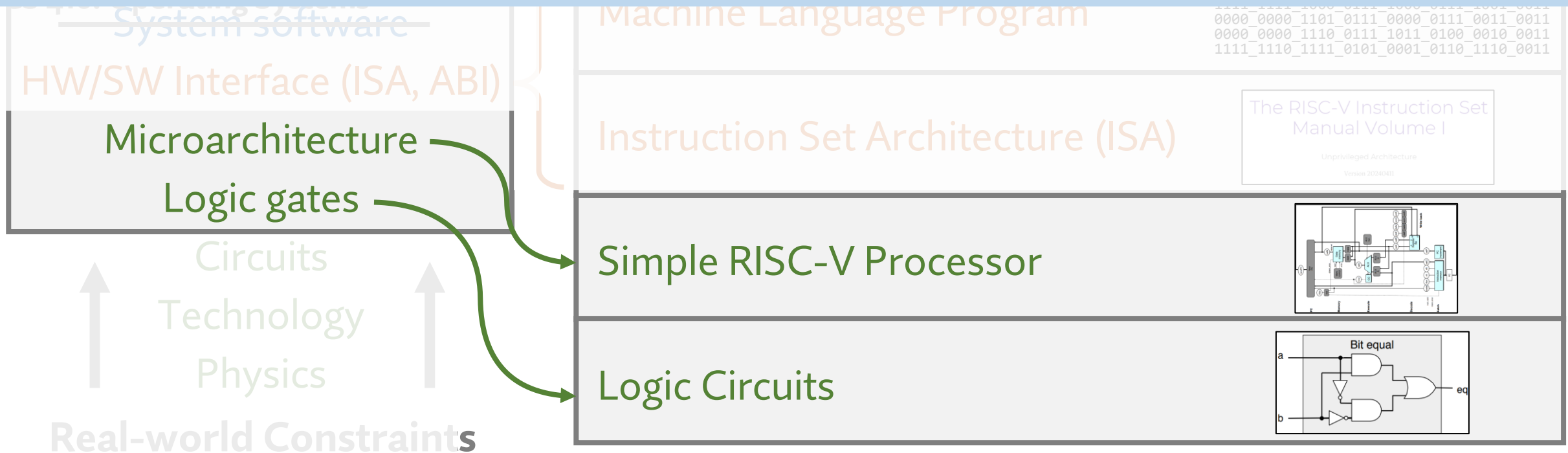
Layers of Abstraction for CS 211



Layers of Abstraction for CS 211

Next ~2-3 weeks:

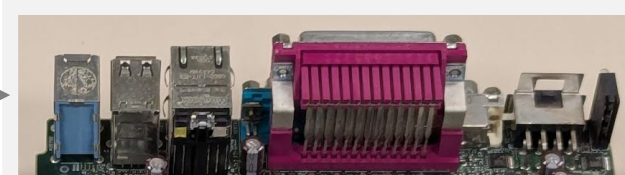
How a **CPU** executes **machine language code**



Recap: Computer Organization

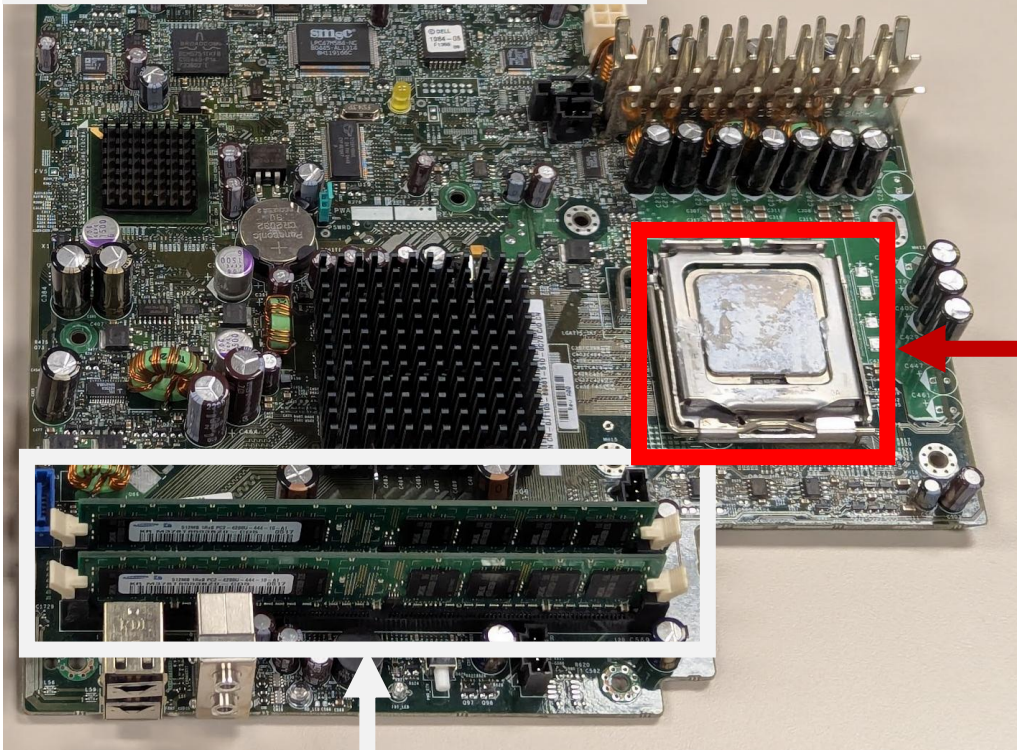
Input/Output

(network, USB, etc.)



Storage

(big, slow, cheap memory)



Processor
(where code runs)

Main Memory

(small, fast, expensive)

Nandgame

Solve Level

Levels

⚙️

Donate

About

Level Help

Nand

Your task is to connect inputs to output through wires and relays such that when both **a** and **b** inputs are 1, the output is 0.

1 represents electrical current, 0 represents no current.

The **V** input carries constant current, i.e. always 1.

The exact specification:

Input		Output
a	b	
0	0	1
0	1	1
1	0	1
1	1	0

Check solution

Clear canvas

Clear all levels

Skip level

Toolbox

relay (default on)

relay (default off)

Output:

Step 1: Drag components from the toolbox to the blue area.

Input:

a

b

V

Power + (always 1)

Nand

Welcome to **The Nand Game!**

You are going to build a computer starting from basic components.

The game consists of a series of levels. In each level, you are tasked with building a component that behaves according to a specification. This component can then be used as a building block in the next level.

The game does not require any previous knowledge about computer architecture or software, and does not require math skills beyond addition and subtraction. (It does require some patience—some of the tasks might take a while to solve!)

Your first task is to build a **nand** component.

On the left of the diagram is the exact specification of the task. Click "Level Help" for further information which might be helpful.

OK

Minecraft Redstone

Explore

Fan Central

CURRENT

MINECRAFT WIKI

Minecraft Wiki

EXPLORE GAMES MINECRAFT MINECRAFT DUNGEONS WIKI COMMUNITY

7,493 PAGES

in: Redstone, Tutorials, Redstone circuits

English

Redstone circuits/Logic

TALK VIEW SOURCE

< Redstone circuits

This article is about a specific category of redstone circuits. For other circuits, see [redstone circuit](#).

A logic gate can be thought of as a simple device that will return a number of *outputs*, determined by the pattern of *inputs and rules* that the logic gate follows. For example, if both inputs in an **AND** gate are in the 'true'/'on'/'powered'/'1' state, then the gate will return 'true'/'on'/'powered'/'1'.

There are many different kinds of logic gates, each of which can be implemented with many different designs. Each design has various advantages and disadvantages, such as size, complexity, speed, maintenance overhead, or cost. The various sections will give many different designs for each gate type.

Contents

[hide]

1. Concepts

2. Logic gate

2.1. NOT gate

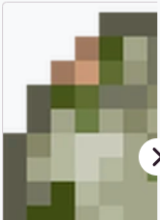
2.2. OR gate

Recent Images



Spawn Egg

15 hours ago



Spawn Egg

15 hours ago

Check out Fandom Quizzes and challenge your friends!

[Redstone circuits/Logic – Minecraft Wiki](#)

8

Nand to Tetris

DOI:10.1145/3626513

CS course walks students through a step-by-step construction of a complete, general-purpose computer system—hardware and software—in one semester.

BY SHIMON SCHOCKEN

Nand to Tetris: Building a Modern Computer System from First Principles

SUPPOSE YOU WERE asked to design an abridged computer science (CS) program consisting of just three courses. How would you go about it? The

Several reasonable options come to mind. One is a hands-on overview of applied CS, building on the programming skills and theoretical knowledge acquired in the first two courses. Such a course could survey key topics in computer architecture, compilation, operating systems, and software engineering, presented in one cohesive framework. Ideally, the course would engage students in significant programming assignments, have them implement classical algorithms and widely used data structures, and expose them to a range of optimization and complexity issues. This hands-on synthesis could benefit students who seek an overarching understanding of computing systems, as well as self-learners and non-majors who cannot commit to more than a few core CS courses.

This article describes such a course, called *Nand to Tetris*, which walks students through a step-by-step construction of a complete, general-purpose computer system—hardware and software—in one semester. As it turns out, construction of the computer system built during the course requires exposure to, and application of, some of the most per-

>> key insights

- In the early days of computers, any curious person could gain a gestalt understanding of how the machine works. As digital technologies became increasingly more complex, this clarity is all but lost: The most fundamental ideas and techniques in applied computer science are now hidden under many layers of obscure interfaces and proprietary implementations.
- Starting from NAND gates only, students build a hardware platform comprising a CPU, RAM, datapath, and a software hierarchy consisting of an assembler.

Downloaded from the ACM Digital Library by Rutgers University on April 8, 2025.

From Nand to Tetris

Building a Modern Computer From First Principles



Home

Projects

Book

Software

Demos

License

Cool Stuff

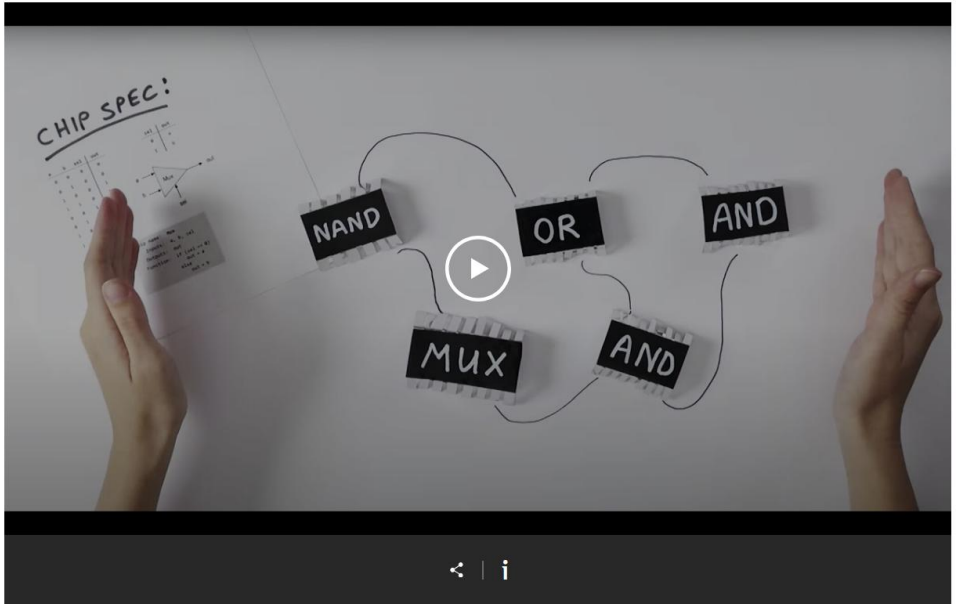
Team

Stay in Touch

Q&A

The official website of Nand to Tetris courses

And of the book [The Elements of Computing Systems](#), By [Noam Nisan](#) and [Shimon Schocken](#) (MIT Press)



This website contains all the lectures, project materials and tools necessary for building a general-purpose computer system and a modern software hierarchy from the ground up.

The materials are aimed at students, instructors, and self-learners. Everything is free and open-source, as long as you operate in a non-profit setting. Here is a recent CACM article about the course: [text](#) / [video](#).

<https://dl.acm.org/doi/pdf/10.1145/3626513>

<https://www.nand2tetris.org/>

Agenda

- **The Digital Logic Abstraction**
- Boolean Algebra
 - Truth Tables
 - Equations and Identities
- Digital Logic Gates

Digital Logic

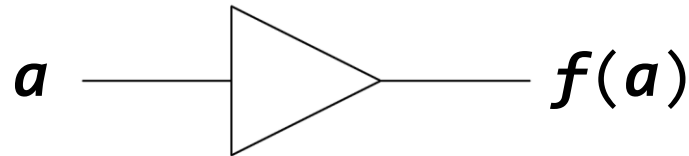
- Digital logic is a **convenient abstraction** for how circuits work
- **1-bit** input/output signals (high/low = 1/0)

Example Logic Circuit: “Buffer”

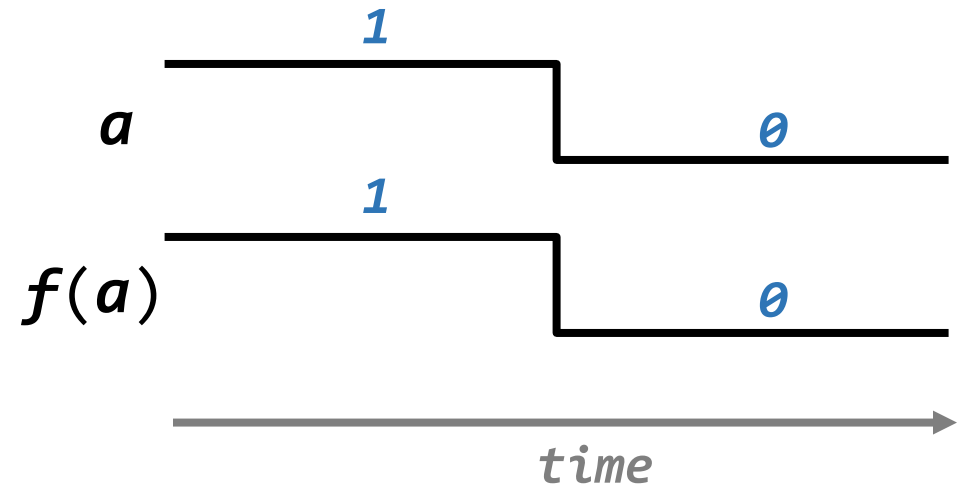
Algebraic Representation

$$f(a) = a$$

Graphical Representation



Input/Output Signal Representation



Digital Logic Abstraction

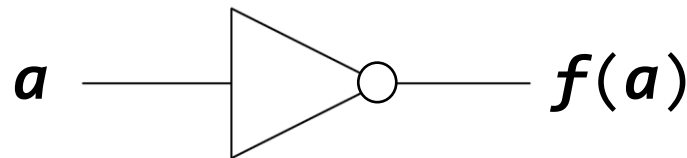
- Digital logic **abstracts away** the hardware
 - Can work with **mathematical equations** (instead of resistors, capacitors, etc.)
 - Unfortunately, **hard to say** whether you can actually build a given circuit

Example Logic Circuit: “Inverter”

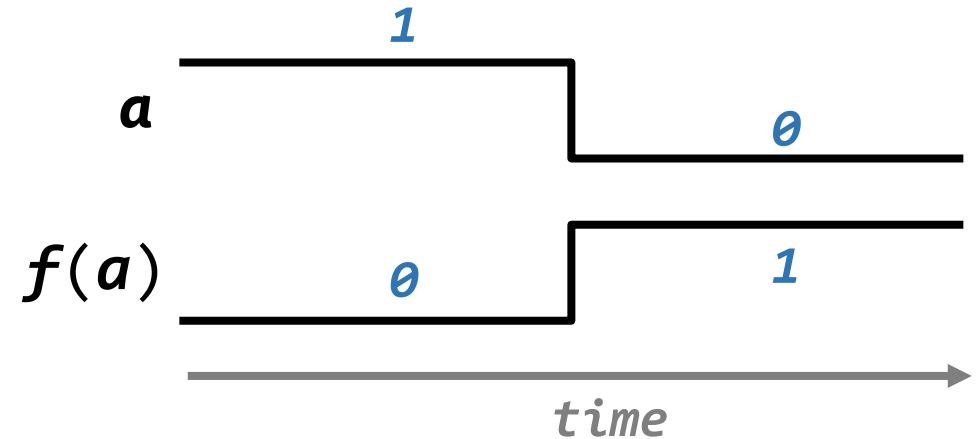
Algebraic Representations

$$\begin{aligned}f(a) &= \sim a \\ &= !a \\ &= \bar{a} \\ &= a'\end{aligned}$$

Graphical Representation



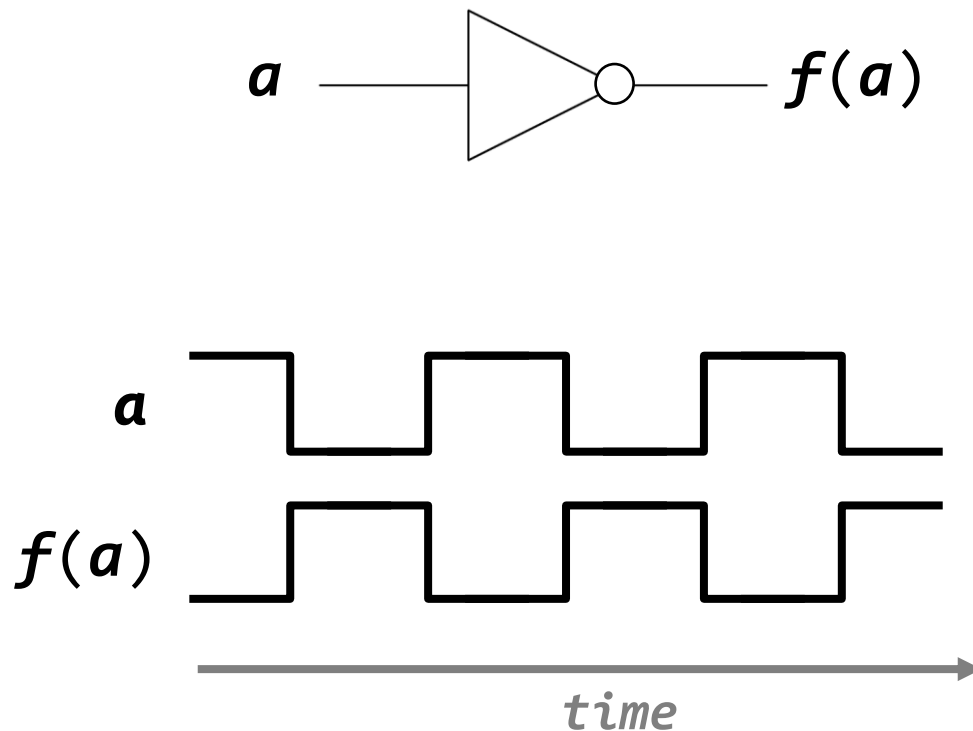
Input/Output Signal Representation



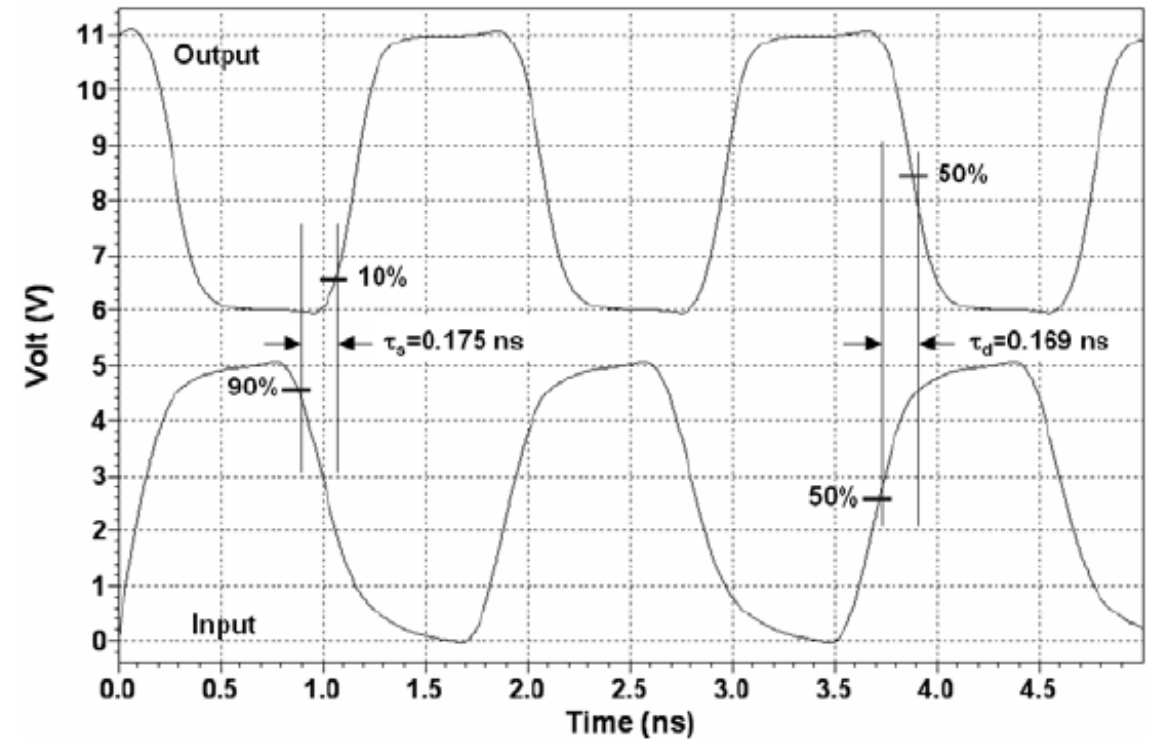
Abstraction vs. Reality

- Voltage **cannot** change instantaneously

Abstraction



Reality



Sandoval-Ibarra, F., and E. S. Hernández-Bernal. "Ring CMOS NOT-based oscillators: Analysis and design." *Journal of applied research and technology*, 2008.

Example: Electrical Wires

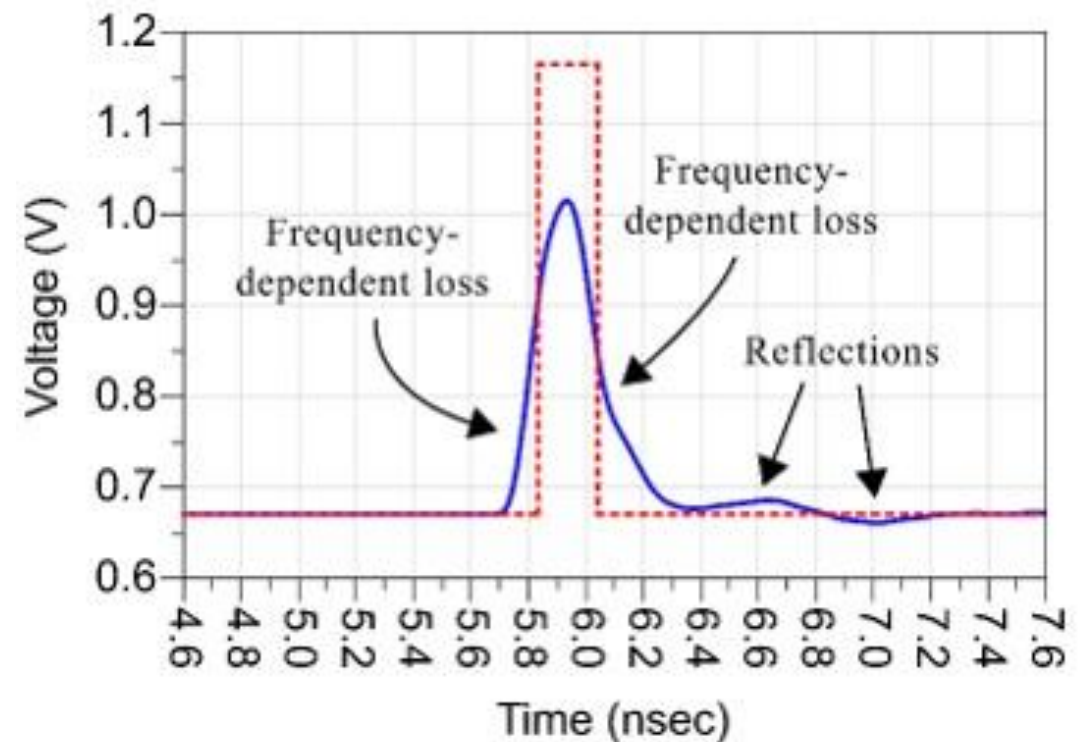
- Real circuit behavior **is highly non-ideal**
 - **Circuit design** (e.g., density, error-mitigation mechanism)
 - **Manufacturing defects** (e.g., thinner/thicker wires)
 - **Environment** (e.g., temperature, altitude)
 - **Operating conditions** (e.g., voltage, frequency)
 - ...

Two Adjacent Wires



Eric Bogatin, "Signal and Power Integrity 2/E," 2010.

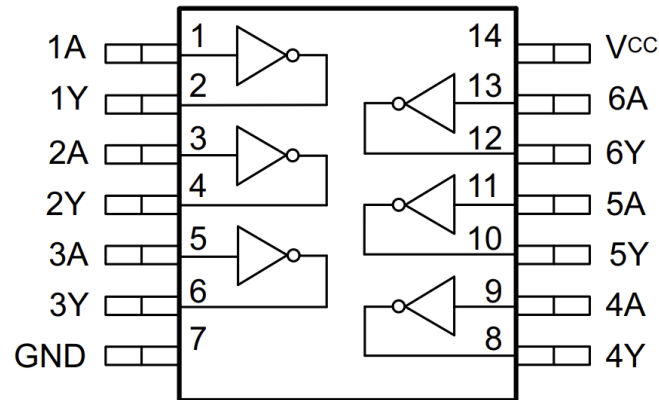
DDR5 Read/Write Interface



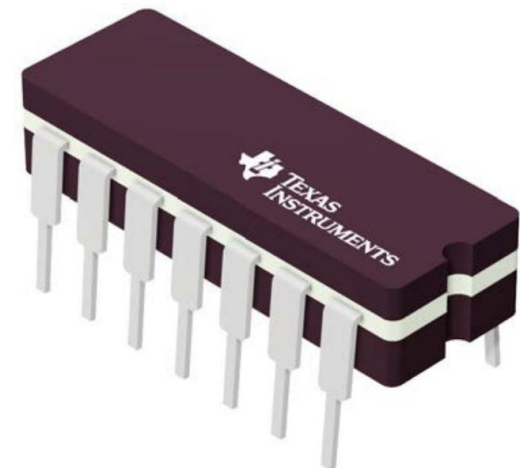
Tim Wang Len, "DDR5 Signal Integrity Fundamentals," 2021.

Example: Real Inverter Circuits

- Performance heavily depends on **voltage** and **temperature**



Functional pinout

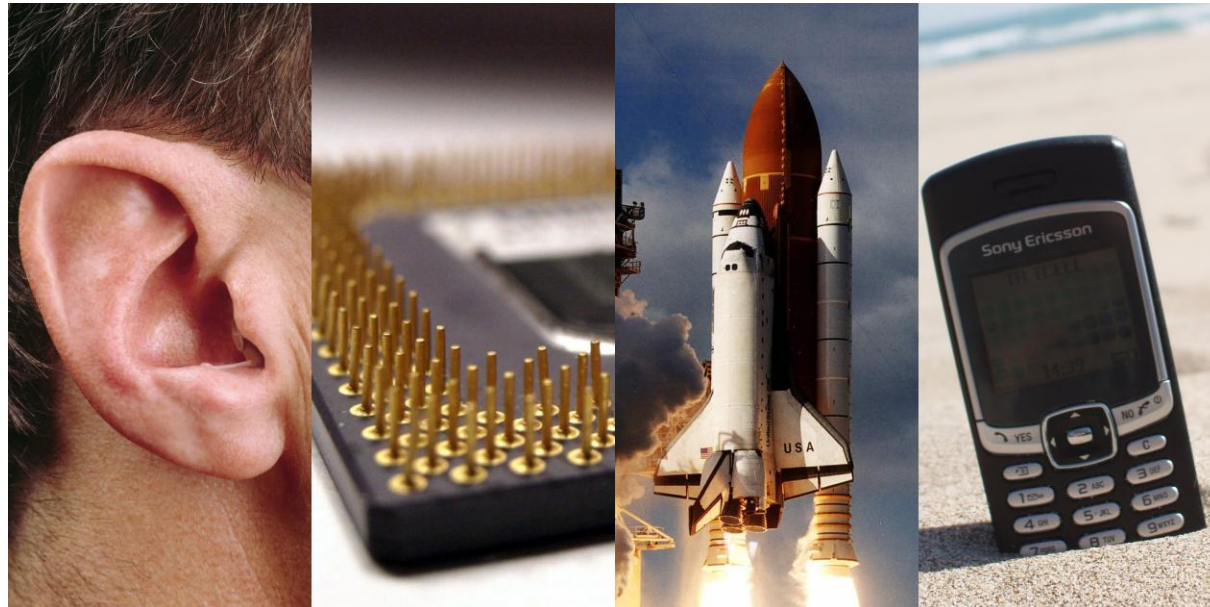


PARAMETER		FROM	TO	V _{CC}	Operating free-air temperature (T _A)						UNIT
					25°C			−40°C to 85°C			
					MIN	TYP	MAX	MIN	TYP	MAX	
t _{pd}	Propagation delay	A	Y	2 V		45	95			120	ns
				4.5 V		9	19			24	
				6 V		8	16			20	

Circuit Design is a Game of Tradeoffs

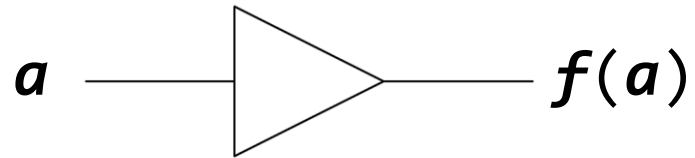
- **Cost:** typically proportional to the **physical circuit size (area)**
- **Performance:** depends on size, complexity, etc.
- **Power / Energy:** worse with larger circuits, higher voltage/frequency
- **Design effort:** circuit designers are **expensive** in time and money

No single approach: depends on **goals** and **requirements**



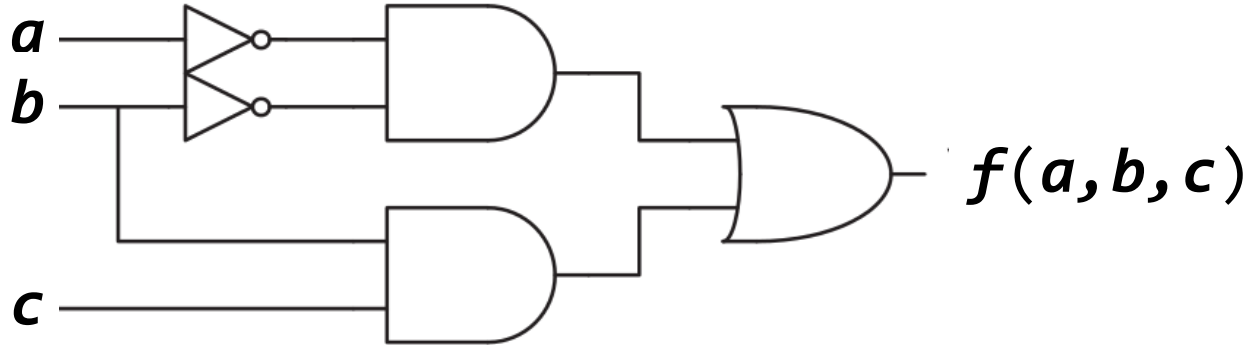
Digital Logic Abstractions for CS 211

Graphical Circuit



Algebraic Equation

$$f(a) = a$$



$$f(a, b, c) = a'b' + bc$$

Agenda

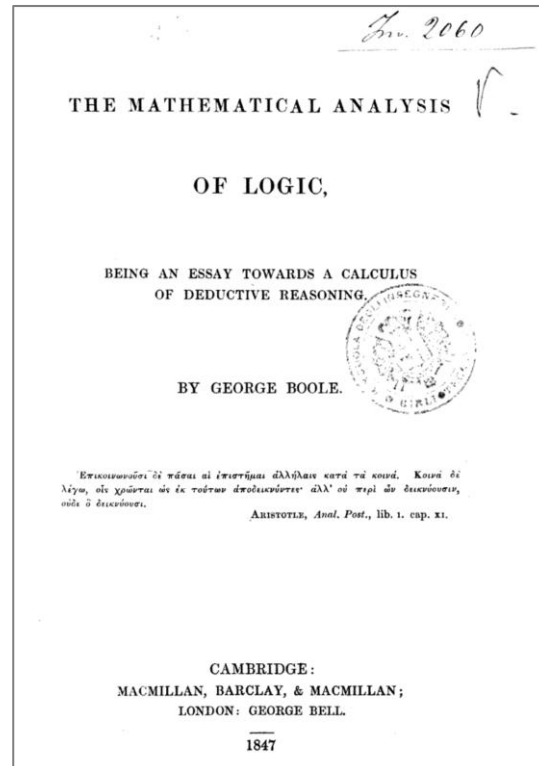
- The Digital Logic Abstraction
- **Boolean Algebra**
 - **Truth Tables**
 - Equations and Identities
- Digital Logic Gates

Boolean Algebra

- Mathematical framework for **binary values**

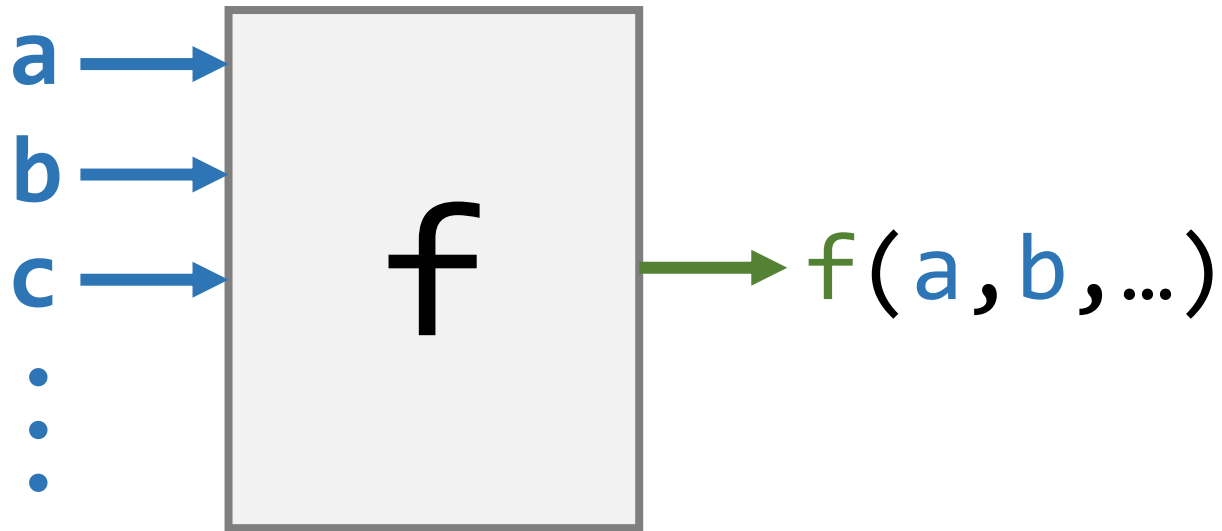
$$a, b \in \{0, 1\}$$

- Formalized by **George Boole** in the mid-1800s



Boolean Functions

- We can define **functions** and **equations** over Boolean variables



Example Function

$$f(a) = \begin{cases} 1 & a = 0 \\ 0 & a = 1 \end{cases}$$

a	$f(a)$
0	1
1	0

Example: Binary Addition

- Adding two **unsigned one-bit numbers**

$$f(a, b) = a + b \quad + \begin{array}{r} a \\ b \\ \hline f(a, b) \end{array}$$

$$\begin{array}{r} 0 \\ + 0 \\ \hline 00 \end{array} \quad \begin{array}{r} 0 \\ + 1 \\ \hline 01 \end{array} \quad \begin{array}{r} 1 \\ + 0 \\ \hline 01 \end{array} \quad \begin{array}{r} 1 \\ + 1 \\ \hline 10 \end{array}$$

Truth Table Representation

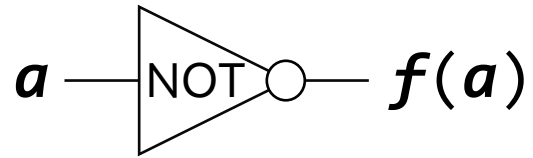
- Fully defines the function (all possible inputs)

$$\begin{array}{r} 0 \\ + 0 \\ \hline 00 \end{array} \quad \begin{array}{r} 0 \\ + 1 \\ \hline 01 \end{array} \quad \begin{array}{r} 1 \\ + 0 \\ \hline 01 \end{array} \quad \begin{array}{r} 1 \\ + 1 \\ \hline 11 \end{array}$$

“Truth Table”

inputs		outputs
a	b	$f(a, b)$
0	0	00
0	1	01
1	0	01
1	1	10

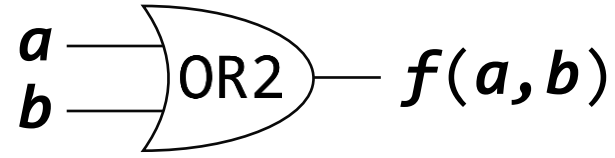
Truth Tables: Examples



a	$f(a)$
-----	--------

0	1
---	---

1	0
---	---



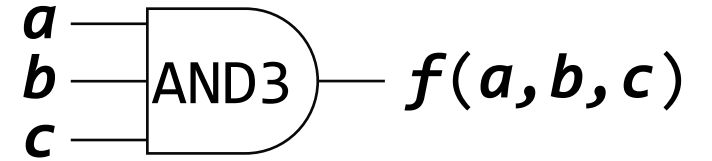
ab	$f(a, b)$
------	-----------

00	0
----	---

01	1
----	---

10	1
----	---

11	1
----	---



abc	$f(a, b, c)$
-------	--------------

000	0
-----	---

001	0
-----	---

010	0
-----	---

011	0
-----	---

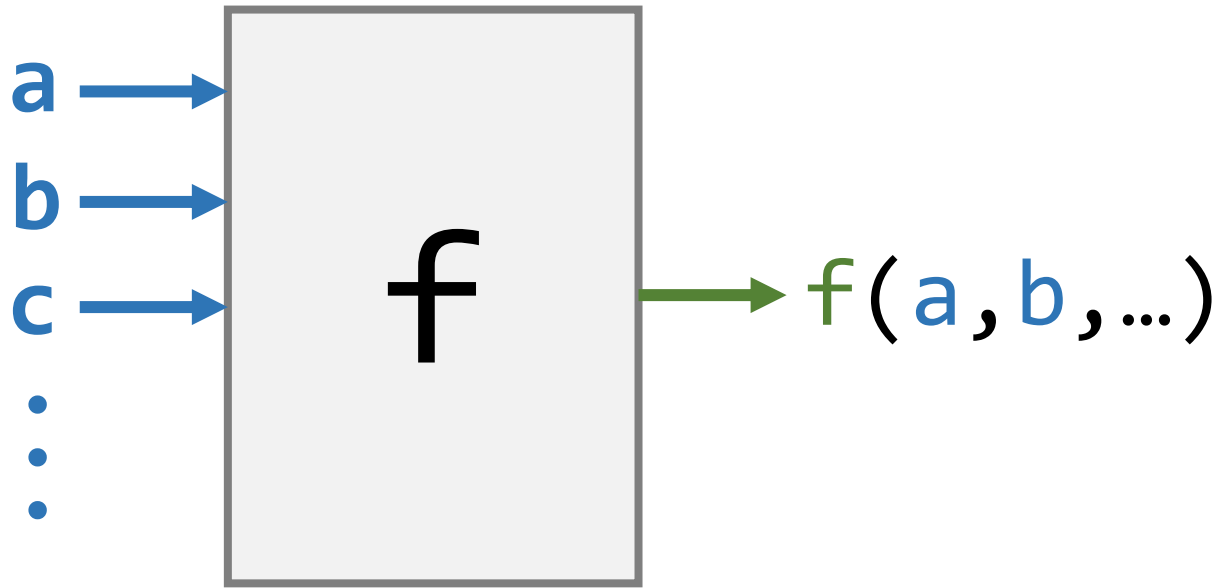
100	0
-----	---

101	0
-----	---

110	0
-----	---

111	1
-----	---

Truth Tables (Generalized)

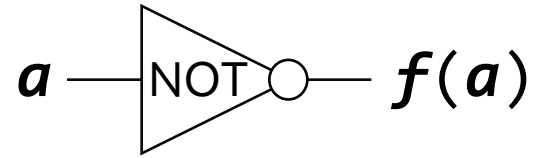


inputs	outputs
abc	$f(a, b, c)$
000	$f(0, 0, 0)$
001	$f(0, 0, 1)$
010	$f(0, 1, 0)$
011	$f(0, 1, 1)$
100	$f(1, 0, 0)$
101	$f(1, 0, 1)$
110	$f(1, 1, 0)$
111	$f(1, 1, 1)$

Agenda

- The Digital Logic Abstraction
- Boolean Algebra
 - Truth Tables
 - **Equations and Identities**
- Digital Logic Gates

Boolean Algebra Representations



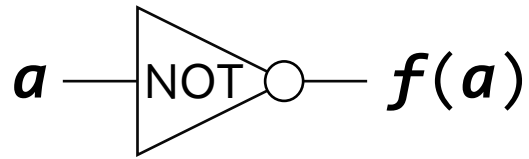
a	$f(a)$
0	1
1	0

```
 $f(a)$  =  $!a$  // C-style notation  
      =  $a'$  // Boolean algebra notation
```

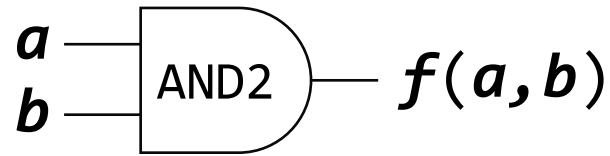
“a prime”

Functional Completeness

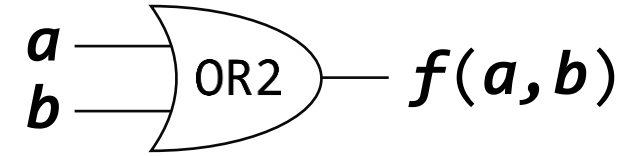
- We can express **any Boolean function** in terms of {**NOT**, **AND**, **OR**}



a	f
0	1
1	0



ab	f
00	0
01	0
10	0
11	1



ab	f
00	0
01	1
10	1
11	1

$$f(a) = a'$$

= ¬a

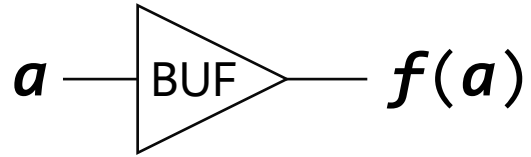
$$f(a, b) = a * b$$

$$= ab = a \wedge b$$

$$f(a, b) = a + b$$

$$a \vee b$$

Example: Buffer and Inverter



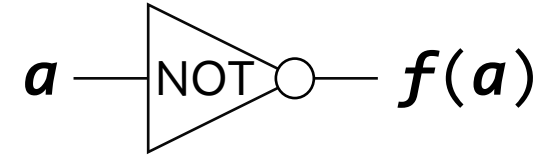
a	$f(a)$
-----	--------

0	0
---	---

1	1
---	---

$\leftarrow a = 1$

$$f(a) = a$$



a	$f(a)$
-----	--------

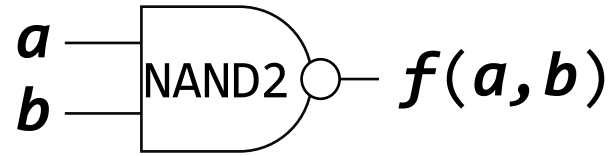
0	1
---	---

1	0
---	---

$\leftarrow a = 0$

$$f(a) = a'$$

Example: NAND2



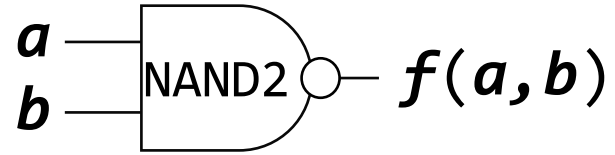
ab	f
00	1
01	1
10	1
11	0

$$f(a, b) = (ab=00) \vee (ab=01) \vee (ab=10)$$

$\downarrow \qquad \qquad \downarrow \quad \underbrace{\qquad \qquad} \quad \downarrow \quad \underbrace{\qquad \qquad} \quad \downarrow$

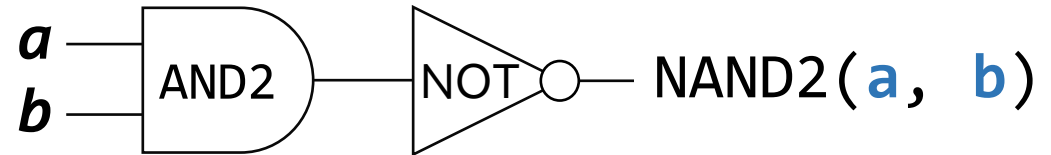
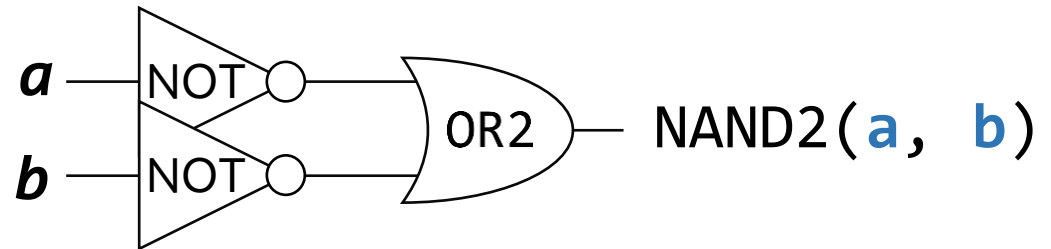
$$= a'b' + a'b + ab'$$

Example: NAND2

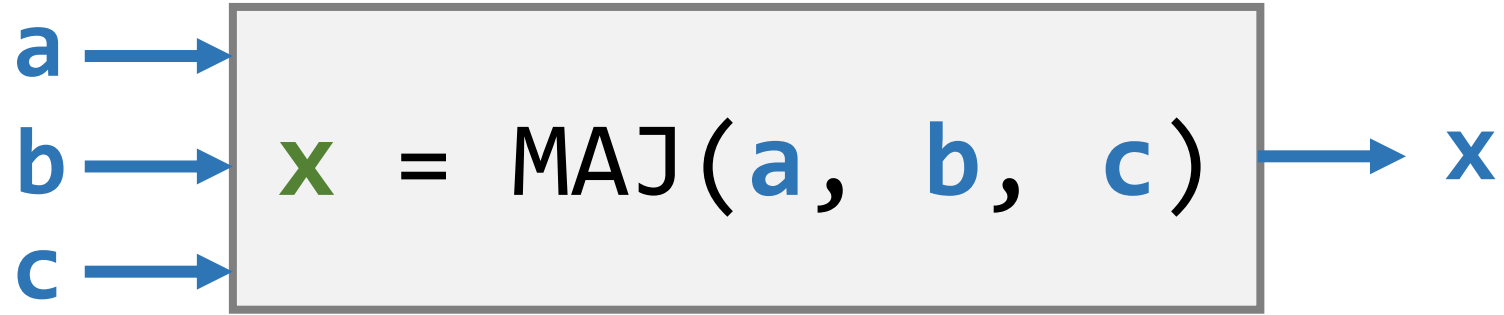


ab	f
00	1
01	1
10	1
11	0

$$\begin{aligned} f(a, b, c) &= a' + b' \\ &= (ab)' \end{aligned}$$



Example: Majority Function

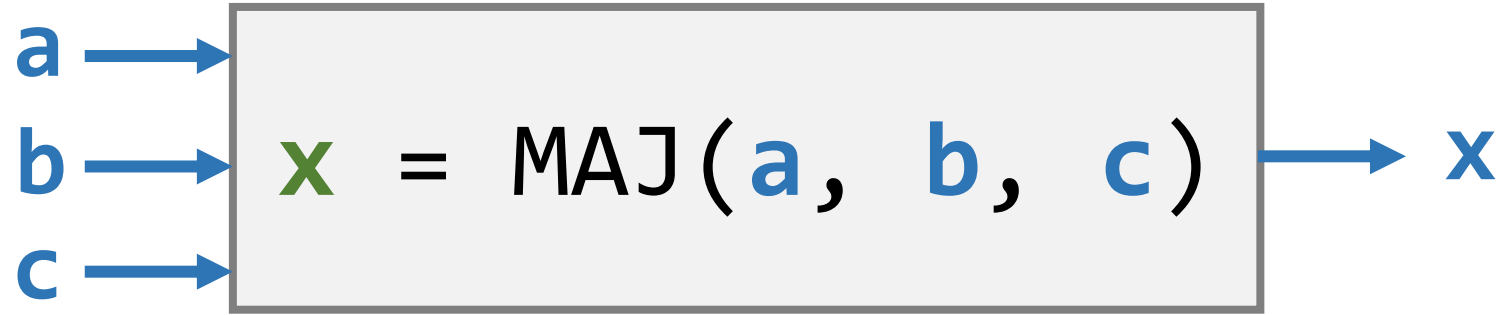


abc	x
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

$$f(a, b, c) = \underbrace{a'bc + ab'c + abc}_{c(a'b + ab')} + \underbrace{abc' + abc}_{a(bc' + bc)}$$

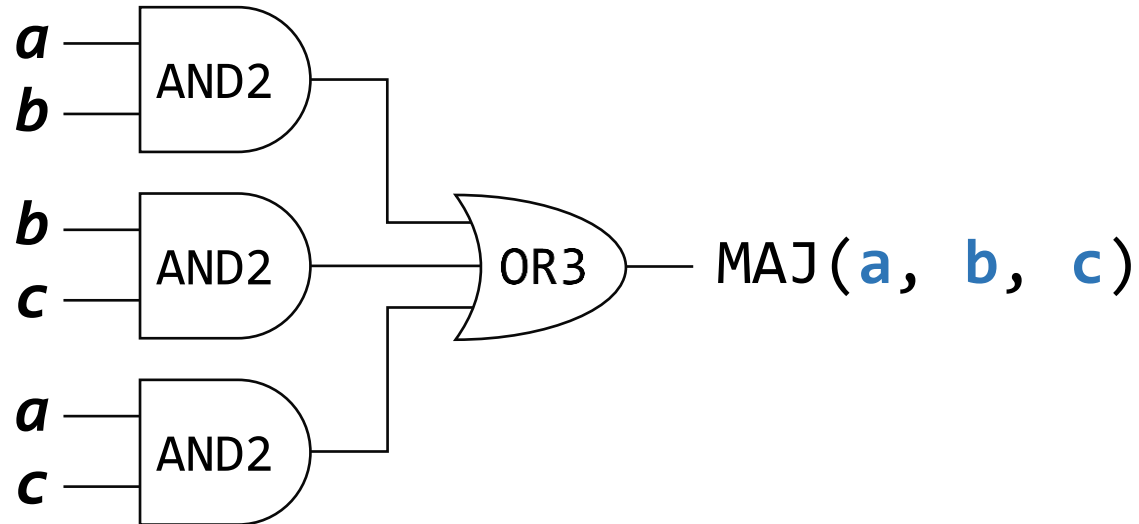
A red arrow originates from the red box around the row (011, 1) in the truth table and points to the first term of the handwritten expression, $a'bc + ab'c + abc$.

Example: Majority Function



abc	x
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

$$f(a, b, c) = ab + bc + ac$$



Truth Table to Algebraic Equation

Truth Table

ab $f(a, b)$

00 $f(00)$

01 $f(01)$

10 $f(10)$

11 $f(11)$

Algebraic Representation

$f(a, b) =$

$a'b' * f(00) +$

$a'b * f(01) +$

$ab' * f(10) +$

$ab * f(11)$

NOT

a f

0 1

1 0

$$f(a) = a' \cdot 1 + a \cdot 0$$

$$f(a) = a'$$

Truth Table to Algebraic Equation

Truth Table

ab	$f(a, b)$
------	-----------

00	$f(00)$
----	---------

01	$f(01)$
----	---------

10	$f(10)$
----	---------

11	$f(11)$
----	---------

Algebraic Representation

$f(a, b) =$

$a'b' * f(00) +$

$a'b * f(01) +$

$ab' * f(10) +$

$ab * f(11)$

NOT

a	f
-----	-----

0	1
---	---

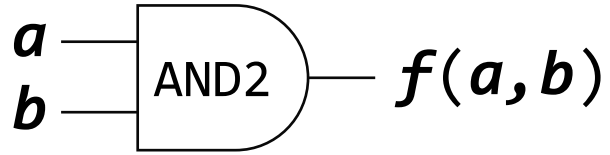
1	0
---	---

$f(a) = a' * f(0) + a * f(1)$

$= a' * 1 + a * 0$

$= a'$

Example: AND



ab	f
00	0
01	0
10	0
11	1

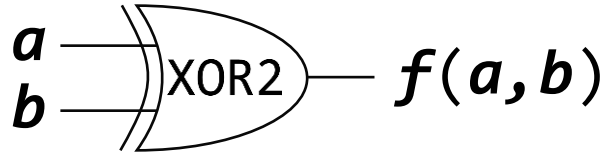
Ignore
 $f=0$ cases

$$f = \begin{cases} 1 & \text{iff } ab == 11 \\ 0 & \text{otherwise} \end{cases}$$

Algebraic Representation

$$\begin{aligned} f(a, b) &= a * b * f(1, 1) \\ &= ab \end{aligned}$$

Example: XOR



$$f = \begin{cases} 1 & \text{iff } ab == 01 \\ & \text{OR } ab == 10 \\ 0 & \text{otherwise} \end{cases}$$






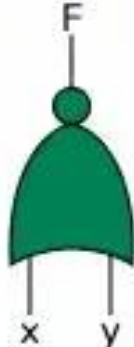


ab	f
00	0
01	1
10	1
11	0

Algebraic Representation

$$f(a, b) = a'b + ab'$$

$$a \oplus b$$

Standard Logic Blocks

Name	AND	OR	Inverter	Buffer	NAND	NOR	Exclusive-OR (XOR)	Exclusive-NOR or equivalence																																																																																																						
Graphic Symbol																																																																																																														
Algebraic Function	$F=xy$	$F=x+y$	$F=x'$	$F=x$	$F=(xy)'$	$F=(x+y)'$	$F=(x\oplus y)$	$F=(x\oplus y)'$																																																																																																						
Truth Table	<table><tr><td>x</td><td>y</td><td>F</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1	<table><tr><td>x</td><td>y</td><td>F</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1	<table><tr><td>x</td><td>F</td></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0	<table><tr><td>x</td><td>F</td></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	F	0	0	1	1	<table><tr><td>x</td><td>y</td><td>F</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0	<table><tr><td>x</td><td>y</td><td>F</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0	<table><tr><td>x</td><td>y</td><td>F</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0	<table><tr><td>x</td><td>y</td><td>F</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																																																																																																												
0	0	0																																																																																																												
0	1	0																																																																																																												
1	0	0																																																																																																												
1	1	1																																																																																																												
x	y	F																																																																																																												
0	0	0																																																																																																												
0	1	1																																																																																																												
1	0	1																																																																																																												
1	1	1																																																																																																												
x	F																																																																																																													
0	1																																																																																																													
1	0																																																																																																													
x	F																																																																																																													
0	0																																																																																																													
1	1																																																																																																													
x	y	F																																																																																																												
0	0	1																																																																																																												
0	1	1																																																																																																												
1	0	1																																																																																																												
1	1	0																																																																																																												
x	y	F																																																																																																												
0	0	1																																																																																																												
0	1	0																																																																																																												
1	0	0																																																																																																												
1	1	0																																																																																																												
x	y	F																																																																																																												
0	0	0																																																																																																												
0	1	1																																																																																																												
1	0	1																																																																																																												
1	1	0																																																																																																												
x	y	F																																																																																																												
0	0	1																																																																																																												
0	1	0																																																																																																												
1	0	0																																																																																																												
1	1	1																																																																																																												

Basic Identities of Boolean Algebra

- Used to simplify complex equations

$a'' = a$		involution
$a \cdot 1 = a$	\longleftrightarrow	$a + 0 = a$ identity
$a \cdot 0 = 0$	\longleftrightarrow	$a + 1 = 1$ null
$a \cdot a = a$	\longleftrightarrow	$a + a = a$ idempotent
$a \cdot a' = 0$	\longleftrightarrow	$a + a' = 1$ complementarity
$ab = ba$	\longleftrightarrow	$a + b = b + a$ commutativity
$(ab)c = a(bc)$	\longleftrightarrow	$(a + b) + c = a + (b + c)$ associativity
$a(b + c) = ab + ac$	\longleftrightarrow	$a + bc = (a + b)(a + c)$ distributivity
$ab + a = a$	\longleftrightarrow	$(a + b)a = a$ absorption
$(ab)' = a' + b'$	\longleftrightarrow	$(a + b)' = a'b'$ DeMorgan's law

Exercise: Simplify the Expressions

→

$$\boxed{a'bc + a'bc' + ab}$$

$$a'b(\overset{\bullet}{c+c'})^1 + ab$$

$$a'b + ab$$

$$b(\overset{\bullet}{a'+a})^1$$

$$\boxed{b}$$

$$\boxed{(a' + b')(a' + b)}$$

$$\underline{a'a' + a'b + b'a' + b'b}^0$$

$$\underline{a' + a'b + b'a'}$$

$$a'(\cancel{1 + b + b'})^1$$

$$\boxed{a'}$$

a	f
0	1.1 = 1
1	0.0 = 0

a'

Exercise: Simplify the Expressions

$$((a + d)' (b' + c)')'$$

$\underbrace{(a + d)'}_x \quad \underbrace{(b' + c)'}_y$

$$(x' y')' \Rightarrow ((x + y)')'$$

deMorgan's

$$x + y$$

$$a + d + b' + c$$

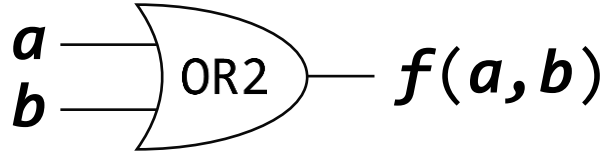
$$\overline{abc} + \overline{a'c} + bc'$$

$$\begin{aligned} & c(\overline{ab} + \overline{a'}) + bc' \\ & c(\overline{b} + \overline{a'}) + bc' \\ & c\overline{b} + c\overline{a'} + bc' \\ & b(\overline{c} + \overline{c'}) + c\overline{a'} \\ & \quad \downarrow \\ & \quad 1 \end{aligned}$$

$$b + c\overline{a'}$$

ab	$\overline{ab} + \overline{a'}$
00	1
01	1
10	0
11	1
$\overline{b} + \overline{a'}$	
00	1
01	1
10	0
11	1

Example: OR



ab	f
00	0
01	1
10	1
11	1

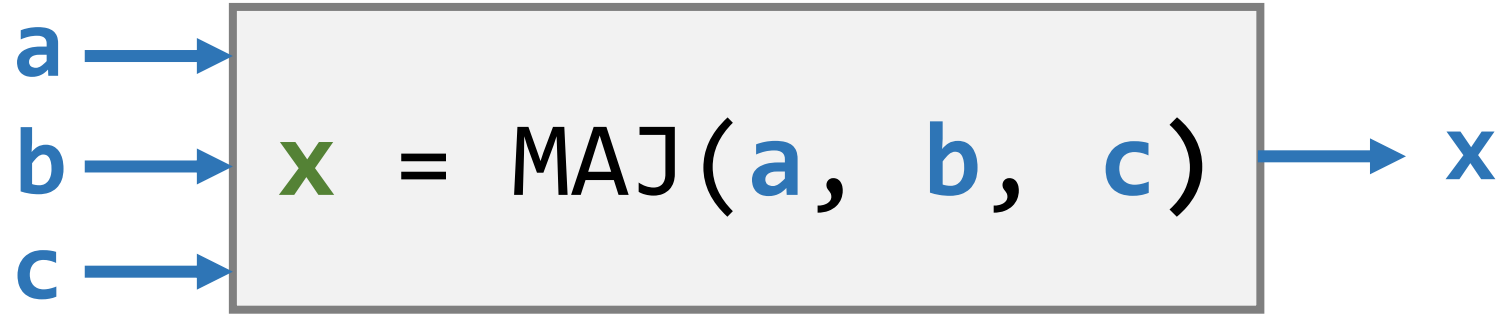
Boolean algebra
(simplification using identities)

$$f = \begin{cases} 1 & \text{iff } ab == 01 \\ & \text{OR } ab == 10 \\ & \text{OR } ab == 11 \\ 0 & \text{otherwise} \end{cases}$$

Algebraic Representation

$$\begin{aligned} f(a, b) &= a'b + ab' + ab \\ &= a'b + a(b' + b) \\ &= a'b + a \\ &= a + b \end{aligned}$$

Example: Majority Function



abc	x
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

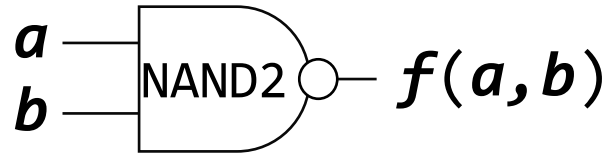
Algebraic Representation

$$\begin{aligned} f(a, b, c) &= a'bc + ab'c + abc' + abc \\ &= \dots \\ &= ac + ab + bc \end{aligned}$$

Simplification
left as an exercise 😊

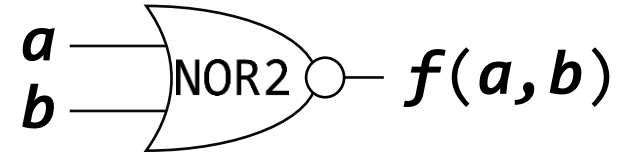
Aside: Functional Completeness

- {NAND} and {NOR} are **functionally complete** singleton sets



↙

ab	f
00	1
01	1
10	1
11	0



ab	f
00	1
01	0
10	0
11	0

Agenda

- The Digital Logic Abstraction
- Boolean Algebra
 - Truth Tables
 - Equations and Identities
- **Digital Logic Gates**

Representations of Boolean Logic

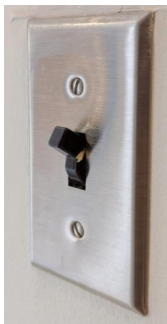
- Any **two-state system** can represent Boolean values

Truth Table

a	f
0	1
1	0

Boolean Equation

$$f(a) = a'$$



Switch
up/down



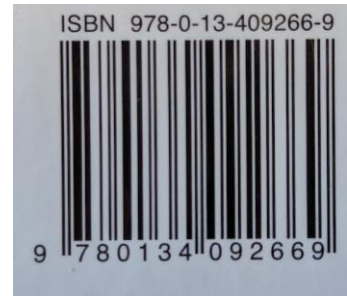
Lever
up/down



State
clipped/unclipped



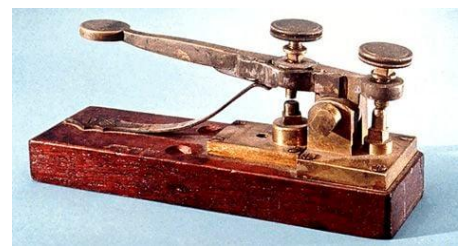
Color
red/black



Line width
narrow/wide



Pixel color
Black/white



Sound Length
Long/short tap

<https://www.smithsonianmag.com/arts-culture/how-the-telegraph-went-from-semaphore-to-communication-game-changer-1403433/>

Building a Logic Gate: Mechanical Switches

- Mechanical switches **controlled manually**



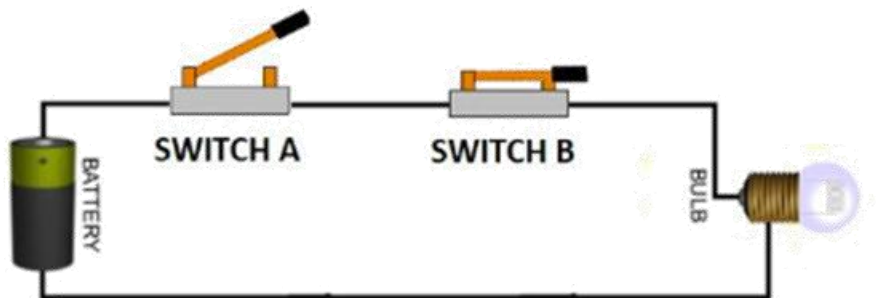
Truth Table

switch	light
0	1
1	0

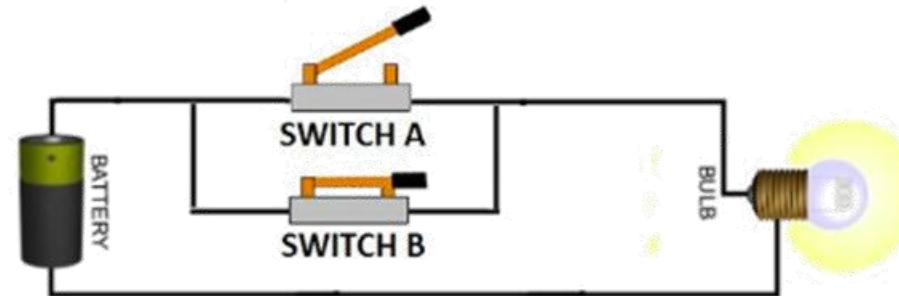
Boolean Equation

$$\text{light}(sw) = sw'$$

AND circuit

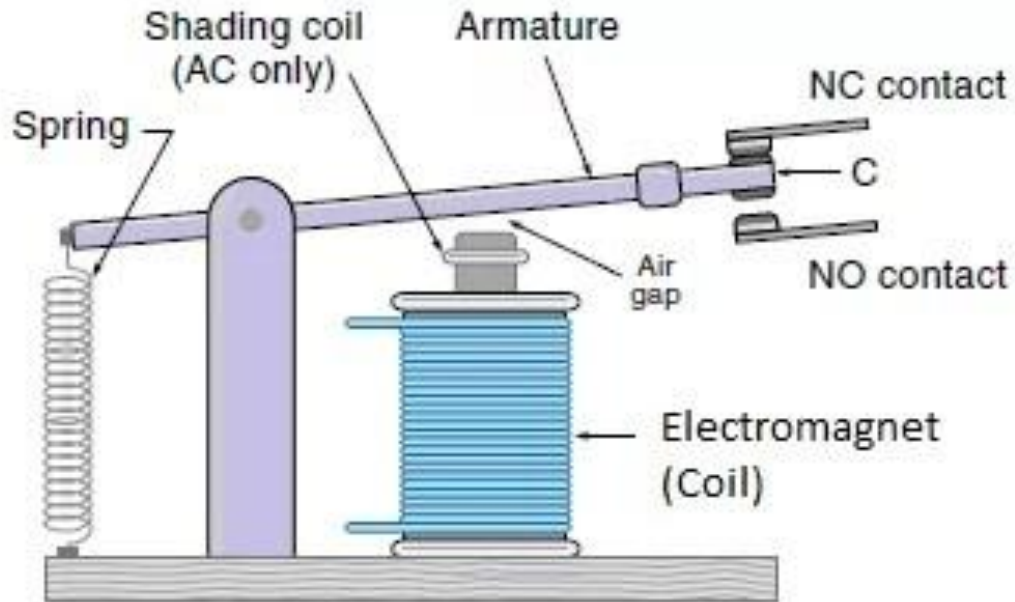


OR circuit



Building a Logic Gate: Electromechanical Switches

- Mechanical switches **controlled by electricity**



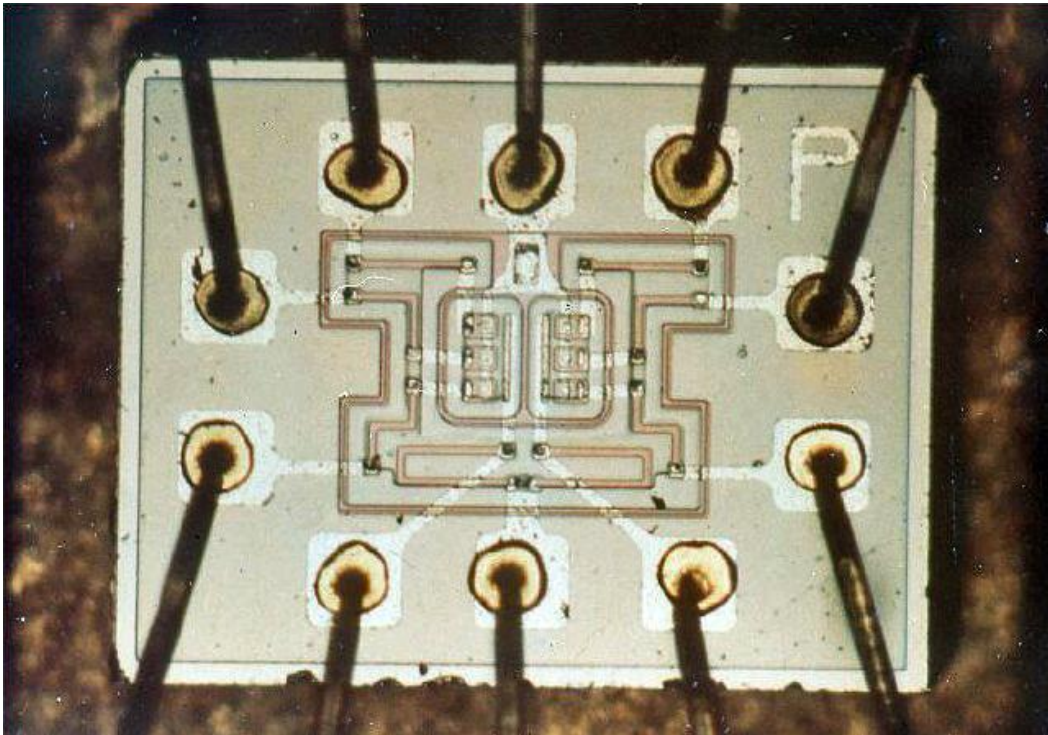
<https://www.electricalandcontrol.com/electromechanical-relays-emr/>

Truth Table	
electromagnet	contact
off	NC
on	NO

Building a Logic Gate: Electrical Switches

- **Fully electronic** switch (transistor): no mechanical/moving parts

NOR3 Gate from the
Apollo Guidance Computer (1960s)



https://commons.wikimedia.org/wiki/File:Agc_nor2.jpg

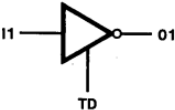
abc	NOR3(a, b, c)
000	1
001	0
010	0
011	0
100	0
101	0
110	0
111	0

Example: “Standard Cell” Library for Chip Design

intel

INTRODUCTION TO CELL-BASED DESIGN

INVTD — 3-STATE INVERTER



Logic Table

Inputs		Outputs
I1	TD	O1
0	1	1
1	1	0
X	0	Z

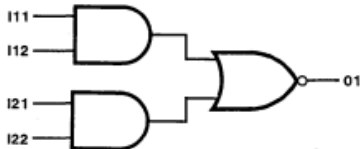
INVTD Description

Function: 3-State Inverter with Active High Output Enable, Normal Drive

intel

INTRODUCTION TO CELL-BASED DESIGN

AOI22 — AND-OR INVERT GATE



Logic Table

Inputs				Outputs
I11	I12	I21	I22	O1
1	1	X	X	0
X	X	1	1	0
Any other combination				1

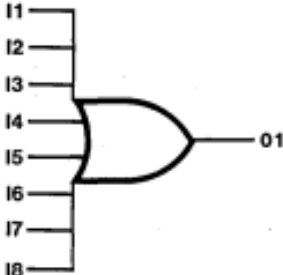
AOI22 Description

Function: 2 AND2 into NOR2, Normal Drive

intel

INTRODUCTION TO CELL-BASED DESIGN

OR8 — 8 INPUT OR GATE



Logic Table

Inputs								Outputs
I1	I2	I3	I4	I5	I6	I7	I8	O1
0	0	0	0	0	0	0	0	0
1	X	X	X	X	X	X	X	1
X	1	X	X	X	X	X	X	1
X	X	1	X	X	X	X	X	1
X	X	X	1	X	X	X	X	1
X	X	X	X	1	X	X	X	1
X	X	X	X	X	1	X	X	1
X	X	X	X	X	X	1	X	1
X	X	X	X	X	X	X	1	1

OR8 Description

Function: 8 Input OR, Normal Drive

CS 211: Intro to Computer Architecture

11.1: Digital Logic and Boolean Algebra

Minesh Patel

Spring 2025 – Tuesday 8 April