Gates and Logic: From Transistors to Logic Gates and Logic Circuits

Prof. Hakim Weatherspoon
CS 3410
Computer Science
Cornell University

The slides are the product of many rounds of teaching CS 3410 by Professors Weatherspoon, Bala, Bracy, and Sirer.

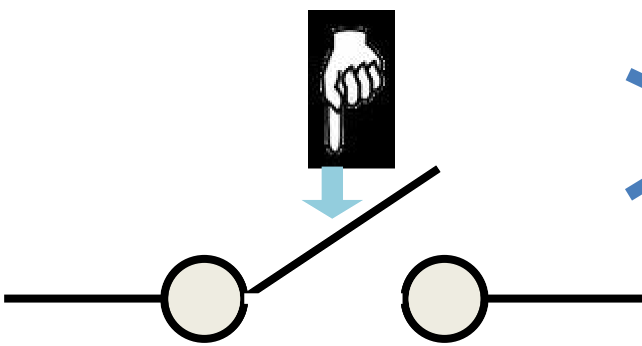
Goals for Today

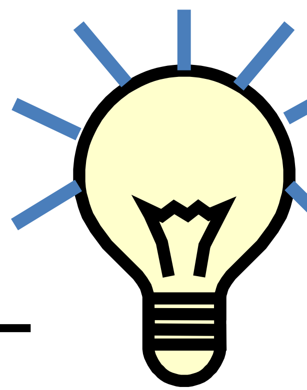
- From Switches to Logic Gates to Logic Circuits
- Logic Gates
  - From switches
  - Truth Tables
- Logic Circuits
  - Identity Laws
  - From Truth Tables to Circuits (Sum of Products)
- Logic Circuit Minimization
  - Algebraic Manipulations
  - Truth Tables (Karnaugh Maps)
- Transistors (electronic switch)

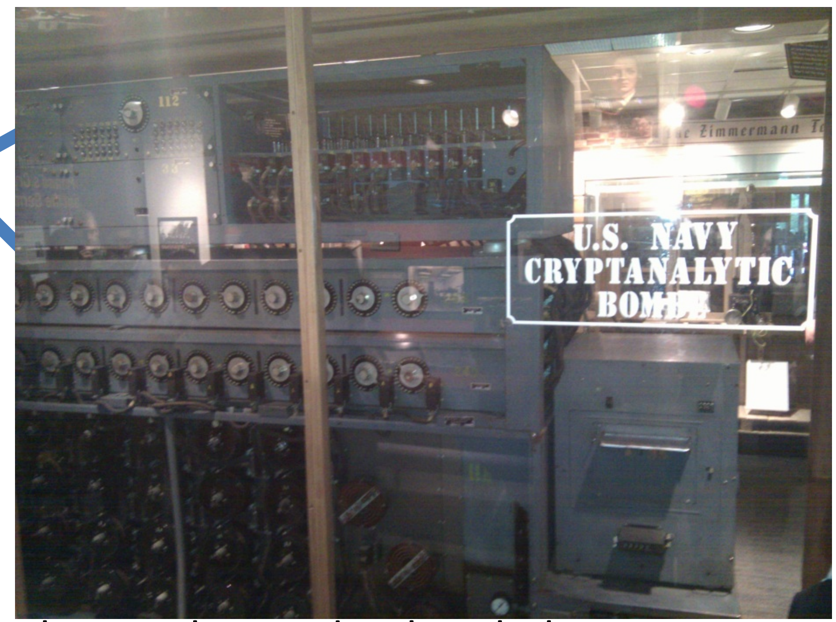
A switch

- Acts as a *conductor* or *insulator*

- Can be used to build amazing things…

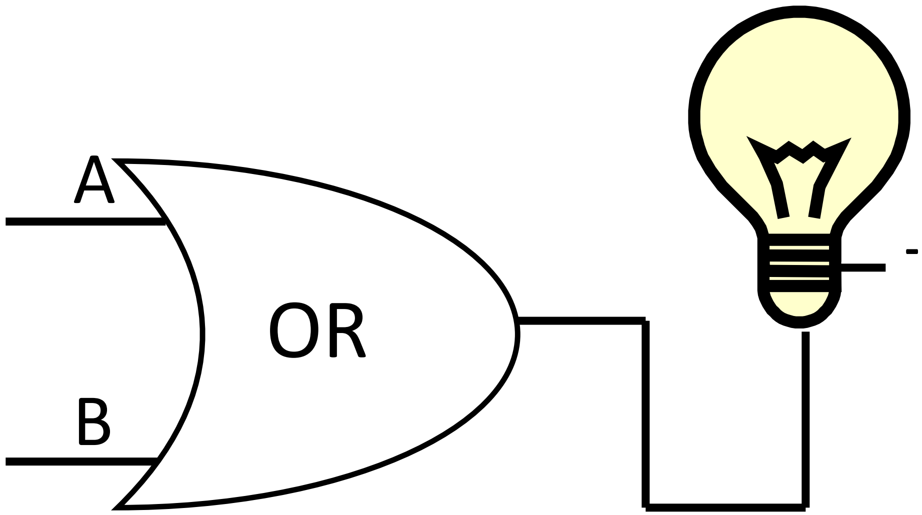






The Bombe used to break the German Enigma machine during World War II

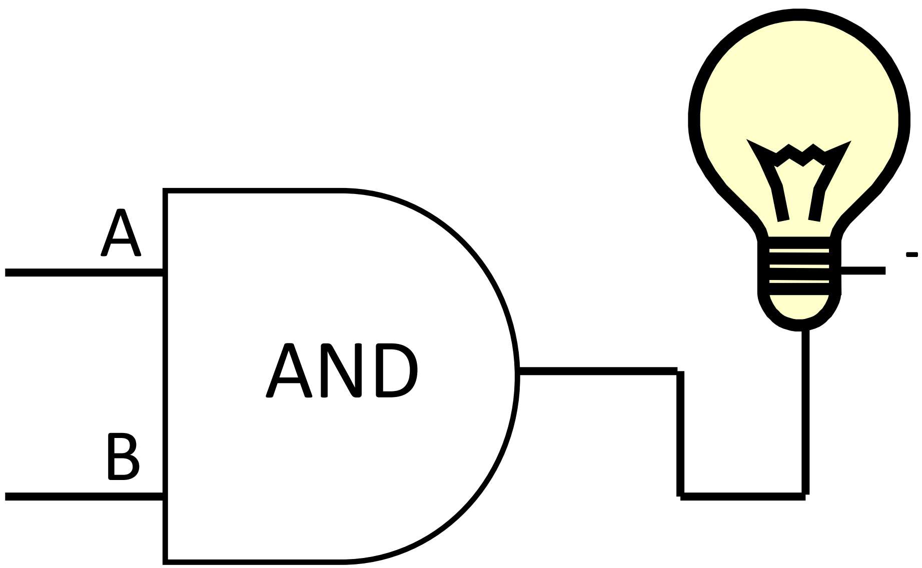
Basic Building Blocks: Switches to Logic Gates



- Either (OR)

Truth Table

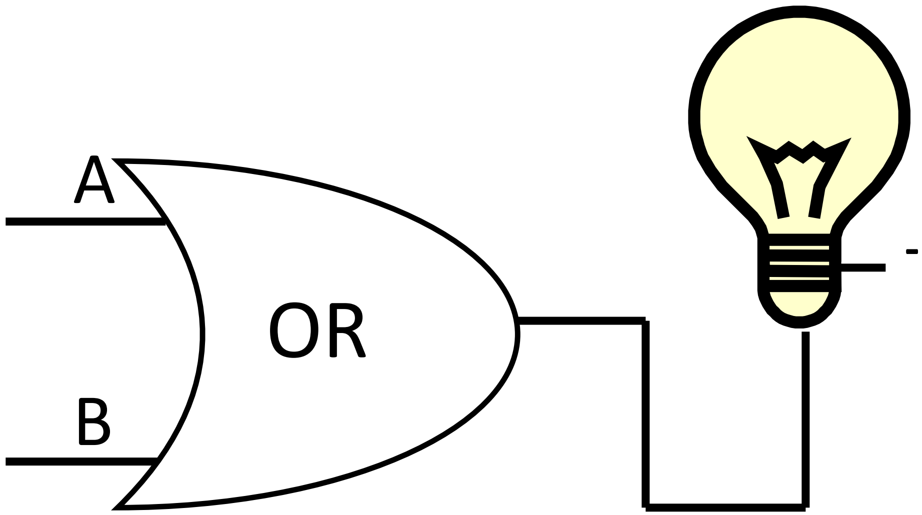
A	B	Light
OFF	OFF	
OFF	ON	
ON	OFF	
ON	ON	



- Both (AND)

A	B	Light
OFF	OFF	
OFF	ON	
ON	OFF	
ON	ON	

Basic Building Blocks: Switches to Logic Gates

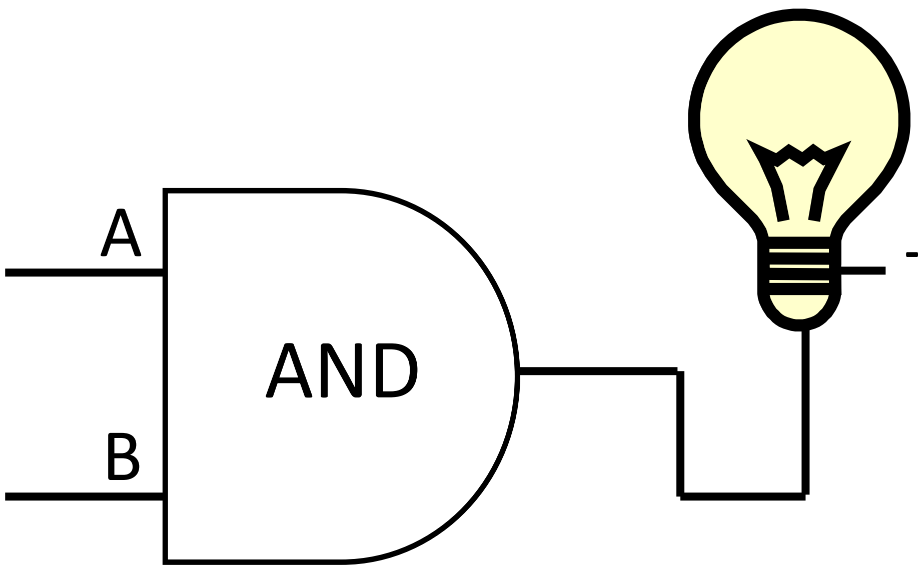


- Either (OR)

Truth Table

A	B	Light
0	0	
0	1	
1	0	
1	1	

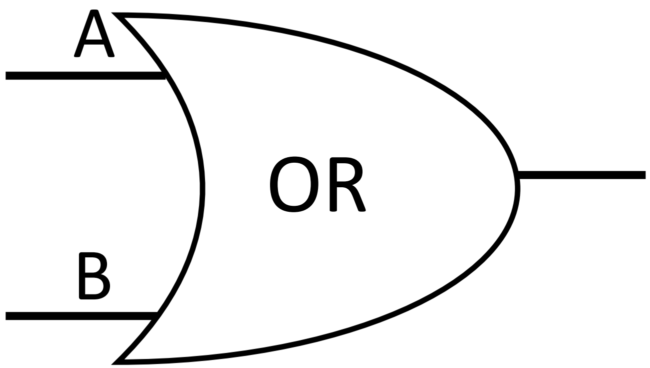
0 = OFF
1 = ON

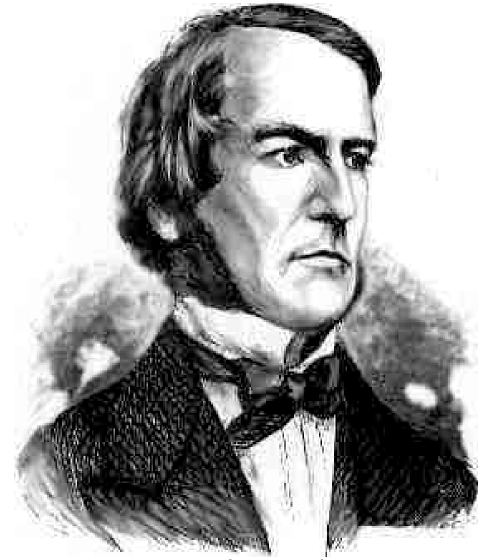


- Both (AND)

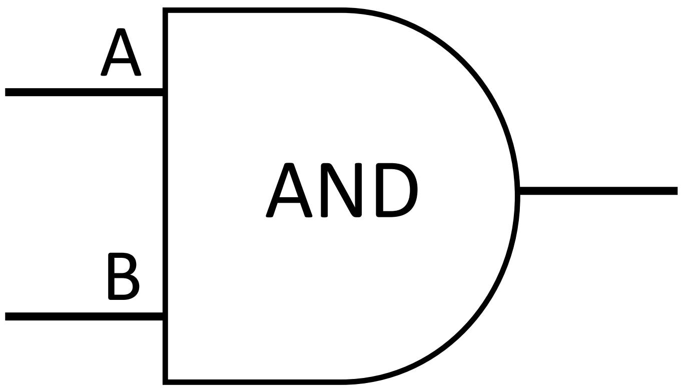
A	B	Light
0	0	
0	1	
1	0	
1	1	

Basic Building Blocks: Switches to Logic Gates





George Boole,(1815-1864)



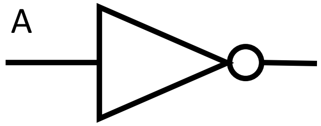
- Did you know?
- George Boole Inventor of the idea of logic gates. He was born in Lincoln, England and he was the son of a shoemaker in a low class family.

Takeaway

- Binary (two symbols: true and false) is the basis of Logic Design

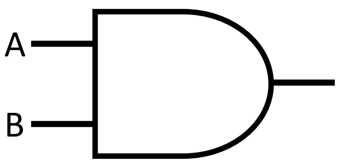
Building Functions: Logic Gates

- NOT:



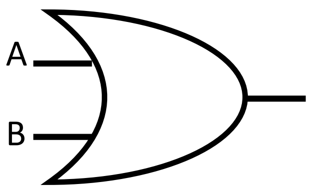
A	Out
0	1
1	0

- AND:



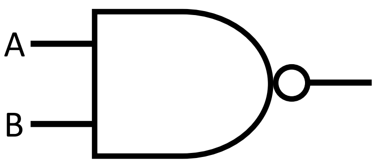
A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

- OR:



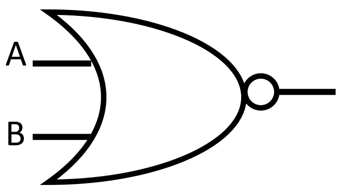
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

NAND:



A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

NOR:



A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

- Logic Gates
  - digital circuit that either allows a signal to pass through it or not.
  - Used to build logic functions
  - There are seven basic logic gates:
    AND, OR, **NOT**,
    NAND (not AND), NOR (not OR), XOR, and XNOR (not XOR) [later]

Goals for Today

- From Switches to Logic Gates to Logic Circuits
- Logic Gates
  - From switches
  - Truth Tables
- Logic Circuits
  - Identity Laws
  - From Truth Tables to Circuits (Sum of Products)
- Logic Circuit Minimization
  - Algebraic Manipulations
  - Truth Tables (Karnaugh Maps)
- Transistors (electronic switch)

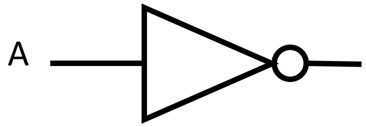
Next Goal

- Given a Logic function, create a Logic Circuit that implements the Logic Function…
- …and, *with the minimum number of logic gates*

- Fewer gates: A cheaper ($$$) circuit!

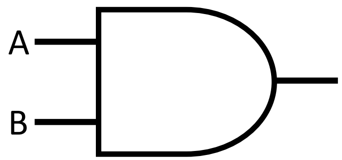
Logic Gates

NOT:



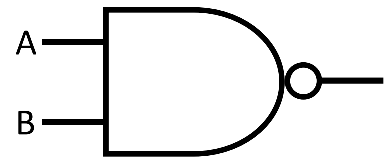
A	Out
0	1
1	0

AND:



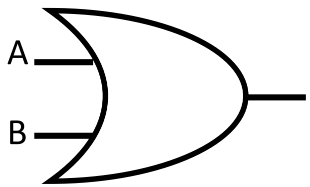
A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

NAND:



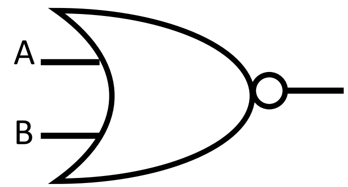
A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

OR:



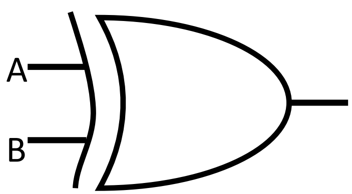
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

NOR:



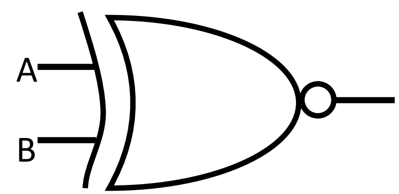
A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

XOR:



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

XNOR:



A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

Logic Implementation

- How to implement a desired logic function?

a	b	c	out
0	**0**	**0**	**0**
0	**0**	**1**	**1**
0	**1**	**0**	**0**
0	**1**	**1**	**1**
1	**0**	**0**	**0**
1	**0**	**1**	**1**
1	**1**	**0**	**0**
1	**1**	**1**	**0**

Logic Implementation

- How to implement a desired logic function?

a	b	c	out	minterm
0	**0**	**0**	**0**	$\bar{a}\,\bar{b}\,\bar{c}$
0	**0**	**1**	**1**	$\bar{a}\,\bar{b}\,c$
0	**1**	**0**	**0**	$\bar{a}\,b\,\bar{c}$
0	**1**	**1**	**1**	$\bar{a}\,b\,c$
1	**0**	**0**	**0**	$a\,\bar{b}\,\bar{c}$
1	**0**	**1**	**1**	$a\,\bar{b}\,c$
1	**1**	**0**	**0**	$a\,b\,\bar{c}$
1	**1**	**1**	**0**	$a\,b\,c$

1) Write minterms
2) sum of products:

- OR of all minterms where out=1

Logic Equations

- NOT:
  - out = $\bar{a}$ = !a = $\neg$a
- AND:
  - out = $a \cdot b$ = a & b = $a \wedge b$
- OR:
  - out = $a + b$ = a | b = $a \vee b$
- XOR:
  - out = $a \oplus b = a\bar{b} + \bar{a}b$
- Logic Equations
  - Constants: true = 1, false = 0
  - Variables: a, b, out, ...
  - Operators (above): AND, OR, NOT, etc.

NAND:
- out = $\overline{a \cdot b}$ = !(a & b) = $\neg$ (a $\wedge$ b)

NOR:
- out = $\overline{a + b}$ = !(a | b) = $\neg$ (a $\vee$ b)

XNOR:
- out = $\overline{a \oplus b} = ab + \overline{ab}$

Identities

Identities useful for manipulating logic equations

- For optimization & ease of implementation

$a + 0 =$

$a + 1 =$

$a + \bar{a} =$

$a \cdot 0 =$

$a \cdot 1 =$

$a \cdot \bar{a} =$

Identities

Identities useful for manipulating logic equations

- For optimization & ease of implementation

$$\overline{(a + b)} =$$

$$\overline{(a \cdot b)} =$$

$$a + a\,b =$$

$$a(b+c) =$$

$$\overline{a(b + c)} =$$

Goals for Today

- From Switches to Logic Gates to Logic Circuits
- Logic Gates
  - From switches
  - Truth Tables
- Logic Circuits
  - From Truth Tables to Circuits (Sum of Products)
  - Identity Laws
- **Logic Circuit Minimization – *why?***
  - Algebraic Manipulations
  - Truth Tables (Karnaugh Maps)
- Transistors (electronic switch)

Checking Equality w/Truth Tables

circuits $\leftrightarrow$ truth tables $\leftrightarrow$ equations

Example: $(a+b)(a+c) = a + bc$

a	b	c					
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

Takeaway

- Binary (two symbols: true and false) is the basis of Logic Design
- More than one Logic Circuit can implement same Logic function. Use Algebra (Identities) or Truth Tables to show equivalence.

Goals for Today

- From Switches to Logic Gates to Logic Circuits
- Logic Gates
  - From switches
  - Truth Tables
- Logic Circuits
  - From Truth Tables to Circuits (Sum of Products)
  - Identity Laws
- Logic Circuit Minimization
  - Algebraic Manipulations
  - Truth Tables (Karnaugh Maps)
- Transistors (electronic switch)

Next Goal

- How to standardize minimizing logic circuits?

Karnaugh Maps

How does one find the most efficient equation?

– Manipulate algebraically until…?
– Use Karnaugh Maps (optimize visually)
– Use a software optimizer

For large circuits

– Decomposition & reuse of building blocks

Minimization with Karnaugh maps (1)

a	b	c	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

- Sum of minterms yields
  - out =

Minimization with Karnaugh maps (2)

a	b	c	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

c \ ab	00	01	11	10
0	0	0	0	1
1	1	1	0	1

- Sum of minterms yields
  - $out = \overline{a}\overline{b}c + \overline{a}bc + a\overline{b}\overline{c} + a\overline{b}c$
- Karnaugh map minimization
  - Cover all 1's
  - Group adjacent blocks of 2^n 1's that yield a rectangular shape
  - Encode the common features of the rectangle
    - $out = a\overline{b} + \overline{a}c$

Karnaugh Minimization Tricks (1)

c \ ab	00	01	11	10
0	0	1	1	1
1	0	0	1	0

- Minterms can overlap
  - out =

c \ ab	00	01	11	10
0	1	1	1	1
1	0	0	1	0

- Minterms can span 2, 4, 8 or more cells
  - out =

Karnaugh Minimization Tricks (2)

cd \ ab	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	1	0	0	1
10	0	0	0	0

- The map wraps around
  - out =

cd \ ab	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

  - out =

Karnaugh Minimization Tricks (3)

cd \ ab	00	01	11	10
00	0	0	0	0
01	1	x	x	x
11	1	x	x	1
10	0	0	0	0

- “Don’t care” values can be interpreted individually in whatever way is convenient
  - assume all x’s = 1
  - out =

cd \ ab	00	01	11	10
00	1	0	0	x
01	0	x	x	0
11	0	x	x	0
10	1	0	0	1

  - assume middle x’s = 0
  - assume 4th column x = 1
  - out =

Minimization with K-Maps

c \ ab	00	01	11	10
0	0	0	0	1
1	1	1	0	1

(1) Circle the 1's (see below)

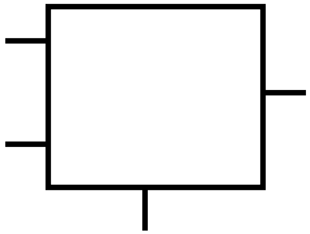
(2) Each circle is a logical component of the final equation

$$= a\bar{b} + \bar{a}c$$

Rules:

- Use fewest circles necessary to cover all 1's
- Circles must cover *only* 1's
- Circles span rectangles of size power of 2 (1, 2, 4, 8…)
- Circles should be as large as possible (all circles of 1?)
- Circles may wrap around edges of K-Map
- 1 may be circled multiple times *if* that means fewer circles

Multiplexer



- A multiplexer selects between multiple inputs
  - out = a, if d = 0
  - out = b, if d = 1

a	b	d	out
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

- Build truth table
- Minimize diagram
- Derive logic diagram

Takeaway

- Binary (two symbols: true and false) is the basis of Logic Design

- More than one Logic Circuit can implement same Logic function. Use Algebra (Identities) or Truth Tables to show equivalence.

- Any logic function can be implemented as "sum of products". Karnaugh Maps minimize number of gates.

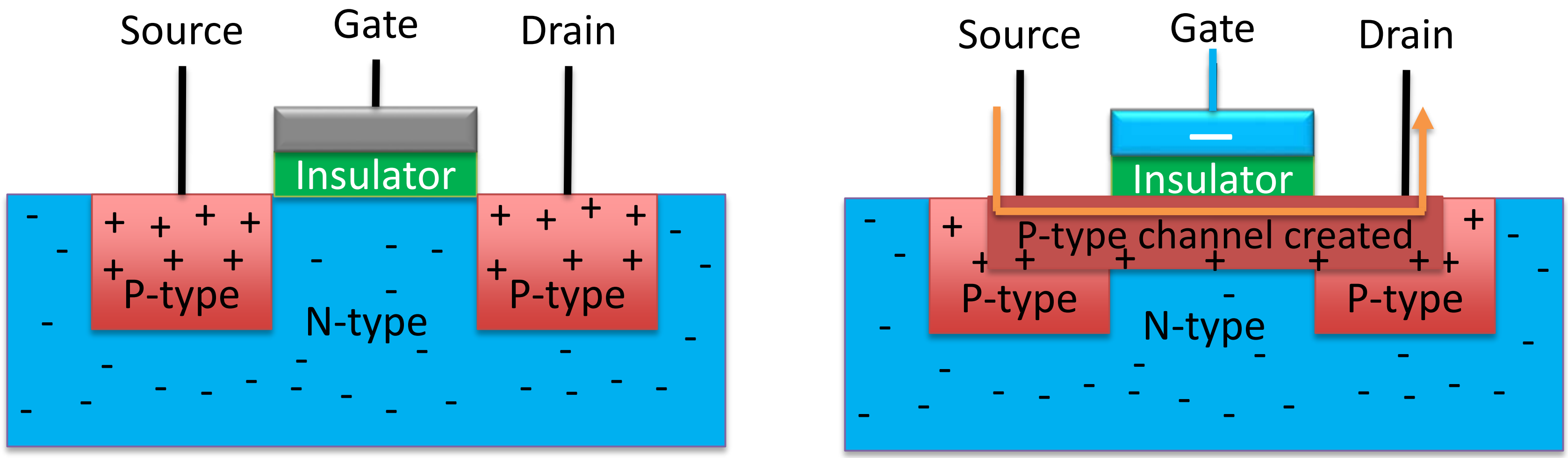
Goals for Today

- From Transistors to Gates to Logic Circuits
- Logic Gates
  - From transistors
  - Truth Tables
- Logic Circuits
  - From Truth Tables to Circuits (Sum of Products)
  - Identity Laws
- Logic Circuit Minimization
  - Algebraic Manipulations
  - Truth Tables (Karnaugh Maps)
- Transistors (electronic switch)

Silicon Valley & the Semiconductor Industry

- Transistors:
- Youtube video “How does a transistor work”
  https://www.youtube.com/watch?v=IcrBqCFLHIY
- Break: show some Transistor, Fab, Wafer photos

Transistors 101



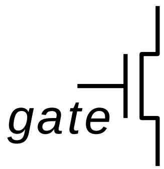
Off **On**

N-Type Silicon: negative free-carriers (electrons)
P-Type Silicon: positive free-carriers (holes)
P-Transistor: negative charge on gate generates electric field that creates a (+ charged) p-channel connecting source & drain
N-Transistor: works the opposite way
Metal-Oxide Semiconductor (Gate-Insulator-Silicon)

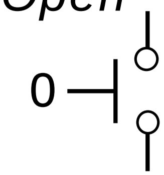
- Complementary MOS = **CMOS** technology uses both p- & n-type transistors

CMOS Notation

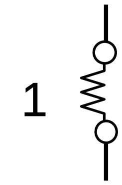
N-type



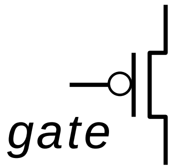
Off/Open



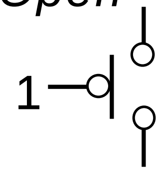
On/Closed



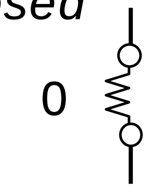
P-type



Off/Open



On/Closed

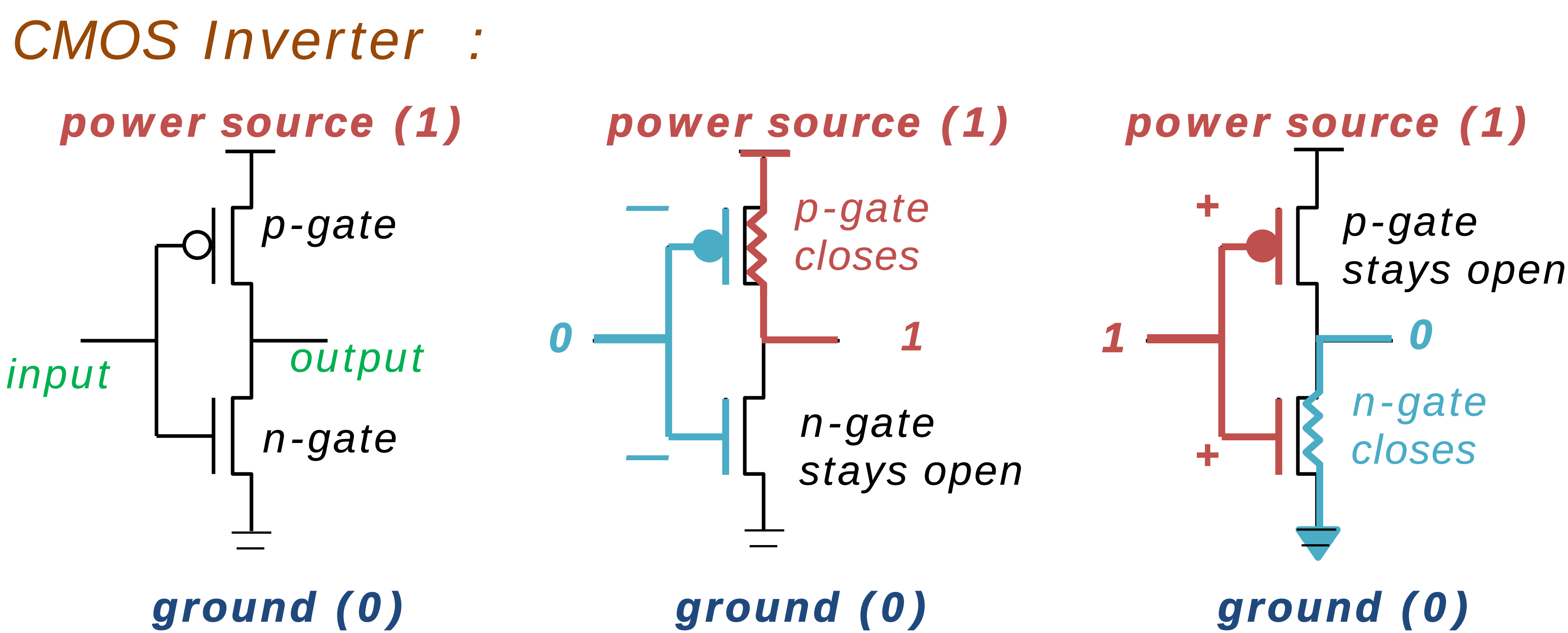


Gate input controls whether current can flow between the other two terminals or not.

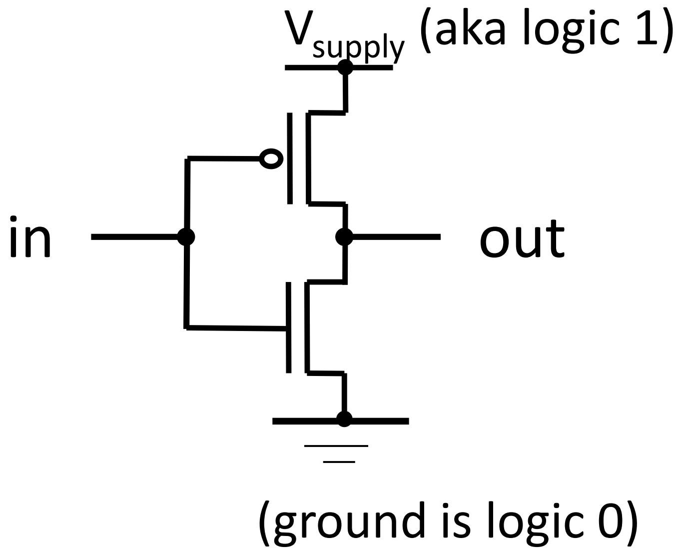
Hint: the “o” bubble of the p-type tells you that this gate wants a 0 to be turned on

2-Transistor Combination: NOT

- Logic gates are constructed by combining transistors in complementary arrangements
- Combine p&n transistors to make a NOT gate:

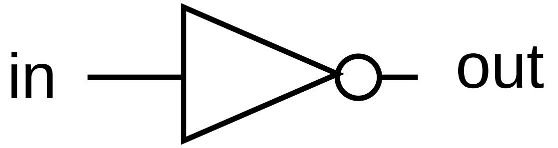


Inverter



Function: NOT

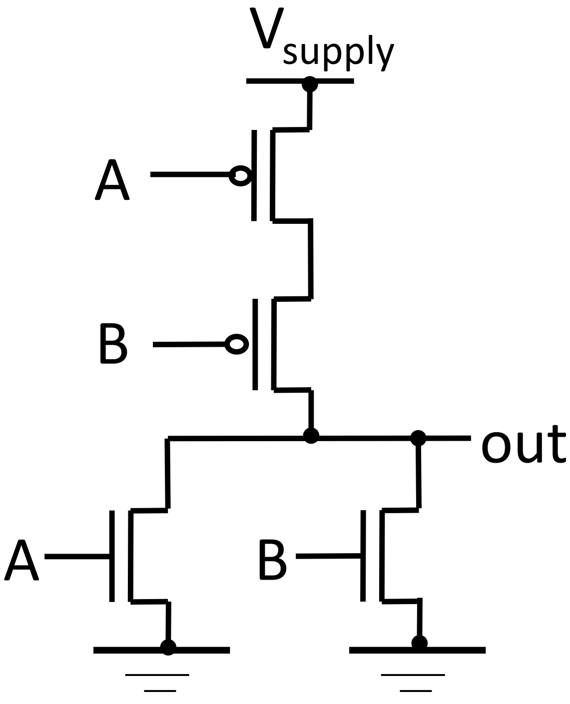
Symbol:



Truth Table:

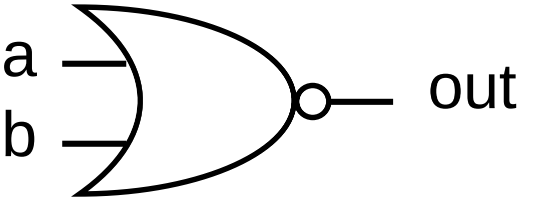
In	Out
0	1
1	0

NOR Gate



Function: NOR

Symbol:

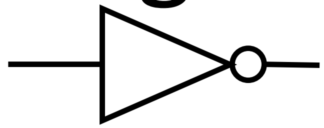
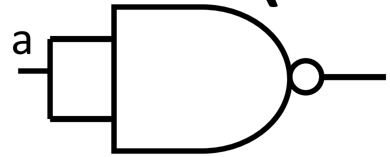


Truth Table:

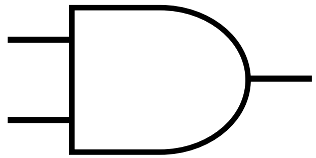
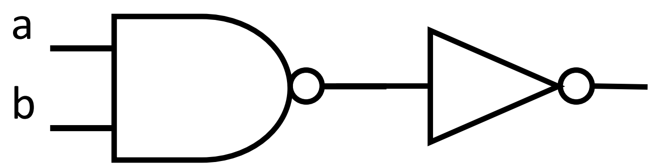
A	B	out
0	0	1
0	1	0
1	0	0
1	1	0

Building Functions (Revisited)

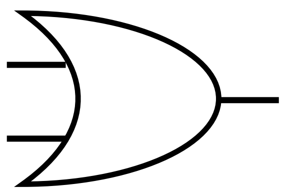
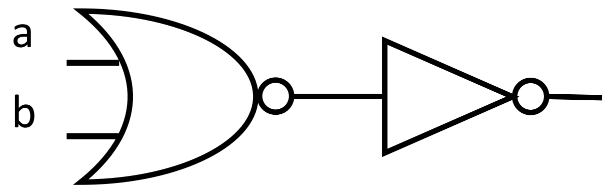
- NOT:

- AND:

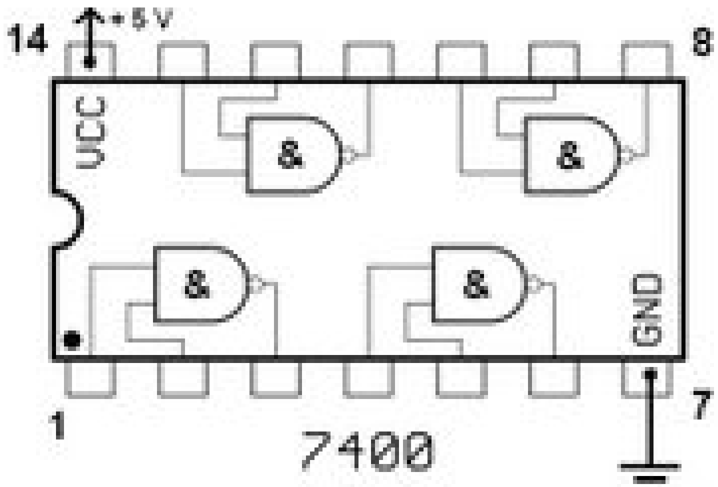



- OR:

- NAND and NOR are universal
  - Can implement *any* function with NAND or just NOR gates
  - useful for manufacturing

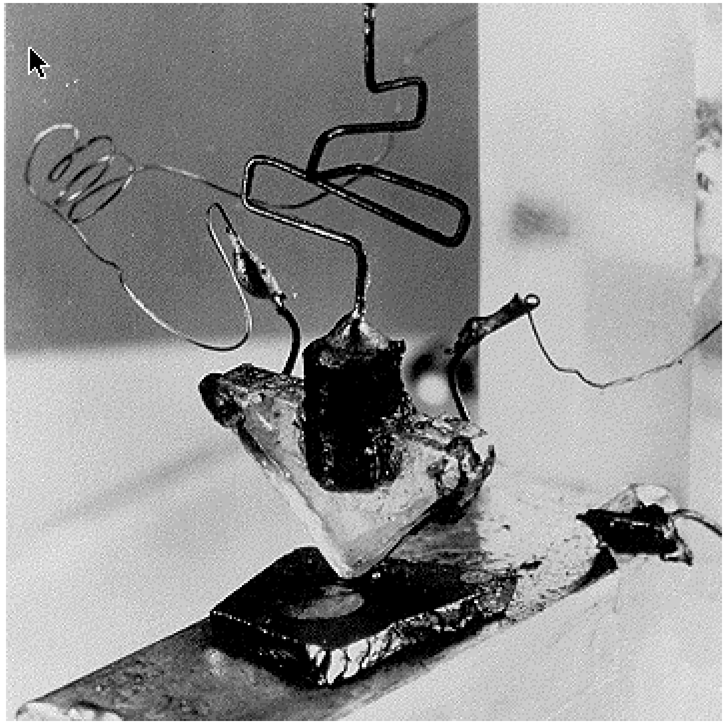
Logic Gates

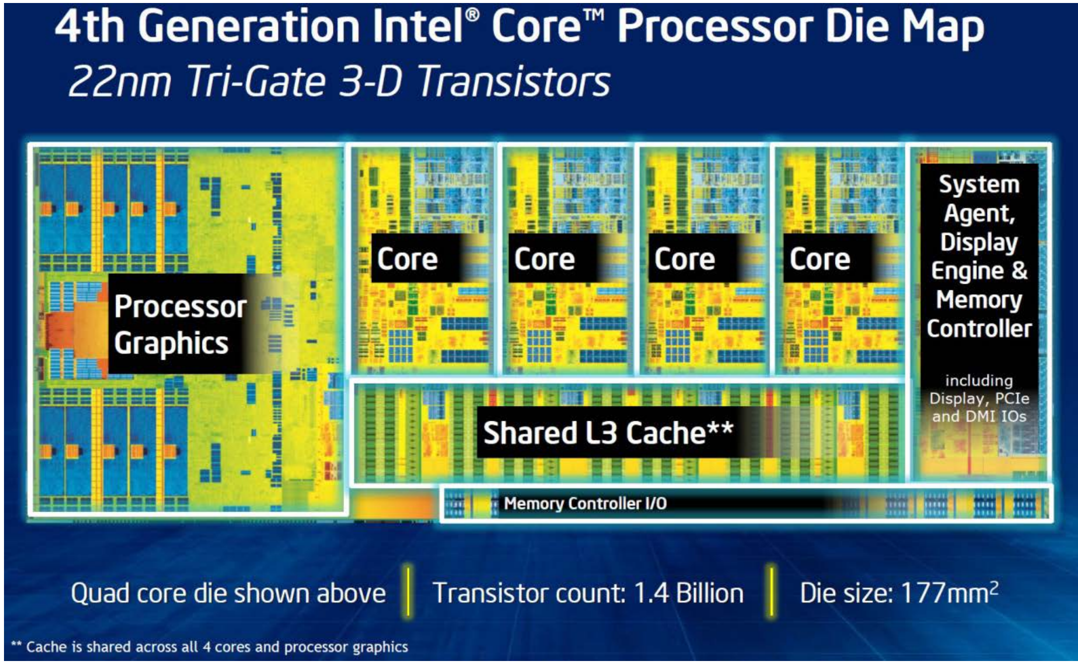


- One can buy gates separately
  - ex. 74xxx series of integrated circuits
  - cost ~$1 per chip, mostly for packaging and testing

- Cumbersome, but possible to build devices using gates put together manually

Then and Now



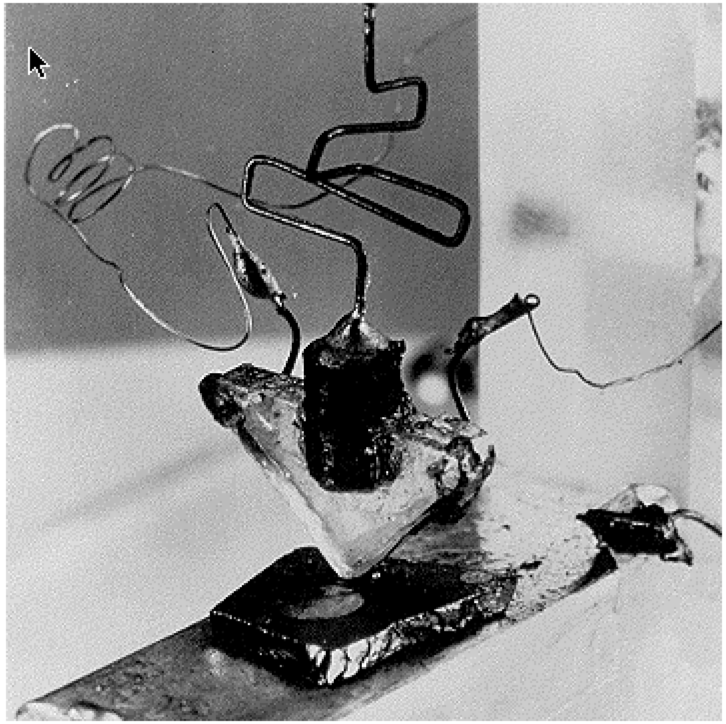


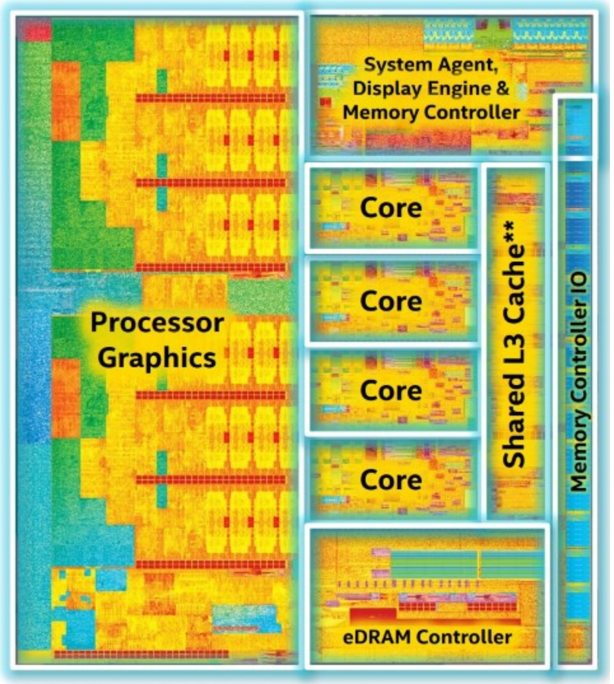
http://techguru3d.com/4th-gen-intel-haswell-processors-architecture-and-lineup/

- The first transistor
  - One workbench at AT&T Bell Labs
  - 1947
  - Bardeen, Brattain, and Shockley
- Intel Haswell
  - 1.4 billion transistors, 22nm
  - 177 square millimeters
  - Four processing cores

https://en.wikipedia.org/wiki/Transistor_count

Then and Now





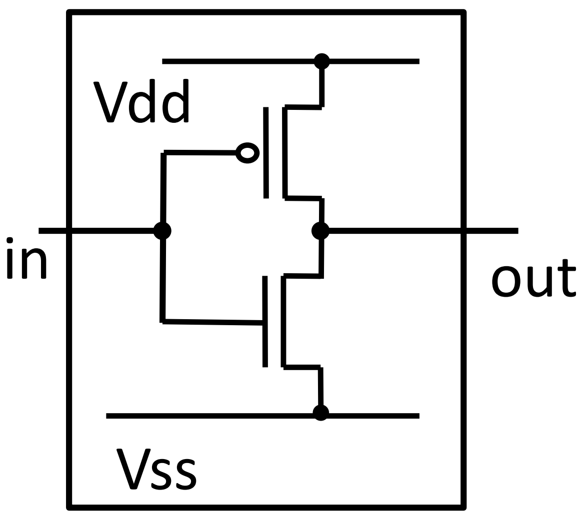
https://www.computershopper.com/computex-2015/performance-preview-desktop-broadwell-at-computex-2015

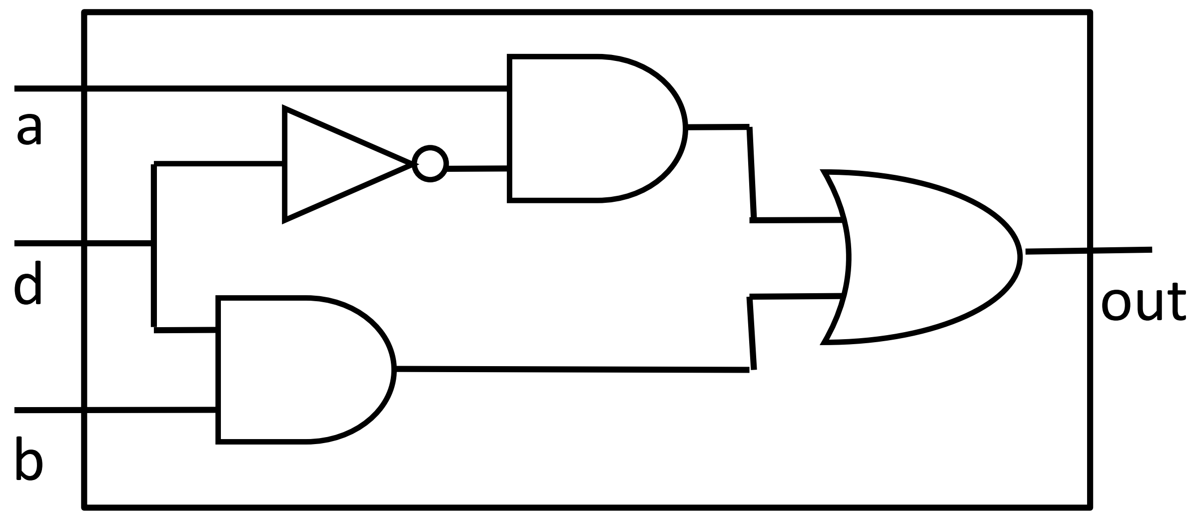
- The first transistor
  - One workbench at AT&T Bell Labs
  - 1947
  - Bardeen, Brattain, and Shockley
- Intel Broadwell
  - 7.2 billion transistors, 14nm
  - 456 square millimeters
  - Up to 22 processing cores

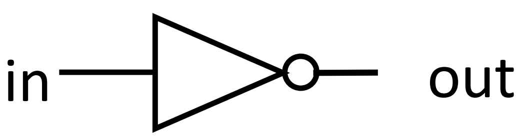
https://en.wikipedia.org/wiki/Transistor_count

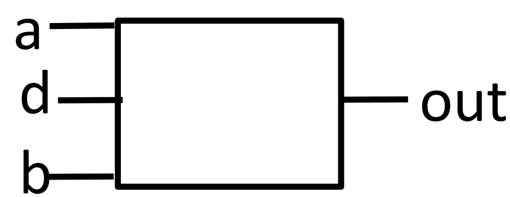
Big Picture: Abstraction

- Hide complexity through simple abstractions
  - Simplicity
    - Box diagram represents inputs and outputs
  - Complexity
    - Hides underlying NMOS- and PMOS-transistors and atomic interactions









Summary

- Most modern devices made of billions of transistors
  - You will build a processor in this course!
  - Modern transistors made from semiconductor materials
  - Transistors used to make logic gates and logic circuits
- We can now implement any logic circuit
  - Use P- & N-transistors to implement NAND/NOR gates
  - Use NAND or NOR gates to implement the logic circuit
  - *Efficiently*: use K-maps to find required minimal terms