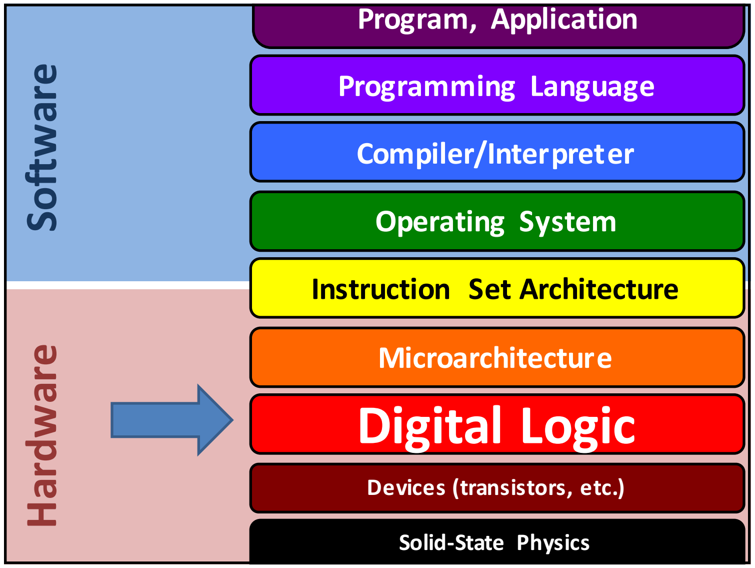
Digital Logic

*Gate*way to computer science


Software
Hardware
Program, Application
Programming Language
Compiler/Interpreter
Operating System
Instruction Set Architecture
Microarchitecture
Digital Logic
Devices (transistors, etc.)
Solid-State Physics

Boolean value (*bit*): 0 or 1

Abstraction!

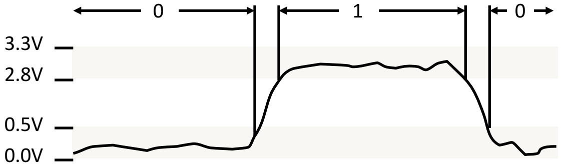
basis of all digital data representations

Boolean functions (AND, OR, NOT, …):

basis of all digital computations

Electronically:

bit = high voltage vs. low voltage

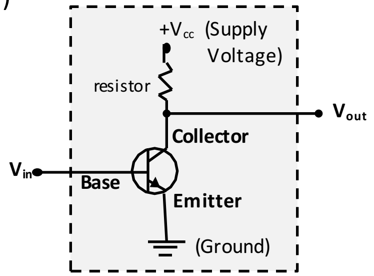


Boolean functions = logic gates, built from transistors

Transistors (more in lab)

If *Base* voltage is high:
Current may flow freely
from *Collector* to *Emitter*.

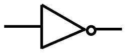
If *Base* voltage is low:
Current may not flow
from *Collector* to *Emitter*.



Truth table

V_{in}	V_{out}		in	out		in	out
low	high	=	0	1	=	F	T
high	low		1	0		T	F

NOT gate



Abstraction!

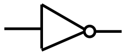
Digital Logic Gates

Abstraction!

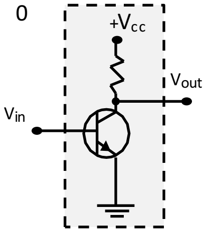
ex

Tiny electronic devices that compute basic Boolean functions.

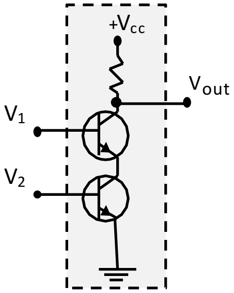
NOT



V_{in}	V_{out}
0	1
1	0

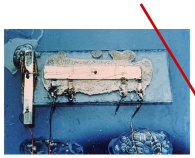


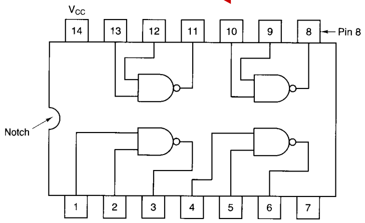
V_1 \ V_2	0	1
0		
1		



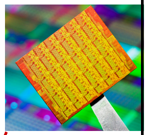
Integrated Circuits (invented 1950s)

Gates are manufactured in units called ***integrated circuits***.
From SSI (tens) to VLSI (hundreds of thousands to billions)

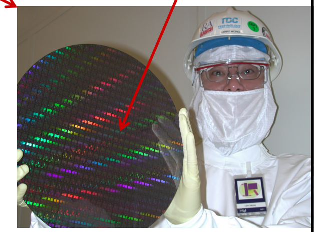




Chip

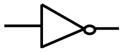


Wafer

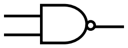


Five basic gates: define with truth tables

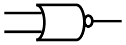
ex



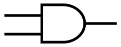
NOT	
0	1
1	0



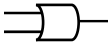
NAND	0	1
0	1	1
1	1	0



NOR	0	1
0		
1		



AND	0	1
0		
1		



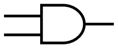
OR	0	1
0		
1		

Boolean Algebra

for combinational logic

inputs	=	*variables*
wires	=	*expressions*
gates	=	*operators/functions*
circuits	=	*functions*

A
B

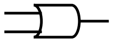


A B
(A · B)

AND = Boolean product

·	**0**	**1**
0	0	0
1	0	1

A
B

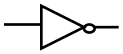


A + B

OR = Boolean sum

+	**0**	**1**
0	0	1
1	1	1

A



$\overline{A}$

NOT = inverse or complement

0	1
1	0

A A

wire = identity

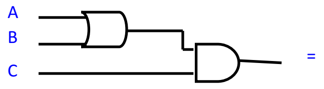
0	0
1	1

Circuits

ex

Connect inputs and outputs of gates with wires.
Crossed wires touch *only if* there is a dot.

What is the output if A=1, B=0, C=1?
What is the truth table of this circuit?
What is an equivalent Boolean expression?



Translation

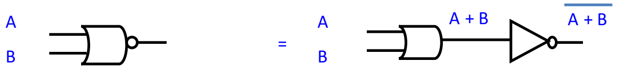
ex

Connect gates to implement these functions. Check with truth tables.
A one-to-one translation is straightforward and bidirectional.

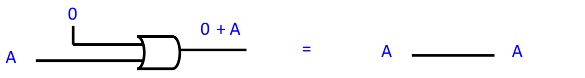
$F = (A\overline{B} + C)D$

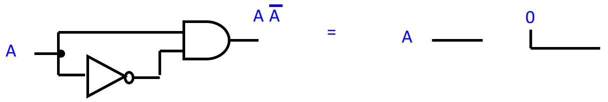
$Z = \overline{W} + (X + \overline{WY})$

Note on notation: bubble = inverse/complement

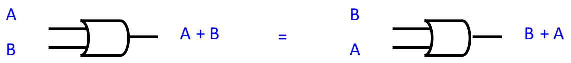


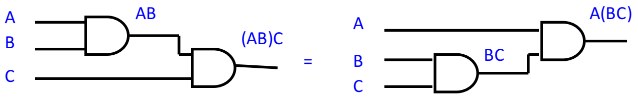
Identity law, inverse law



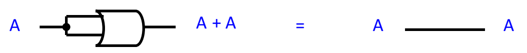


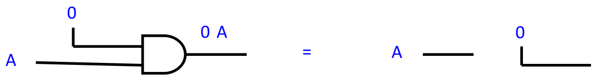
Commutativity, Associativity



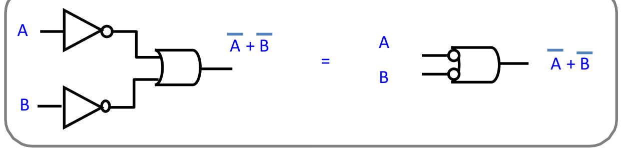


Idempotent law, Null/Zero law



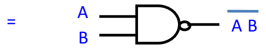


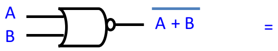
Note on notation: bubble = inverse/complement



DeMorgan's Law

(**double bubble**, toil and trouble, in Randy's words…)

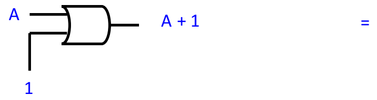


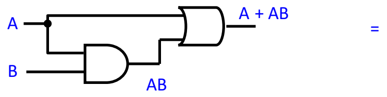


ex

One law, Absorption law

Write truth tables. Do they correspond to simpler circuits?





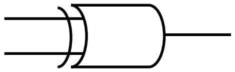
ex

NAND is *universal.*

All Boolean functions can be implemented using only NANDs.
Let's prove it! Build NOT, AND, OR, NOR, using only NAND gates.

XOR: Exclusive OR

ex



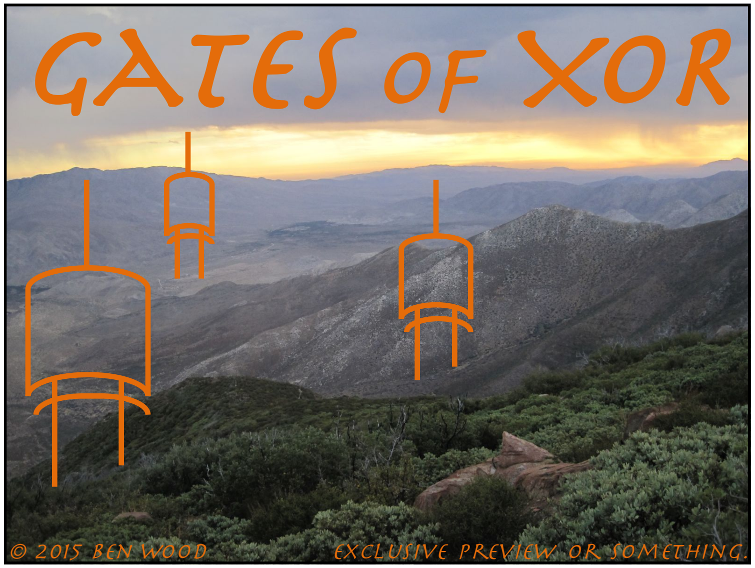
Output = 1 if exactly one input = 1.

Truth table:

Build from earlier gates:

Often used as a one-bit comparator.

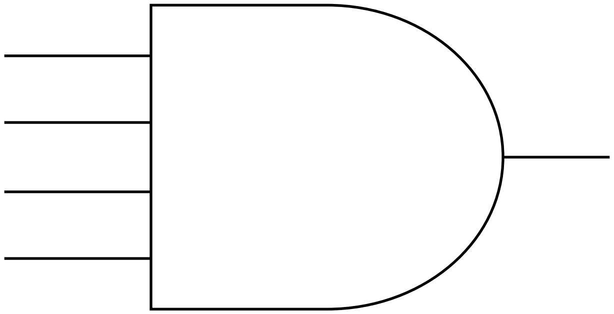
Video game designers, Halloween costumers extraordinaire, sci-fi/fantasy screenwriters, I have an idea…



Larger gates

ex

Build a 4-input AND gate using any number of 2-input gates.

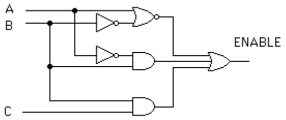


Circuit simplification

Why simplify?
Smaller = cheaper, faster, cooler,
easier to design/build.

ex

Can we find a simpler circuit that performs the same function?



Start with an equivalent Boolean expression, then simplify with algebra.

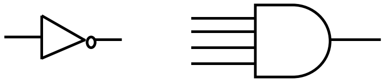
F(A, B, C) =

Check the answer with a truth table.

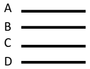
Circuit derivation: ***code detectors***

ex

AND gate + NOT gates = code detector, recognizes exactly one input code.



Design a 4-input code detector to output 1 if ABCD = 1001, and 0 otherwise.



Design a 4-input code detector to accept two codes (ABCD=1001, ABCD=1111) and reject all others. (accept = 1, reject = 0)

Circuit derivation: ***sum-of-products*** form

ex

logical sum (OR)

of products (AND)

of inputs or their complements (NOT)

Draw the truth table and **design a sum-of-products circuit** for a 4-input code detector to accept two codes (ABCD=1001, ABCD=1111) and reject all others.
How are the truth table and the sum-of-products circuit related?

Voting machines

ex

A **majority circuit** outputs 1 if and only if a majority of its inputs equal 1.
Design a majority circuit for three inputs. **Use a sum of products.**

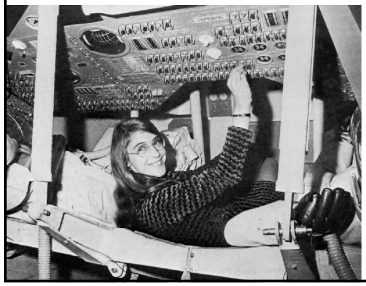
A	B	C	Majority
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Triply redundant computers in spacecraft

- Compute everything on three separate devices and compare answers. Majority wins.
- Cosmic rays can flip bits on wires, especially in space.
- Triple redundancy survives failure in one device.
- Space program also hastened Integrated Circuits.

Margaret Hamilton *(speaking of space and reliability)*

Led software team for **Apollo 11** Guidance Computer.
Developed software engineering techniques for correctness and reliability.
Coined "software engineering".
Software avoided mission abort on first moon landing!



Apollo 11 code print-out