CS 240
Foundations of Computer Systems

WELLESLEY

Digital Logic

*Gate*way to computer science

transistors, gates, circuits, Boolean algebra

Software

- Program, Application
- Programming Language
- Compiler/Interpreter
- Operating System
- Instruction Set Architecture

Hardware

- Microarchitecture
- **Digital Logic**
- Devices (transistors, etc.)
- Solid-State Physics

Digital data/computation = Boolean

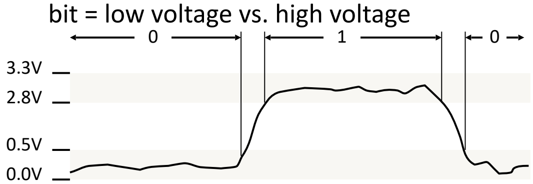
Abstraction!

Boolean value (***bit***): **0 or 1**

Boolean functions (AND, OR, NOT, …)

Electronically:

bit = low voltage vs. high voltage

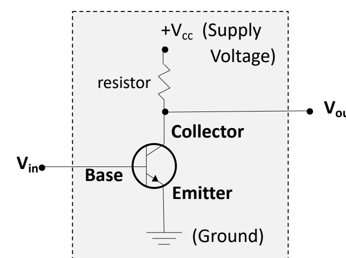


Boolean functions = logic gates, built from transistors

Transistors

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

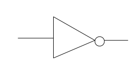
If *Base* voltage is low:
Current may not flow from *Collector* to *Emitter*.



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!

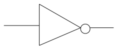
Digital Logic Gates

Abstraction!

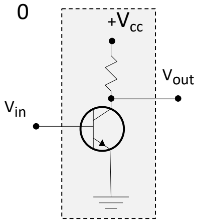
ex

Tiny electronic devices that compute basic Boolean functions.

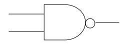
NOT



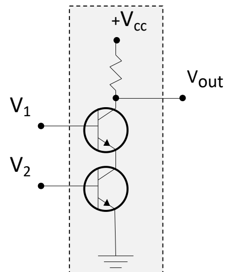
V_{in}	V_{out}
0	1
1	0



NAND

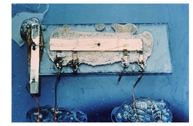


V_1 \ V_2	0	1
0		
1		

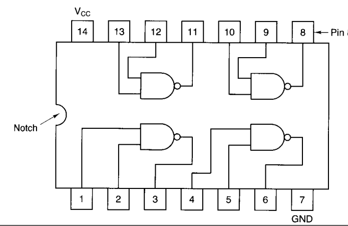


Integrated Circuits (1950s -)

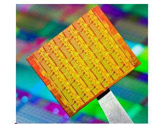
Early (first?) **transistor**



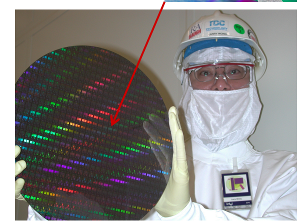
Small integrated circuit



Chip

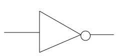


Wafer



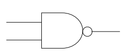
Five basic gates: define with truth tables

ex

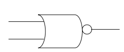


NOT

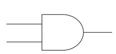
0	1
1	0



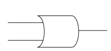
NAND	0	1
0	1	1
1	1	0



NOR	0	1
0		
1		



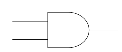
AND	0	1
0		
1		



OR	0	1
0		
1		

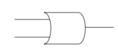
Boolean Algebra

for combinational logic



AND = Boolean product

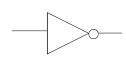
·	0	1
0	0	0
1	0	1



OR = Boolean sum

+	0	1
0	0	1
1	1	1

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



NOT = inverse or complement

0	1
1	0

A ______ A

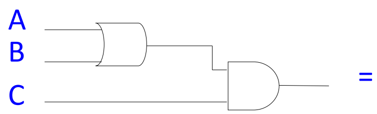
wire = identity

0	0
1	1

A **boolean literal** is a variable or its complement.

E.g., A, A’, B, are literals, but A + B, AB, and (AB)’ are not.

Circuits

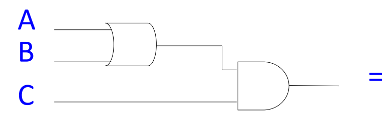


Given input variables, **circuits** specify outputs as functions of inputs using wires & gates.

- Crossed wires touch *only if* there is an explicit dot.
- T intersections copy the value on a wire
- Don't wire together two inputs or two outputs—combine with a gate

Circuits

ex



What is the truth table for Q in the example circuit?

A	B	C	Q
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

What is the output if A=1, B=0, C=1?

What is an equivalent Boolean expression?

Translation

ex

Connect gates to implement these functions. Check with truth tables.

Use a direct translation.

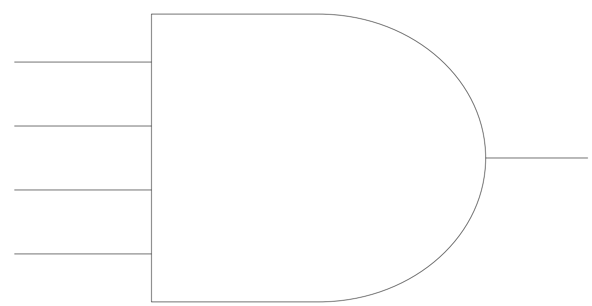
$F = (A\overline{B} + C)D$

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

Larger gates

ex

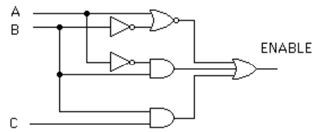
Build a 4-input AND gate using any number of 2-input gates.



Circuit simplification

ex

Is there a simpler circuit that performs the same function?



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

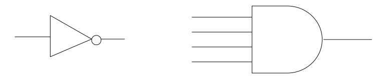
F(A, B, C) =

Check the answer with a truth table.

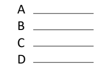
Circuit derivation: ***code detectors***

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)

S*um-of-products* form

ex

logical sum (OR)

of products (AND)

of inputs or their negations (NOT)

A **minterm:** a product of literals (variables or their negations) whose value is 1 for that row. Like a **code detector** for that row!

What is the sum-of-products expression for truth table below?

A	B	C	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

How could you draw the circuit for this expression?

How is it related to **code detectors** from the previous slide?

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

- https://en.wikipedia.org/wiki/Triple_modular_redundancy
- Space program hastened Integrated Circuits.

Boolean Algebra: Simple laws

Boolean algebra laws can be proven by truth tables and used to show equivalences.

For all laws in one place, see the Boolean Laws Reference Sheet

Name of Law / Theorem	Form	Equivalent/Dual form (interchange AND and OR, and 0 and 1)
Involution (or double negation)	$\overline{\overline{A}} = A$	none
Identity	$0 + A = A$	$1 * A = A$
Inverse (or **Complements**)	$A\overline{A} = 0$	$A + \overline{A} = 1$
Commutativity	$A + B = B + A$	$AB = BA$
Associativity	$(AB)C = A(BC)$	$(A+B)+C = A+(B+C)$
Idempotent	$A + A = A$	$AA = A$
Null (or **Null Element**)	$0 * A = 0$ (**the Zero Law**)	$1 + A = 1$ (**the One Law**)

Boolean Algebra: More Complex Laws

Distributive	$A + BC = (A+B)(A+C)$	$A(B+C) = AB + AC$
DeMorgan's	$\overline{A} + \overline{B} + \overline{C} + \ldots = \overline{ABC\ldots}$	$\overline{A + B + C + \ldots} = \overline{A}\ \overline{B}\ \overline{C}\ldots$
Absorption 1 (Covering)	$A + AB = A$	$A(A+B) = A$
Absorption 2	$A + \overline{A}B = A + B$	$A(\overline{A} + B) = AB$
Combining	$AB + A\overline{B} = A$	$(A+B)(A+\overline{B}) = A$
Consensus	$AB + \overline{A}C + BC = AB + \overline{A}C$	$(A+B)(\overline{A}+C)(B+C) = (A+B)(\overline{A}+C)$

You can use truth tables (or other Boolean laws) to convince yourself that these laws hold.

Boolean Algebra: Proving Laws by Truth Tables

ex

Distributive	$A + BC = (A+B)(A+C)$	$A(B+C) = AB + AC$

Right column (distributing multiplication over addition) true in the algebra of numbers;
left column (distributing addition over multiplication) is not true in the algebra of numbers but is true for Boolean algebra!

Complete the truth tables below to show that both distributive laws hold.

A	B	C	A+BC
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

A	B	C	(A+B)(A+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	

A	B	C	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	

A	B	C	AB + AC
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Boolean Algebra: Proving Laws by Algebra

Absorption 1 (Covering)	$A + AB = A$	$A(A+B) = A$

Each step has a **redex** (subexpression which is changed).

Redex and applied law in the same color here.

Step-by-step derivation

explicit Commutativity

A + A*B
= 1*A + A*B [Identity (*)]
= A*1 + A*B [Commutativity (*)]
= A*(1 + B) [Distributivity (*/+)]
= A*1 [One law]
= 1*A [Commutativity (*)]
= A [Identity]

implicit Commutativity

A + A*B
= A*1 + A*B [Identity]
= A*(1 + B) [Distributivity]
= A*1 [One law]
= A [Identity]

***Dual** step-by-step derivation*
*(Swap * ⇔ +, 0 ⇔ 1)*

A*(A + B)
= (0 + A)*(A + B) [Identity (+)]
= (A + 0)*(A + B) [Commutativity (+)]
= A + 0*B [Distributivity (+/*)]
= A + 0 [Zero law]
= 0 + A [Commutativity (+)]
= A [Identity]

A*(A + B)
= (A + 0)*(A + B) [Identity (+)]
= A + 0*B [Distributivity (+/*)]
= A + 0 [Zero law]
= A [Identity]

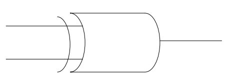
Boolean Algebra: Proving Laws by Algebra

Absorption 2	$A + \overline{A}B = A + B$	$A(\overline{A} + B) = AB$

Boolean Algebra: Proving Laws by Algebra

Combining	$AB + A\overline{B} = A$	$(A + B)(A + \overline{B}) = A$

XOR: Exclusive OR



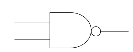
Output = 1 if exactly one input = 1.

Truth table:

Build from earlier gates:

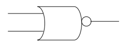
Often used as a one-bit comparator.

NAND is *universal*



All Boolean functions can be implemented using only NANDs.
Build NOT, AND, OR, NOR, using only NAND gates.

NOR is also *universal*



ex

All Boolean functions can also be implemented using only NORs.
Build NAND using only NOR gates; then since NAND is universal,
NOR must be too! (Why?)

Computers

- Manual calculations
- powered all early US **space** missions.
- Facilitated transition to digital computers.

Mary Jackson

Katherine Johnson

- Supported Mercury, Apollo, Space Shuttle, ...

Dorothy Vaughn

- First black supervisor within NACA
- Early self-taught FORTRAN programmer for NASA move to digital computers.

Early pioneers in reliable computing

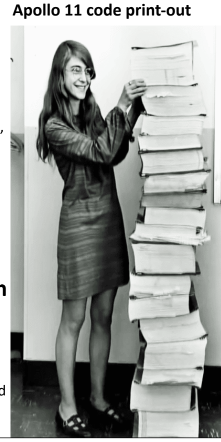
Katherine Johnson

- Calculated first US human space flight trajectories
- Mercury, Apollo 11, Space Shuttle, ...
- Reputation for accuracy in manual calculations, verified early code
- Called to verify results of code for launch calculations for first US human in orbit
- Backup calculations helped save Apollo 13
- Presidential Medal of Freedom 2015

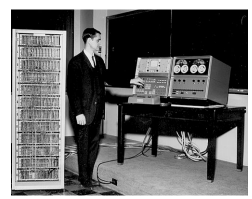
Margaret Hamilton

- Led software team for Apollo 11 Guidance Computer, averted mission abort on first moon landing.
- Coined "software engineering", developed techniques for correctness and reliability.
- Presidential Medal of Freedom 2016

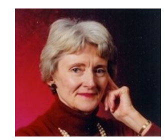
Apollo 11 code print-out



Wellesley Connection: Mary Allen Wilkes '59



Created LAP operating system at MIT Lincoln Labs for Wesley A. Clark's LINC computer, widely regarded as the first personal computer (designed for interactive use in bio labs). Work done 1961—1965.



Created first interactive keyboard-based text editor on 256 character display. LINC had only 2K 12-bit words; (parts of) editor code fit in 1K section; document in other 1K.

In 1965, she developed LAP6 with LINC in Baltimore living room. First home PC user!

Early versions of LAP developed using LINC simulator on MIT TX2 compute, famous for GUI/PL work done by Ivan and Bert Sutherland at MIT.

Later earned Harvard law degree and headed Economic Crime and Consumer Protection Division in Middlesex (MA) County District Attorney's office.