



CS 240

Foundations of Computer Systems



Digital Logic

Gateway to computer science

transistors, gates, circuits, Boolean algebra

Software

Program, Application

Programming Language

Compiler/Interpreter

Operating System

Instruction Set Architecture

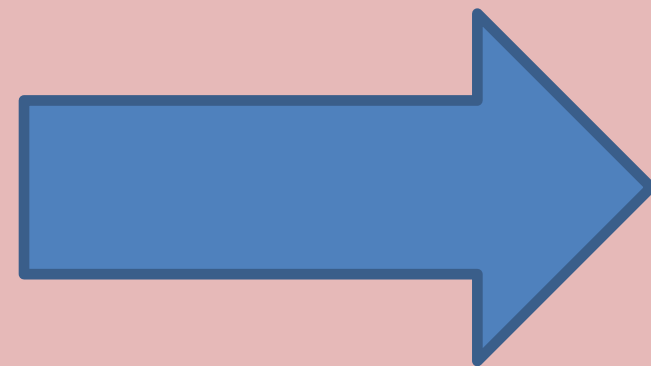
Microarchitecture

Digital Logic

Devices (transistors, etc.)

Solid-State Physics

Hardware

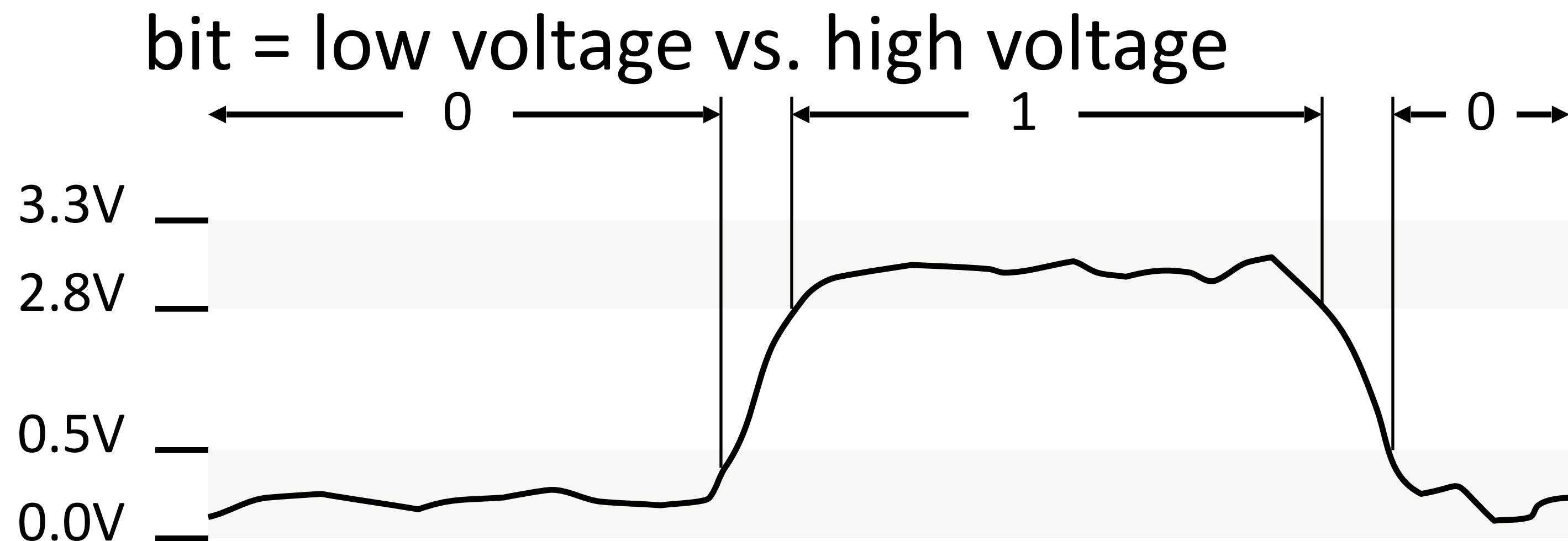


Digital data/computation = Boolean

Boolean value (*bit*): 0 or 1

Boolean functions (AND, OR, NOT, ...)

Electronically:



Abstraction!

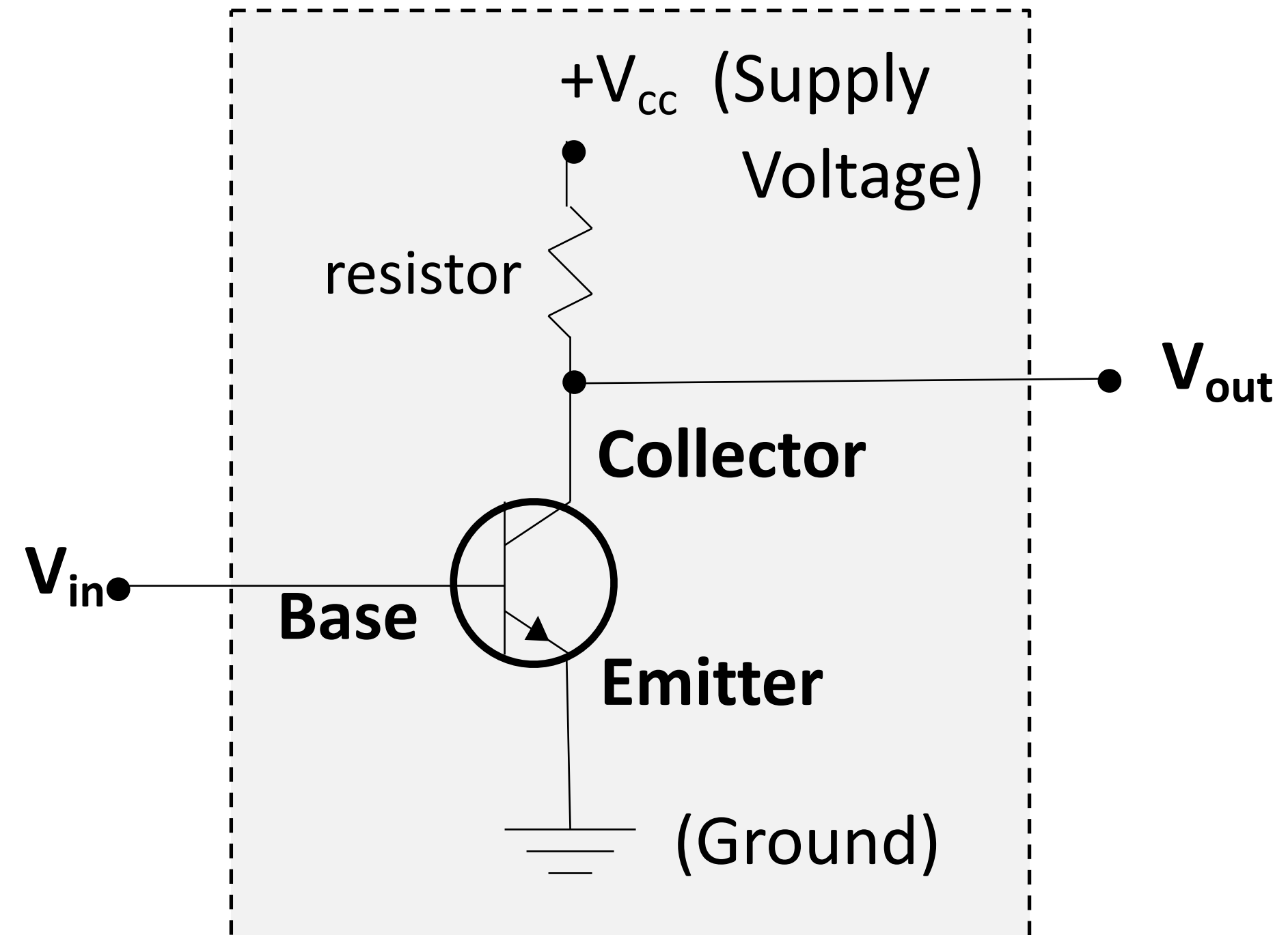


Boolean functions = logic gates, built from transistors

Transistors

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

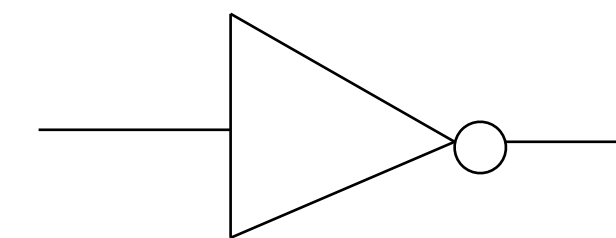
If **Base voltage is low**:
Current may not flow
from *Collector* to *Emitter*.



Truth table

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

NOT gate



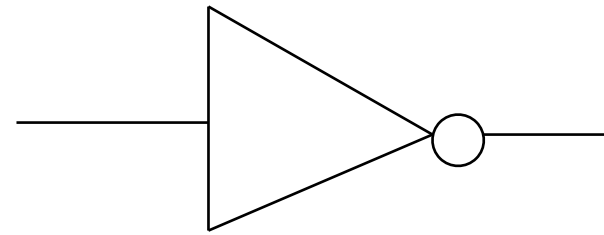
Abstraction!

Digital Logic Gates

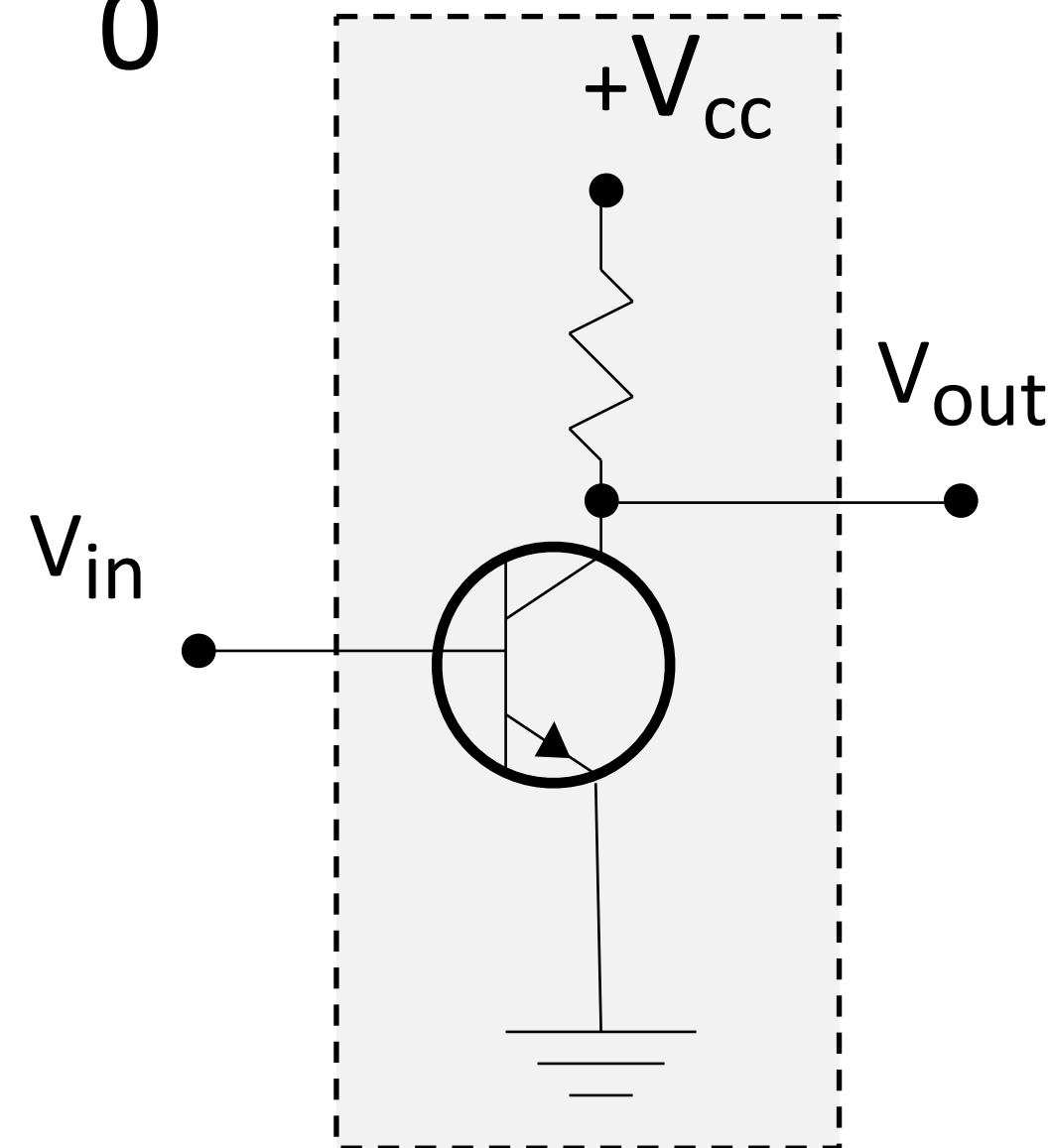
ex

Tiny electronic devices that compute basic Boolean functions.

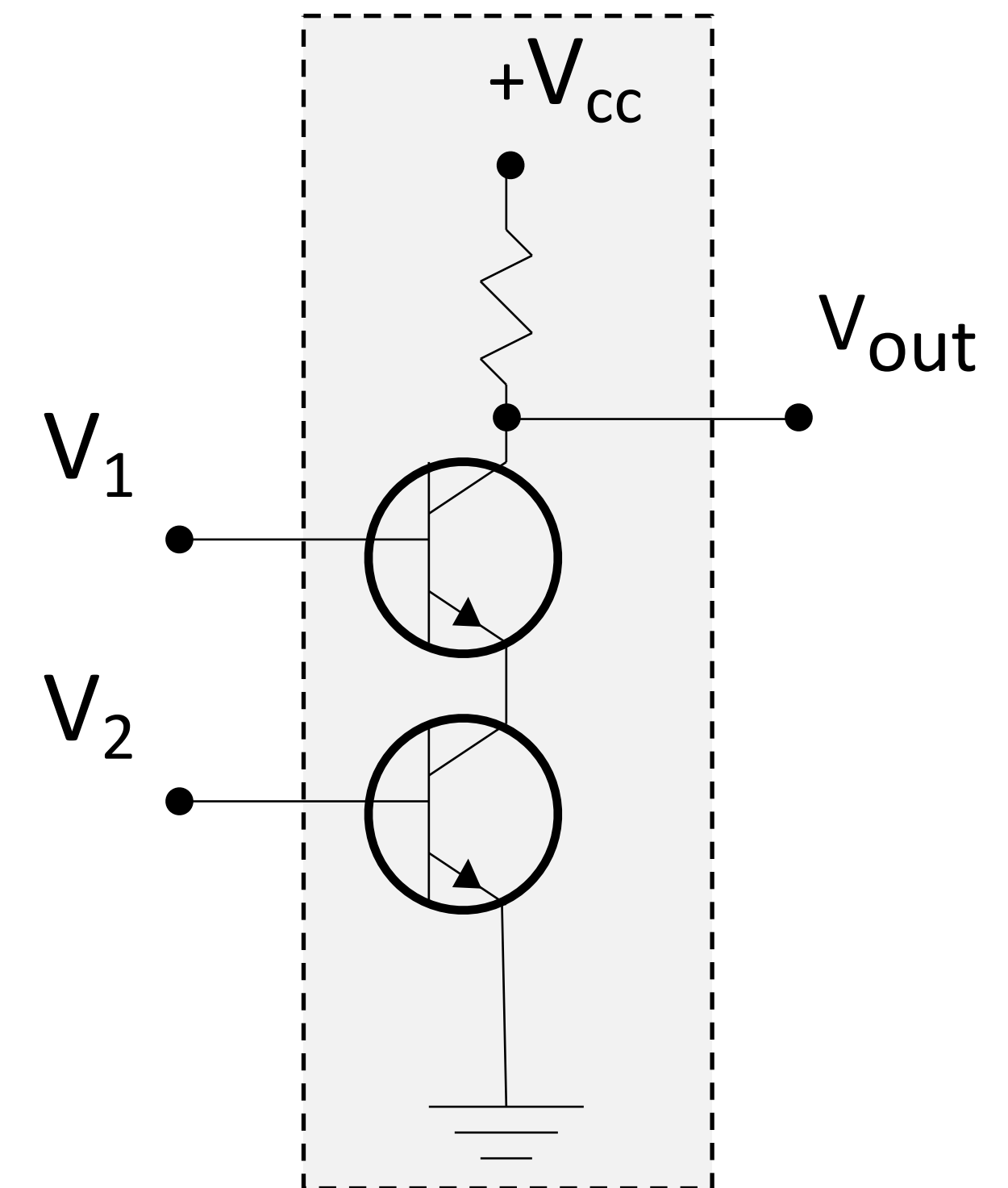
NOT



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

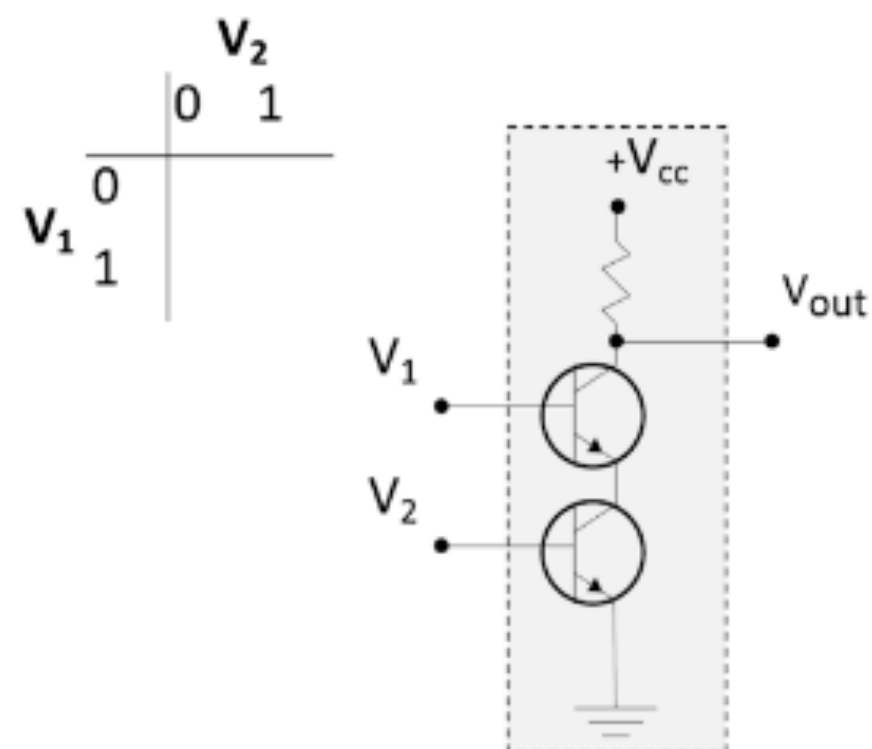
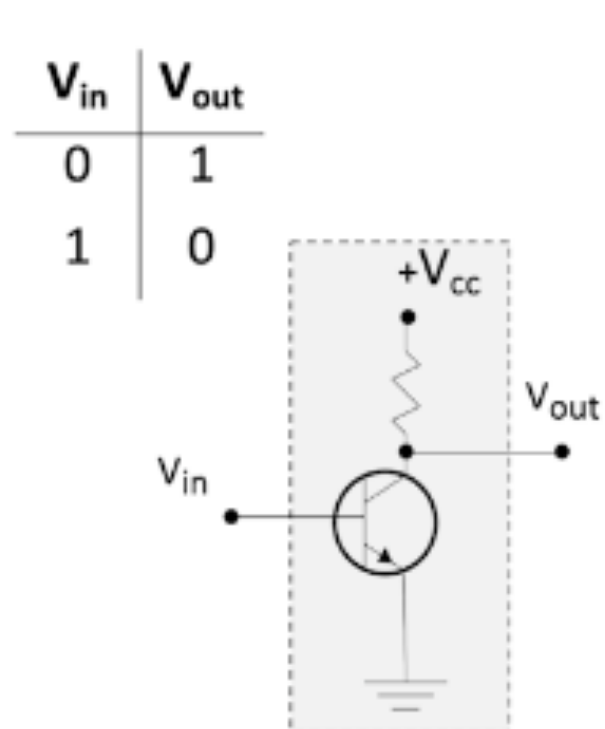


	V_2	
	0	1
V_1	0	
	1	



Which is the correct truth table for the diagram on the right? (It's ok to make an educated guess!)

0



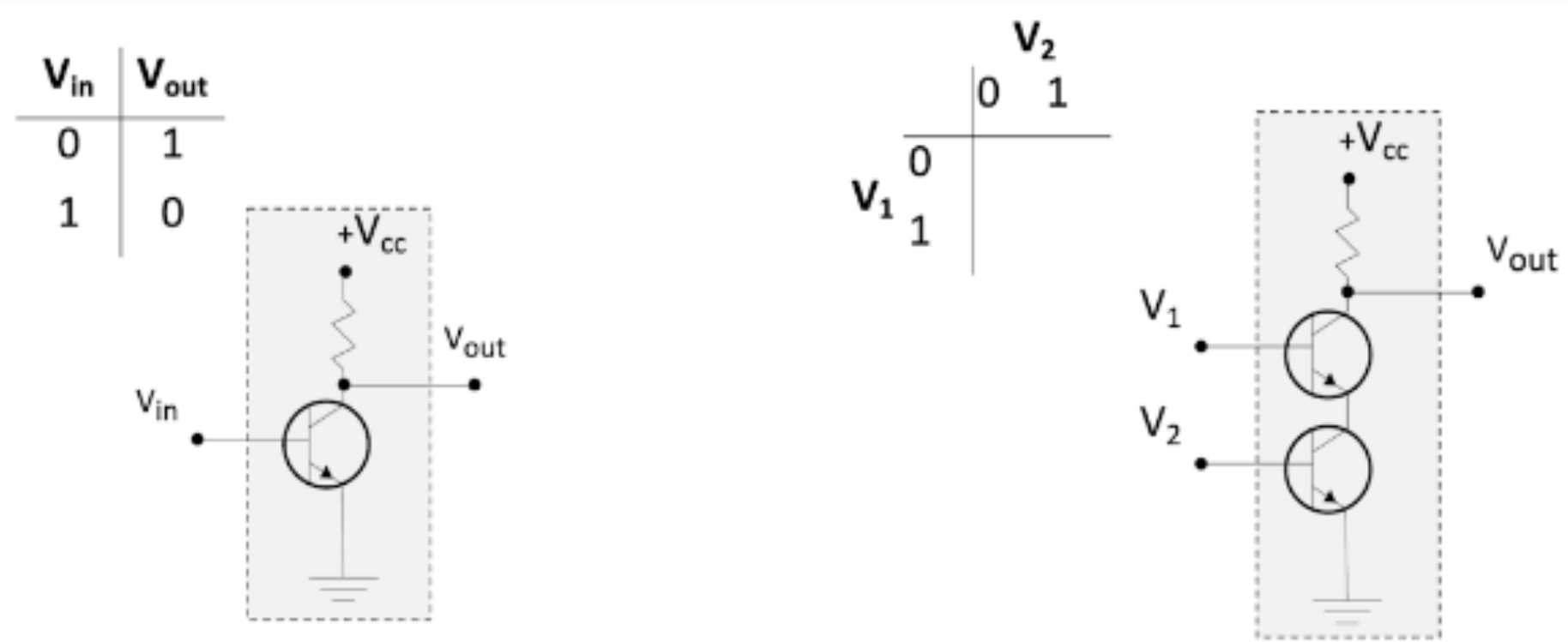
	0	1
0	0	1
1	1	0

	0	1
0	1	0
1	0	0

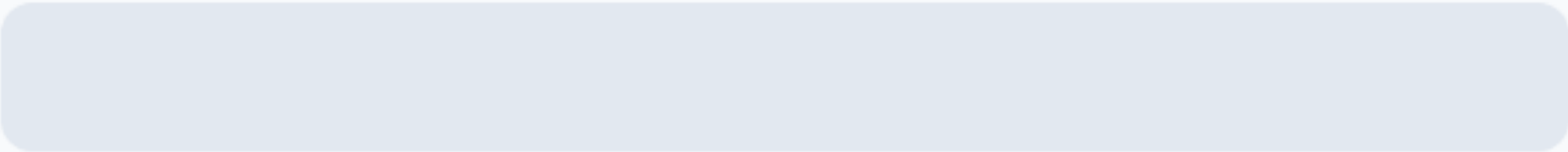
	0	1
0	1	1
1	1	0

Which is the correct truth table for the diagram on the right? (It's ok to make an educated guess!)

0

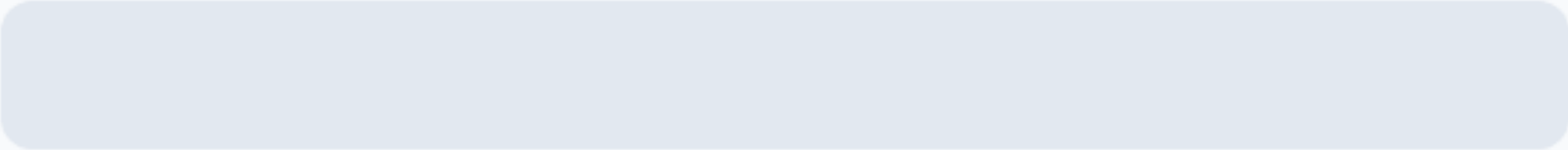


	0	1
0	0	1
1	1	0



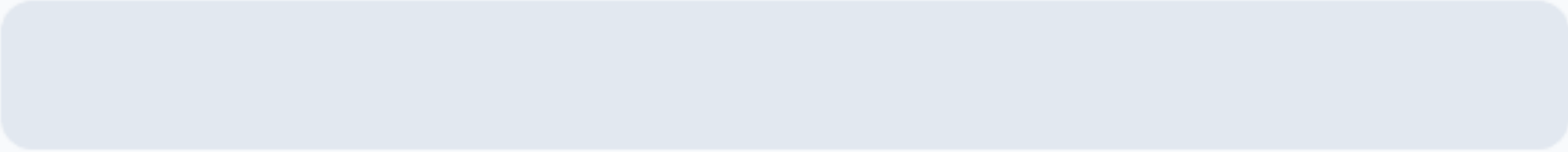
0%

	0	1
0	1	0
1	0	0



0%

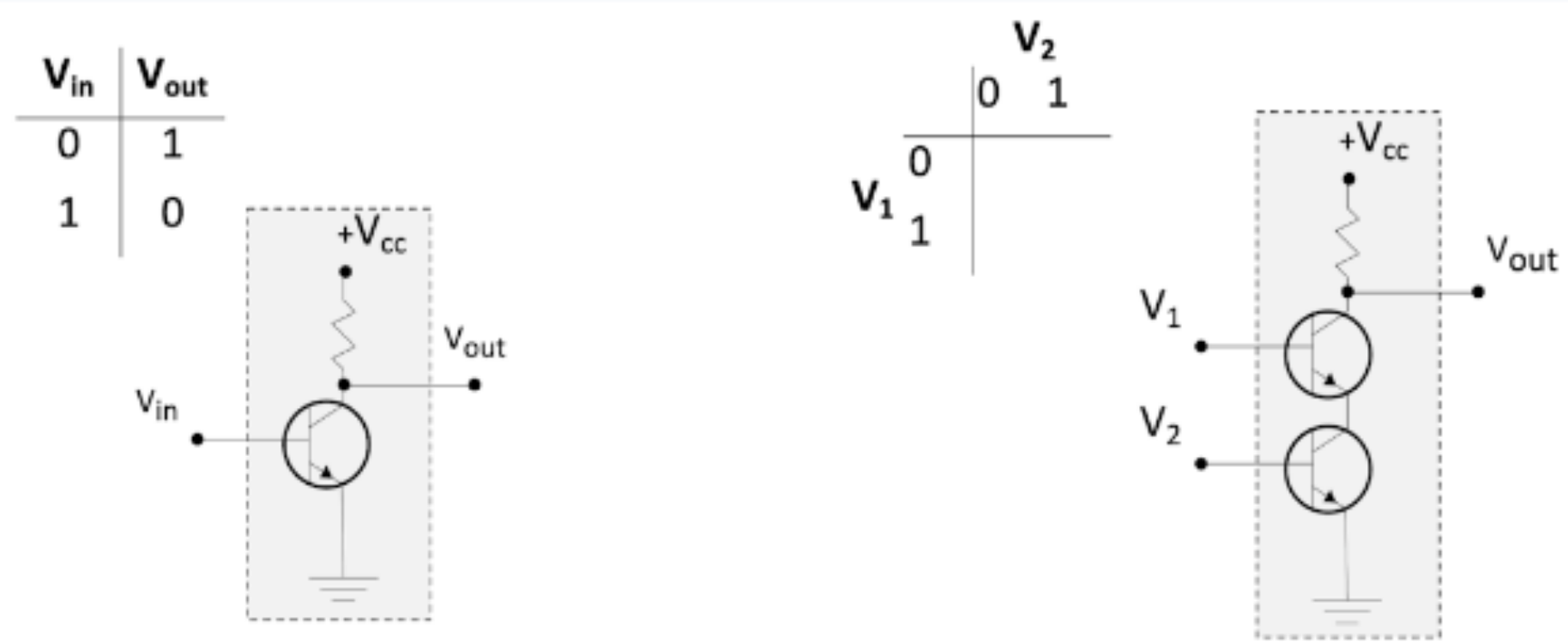
	0	1
0	1	1
1	1	0



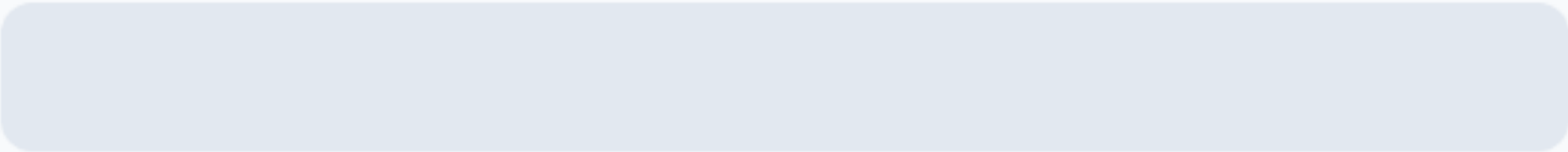
0%

Which is the correct truth table for the diagram on the right? (It's ok to make an educated guess!)

0

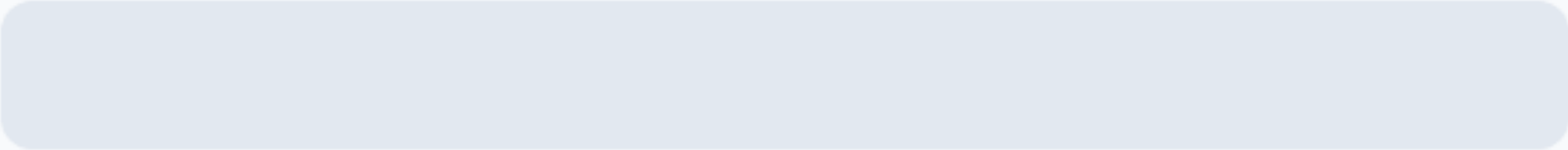


	0	1
0	0	1
1	1	0



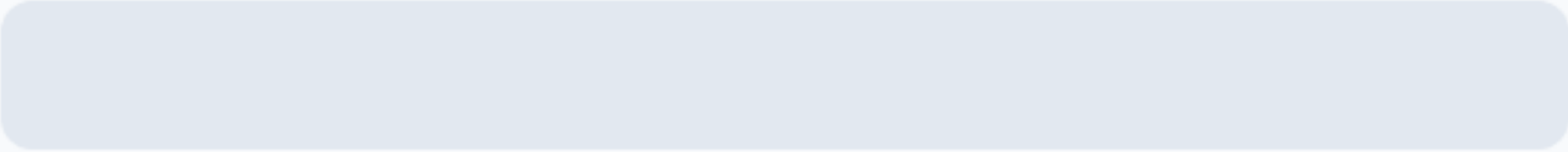
0%

	0	1
0	1	0
1	0	0



0%

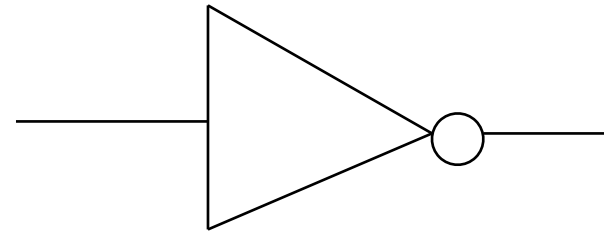
	0	1
0	1	1
1	1	0



0%

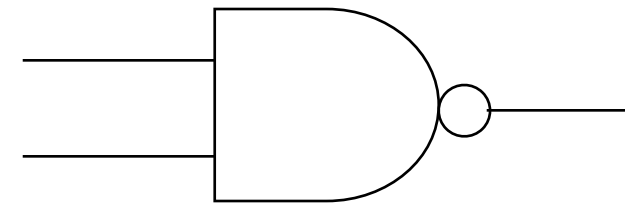
Five basic gates: define with truth tables

ex



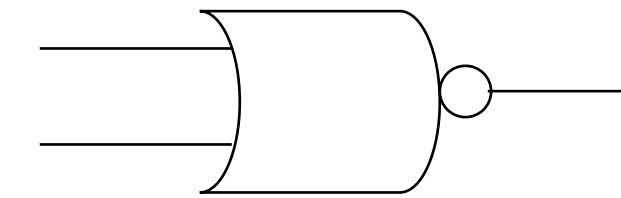
NOT

	0	1
0	1	0
1	0	1



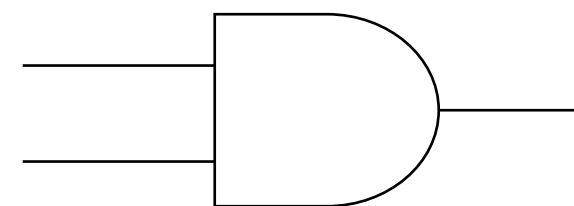
NAND

	0	1
0	1	1
1	1	0



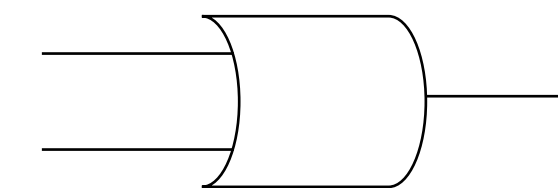
NOR

	0	1
0		
1		



AND

	0	1
0		
1		

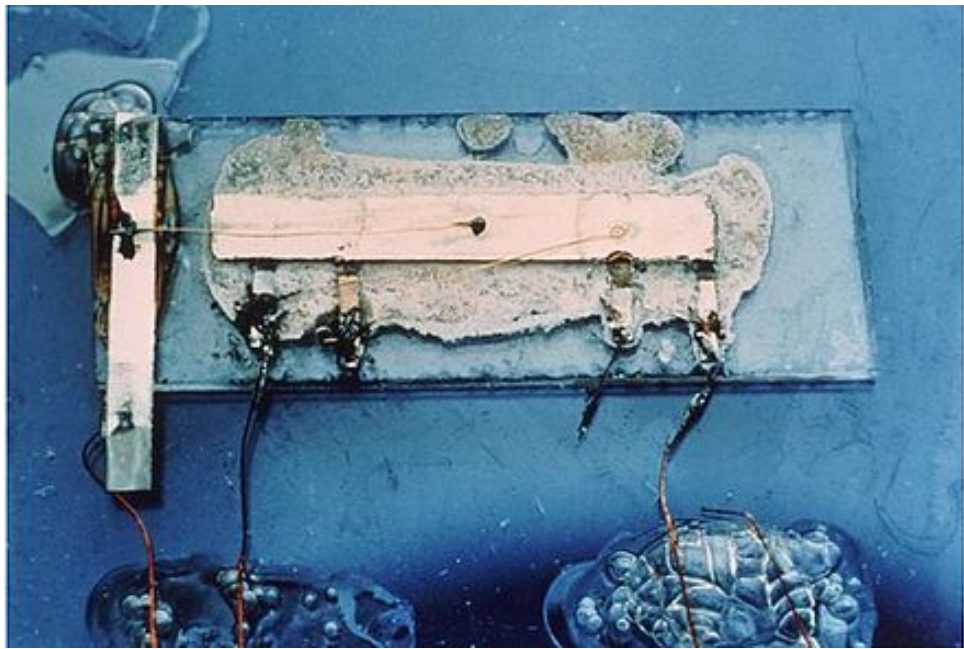


OR

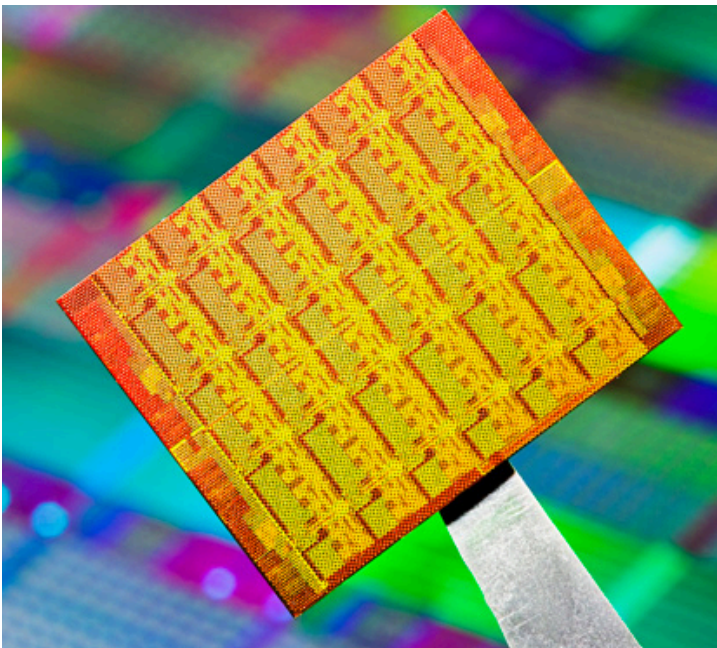
	0	1
0		
1		

Integrated Circuits (1950s -)

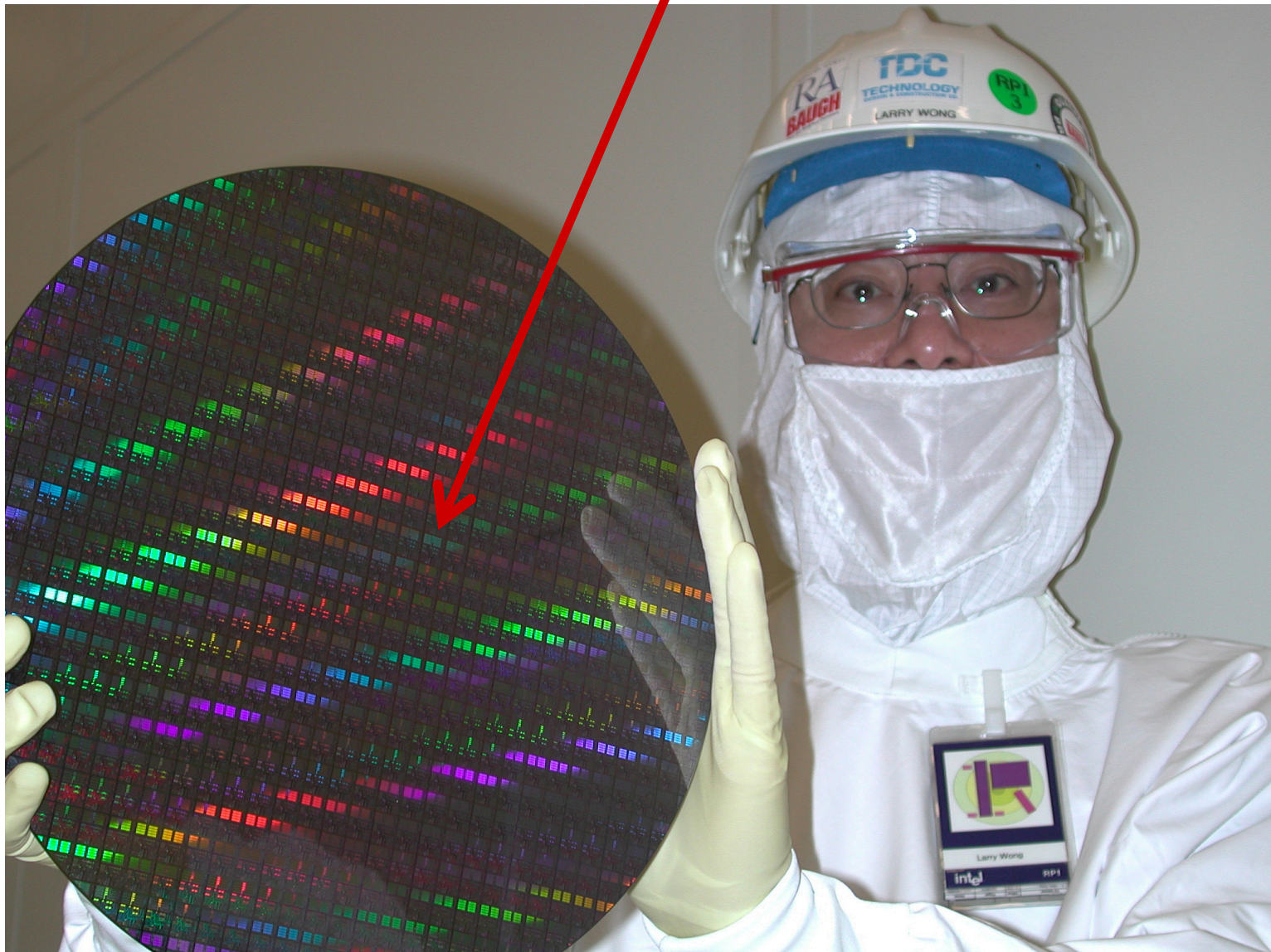
Early (first?) transistor



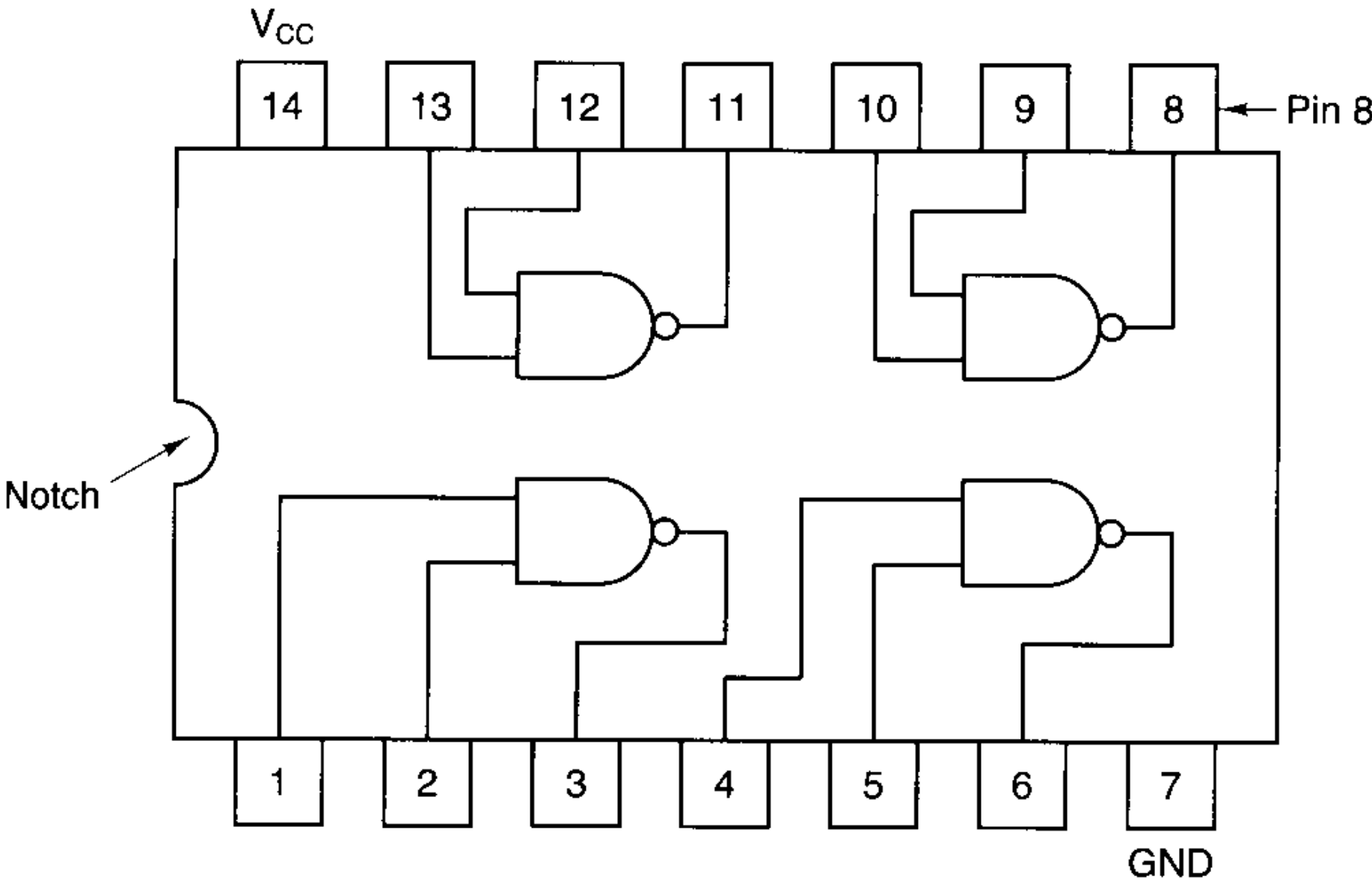
Chip



Wafer

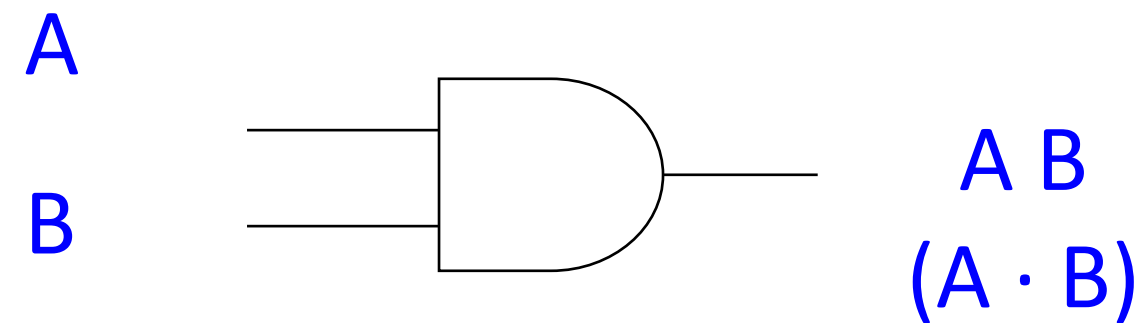


Small integrated circuit



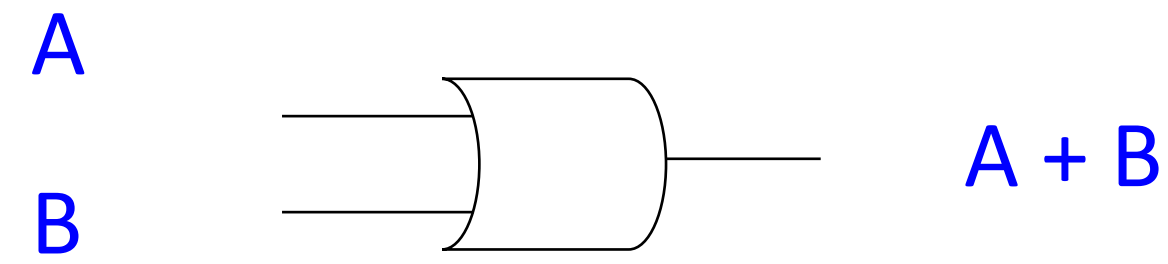
Boolean Algebra

for combinational logic



AND = Boolean product

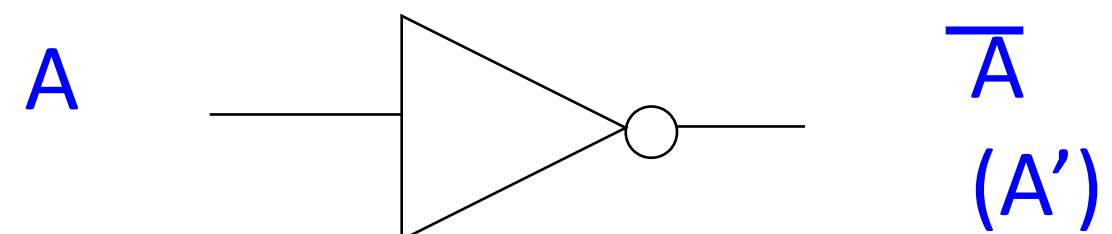
\cdot	0	1
0	0	0
1	0	1



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
0	1	0
1	0	1



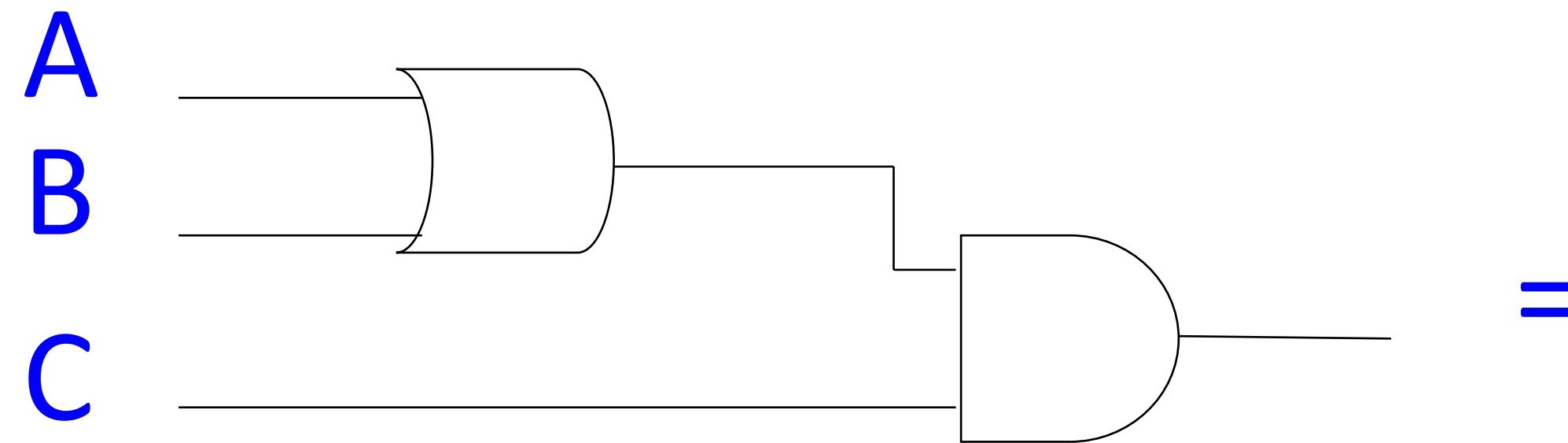
wire = identity

	0	1
0	0	0
1	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

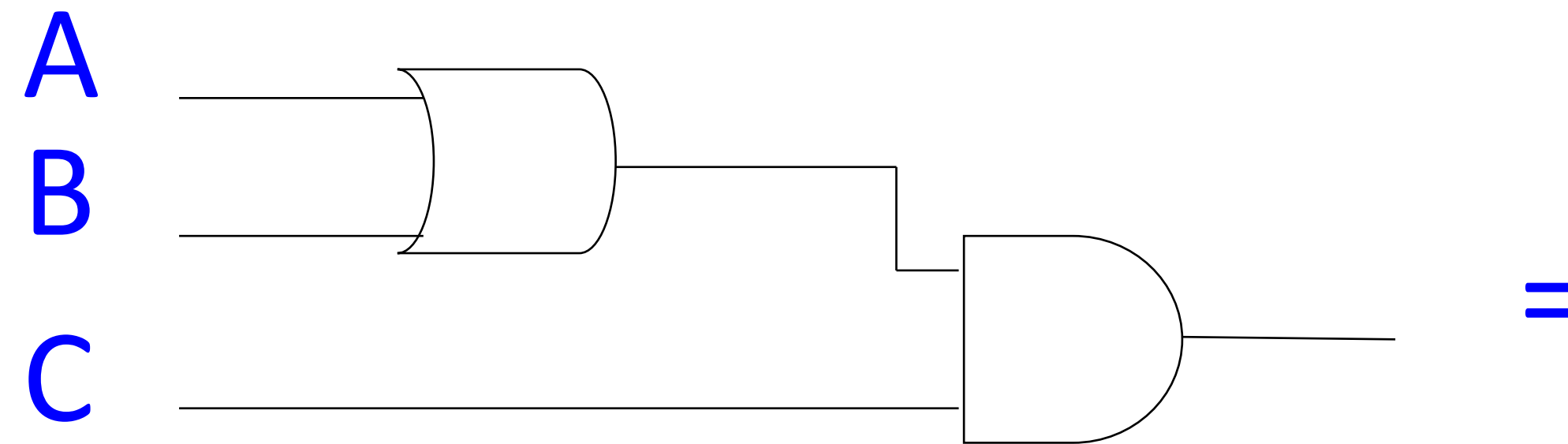


Given input variables, **circuits** specify outputs as functions of inputs using wires & gates.

- 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?

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

What is an equivalent Boolean expression?

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	

Translation

ex

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

Use a direct translation.

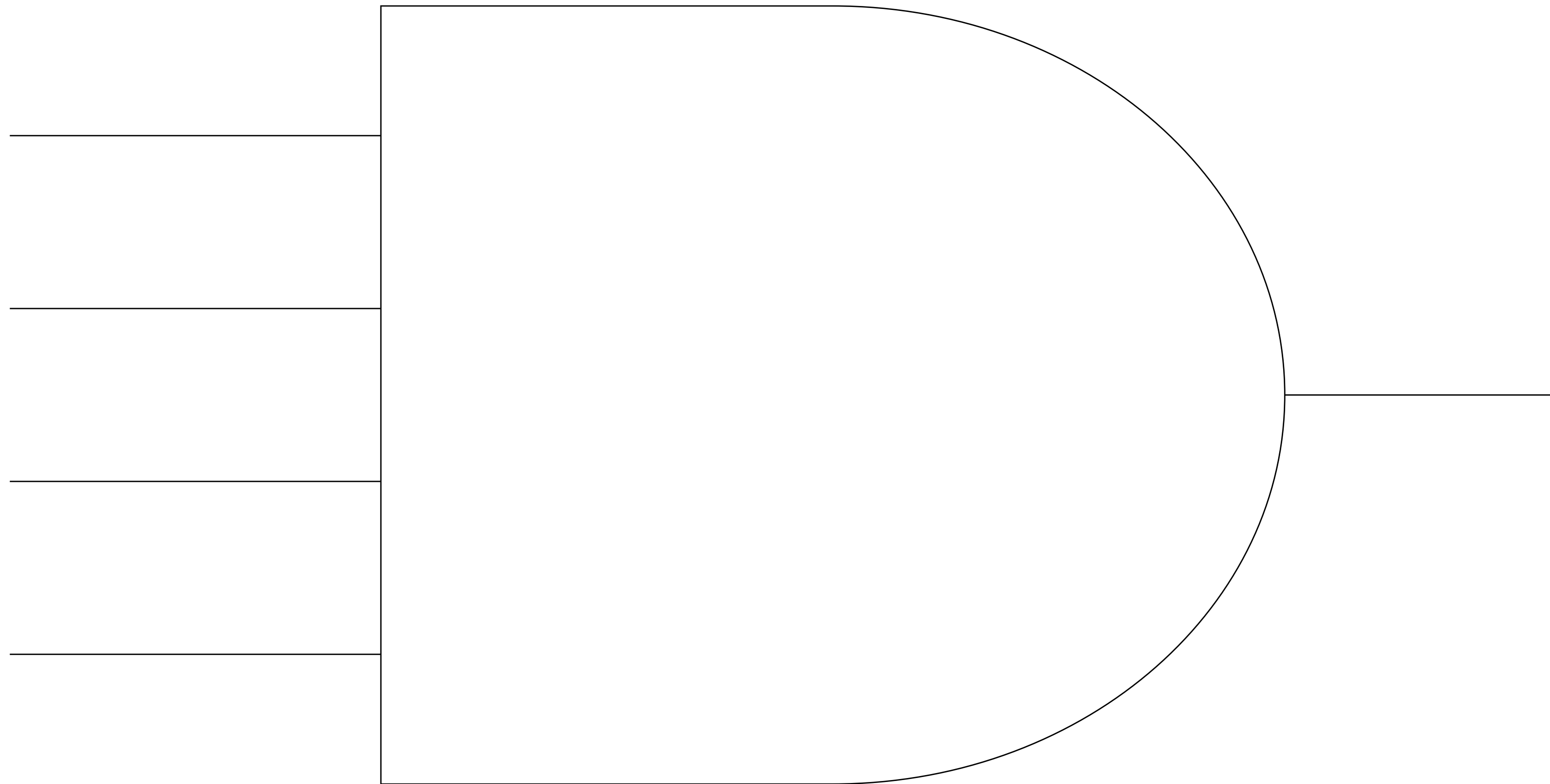
$$F = (AB + C)D$$

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

Larger gates



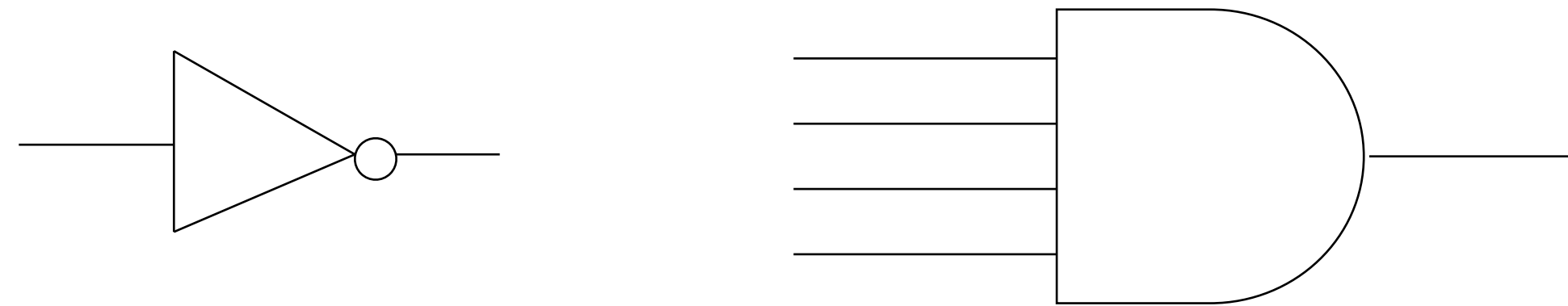
Build a 4-input AND gate using any number of 2-input gates.



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.

A _____
B _____
C _____
D _____

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

Sum-of-products form



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: DeMorgan's

DeMorgan's	$\overline{A} + \overline{B} + \overline{C} + \dots = \overline{ABC\dots}$	$= \overline{A + B + C + \dots} = \overline{A} \overline{B} \overline{C} \dots$
------------	--	---

Intuition:

“This apartment does **not** allow pets (cats or dogs)”

You can **not** have a cat and you can **not** have a dog.

$$\overline{C + D} = \overline{C} \overline{D}$$

“This apartment does **not** allow you to have both cats and dogs”

ex

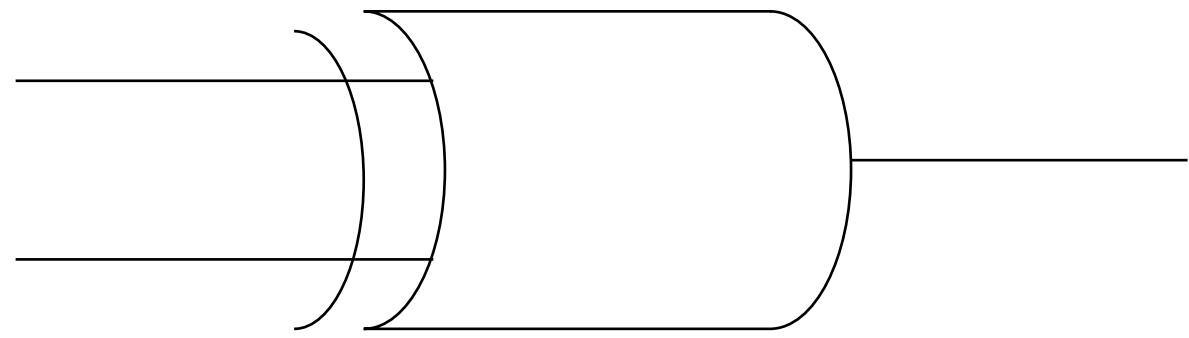
Boolean Algebra: more laws

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

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

XOR: Exclusive OR

ex



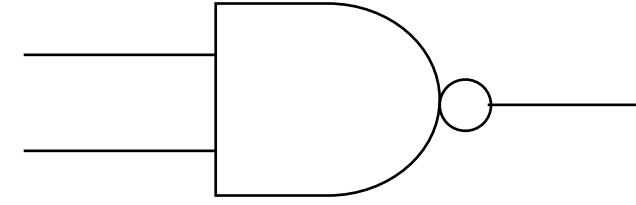
Output = 1 if exactly one input = 1.

Truth table:

Build from earlier gates:

Often used as a one-bit comparator.

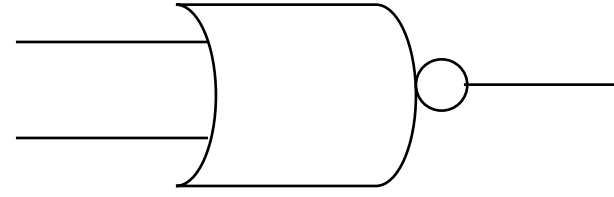
NAND is *universal*



ex

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

NOR is also *universal*

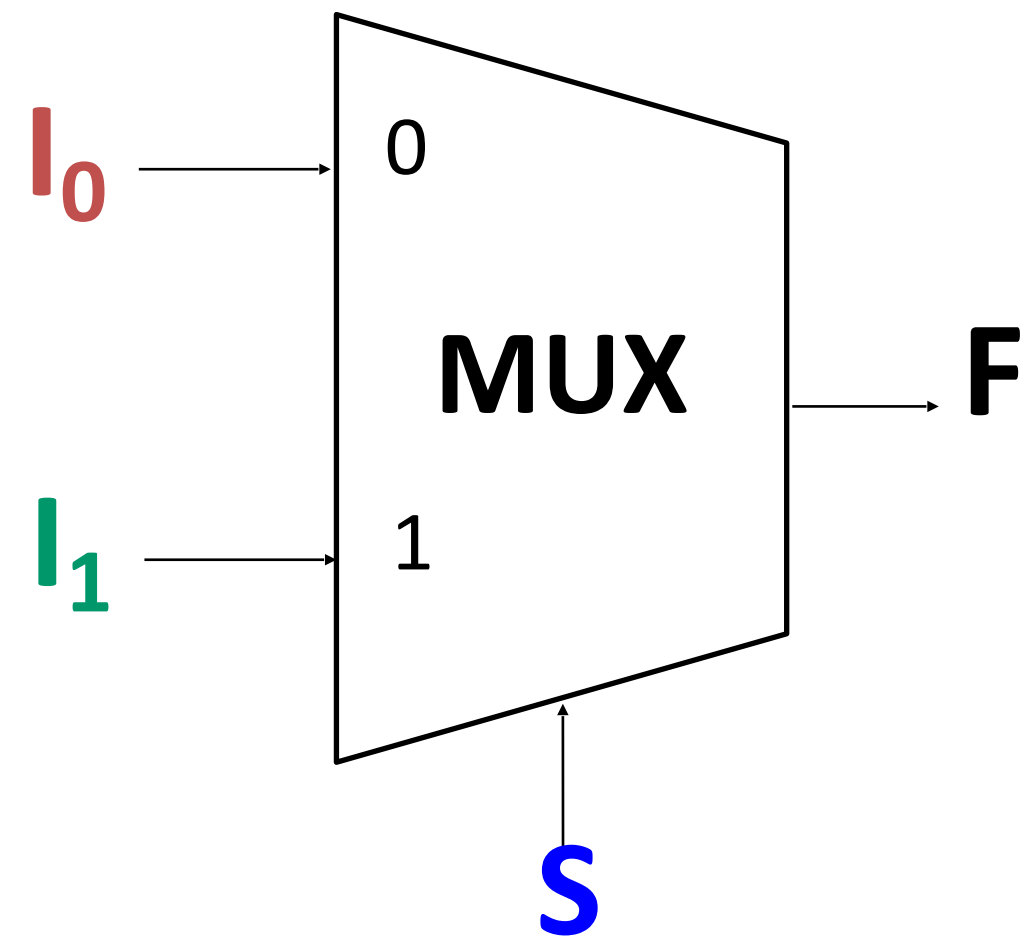


ex

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?)

Encoding *choice* with gates: multiplexing

ex



A multiplexer (MUX) chooses between two inputs (I_0 , I_1) based on a selector input, S .

If $S=0$, then $F=I_0$.

If $S=1$, then $F=I_1$.

1. Construct the truth table.

2. Build a MUX using AND, OR, and NOT.



Mary Jackson



Katherine Johnson

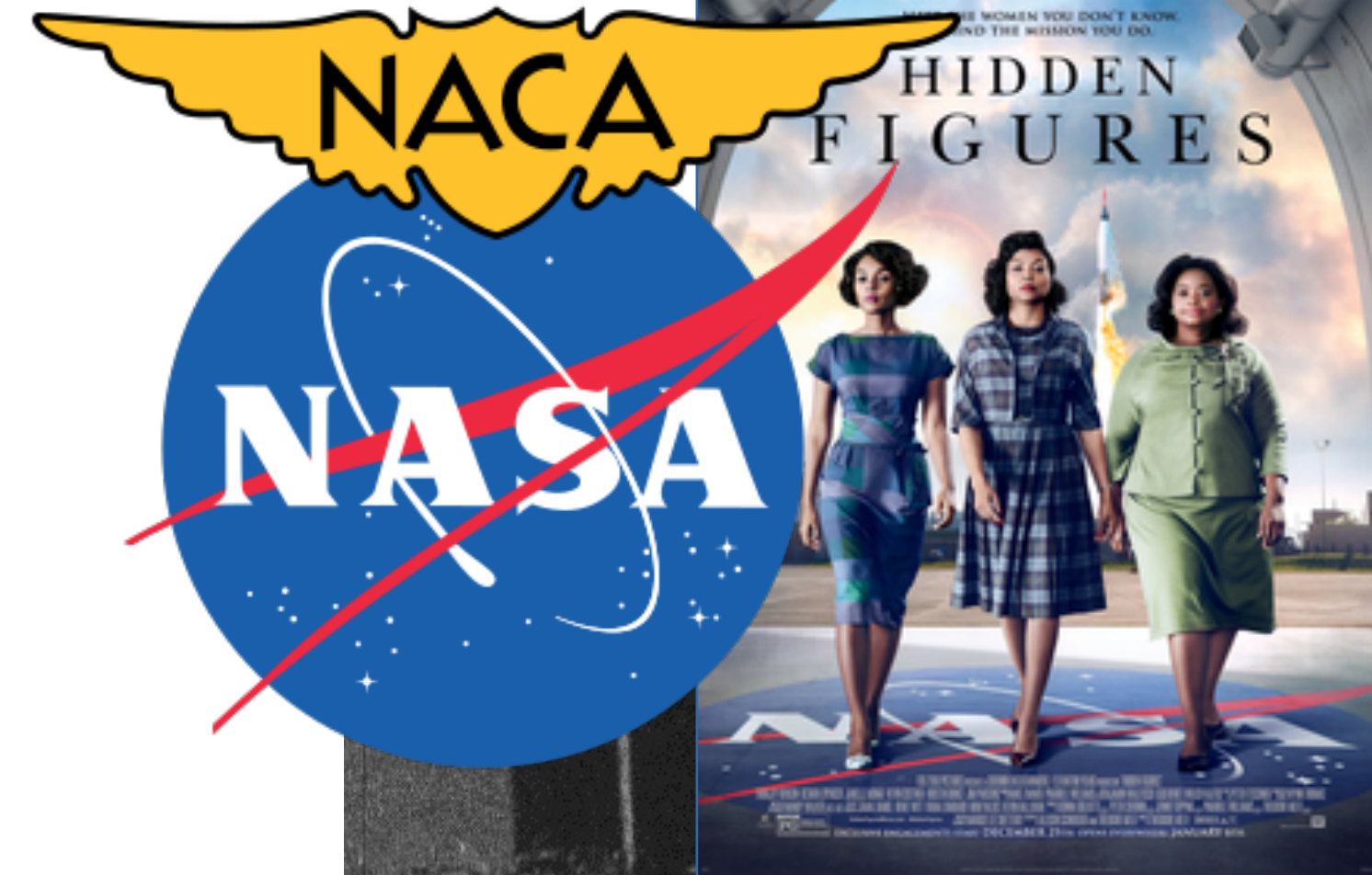
- Supported Mercury, Apollo, Space Shuttle, ...

Dorothy Vaughn

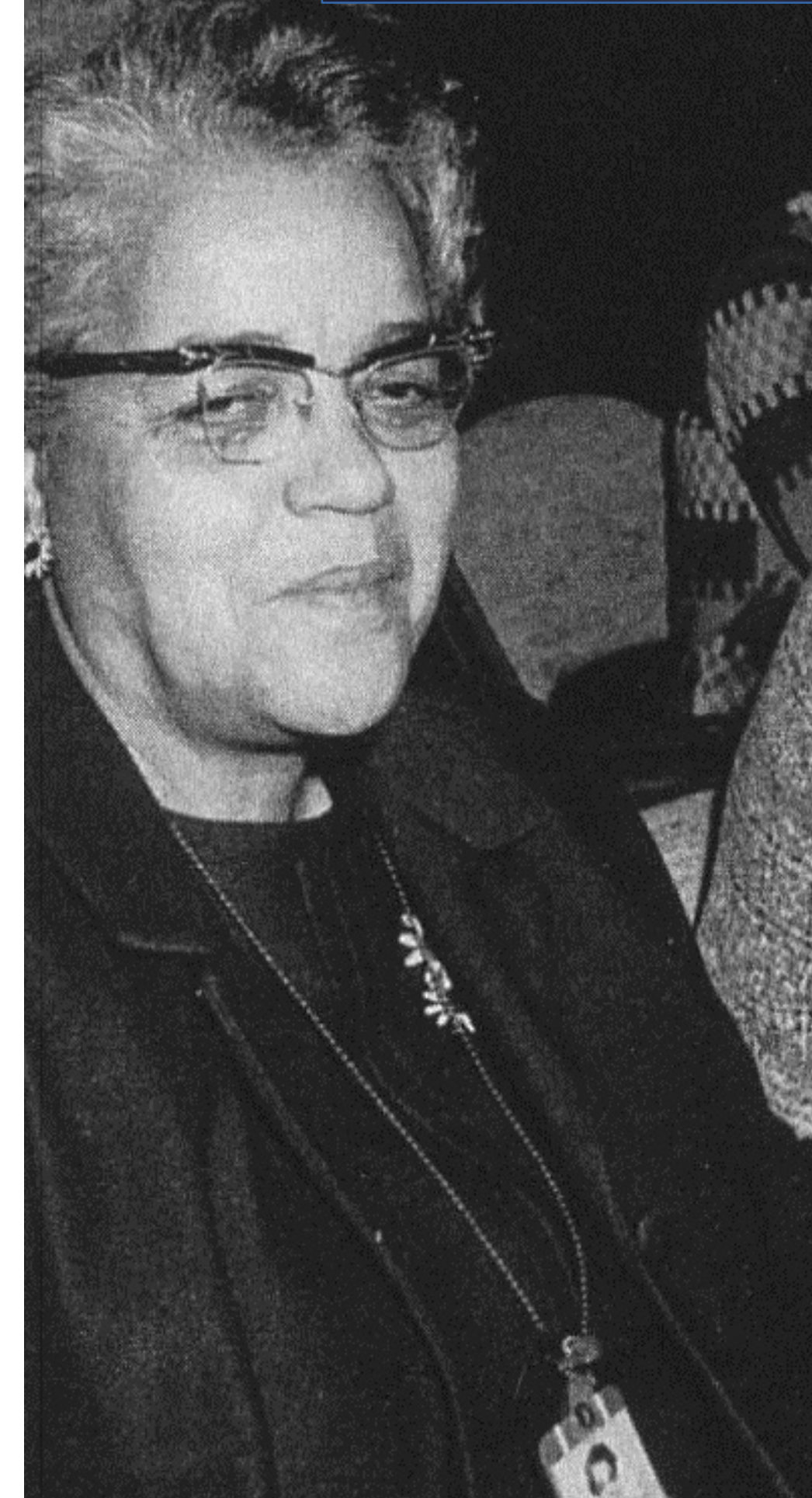
- First black supervisor within NACA
- Early self-taught FORTRAN programmer for NASA move to digital computers.

Computers

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



optional



Early pioneers in reliable computing

optional



Katherine Johnson

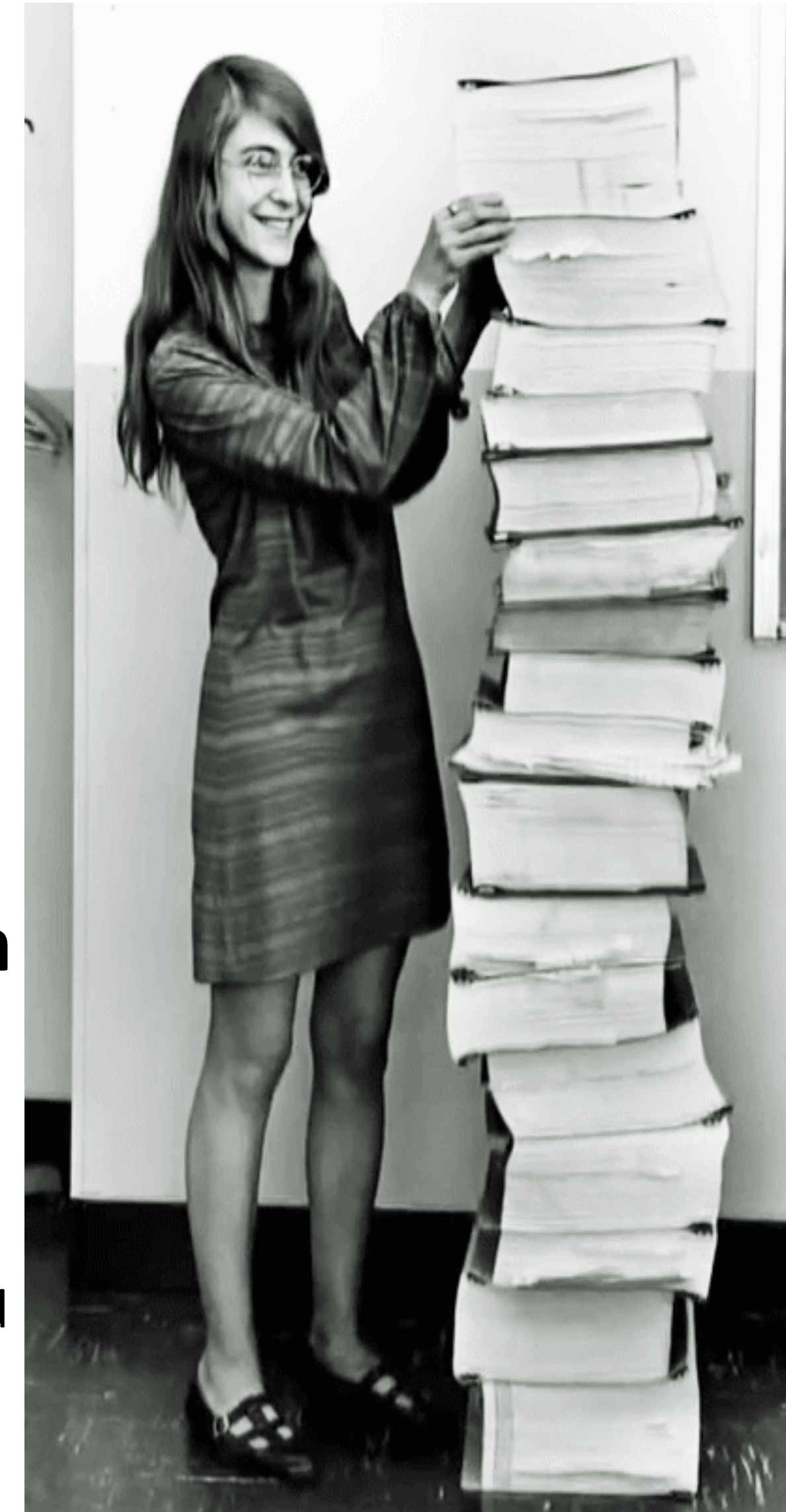
- Calculated first US human space flight trajectories
- Mercury, Apollo 11, Space Shuttle, ...
- Reputation for accuracy in manual calculations, verified early code
- Called to verify results of code for launch calculations for first US human in orbit
- Backup calculations helped save Apollo 13
- Presidential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

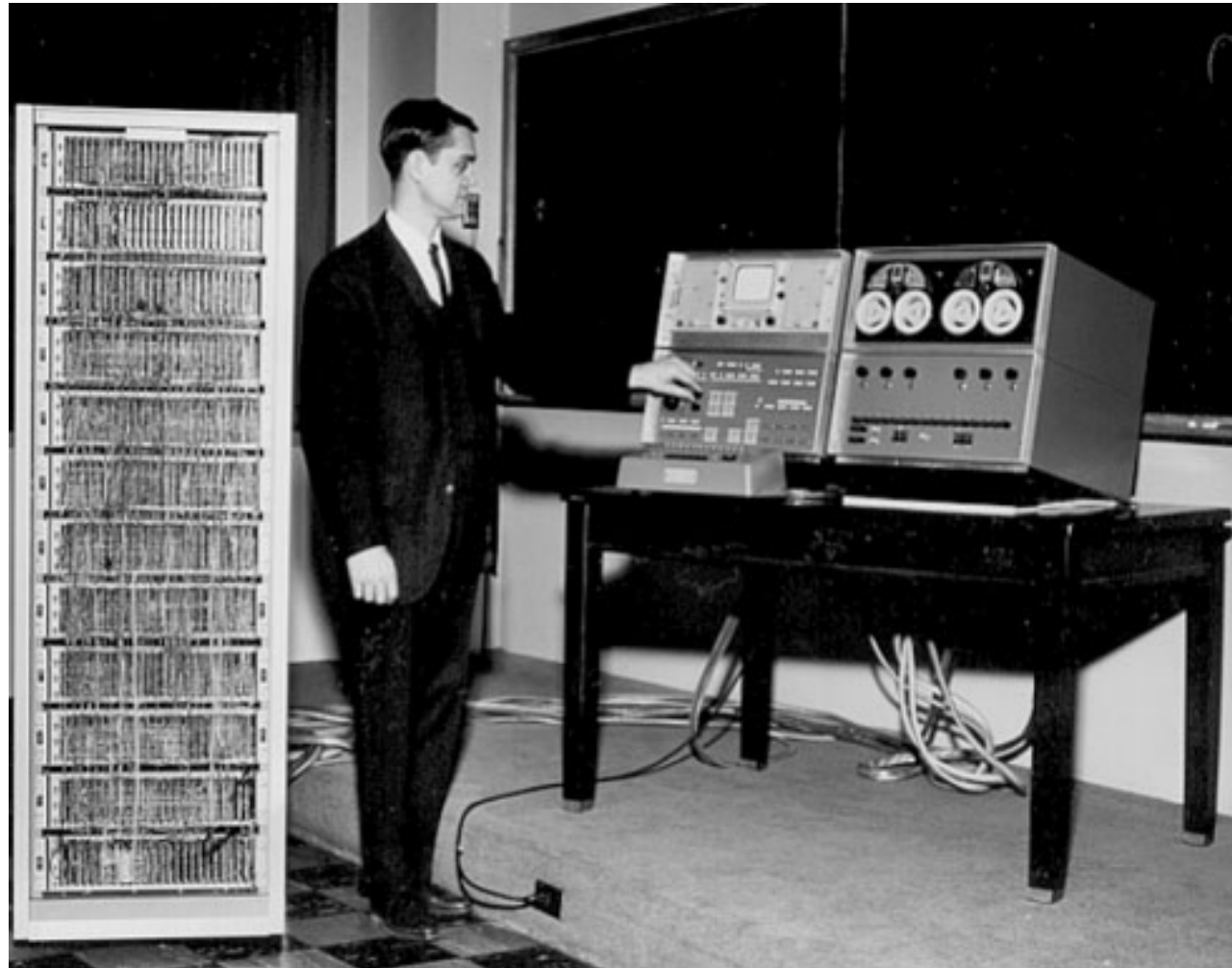
Apollo 11 code print-out



Digital Logic

Wellesley Connection: Mary Allen Wilkes '59

optional



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.