18-100 Introduction to Electrical and Computer Engineering

Lecture 09

Boolean Logic, Logic Gates, and Latches

Learning Objectives for This Lecture

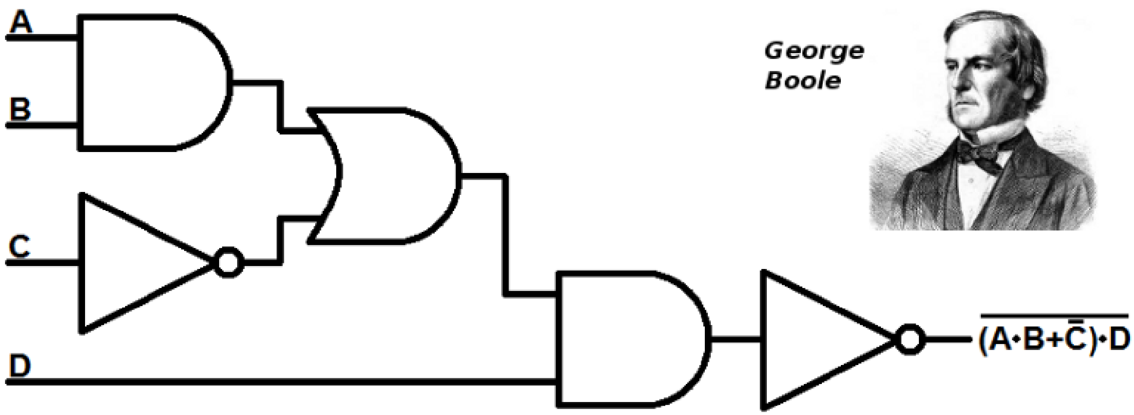
- *What is Boolean logic and what are logic gates.*
- *How to use logic gates to build logic functions.*
- *How computer uses binary numbers to control "things".*
- *How computer uses logic gates to do "computation".*
- *Latches and maintaining state*

A Simple Truth of "1" and "0"

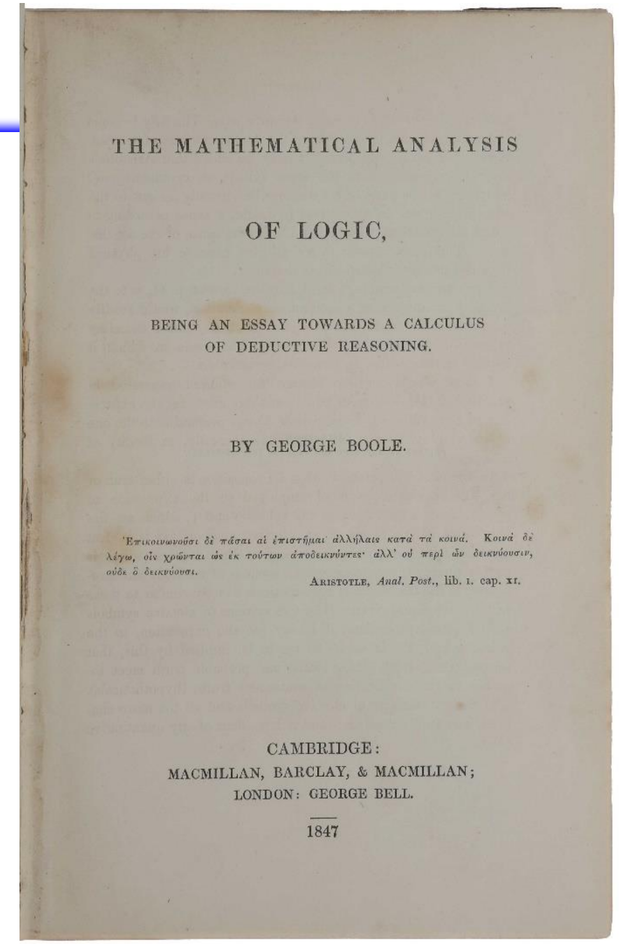
Each transistor only controls a "1" and a "0", however,

8,500,000,000 Transistors can carry out very complex functions!

Binary Logic:



George Boole


THE MATHEMATICAL ANALYSIS

OF LOGIC,

BEING AN ESSAY TOWARDS A CALCULUS
OF DEDUCTIVE REASONING.

BY GEORGE BOOLE.

Ἐπικοινωνοῦσι δὲ πᾶσαι αἱ ἐπιστῆμαι ἀλλήλαις κατὰ τὰ κοινά. Κοινὰ δὲ λέγω, οἷς χρῶνται ὡς ἐκ τούτων ἀποδεικνύντες· ἀλλ' οὐ περὶ ὧν δεικνύουσιν, οὐδὲ ὃ δεικνύουσι.

ARISTOTLE, *Anal. Post.*, lib. I. cap. XI.

CAMBRIDGE:
MACMILLAN, BARCLAY, & MACMILLAN;
LONDON: GEORGE BELL.

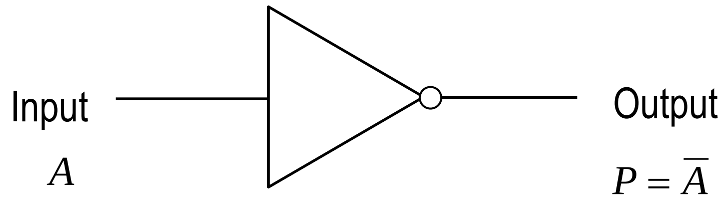
1847

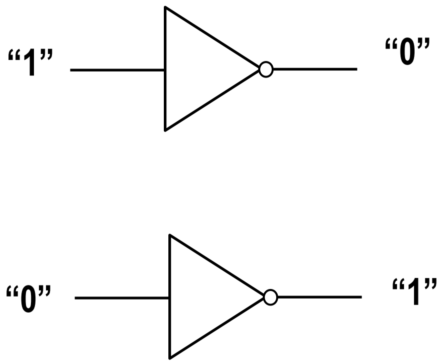
"True" = "1"

"False" = "0"

Binary bit operation: Simplest decision making

NOT Gate: Inverter

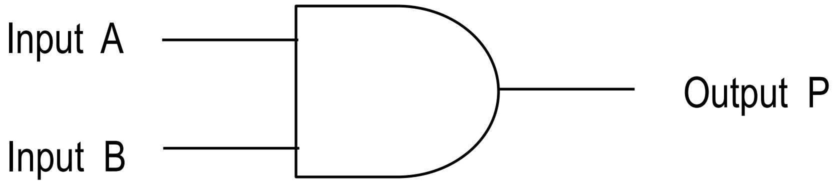


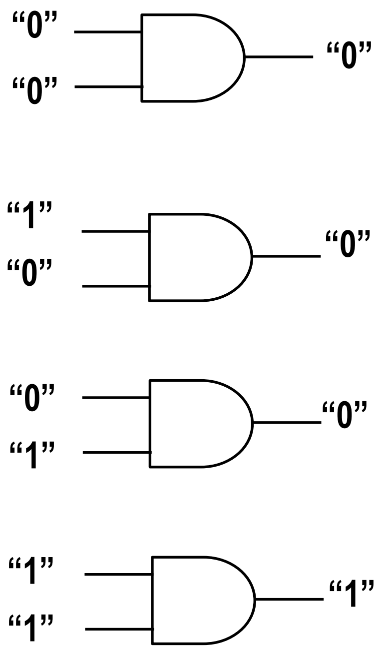


Truth Table $P = \overline{A}$

Input	Output
1	0
0	1

AND Gate



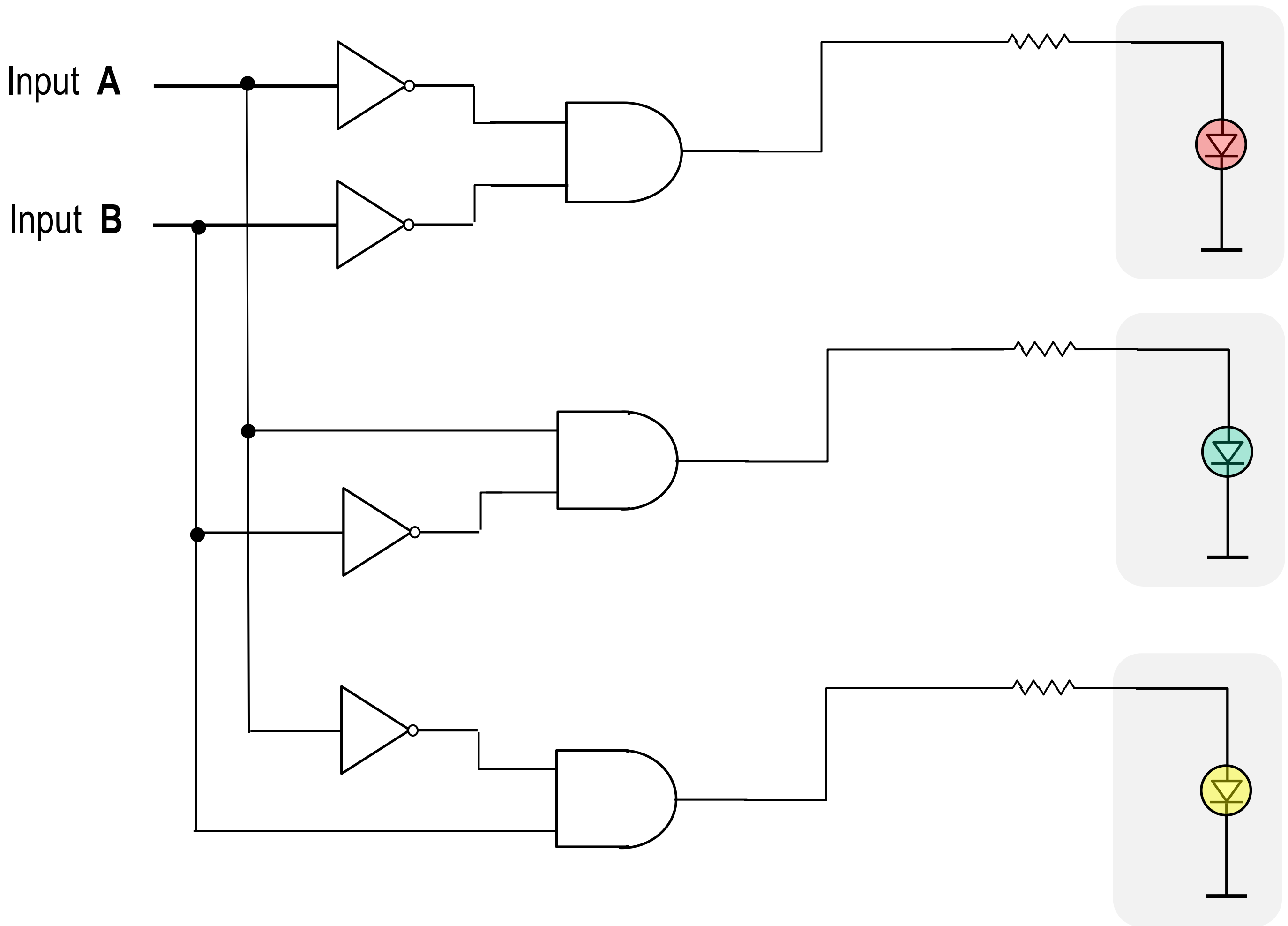


Truth Table $\boxed{P = A \cdot B}$

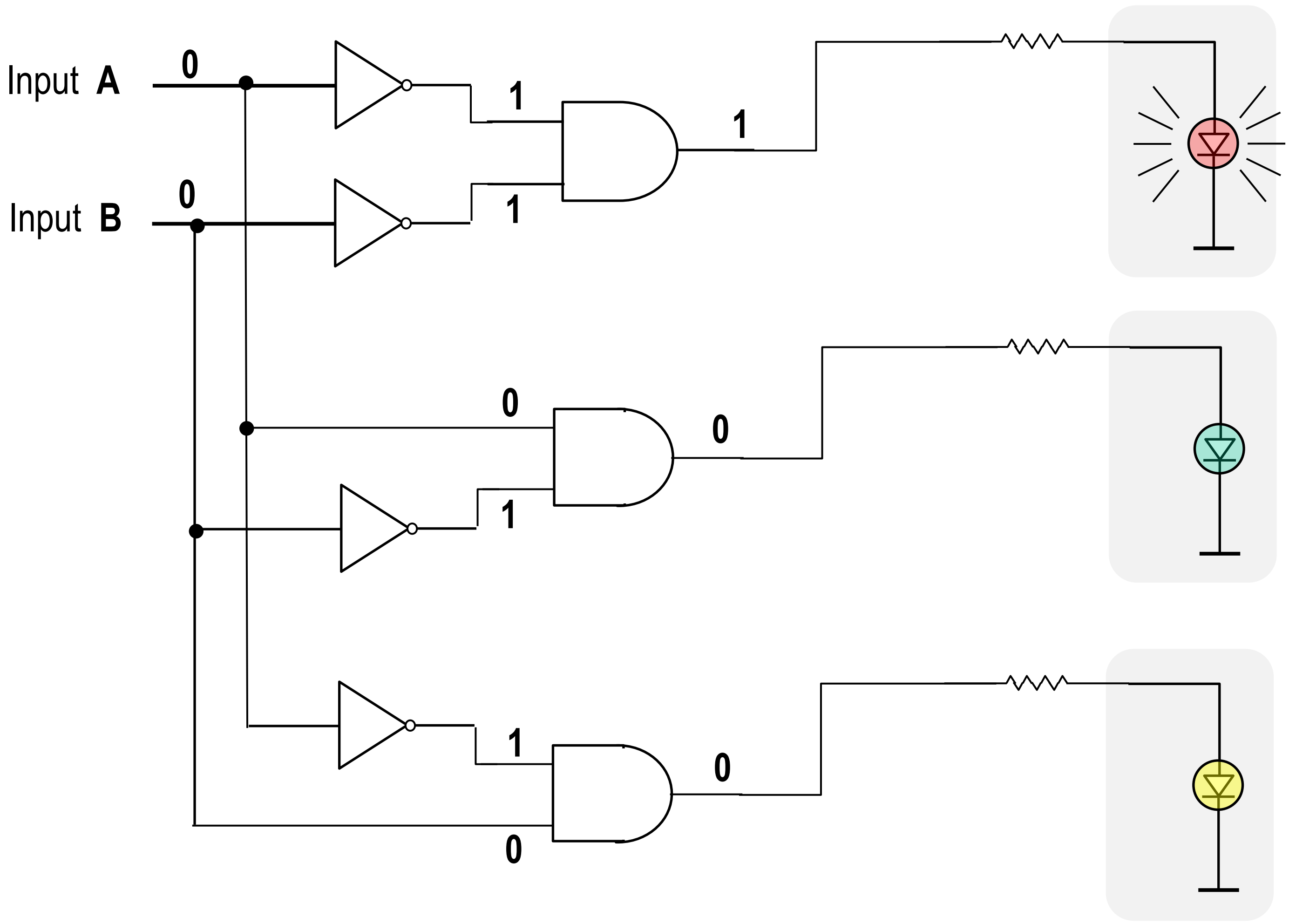
Input A	Input B	Output P
0	0	0
1	0	0
0	1	0
1	1	1

$$\begin{cases} 0 \cdot 0 = 0 \\ 1 \cdot 0 = 0 \\ 0 \cdot 1 = 0 \\ 1 \cdot 1 = 1 \end{cases}$$

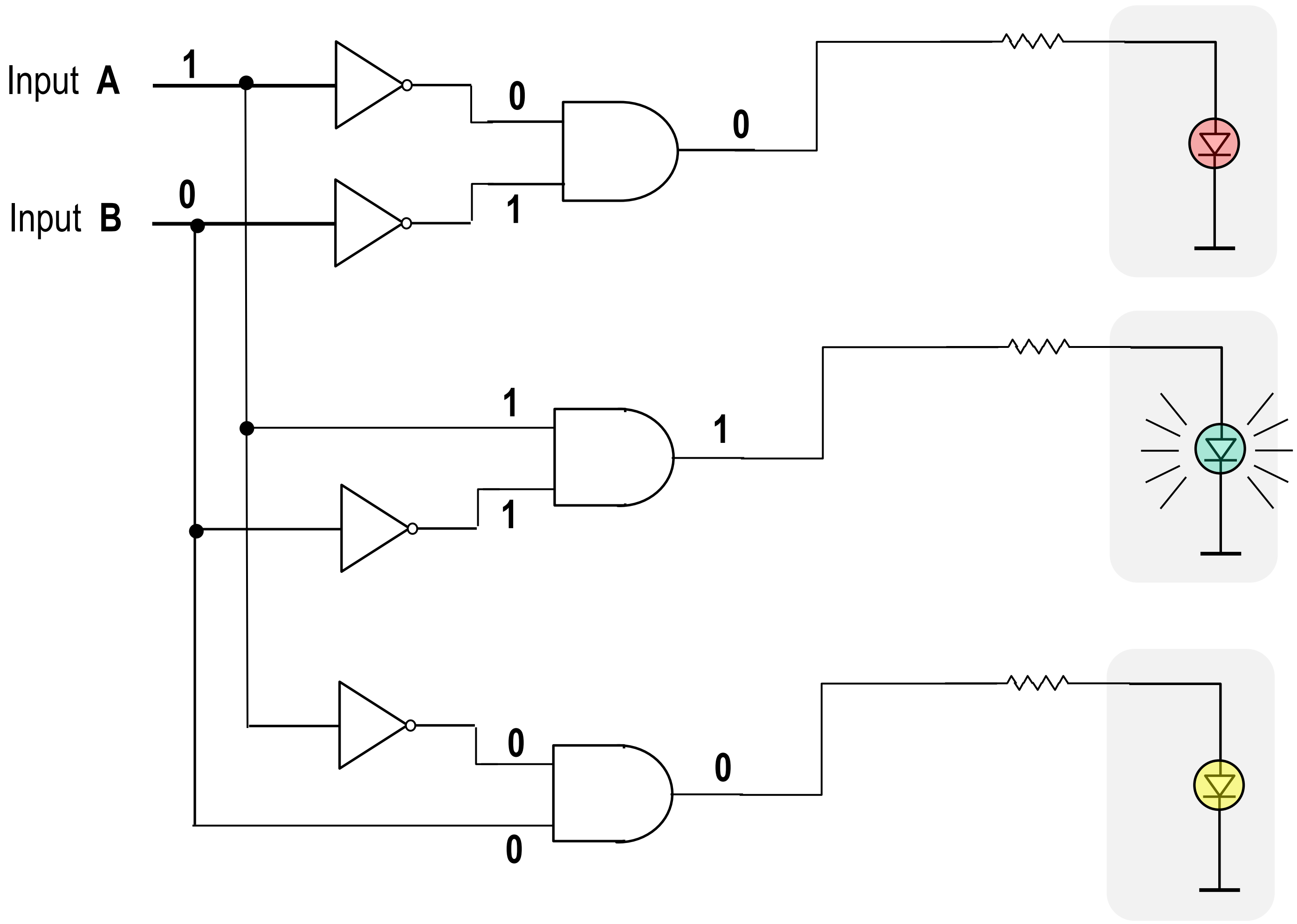
Using Logic Gates to Control Action: The Use of "1" and "0"



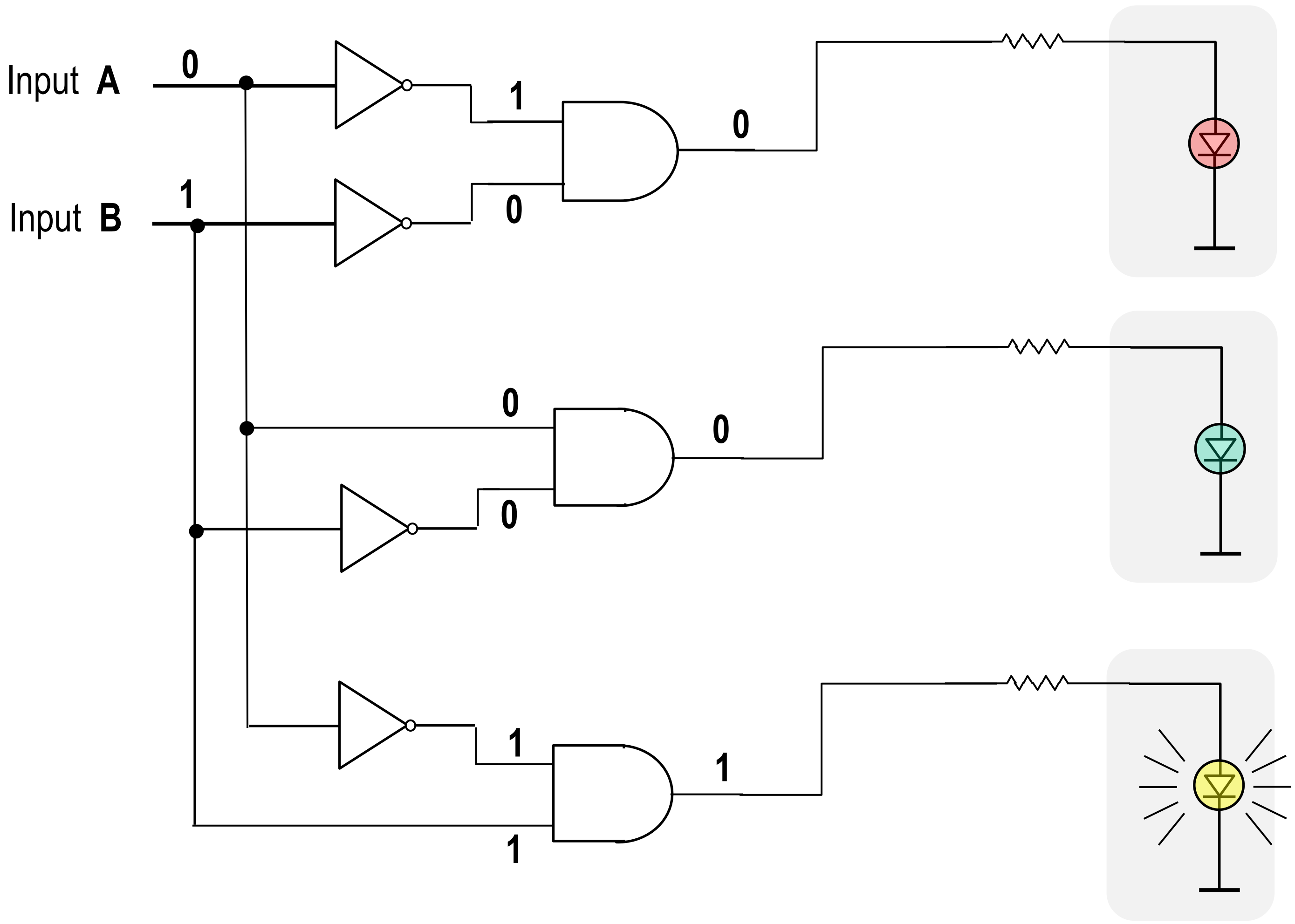
Using Logic Gates to Control Action: The Use of “1” and “0”



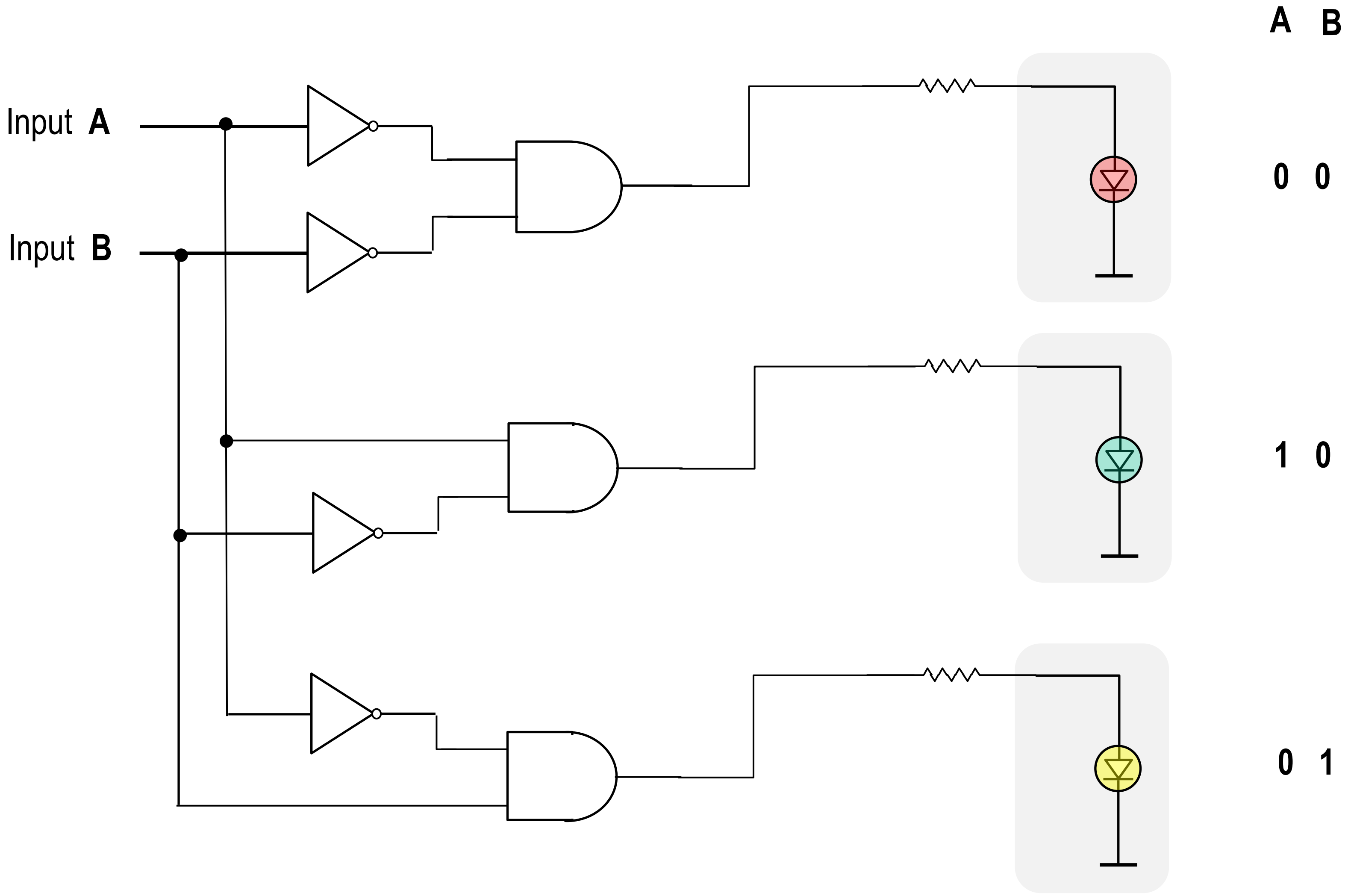
Using Logic Gates to Control Action: The Use of “1” and “0”



Using Logic Gates to Control Action: The Use of "1" and "0"



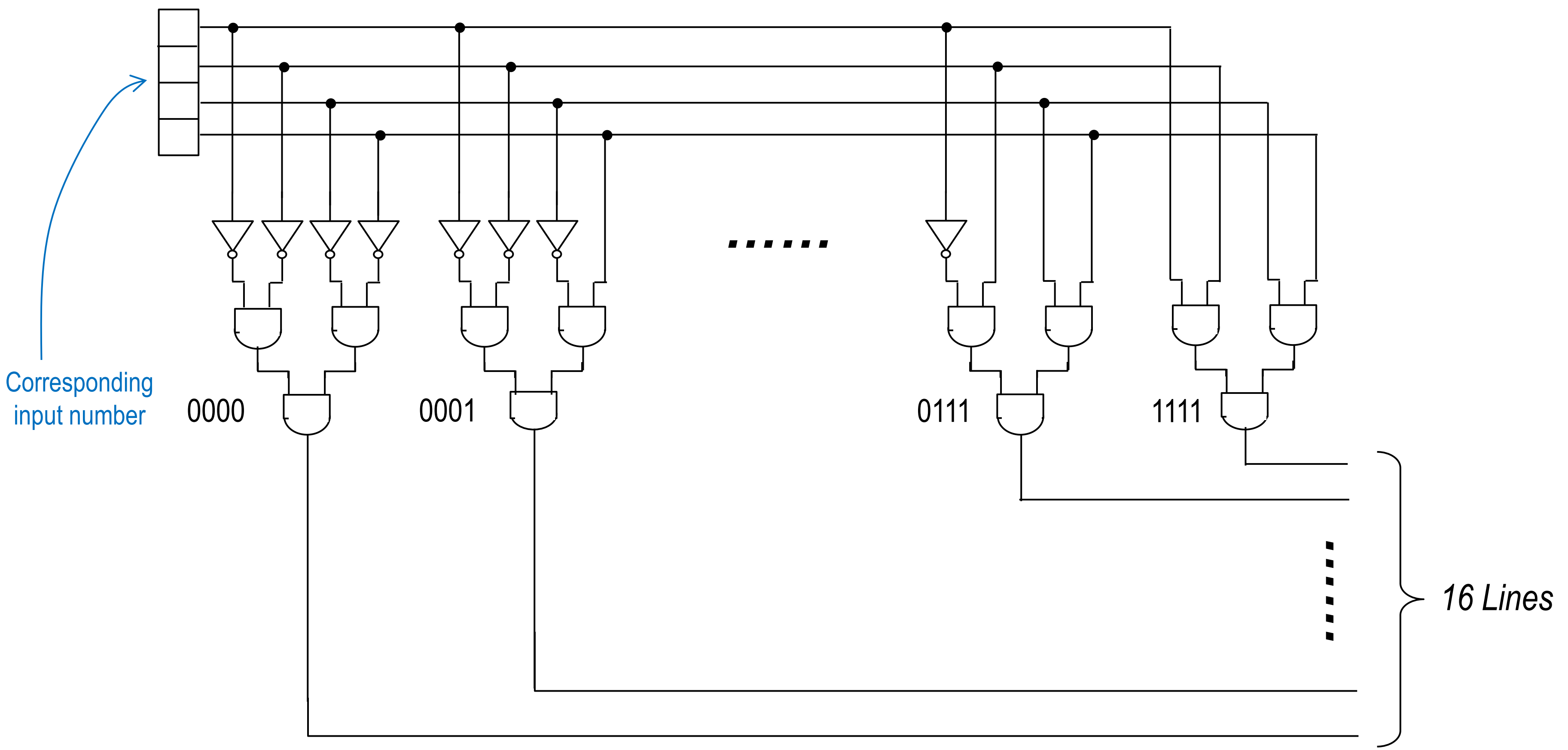
Using Logic Gates to Control Action: The Use of “1” and “0”



n-bit Input Controlling/Selecting 2^n Different Actions

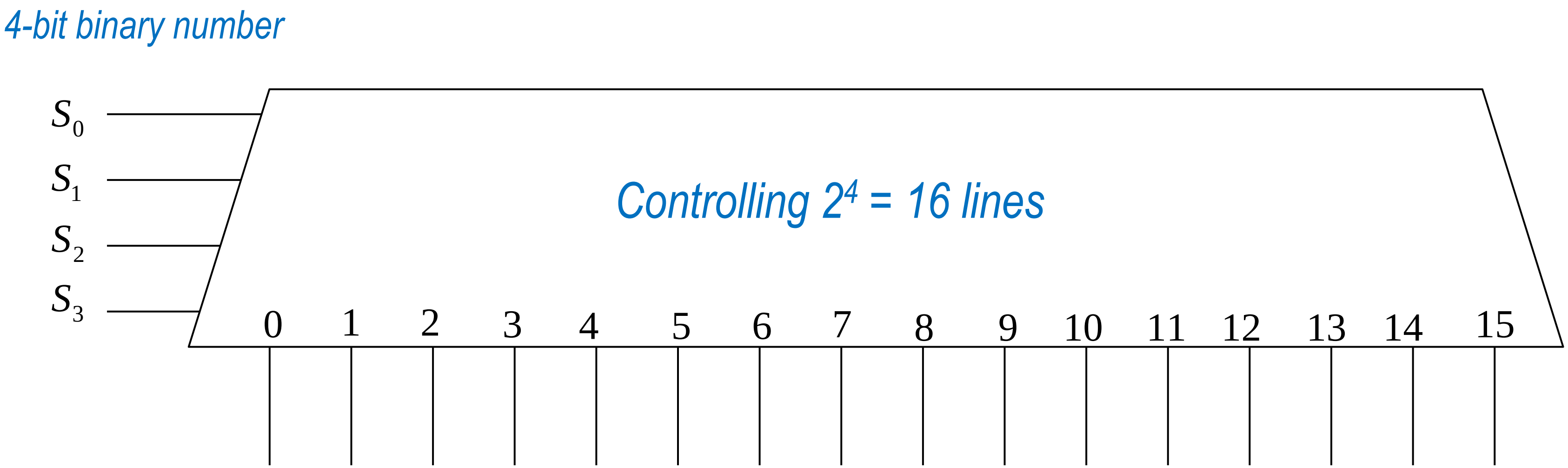
A n-bit number has 2^n different values: Can be used to control 2^n different actions!

Example: 4-bit number controlling/selecting 16 lines



n-bit Input Controlling/Selecting 2^n Different Actions

4-bit Decode

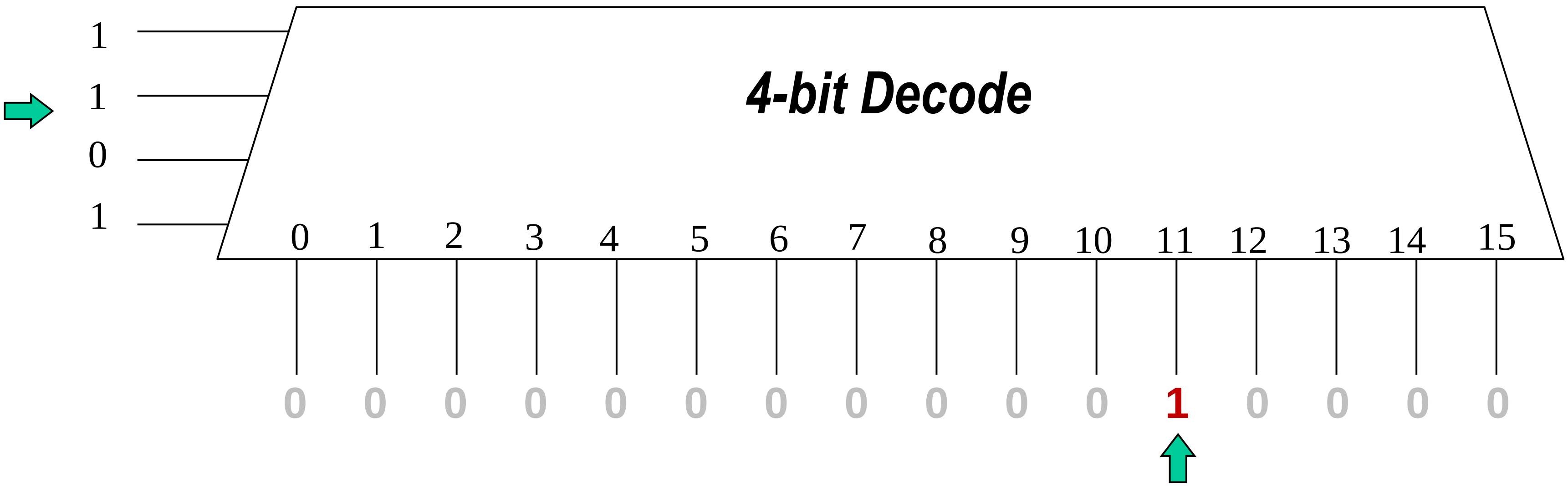


This is how machine(computer) uses binary numbers to perform different actions!

n-bit Input Controlling/Selecting 2^n Different Actions

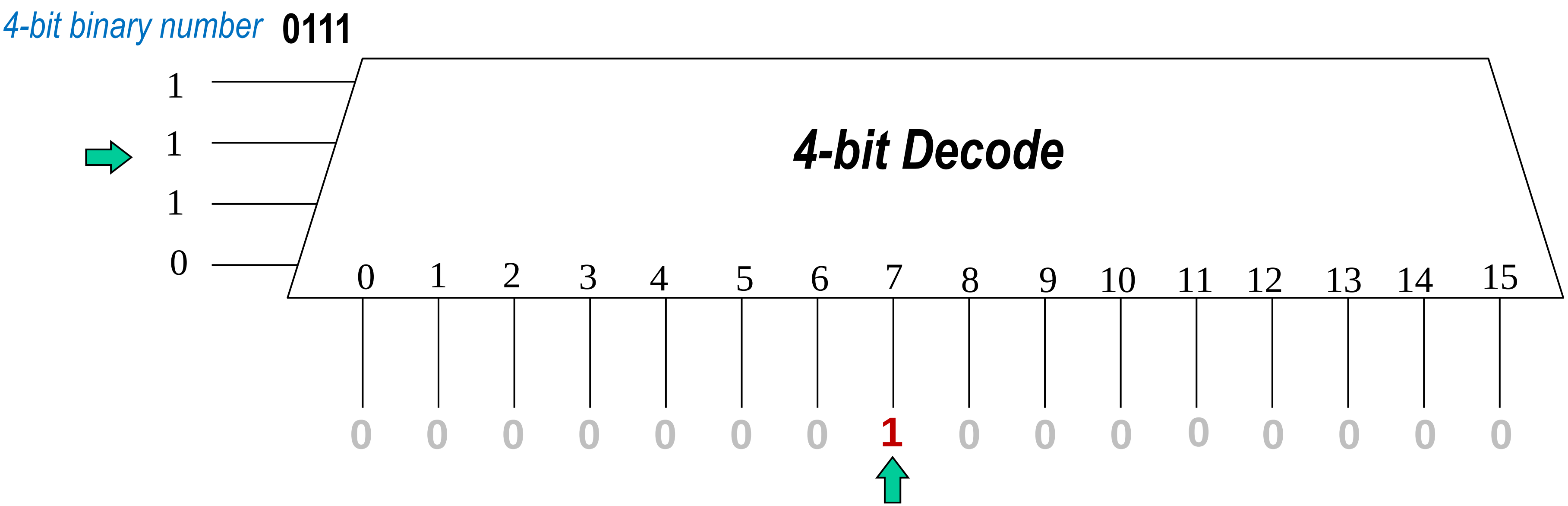
4-bit Decode

4-bit binary number: **1011**



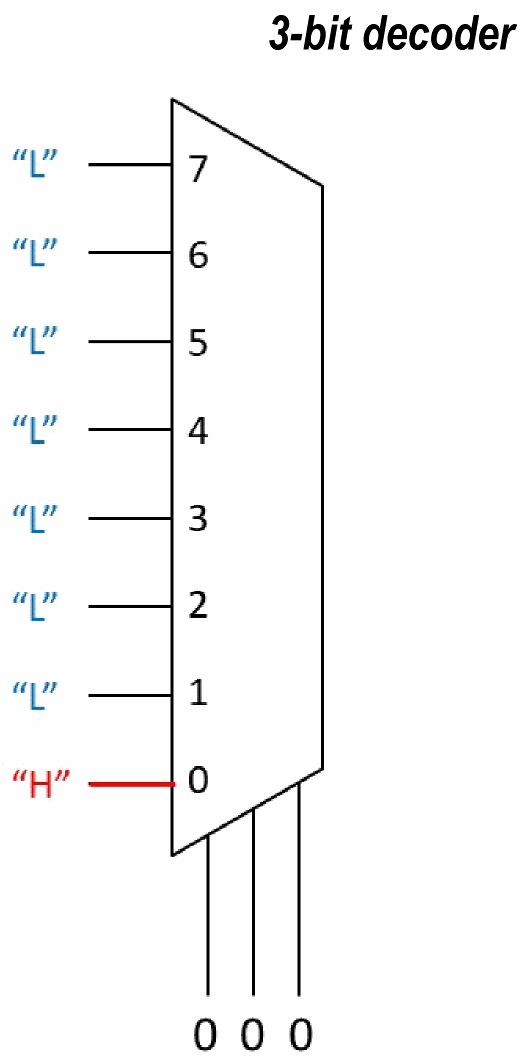
A 4-bit binary number selects a single line (out of the 16 lines) to be "HIGH".
(The rest all remain "LOW")

4-bit Input Controlling/Selecting 2^4 Different Actions

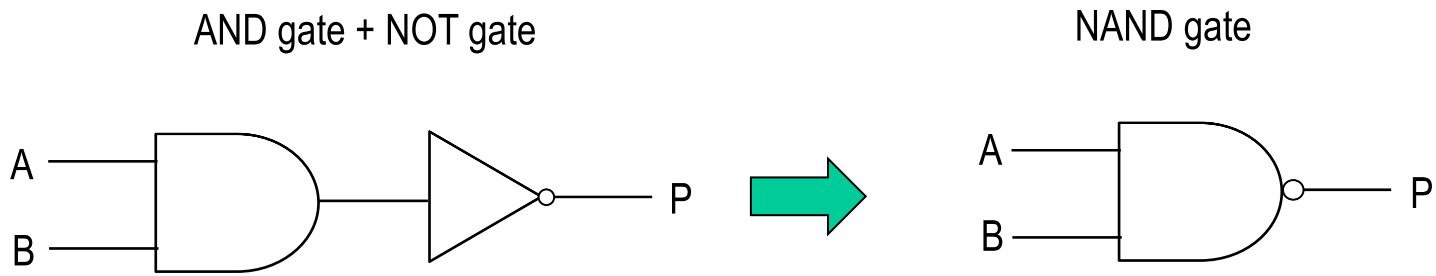


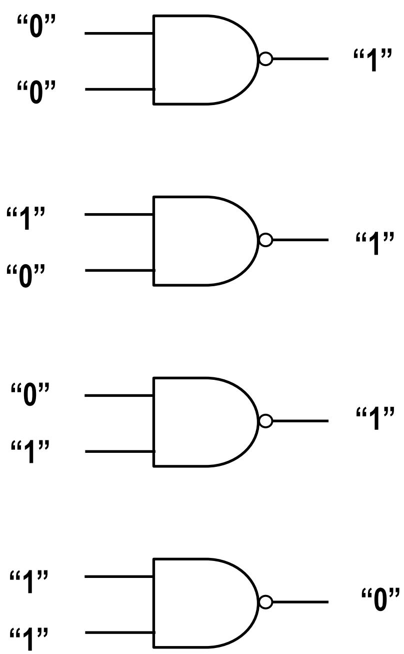
A 4-bit binary number selects a single line (out of the 16 lines) to be "HIGH".
(The rest all remain "LOW")

Does This Look Familiar ?



NAND Gate :



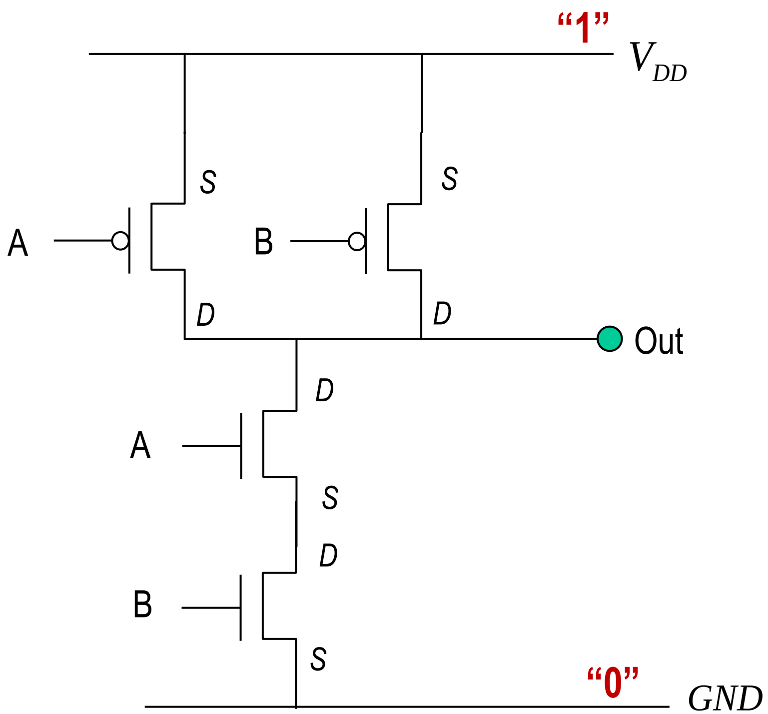


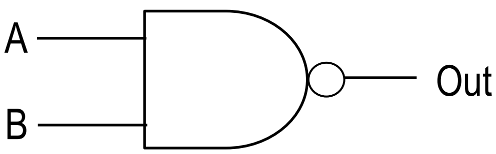
Truth Table $\boxed{P = \overline{A \cdot B}}$

Input A	Input B	Output P
0	0	1
1	0	1
0	1	1
1	1	0

$$\left\{\begin{array}{l} \overline{0 \cdot 0} = \overline{0} = 1 \\ \overline{1 \cdot 0} = \overline{0} = 1 \\ \overline{0 \cdot 1} = \overline{0} = 1 \\ \overline{1 \cdot 1} = \overline{1} = 0 \end{array}\right.$$

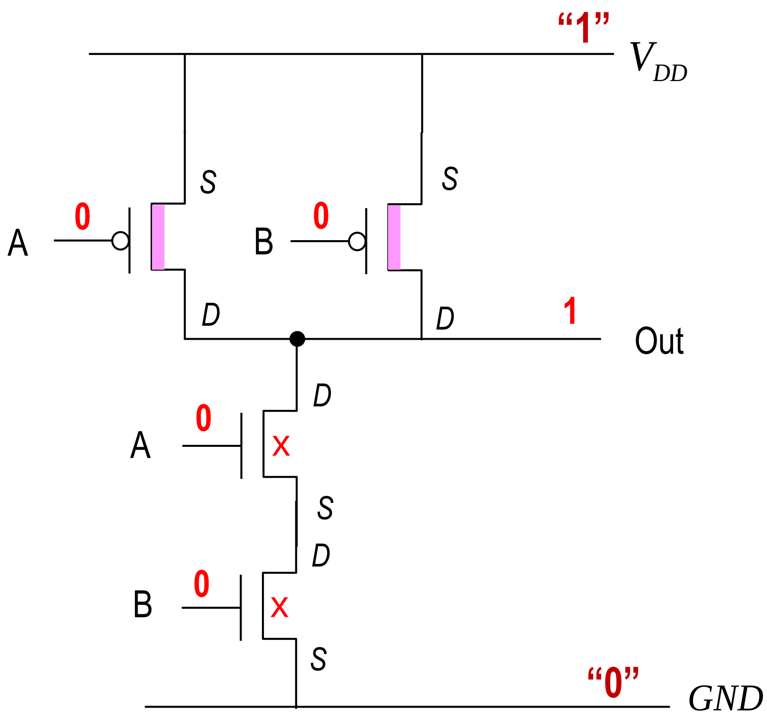
CMOS Logic NAND Gate

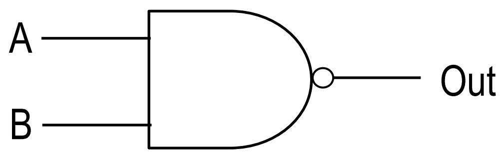




A	B	Out
0	0	1
1	0	1
0	1	1
1	1	0

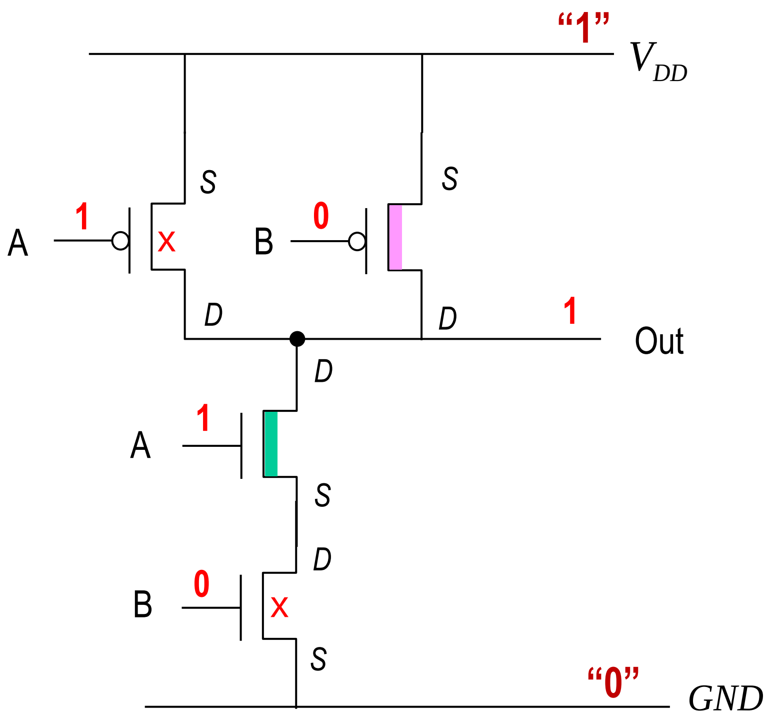
CMOS Logic NAND Gate

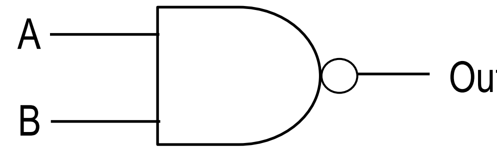




A	B	Out
0	0	1
1	0	1
0	1	1
1	1	0

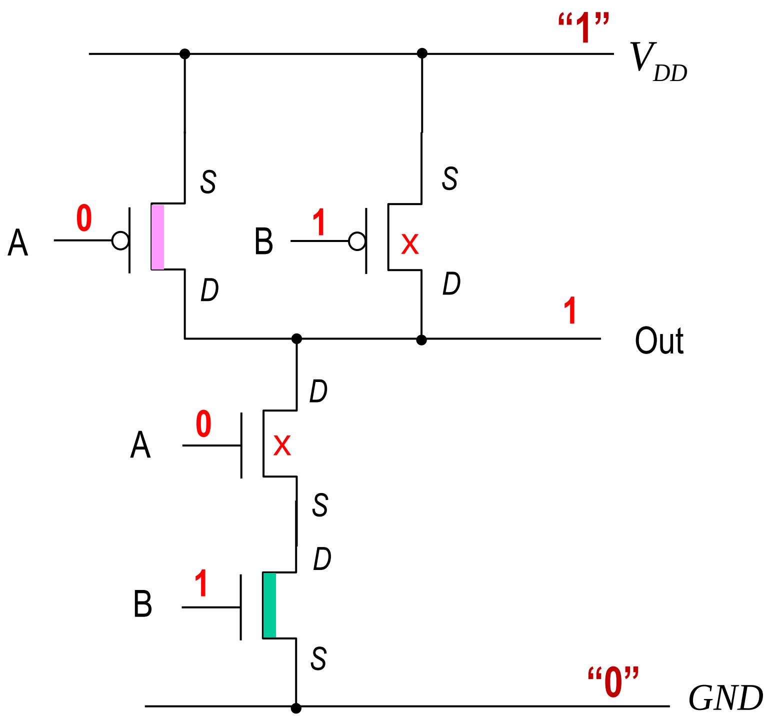
CMOS Logic NAND Gate

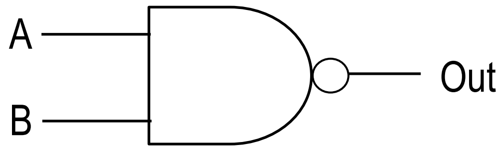




A	B	Out
0	0	1
1	0	1
0	1	1
1	1	0

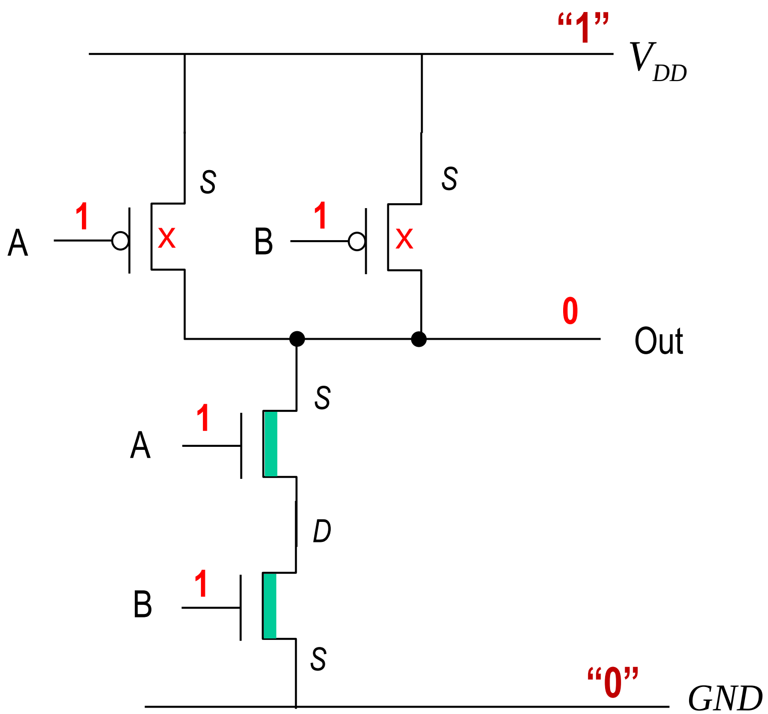
CMOS Logic NAND Gate

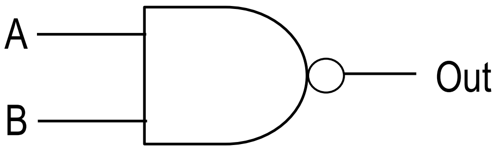




A	B	Out
0	0	1
1	0	1
0	1	1
1	1	0

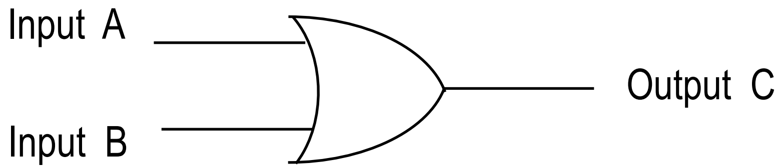
CMOS Logic NAND Gate

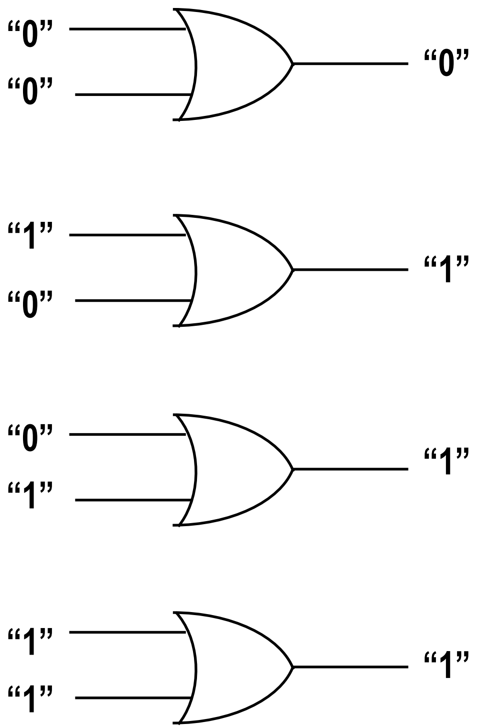




A	B	Out
0	0	1
1	0	1
0	1	1
1	1	0

OR Gate



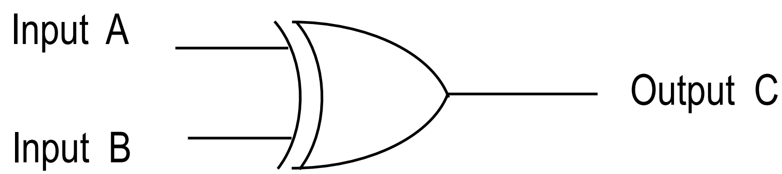


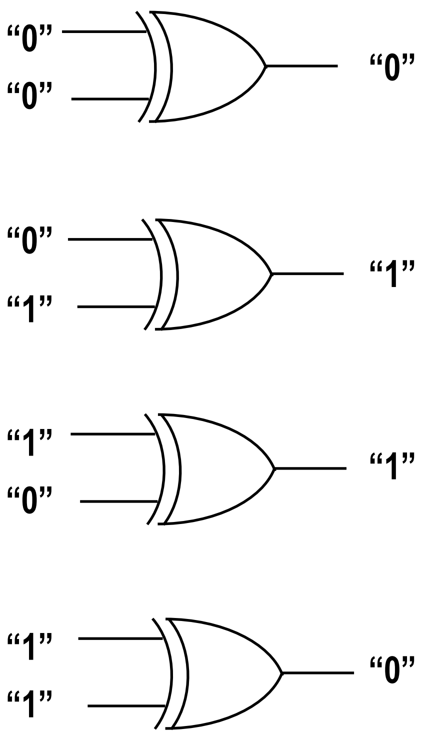
Truth Table $C = A + B$

Input A	Input B	Output C
0	0	0
1	0	1
0	1	1
1	1	1

$$\begin{cases} 0+0=0 \\ 1+0=1 \\ 0+1=1 \\ 1+1=1 \end{cases}$$

Exclusive OR Gate: XOR





Truth Table $C = A \oplus B$

Input A	Input B	Output C
0	0	0
0	1	1
1	0	1
1	1	0

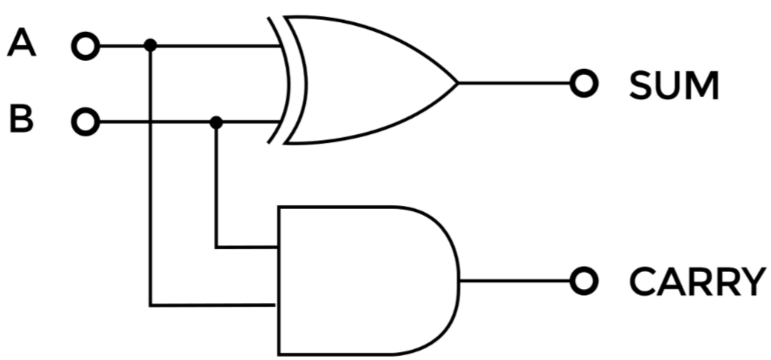
$$\left\{ \begin{array}{l} 0 \oplus 0 = 0 \\ 1 \oplus 0 = 1 \\ 0 \oplus 1 = 1 \\ 1 \oplus 1 = 0 \end{array} \right.$$

Computing Using Logic Gates: Addition

Adding Binary Numbers for the Rightest Digit (No Input Carry)

$$\begin{array}{r} A \\ +\ B \\ \hline \\ \end{array} \quad \Bigg| \quad \begin{array}{r} 0 \\ +\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ {}_{1}1 \\ \hline 0 \end{array}$$

Half Adder (no input carry)



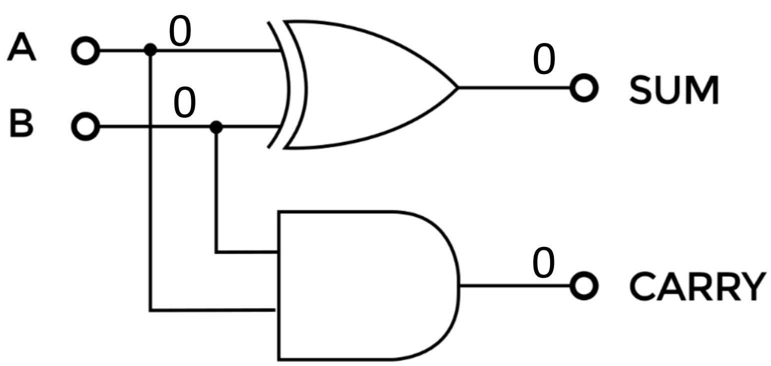
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Computing Using Logic Gates: Addition

Adding Binary Numbers for the Rightest Digit (No Input Carry)

A + B	0 + 0	1 + 0	0 + 1	1 + 1
	0	1	1	0 (carry 1)

Half Adder (no input carry)



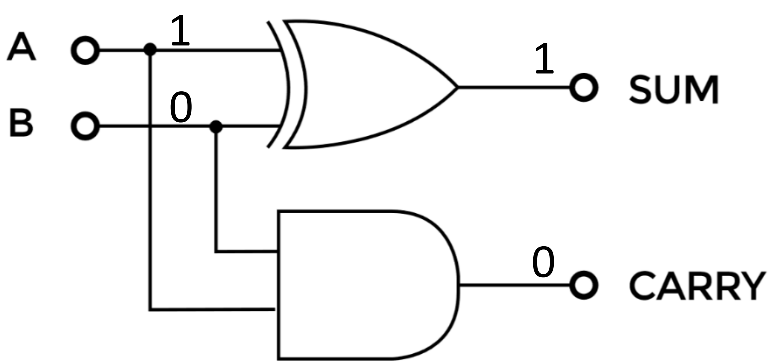
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Computing Using Logic Gates: Addition

Adding Binary Numbers for the Rightest Digit (No Input Carry)

$$\begin{array}{r} A \\ +\ B \\ \hline \\ \end{array} \quad \Bigg| \quad \begin{array}{r} 0 \\ +\ 0 \\ \hline 0 \end{array} \qquad \boxed{\begin{array}{r} 1 \\ +\ 0 \\ \hline 1 \end{array}} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ {}_{1}\,1 \\ \hline 0 \end{array}$$

Half Adder (no input carry)



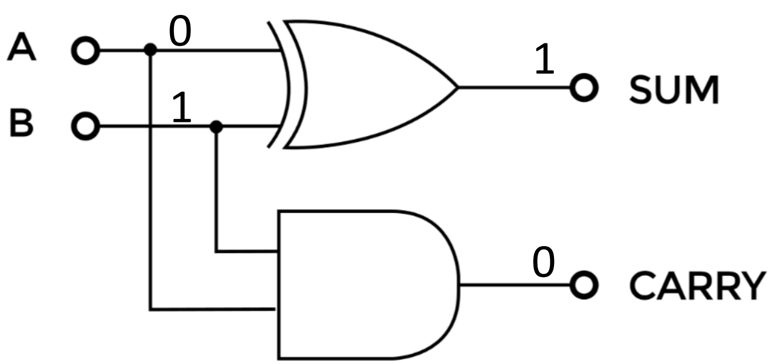
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Computing Using Logic Gates: Addition

Adding Binary Numbers for the Rightest Digit (No Input Carry)

A	0	1	0	1
+ B	+ 0	+ 0	+ 1	+ $_1$ 1
	0	1	1	0

Half Adder (no input carry)



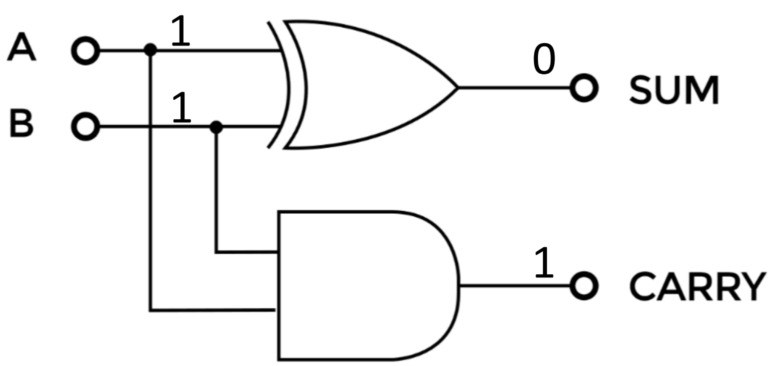
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Computing Using Logic Gates: Addition

Adding Binary Numbers for the Rightest Digit (No Input Carry)

$$\begin{array}{r} A \\ +\ B \\ \hline \end{array} \qquad \begin{array}{r} 0 \\ +\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ {}_{1}\,1 \\ \hline 0 \end{array}$$

Half Adder (no input carry)



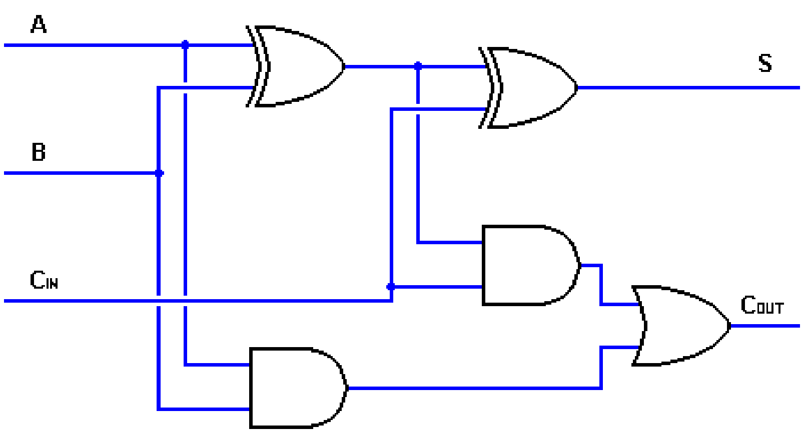
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Addition

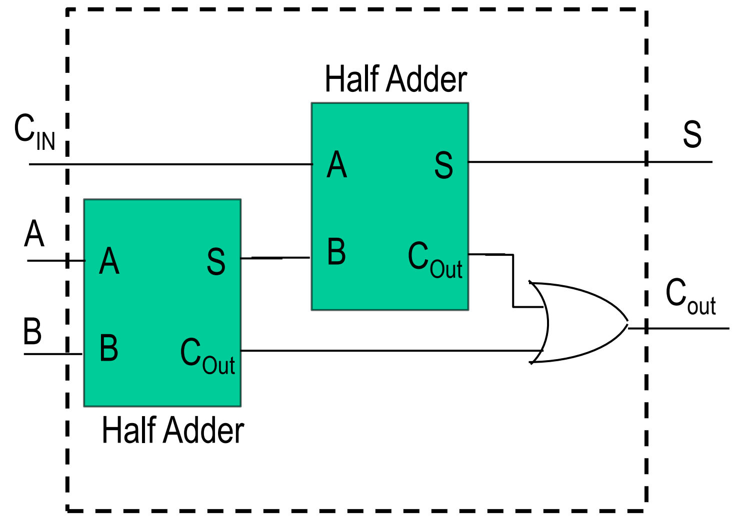
Addition with input carry:

0	1	0	1
+ 0 ₁	+ ₁ 0 ₁	+ ₁ 1 ₁	+ ₁ 1 ₁
1	0	0	1

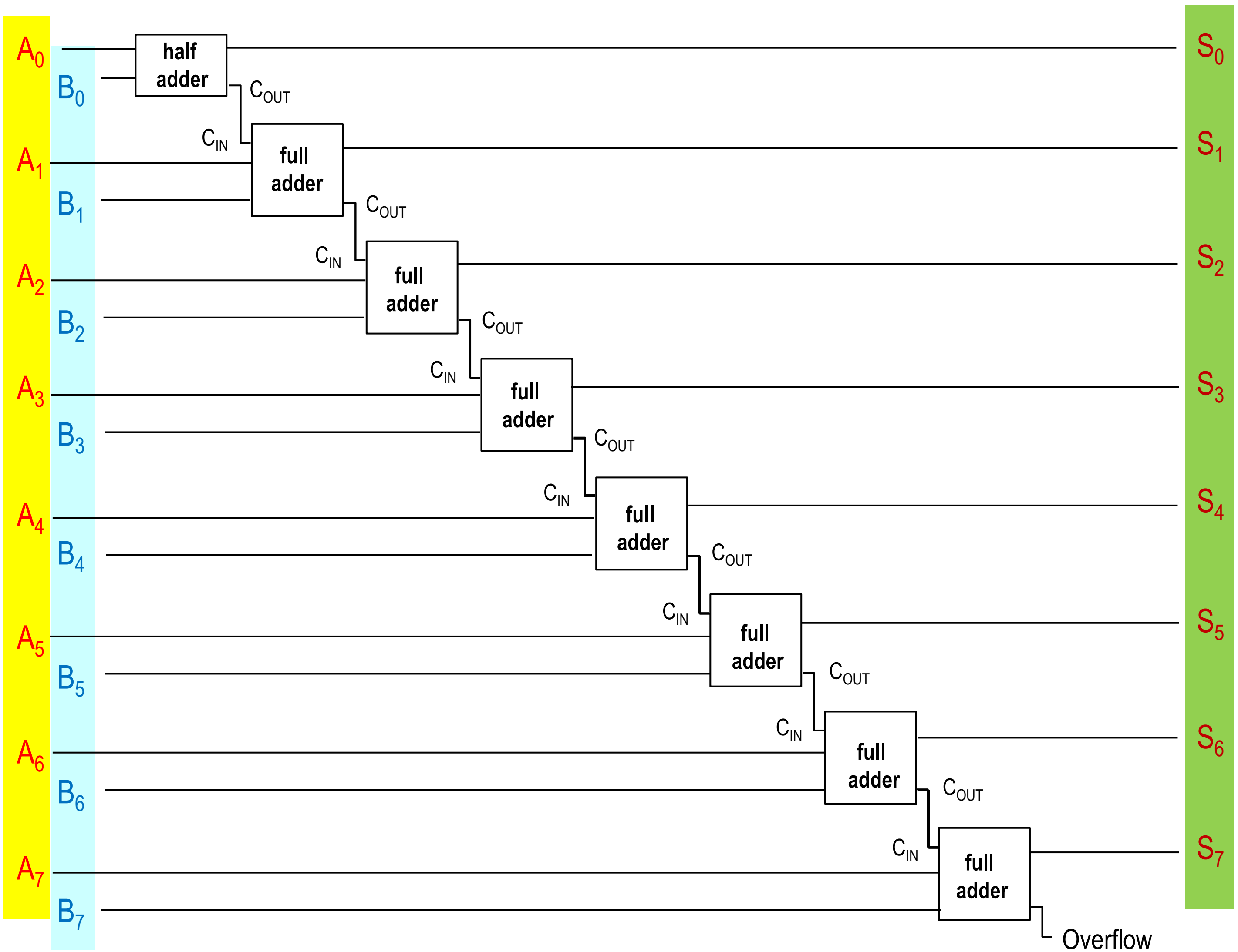
Full Adder (with input carry)



Full Adder



8-bit Adder

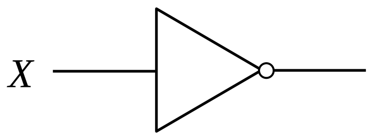

A0
B0
half adder
COUT
CIN
full adder
A1
B1
A2
B2
A3
B3
A4
B4
A5
B5
A6
B6
A7
B7
S0
S1
S2
S3
S4
S5
S6
S7
Overflow

Deriving 2s Complement Signed Representation

(Kesden draws here during lecture)

Boolean Algebra

Rules

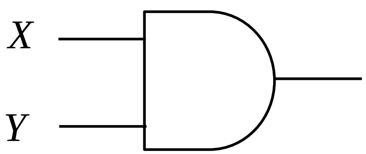


$Z = \overline{X}$

$$\begin{cases} X = 1 \Rightarrow Z = 0 \\ X = 0 \Rightarrow Z = 1 \end{cases}$$

X and $\overline{X}$ are complements

$\overline{\overline{X}} = X$



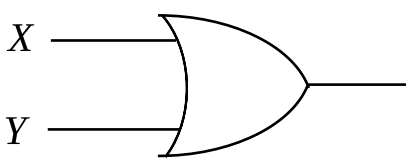
$Z = X \cdot Y$

$$\begin{cases} 0 \cdot 0 = 0 \\ 1 \cdot 0 = 0 \\ 0 \cdot 1 = 0 \\ 1 \cdot 1 = 1 \end{cases}$$

$0 \cdot X = 0$

$X \cdot \overline{X} = 0$

$1 \cdot X = X$



$Z = X + Y$

$$\begin{cases} 0 + 0 = 0 \\ 1 + 0 = 1 \\ 0 + 1 = 1 \\ 1 + 1 = 1 \end{cases}$$

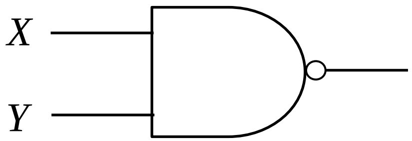
$0 + X = X$

$1 + X = 1$

$X + \overline{X} = 1$

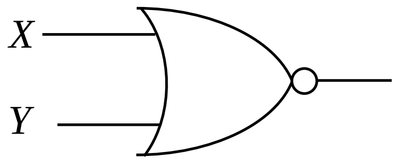
$X + Y = Y + X$

Boolean Algebra



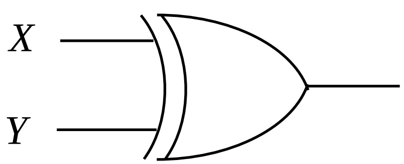
$$Z = \overline{X \cdot Y}$$

$$\begin{cases} \overline{0 \cdot 0} = \overline{0} = 1 \\ \overline{1 \cdot 0} = \overline{0} = 1 \\ \overline{0 \cdot 1} = \overline{0} = 1 \\ \overline{1 \cdot 1} = \overline{1} = 0 \end{cases}$$



$$Z = \overline{X + Y}$$

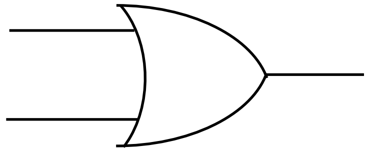
$$\begin{cases} \overline{0 + 0} = \overline{0} = 1 \\ \overline{1 + 0} = \overline{1} = 0 \\ \overline{0 + 1} = \overline{1} = 0 \\ \overline{1 + 1} = \overline{1} = 0 \end{cases}$$

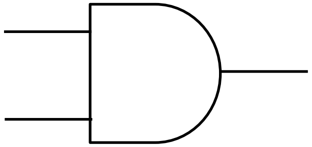


$$Z = X \oplus Y$$

$$\begin{cases} 0 \oplus 0 = 0 \\ 1 \oplus 0 = 1 \\ 0 \oplus 1 = 1 \\ 1 \oplus 1 = 0 \end{cases}$$

Rules for Boolean Algebra





1. $0 + X = X$
2. $1 + X = 1$
3. $X + X = X$
4. $X + \overline{X} = 1$
5. $0 \cdot X = 0$
6. $1 \cdot X = X$
7. $X \cdot X = X$
8. $X \cdot \overline{X} = 0$
9. $\overline{\overline{X}} = X$
10. $X + Y = Y + X$ Commutative Law
11. $X \cdot Y = Y \cdot X$ Commutative Law
12. $X + (Y + Z) = (X + Y) + Z$
13. $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$
14. $X \cdot (Y + Z) = X \cdot Y + X \cdot Z$ Distributive Law
15. $X + X \cdot Z = X$ Absorption Law

De Morgan's Laws

$$\overline{(X+Y)} = \bar{X} \cdot \bar{Y}$$

$$\overline{(X \cdot Y)} = \bar{X} + \bar{Y}$$

- Any logic function can be implemented using only **OR** and **NOT** gates.
- Any logic function can be implemented using only **AND** and **NOT** gates.

Boolean Algebra Example — Fail-Safe Autopilot Logic

Prior to takeoff or landing maneuver, a commercial aircraft requires the following check:

2** of the possible **3 pilots** must be available**: the pilot, the copilot and the autopilot

Set Logic Variables:

X – State of the pilot: $1=$ Present, $0=$Absent

Y – State of the copilot: $1=$ Present, $0=$Absent

Z – State of the autopilot: $1=$ Functioning, $0=$Not Functioning

Logic function corresponding to "System Ready" is

$$f = X \text{ and } Y$$

$$f = X \cdot Y + X \cdot Z + Y \cdot Z$$

$f = 1$*: System Ready ;* $f = 0$ *: System Not Ready*

Boolean Algebra Example — Fail-Safe Autopilot Logic

Positive check

$$f = X \cdot Y + X \cdot Z + Y \cdot Z$$ *sum-of-product*

System Not Ready Condition:

$$\overline{f} = \overline{X \cdot Y + X \cdot Z + Y \cdot Z}$$

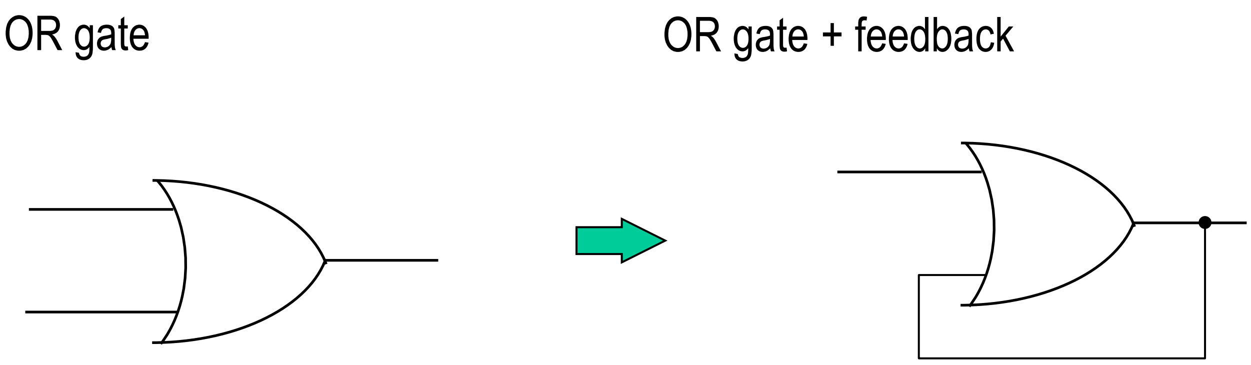
$$= \overline{(X \cdot Y)} \cdot \overline{(X \cdot Z)} \cdot \overline{(Y \cdot Z)}$$ $\overline{(X + Y)} = \overline{X} \cdot \overline{Y}$

$$= (\overline{X} + \overline{Y}) \cdot (\overline{X} + \overline{Z}) \cdot (\overline{Y} + \overline{Z})$$ $\overline{(X \cdot Y)} = \overline{X} + \overline{Y}$

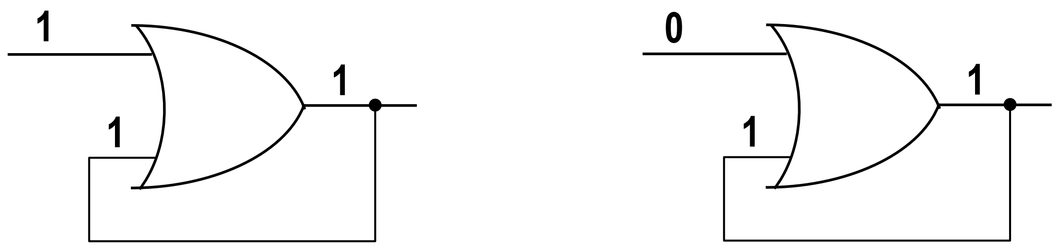
$$\overline{f} = (\overline{X} + \overline{Y}) \cdot (\overline{X} + \overline{Z}) \cdot (\overline{Y} + \overline{Z})$$

Maintaining State w/ Logic Gates: Latches that Never Forget

Logic Gates + Feedback:

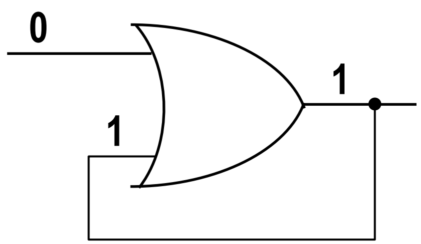


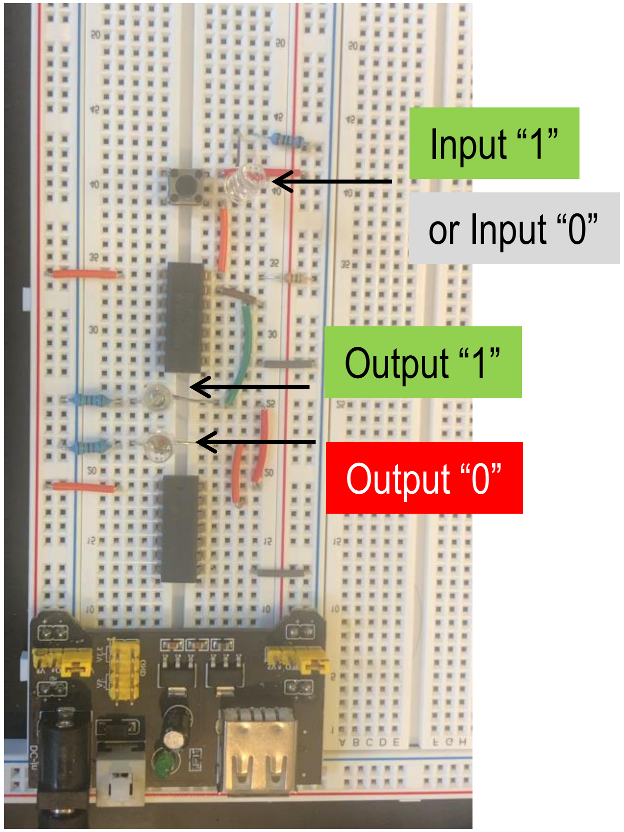
Logic Gates + Feedback: Making a Memory



It stores/memorizes "1"

Store "1"



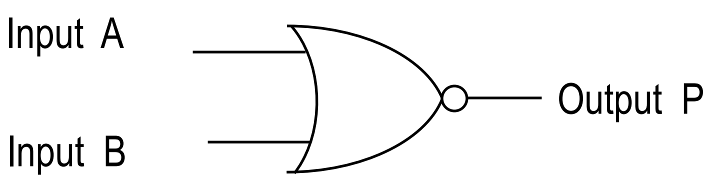


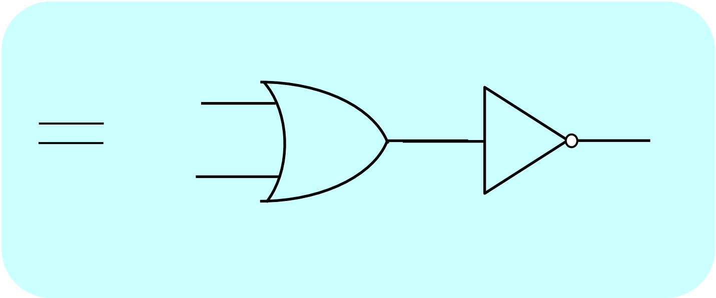
After a "1" is stored, the state cannot be changed.

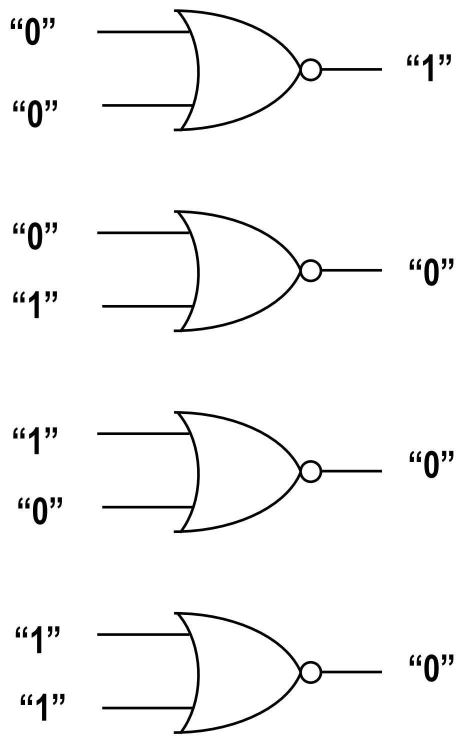
Unless the power is turned-off to reset.

RS Latches: Latches that can Reset

NOR Gate



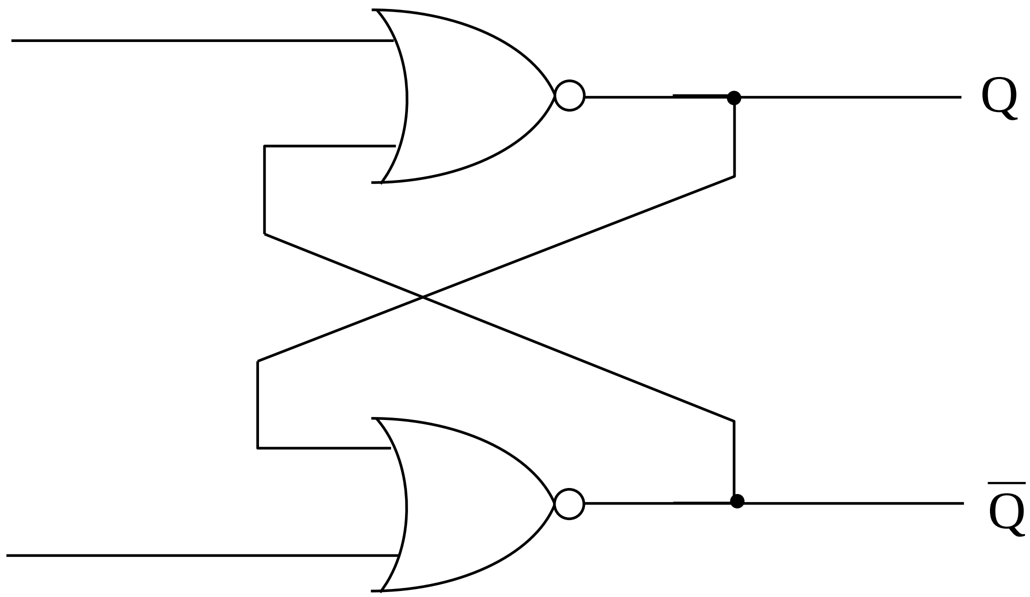




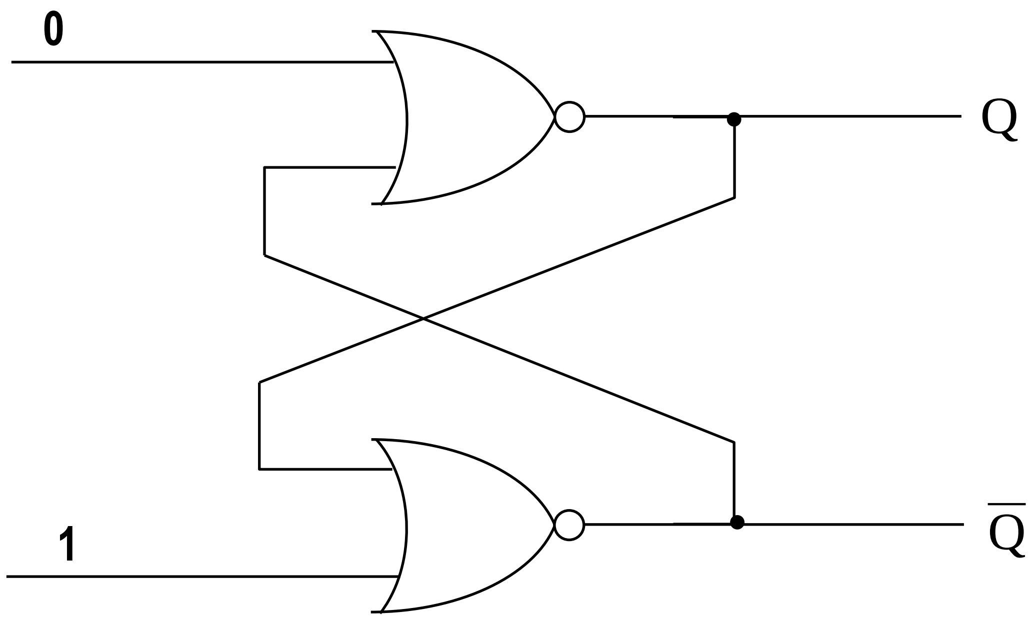
Truth Table

Input A	Input B	Output P
0	0	1
0	1	0
1	0	0
1	1	0

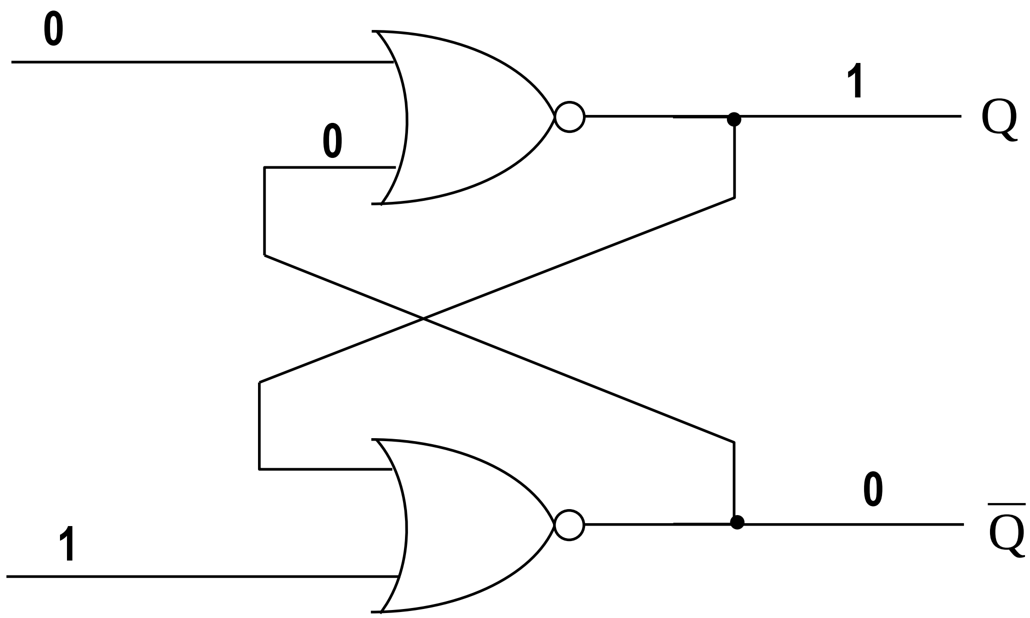
Latch with NOR Gates



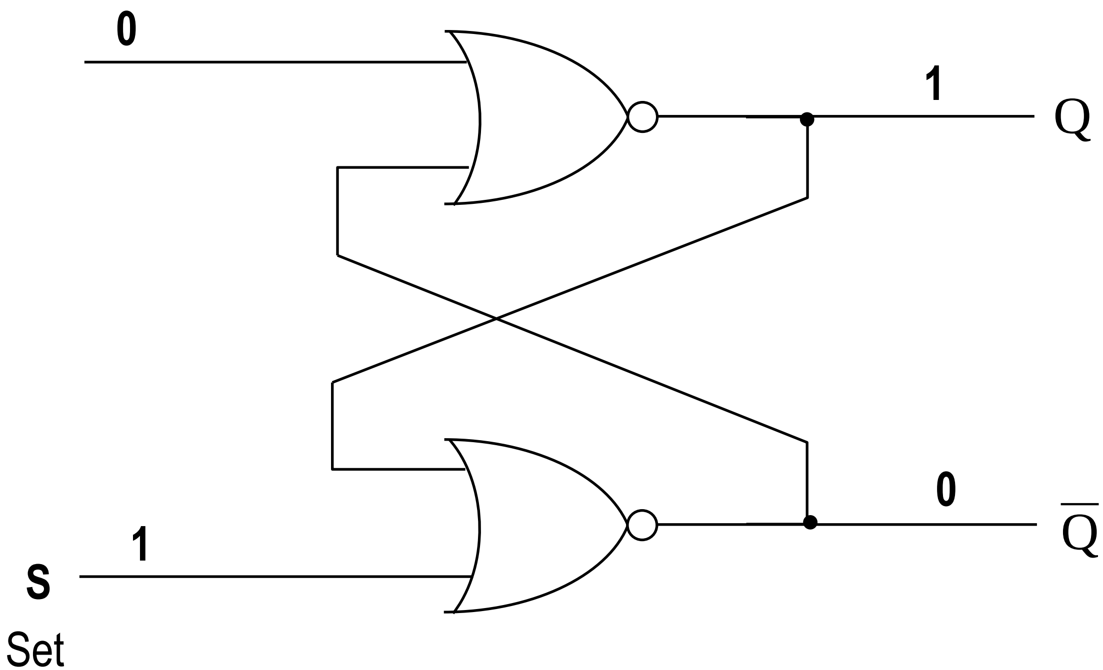
Latch with NOR Gates: Set Q to "1"



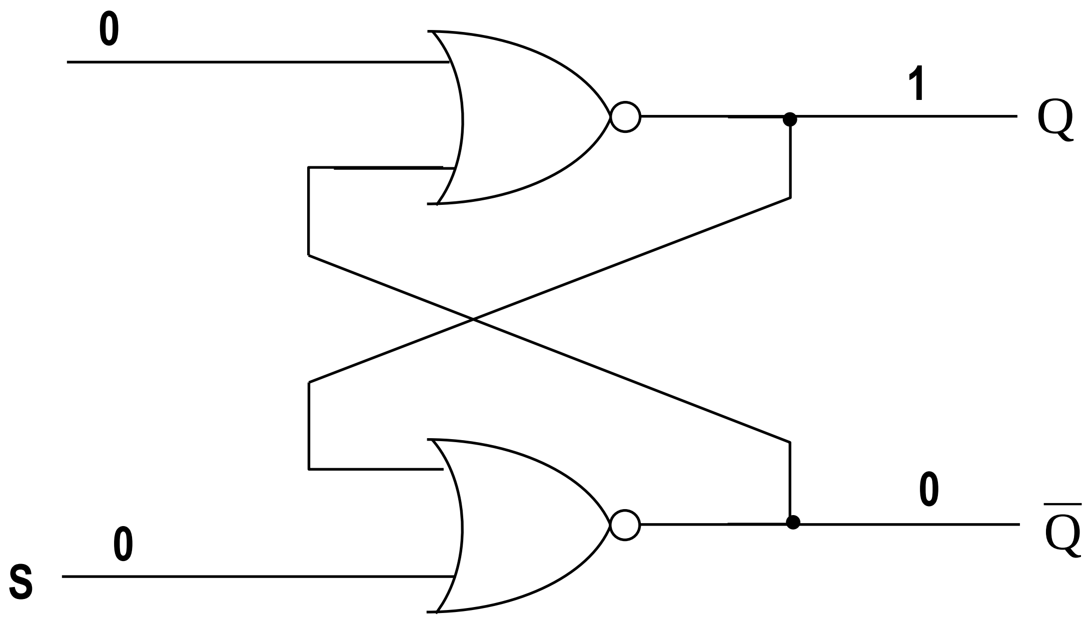
Latch with NOR Gates: Set Q to "1"



Latch with NOR Gates: Set Q to "1"

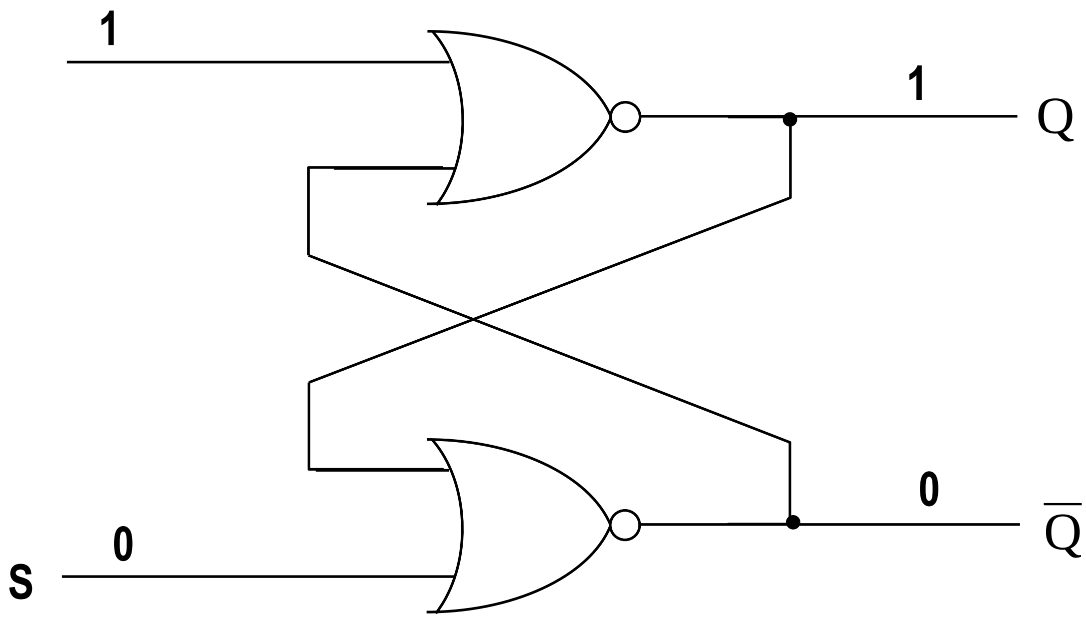


Latch with NOR Gates: Q = “1” is “Latched”

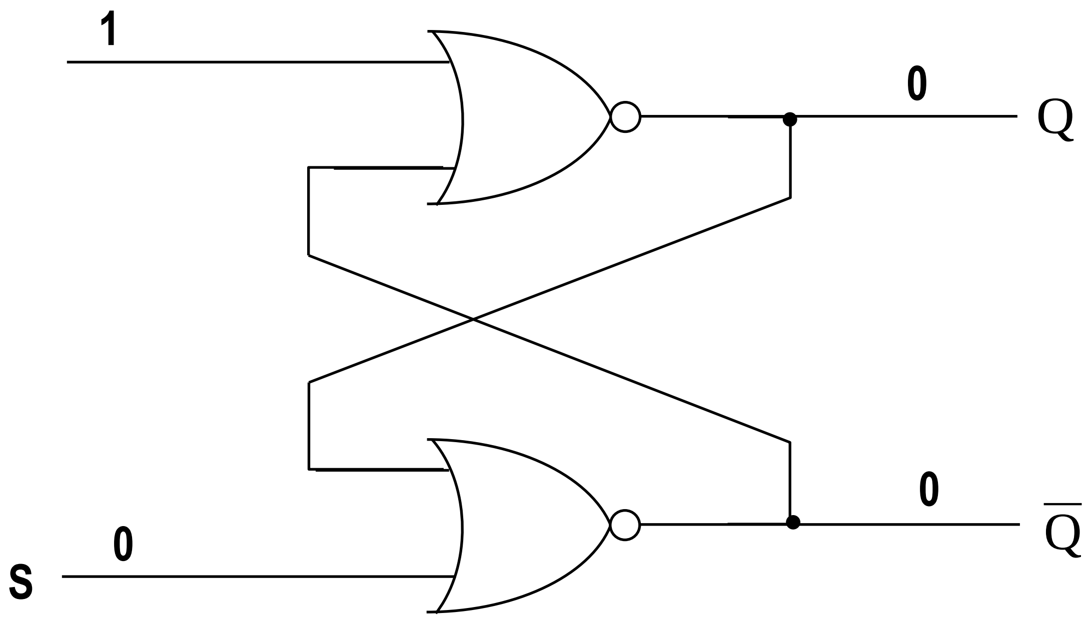


Latch with NOR Gates

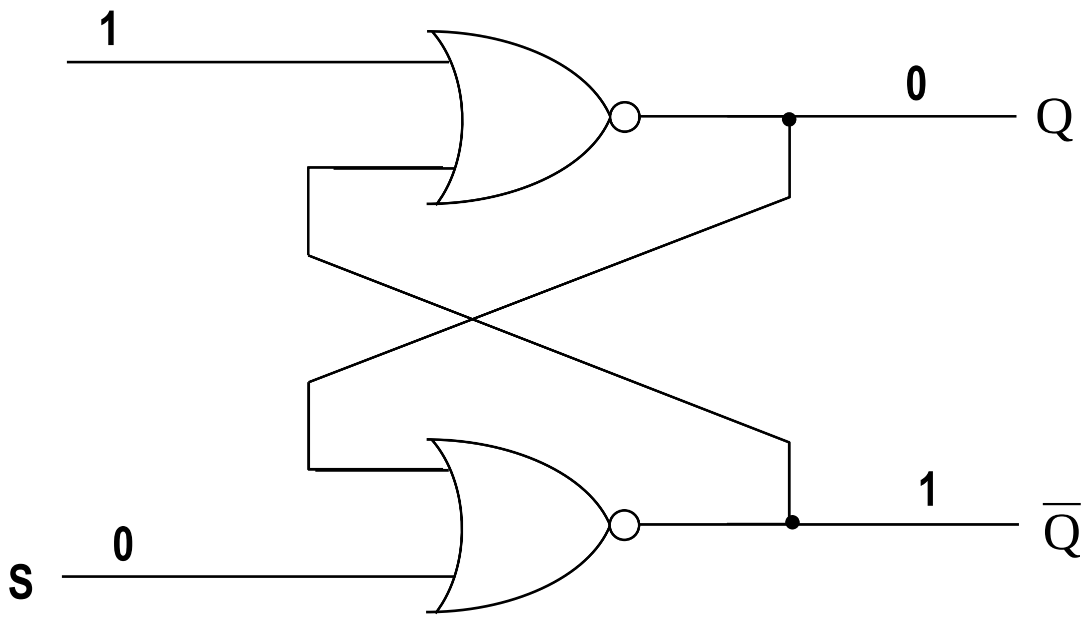
Reset Q to "0"



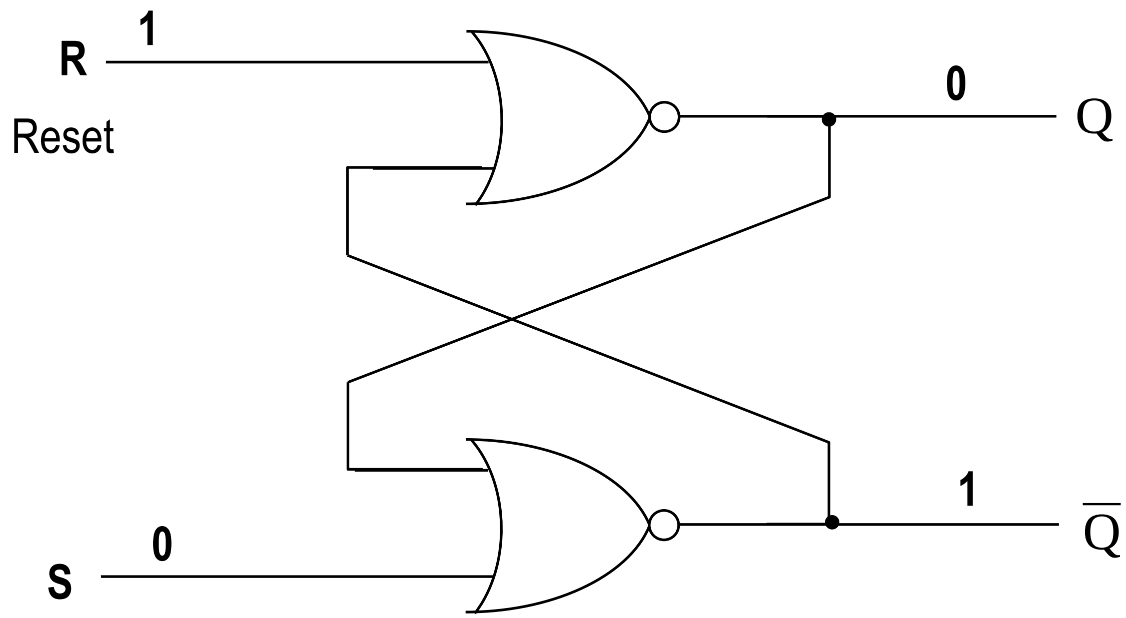
Latch with NOR Gates: Reset Q to “0”



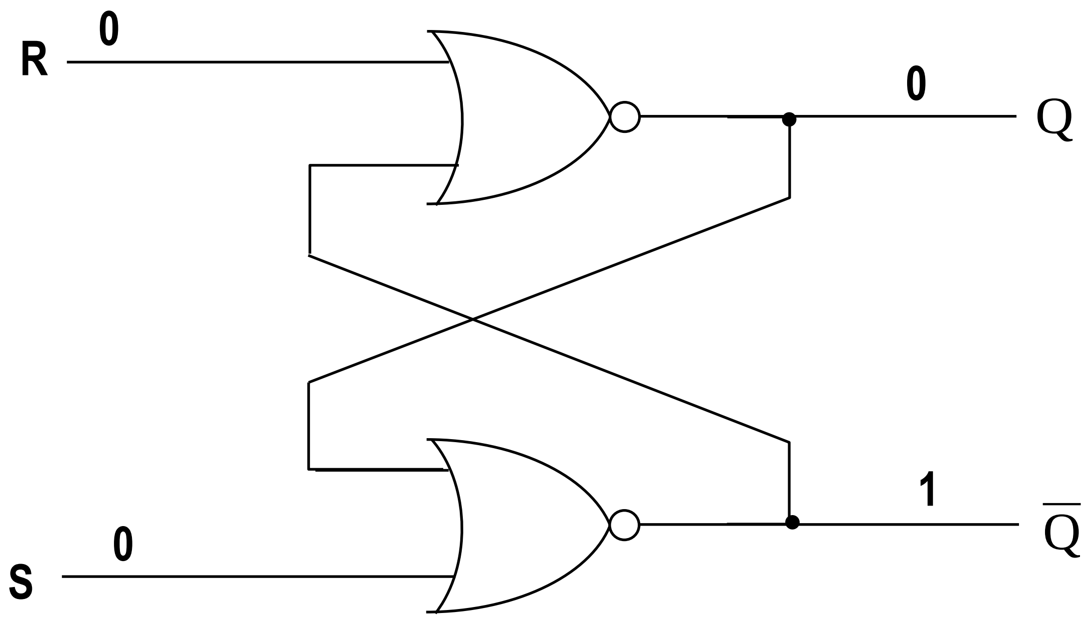
Latch with NOR Gates: Reset Q to "0"



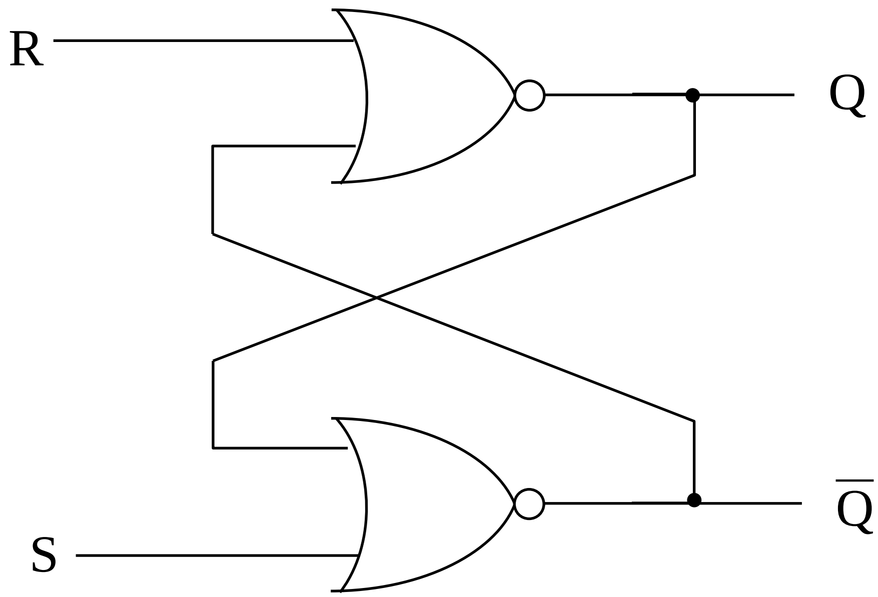
Latch with NOR Gates: Reset Q to "0"

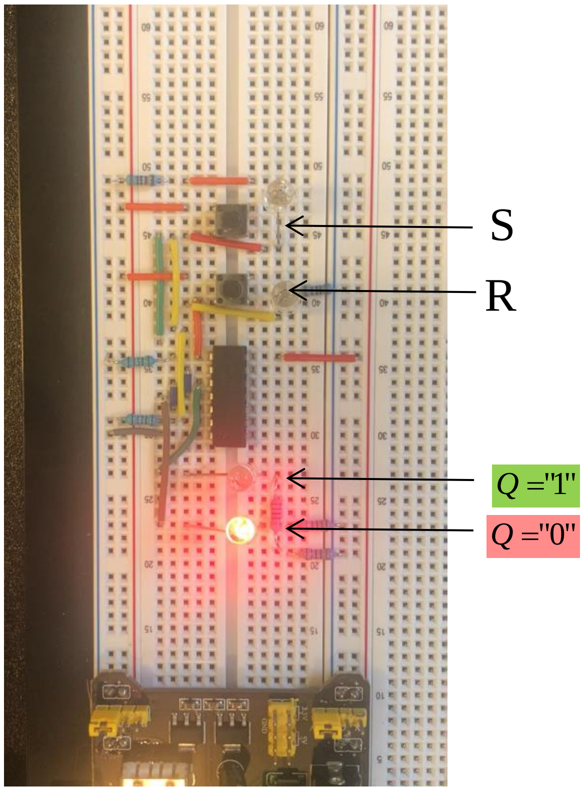


Latch with NOR Gates: Q = "0" is "Latched"



RS Latch



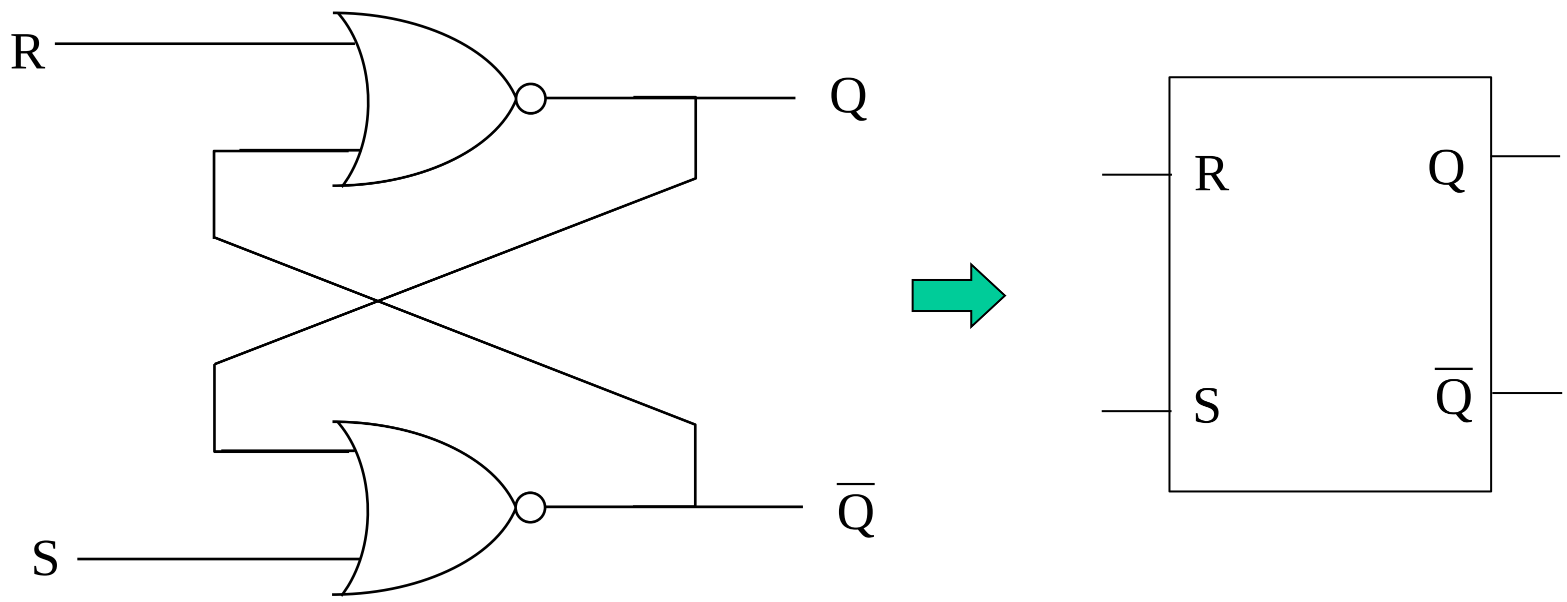


S —— Set

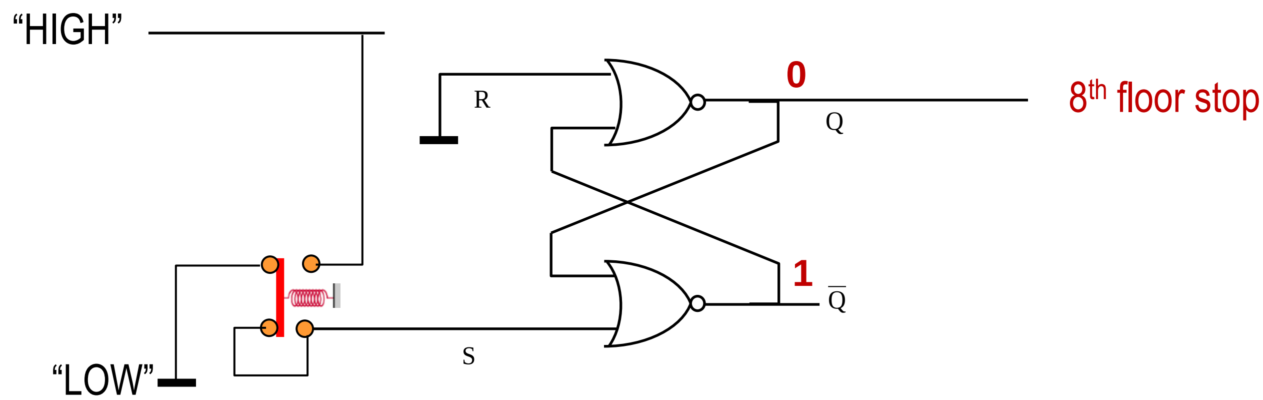
R —— Reset

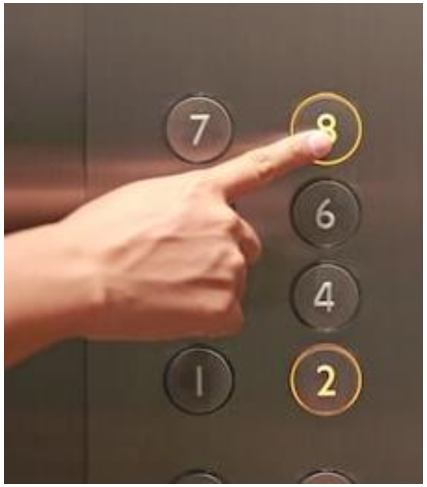
R	S	Q
0	0	no change
0	1	1
1	0	0
1	1	illegal

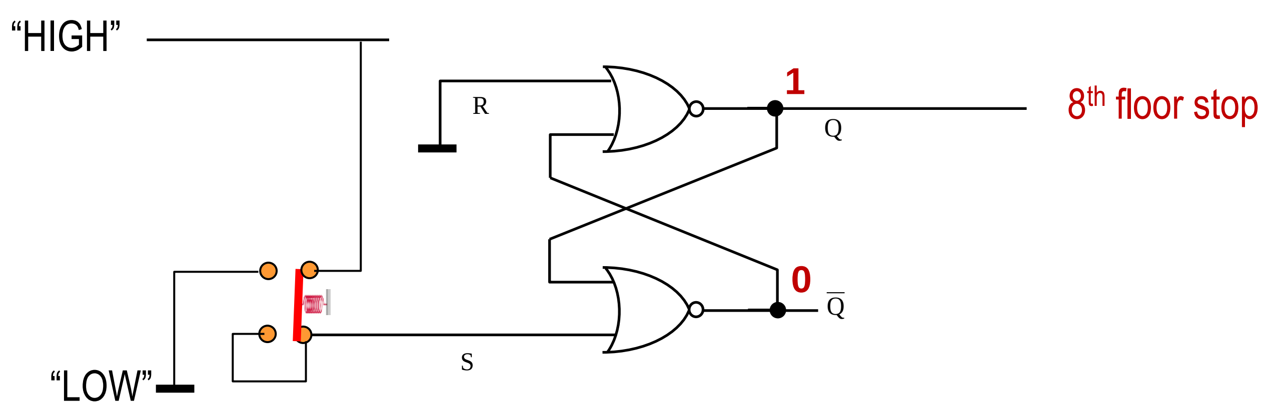
RS Latch



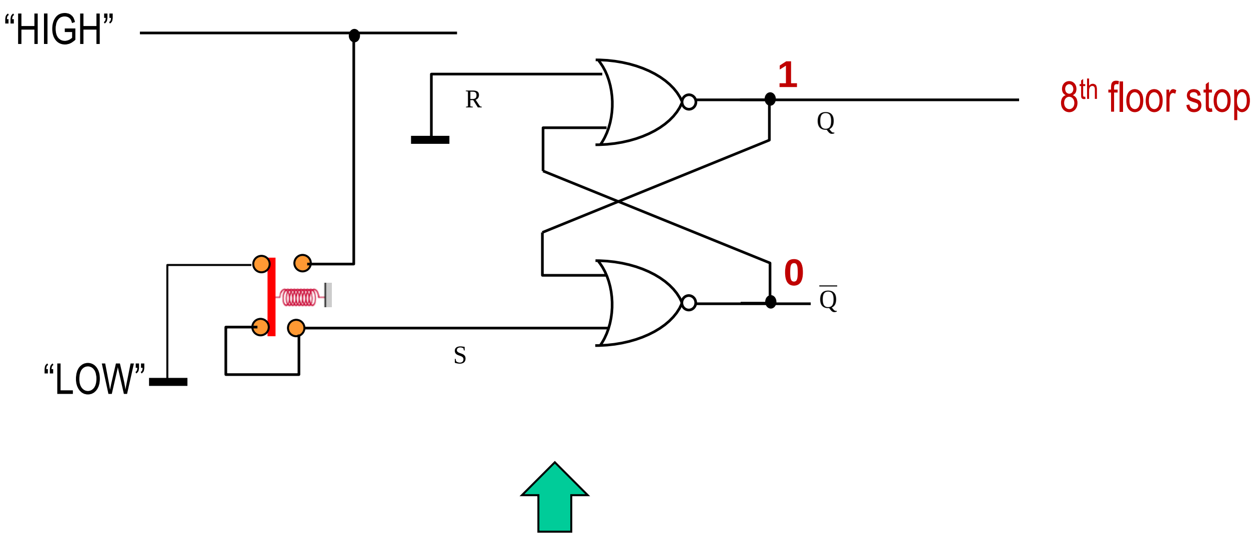
Using Latch to Memorize/Store "Request"

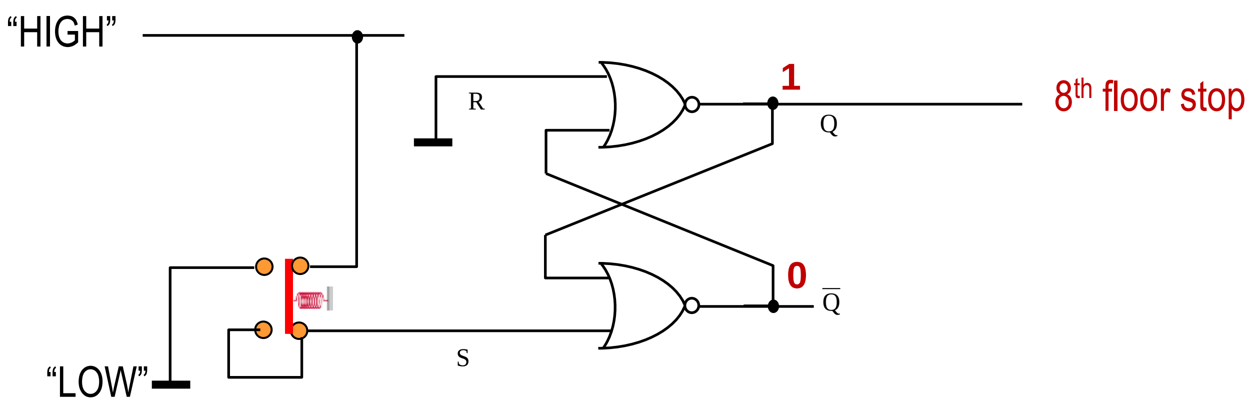




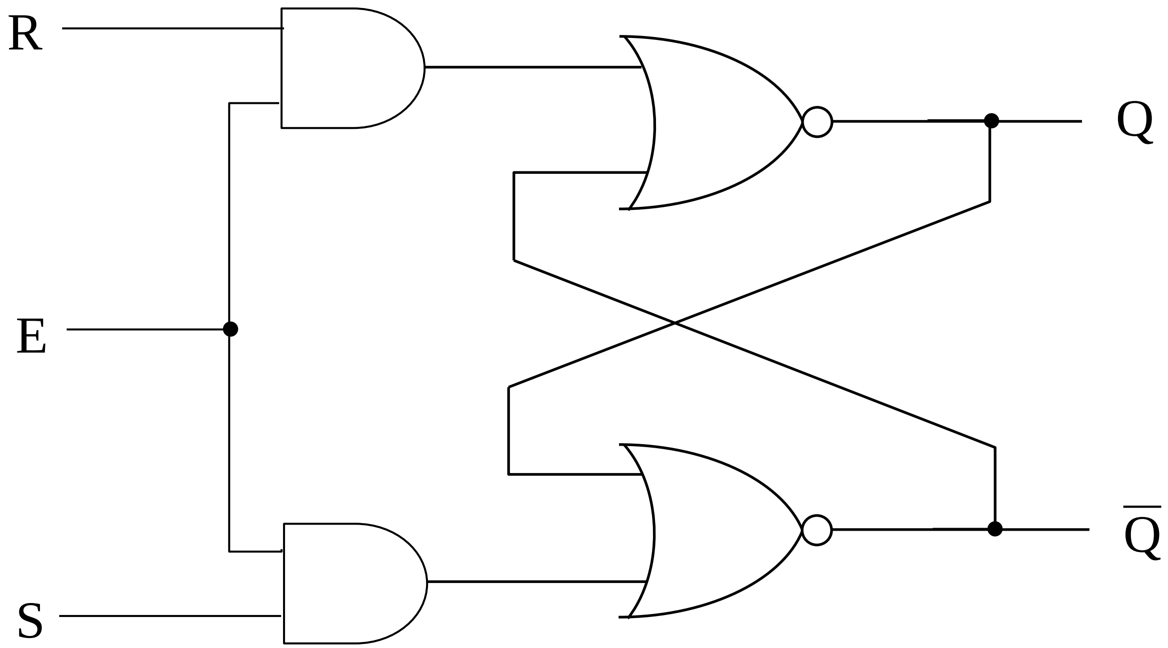


The Use of Latch





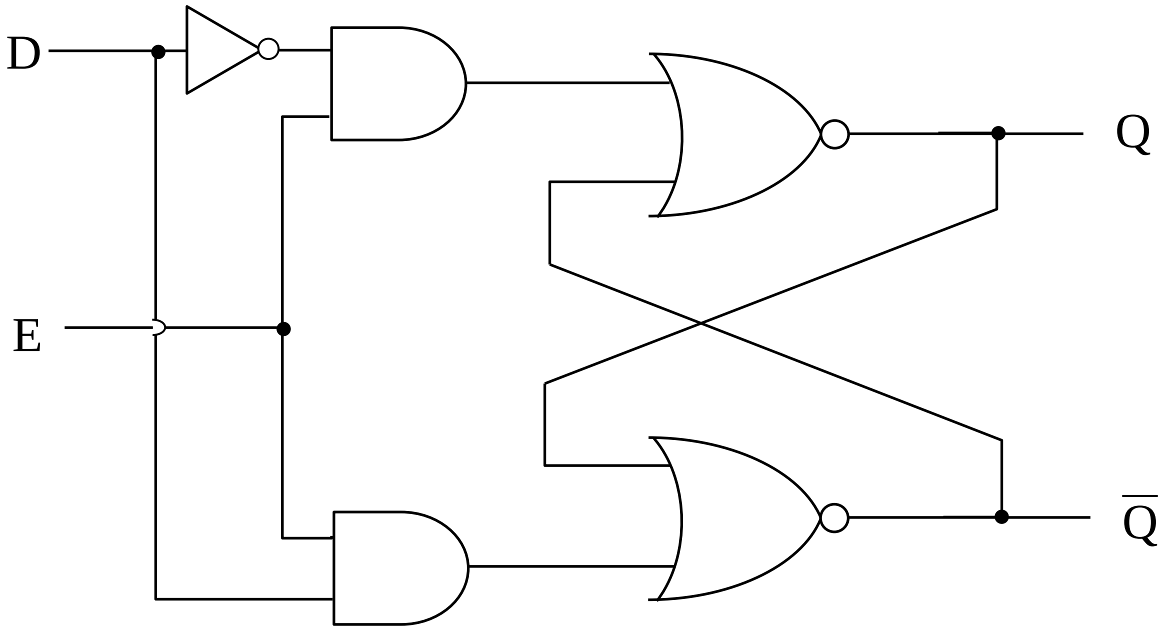
Gated SR Latch (with Enable)

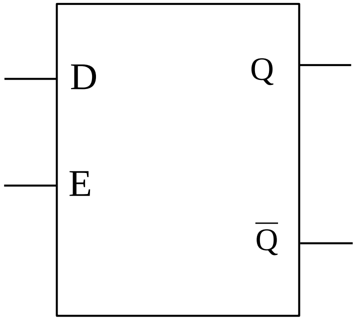


E —— Enable

D Latches and 1-Bit Memories

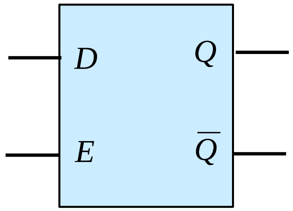
D-Latch

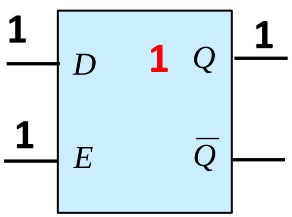


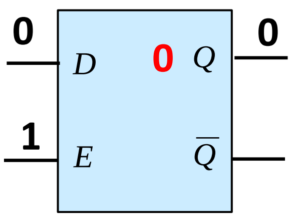


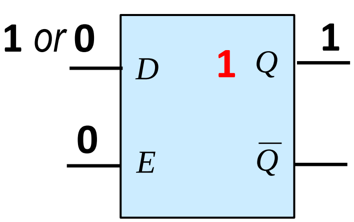
E	D	Q
0	0	Latched/Stored
0	1	Latched/Stored
1	0	0
1	1	1

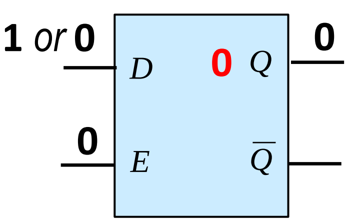
1-Bit Memory: D-Latch

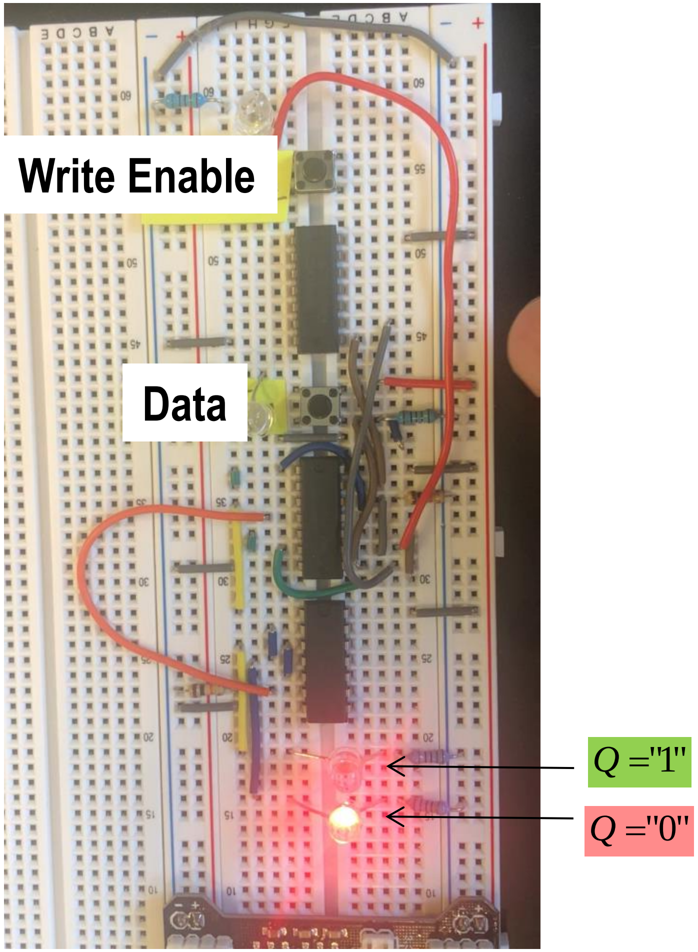












Coming Attractions

- ***Next class: Building Memory***
- ***Monday: Building a Computer!***