Programmering af Mobile Robotter

RB1-PMR – Module 8: Logic Gates

Jes Hundevadt Jepsen | jegj@mmmi.sdu.dk
SDU Drone Center, MMMI, University of Southern Denmark, Odense, Denmark

Agenda

- Recap of last module

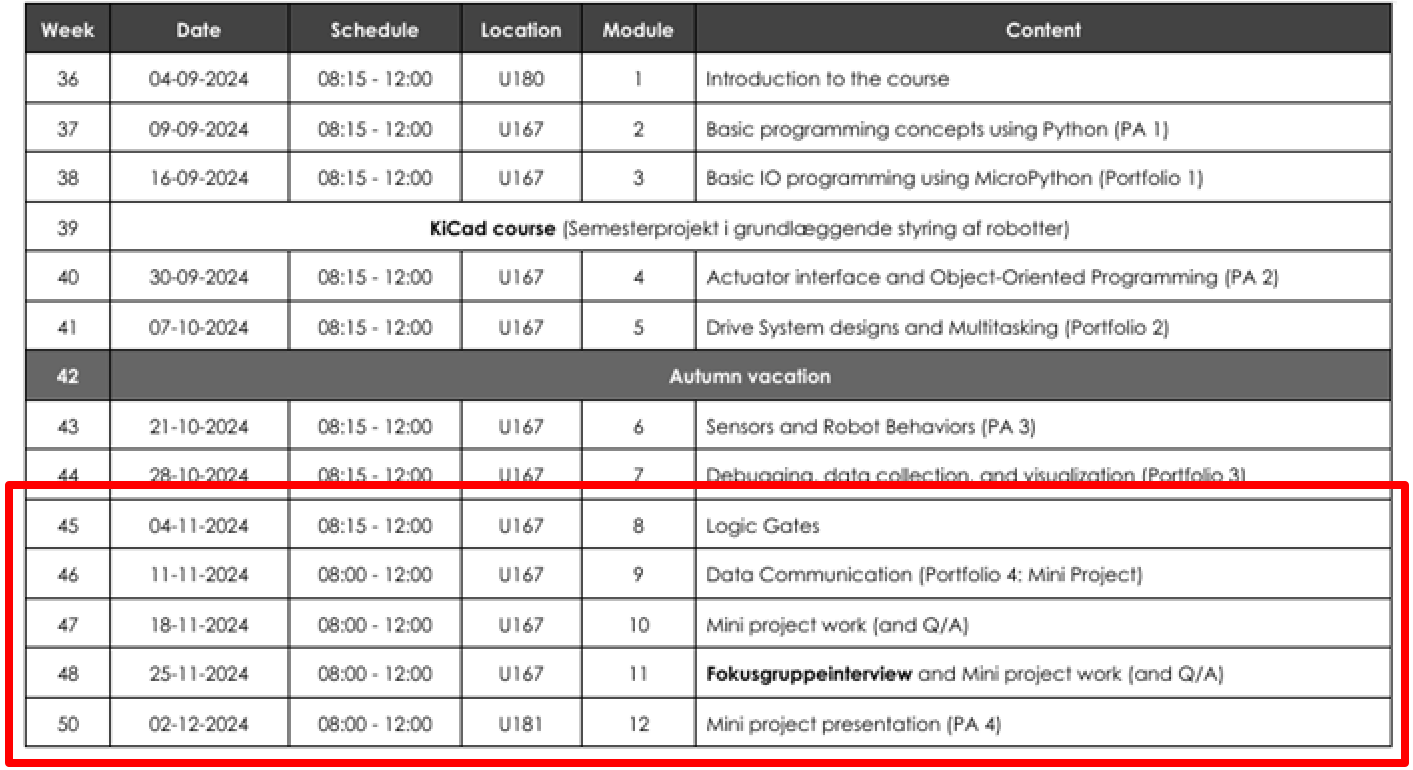
- Logic gates
  - NOT, AND, OR, NAND, NOR, XOR, XNOR

- Combinational Logic Circuits
  - Logic diagram
  - Truth Table
  - Boolean expression
  - Common examples
    - Multiplexers, decode/encoder, full adder

- The Laws of Boolean Algebra
  - How to simplify logic circuits?

- **Next modules**
  - **Next:** Data communication (and Mini Project)
    - Q/A's? (2 modules + Mini Project work)

  - **Remember:** Fokusgruppeinterview (25/11, 8.15 - 9.15)



Week	Date	Schedule	Location	Module	Content
36	04-09-2024	08:15 - 12:00	U180	1	Introduction to the course
37	09-09-2024	08:15 - 12:00	U167	2	Basic programming concepts using Python (PA 1)
38	16-09-2024	08:15 - 12:00	U167	3	Basic IO programming using MicroPython (Portfolio 1)
39	**KiCad course** (Semesterprojekt i grundlæggende styring af robotter)				
40	30-09-2024	08:15 - 12:00	U167	4	Actuator interface and Object-Oriented Programming (PA 2)
41	07-10-2024	08:15 - 12:00	U167	5	Drive System designs and Multitasking (Portfolio 2)
42	Autumn vacation				
43	21-10-2024	08:15 - 12:00	U167	6	Sensors and Robot Behaviors (PA 3)
44	28-10-2024	08:15 - 12:00	U167	7	Debugging, data collection, and visualization (Portfolio 3)
45	04-11-2024	08:15 - 12:00	U167	8	Logic Gates
46	11-11-2024	08:00 - 12:00	U167	9	Data Communication (Portfolio 4: Mini Project)
47	18-11-2024	08:00 - 12:00	U167	10	Mini project work (and Q/A)
48	25-11-2024	08:00 - 12:00	U167	11	**Fokusgruppeinterview** and Mini project work (and Q/A)
50	02-12-2024	08:00 - 12:00	U181	12	Mini project presentation (PA 4)

Recap

- Troubleshooting and Debugging Techniques (continued)
    - Best-practice
        - Basic debugging using Python
        - LED and GPIO for Debugging
        - Data Logging using MicroPython

- Data collection using a Microcontroller
    - Types of data collect
    - End-to-end data collection (design)
        - **Define the objective**
        - **Design the data collection method**
        - **Testing and collect the data**
        - **Analyze and Interpret the Data**

- Visualization of data measurements
    - …using Matplotlib

- Portfolio 3: Data collection of sensors
    - Q/A?
    - …consider storing everything on a container (module 2)
    - …when done with the test, write everything to

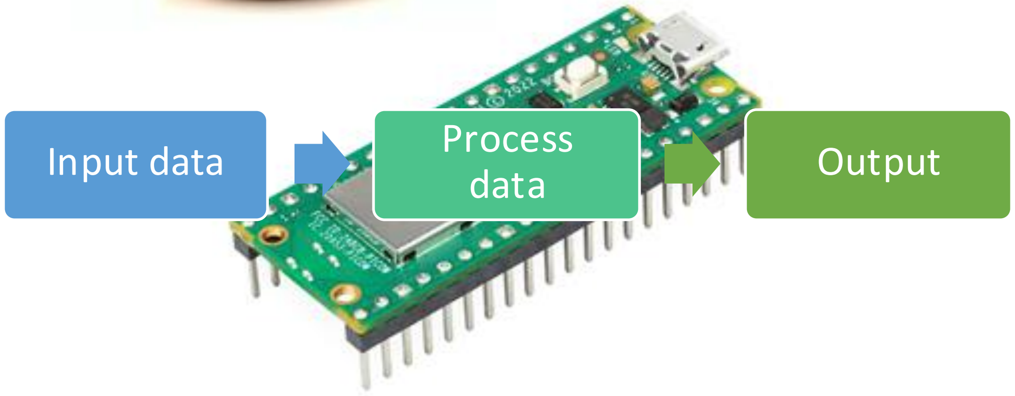
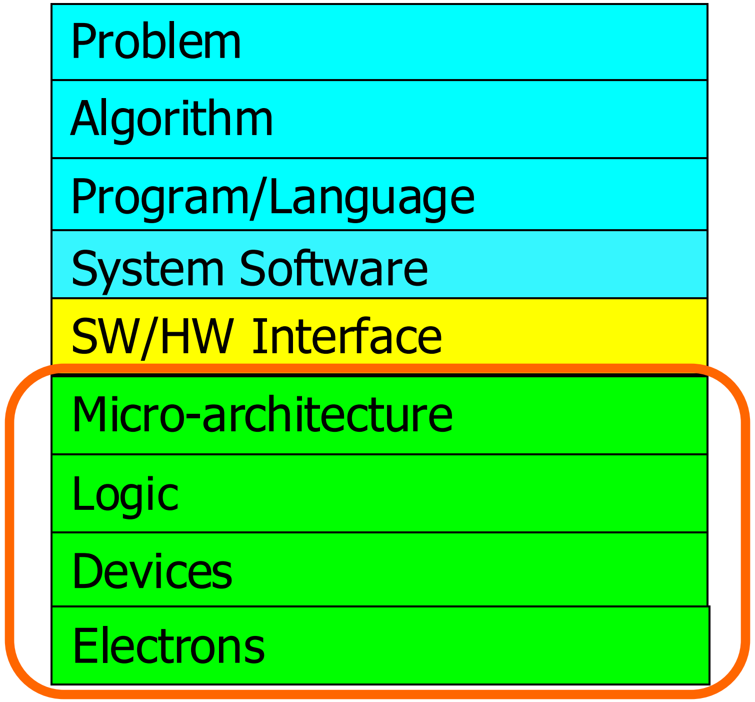


Illustration: from Google Images

Logic Gates

CPU (Central Processing Unit) [module 1]:

- The brain of the microcontroller, responsible for executing instructions.
  - **Control Unit**
  - **Arithmetic Logic Unit (ALU)**
  - **Registers**
  - **Memory**



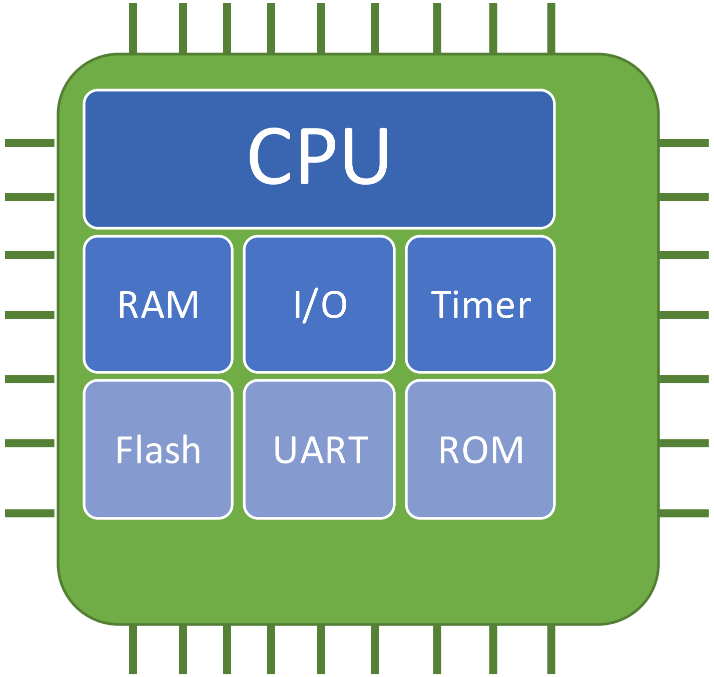
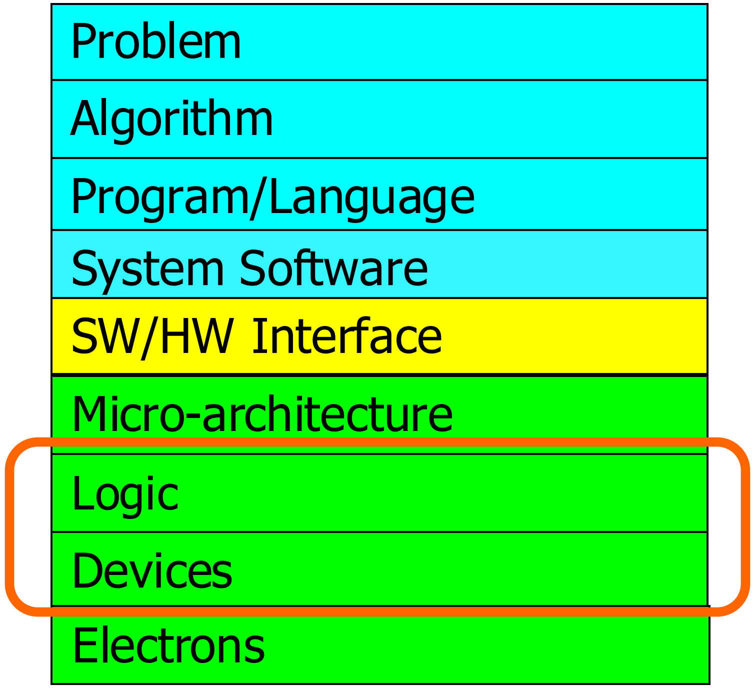
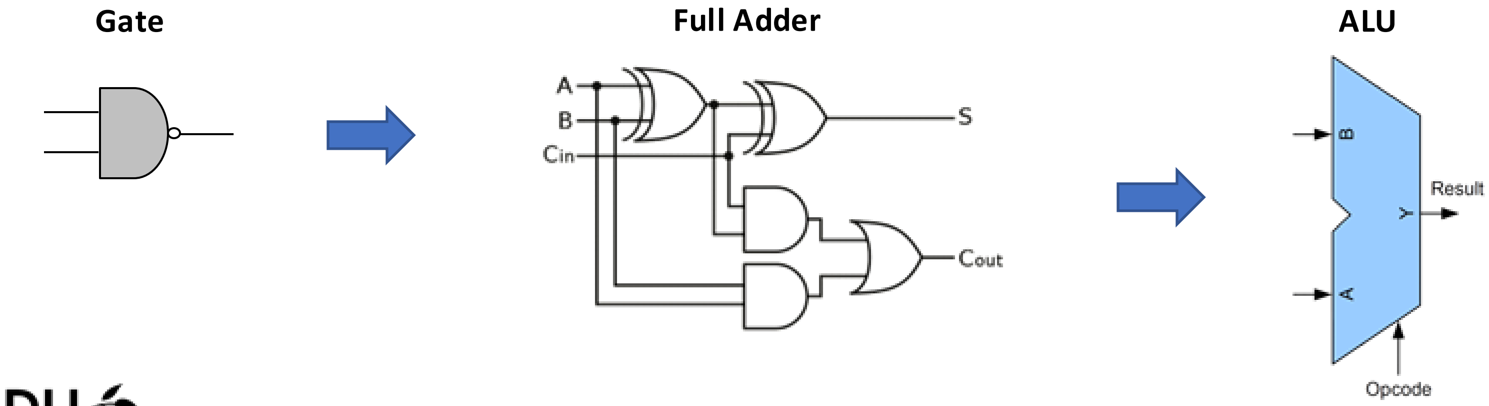


Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

CPU (Central Processing Unit) [module 1]:

- The brain of the microcontroller, responsible for executing instructions.
  - **Control Unit**
  - **Arithmetic Logic Unit (ALU)**
  - **Registers**
  - **Memory**
  - ….all build on logic gates…


Problem
Algorithm
Program/Language
System Software
SW/HW Interface
Micro-architecture
Logic
Devices
Electrons



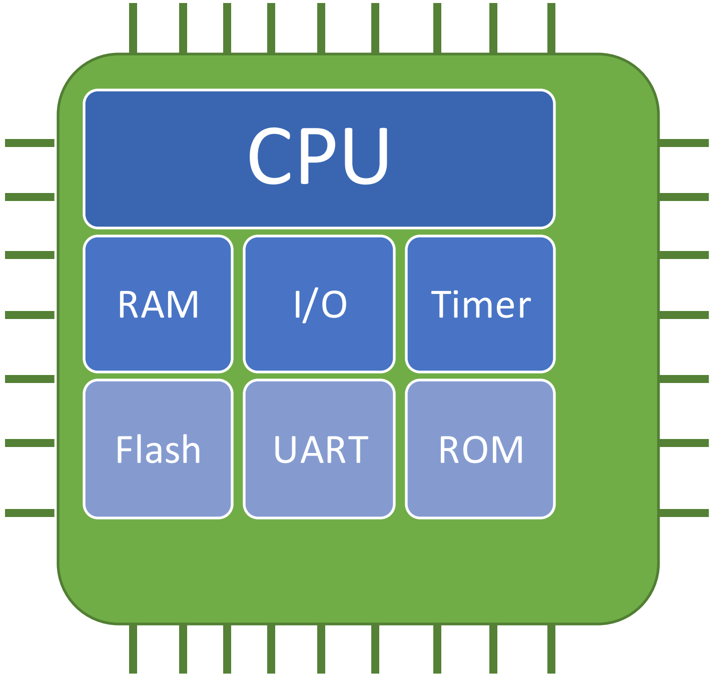


Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

Logic gates

> *“Is a device which accepts one or more **TRUE (1)** or **FALSE (0)** inputs and produces a **TRUE (1)** or **FALSE (0)** output according to a logical rule.”*

- **Binary numbers (Module 1)**
  - …are expressed using only two digits: 0 and 1.

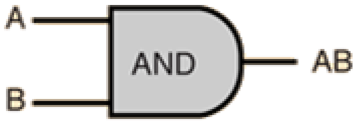
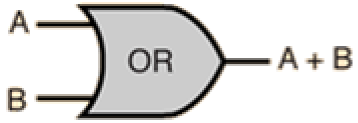
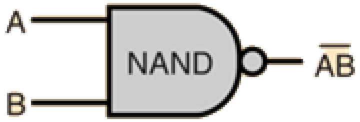
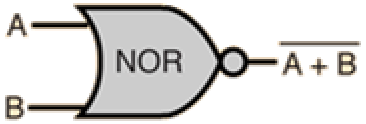
Example: Different basic logic gates	
AND	OR
	
NAND	NOR
	

Illustration: http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/trangate.html#c1

Logic gates

> *“Is a device which accepts one or more **TRUE (1)** or **FALSE (0)** inputs and produces a **TRUE (1)** or **FALSE (0)** output according to a logical rule.”*

- **Binary numbers (Module 1)**
  - …are expressed using only two digits: 0 and 1.

- **Logic gates:** Low-level basic building blocks of digital systems.
  - Input: TRUE (1) or FALSE (0)
  - Output: TRUE (1) or FALSE (0)

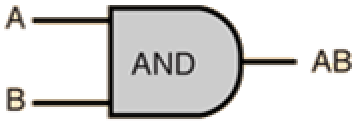
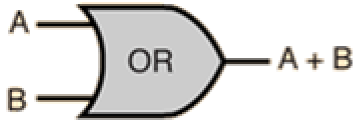
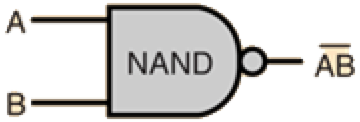
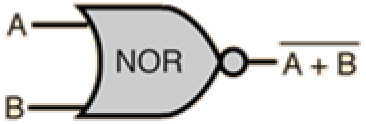
Example: Different basic logic gates	
AND	OR
	
NAND	NOR
	

Illustration: http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/trangate.html#c1

Logic gates

*“Is a device which accepts one or more **TRUE (1)** or **FALSE (0)** inputs and produces a **TRUE (1)** or **FALSE (0)** output according to a logical rule.”*

- **Binary numbers (Module 1)**
  - …are expressed using only two digits: 0 and 1.
- **Logic gates:** Low-level basic building blocks of digital systems.
  - Input: TRUE (1) or FALSE (0)
  - Output: TRUE (1) or FALSE (0)
- **Combinational Logic Circuits:** Multiple interconnected logic gates.
  - …for <u>performing more advanced operations</u> by combining the outputs of individual gates…
  - ..eg. encoders, full adder, memory, CPU, etc.

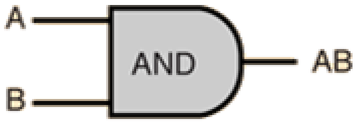
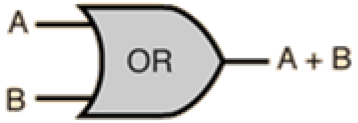
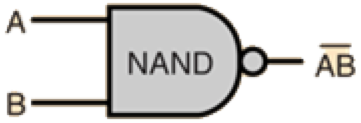
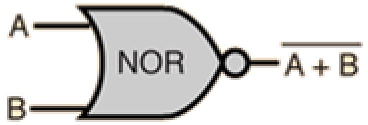
Example: Different basic logic gates	
AND	OR
	
NAND	NOR
	

Illustration: http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/trangate.html#c1

Logic gates

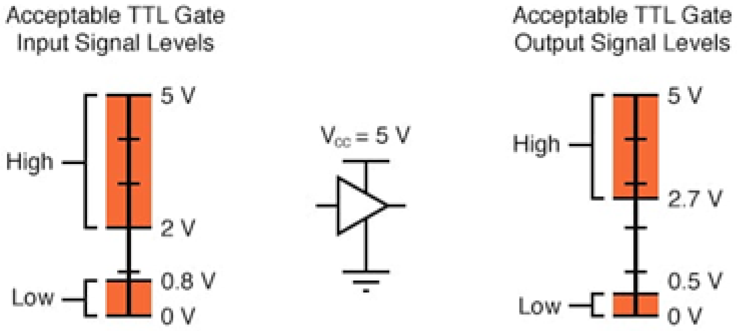
> *“Is a device which accepts one or more TRUE (1) or FALSE (0) inputs and produces a TRUE (1) or FALSE (0) output according to a logical rule.”*

- In microcontroller: Binary digits represent voltage levels
  - (e.g., 0V for 0, 5V for 1),

Build from transistors: the most basic building block of the digital system

- Intel 8051 processor (1980) one of the first processors in the market has 50,000 transistor in it.
- Apple M1 Max processor (2021) has more than 57,000,000,000 transistors

“Acceptable” input signal voltages = TRUE/High or FALSE/Low


Acceptable TTL Gate Input Signal Levels
5 V
High
2 V
Low
0.8 V
0 V
V_{CC} = 5 V
Acceptable TTL Gate Output Signal Levels
5 V
High
2.7 V
Low
0.5 V
0 V

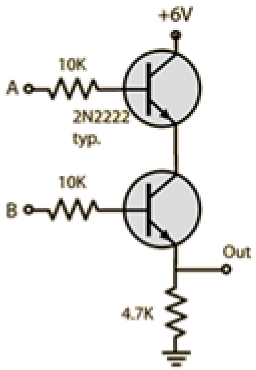
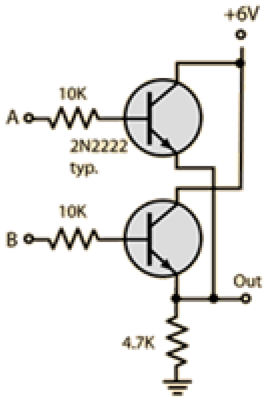
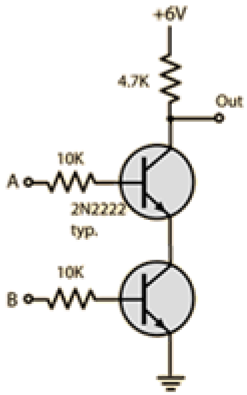
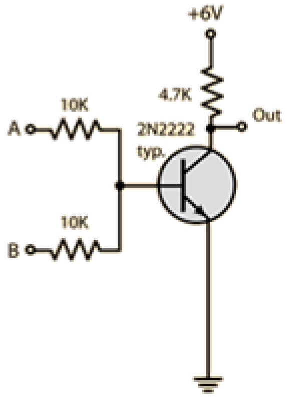
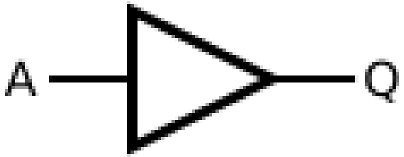
Example: Different basic logic gates (transistor switches)	
AND	OR
	
NAND	NOR
	

Illustration: http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/trangate.html#c1

Logic gates

“Is a device which accepts one or more TRUE (1) or FALSE (0) inputs and produces a TRUE (1) or FALSE (0) output according to a logical rule.”

- **Buffer**
  - One input
  - Outputs its input



Input	Output
A	Q
0	0
1	1

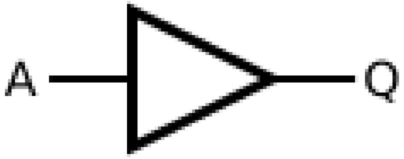
Illustration: https://en.wikipedia.org/wiki/Logic_gate

Logic gates

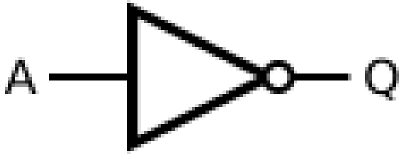
> *“Is a device which accepts one or more TRUE (1) or FALSE (0) inputs and produces a TRUE (1) or FALSE (0) output according to a logical rule.”*

- **Buffer**
  - One input
  - Outputs its input

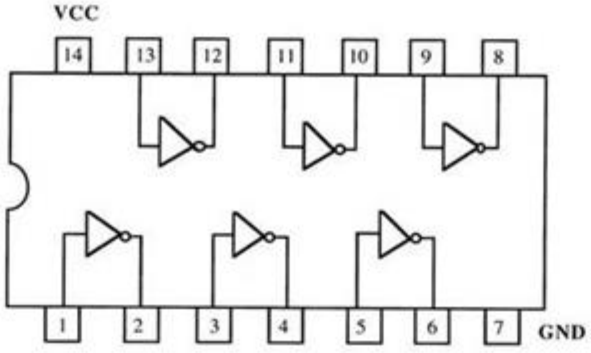
- **Inverting buffer** (Inverter / NOT gate)
  - One input
  - Outputs the opposite of its input
  - **Inversion:** Marked with a dot



Input	**Output**
A	Q
0	0
1	1



Input	**Output**
A	Q
0	1
1	0



7404

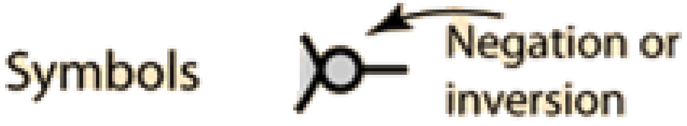


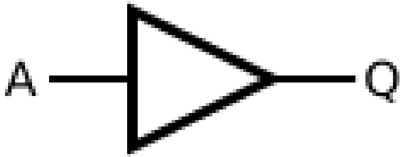
Illustration: https://en.wikipedia.org/wiki/Logic_gate

Logic gates

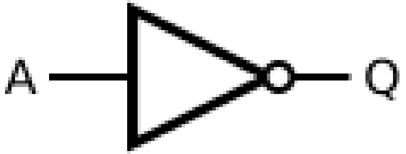
> *“Is a device which accepts one or more TRUE (1) or FALSE (0) inputs and produces a TRUE (1) or FALSE (0) output according to a logical rule.”*

- **Buffer**
  - One input
  - Outputs its input

- **Inverting buffer** (Inverter / NOT gate)
  - One input
  - Outputs the opposite of its input
  - **Inversion:** Marked with a dot



Input	Output
A	Q
0	0
1	1



Input	Output
A	Q
0	1
1	0

Truth table

> *“Shows what is the logic output of the circuit for each possible input”*

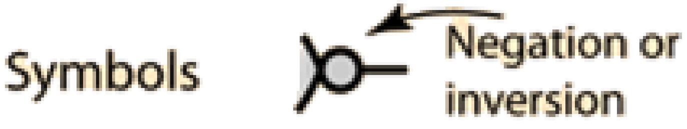
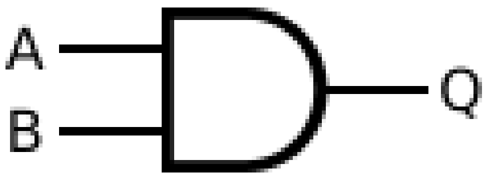


Illustration: https://en.wikipedia.org/wiki/Logic_gate

Logic gates

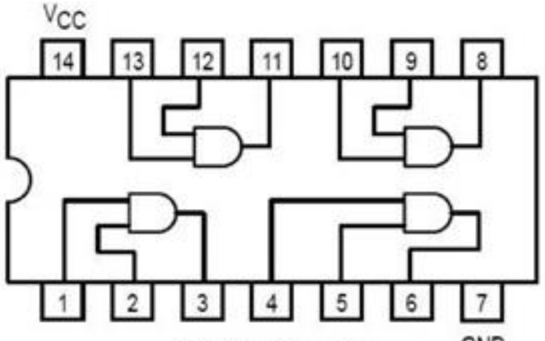
> *"Is a device which accepts one or more TRUE (1) or FALSE (0) inputs and produces a TRUE (1) or FALSE (0) output according to a logical rule."*

- **AND Gate**
  - The AND operation is done on at least two inputs
  - The AND operation on A and B represented as A.B
    - Outputs only TRUE is all inputs are TRUE
    - **Example:** Q=1 if A AND B are 1



A **AND** B

Input		Output
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1



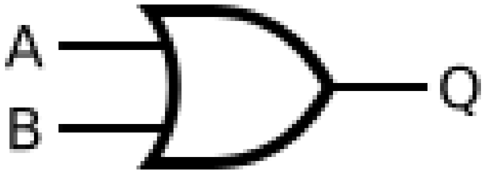
7408 Pinout

Illustration: https://en.wikipedia.org/wiki/Logic_gate

Logic gates

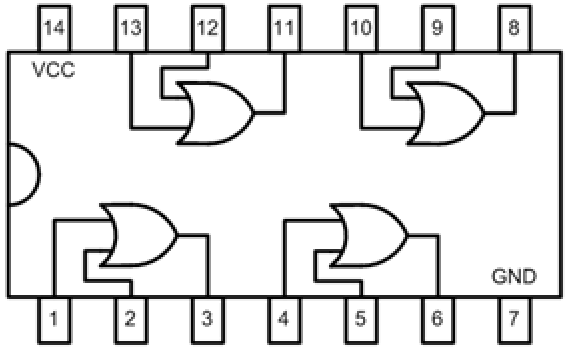
> *"Is a device which accepts one or more TRUE (1) or FALSE (0) inputs and produces a TRUE (1) or FALSE (0) output according to a logical rule."*

- **OR Gate**
  - The OR operation is done on at least two inputs
  - The OR operation on A and B represented as A+B
    - Outputs TRUE is one of the inputs are TRUE
    - Only outputs FALSE if all inputs are FALSE
    - **Example:** Q=1 if A OR B are 1



A **OR** B

Input		**Output**
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1



7432 Quad 2 Input OR

Illustration: https://en.wikipedia.org/wiki/Logic_gate

Logical Operators (Module 2)

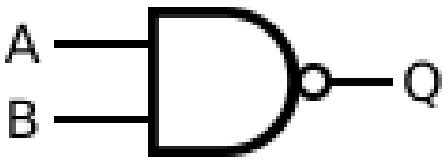
- Used for combining conditions with logical operators (returns either True or False)
    - Commonly in **conditional statements** and **loops**.

a	**b**	a **and** b	a **or** b	not a **and** b	not (a **and** b)
True	**True**	True	True	False	False
True	**False**	False	True	False	True
False	**True**	False	True	True	True
False	**False**	False	False	False	True

Logic gates

> *“Is a device which accepts one or more TRUE (1) or FALSE (0) inputs and produces a TRUE (1) or FALSE (0) output according to a logical rule.”*

- **NAND Gate (Not AND)**
  - The inverse of the AND gate.
  - Outputs TRUE unless all its inputs are TRUE.



NAND		
Input		Output
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

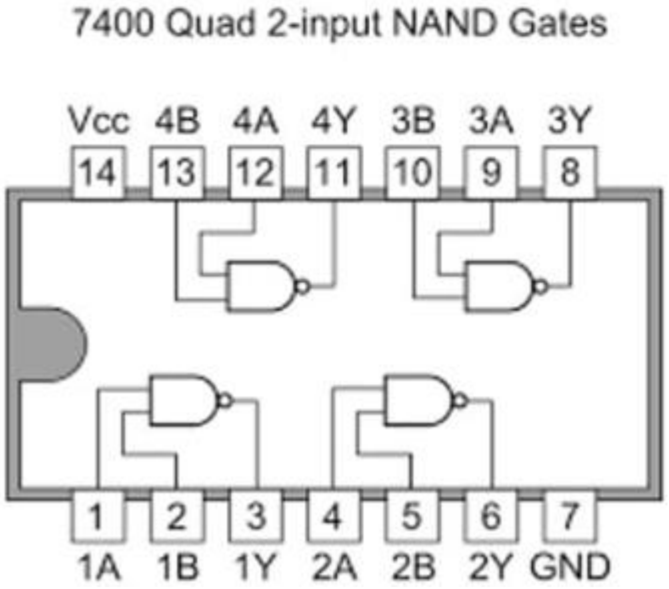


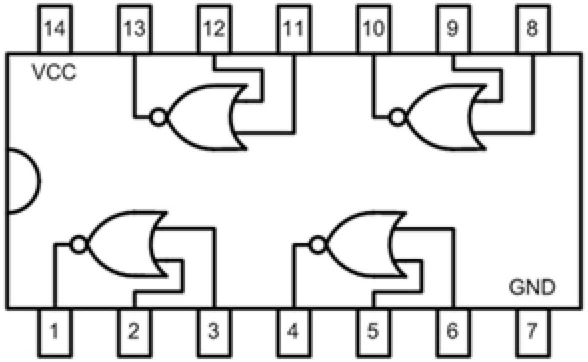
Illustration: https://en.wikipedia.org/wiki/Logic_gate

Logic gates

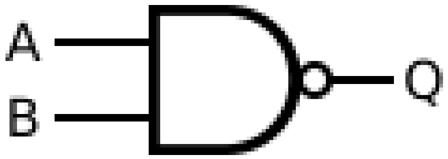
> *"Is a device which accepts one or more TRUE (1) or FALSE (0) inputs and produces a TRUE (1) or FALSE (0) output according to a logical rule."*

- NAND Gate (Not AND)
  - The inverse of the AND gate.
  - Outputs TRUE unless all its inputs are TRUE.

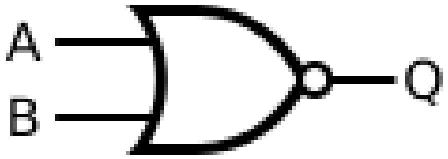
- **NOR Gate (Not OR)**
  - The inverse of the OR gate.
  - Outputs TRUE only if all its inputs are FALSE.



7402 Quad 2 Input NOR

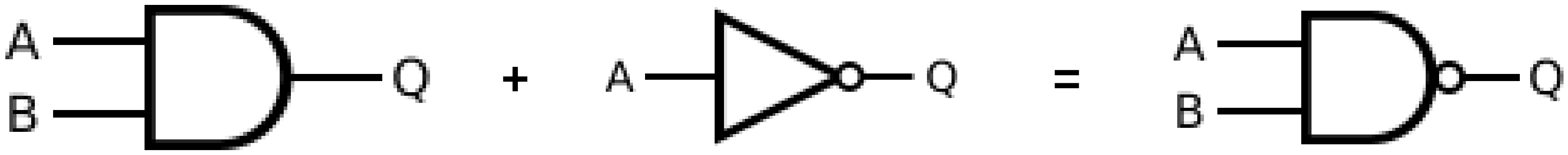


NAND		
Input		Output
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



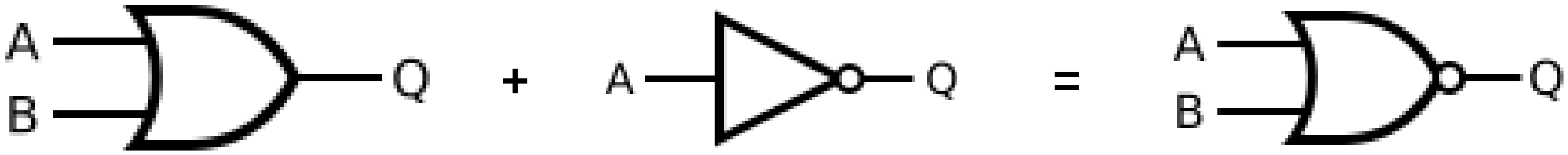
NOR		
Input		Output
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

Illustration: https://en.wikipedia.org/wiki/Logic_gate



NAND		
Input		**Output**
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

(Combinational Logic Circuits?)



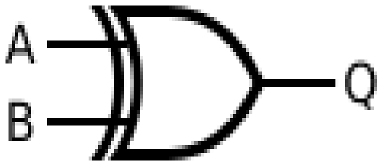
NOR		
Input		**Output**
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

Illustration: https://en.wikipedia.org/wiki/Logic_gate

Logic gates

> *“Is a device which accepts one or more TRUE (1) or FALSE (0) inputs and produces a TRUE (1) or FALSE (0) output according to a logical rule.”*

- **XOR Gate (Exclusive OR)**
  - Outputs TRUE if its inputs are different.
    - one is FALSE and the other is TRUE



XOR		
Input		**Output**
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

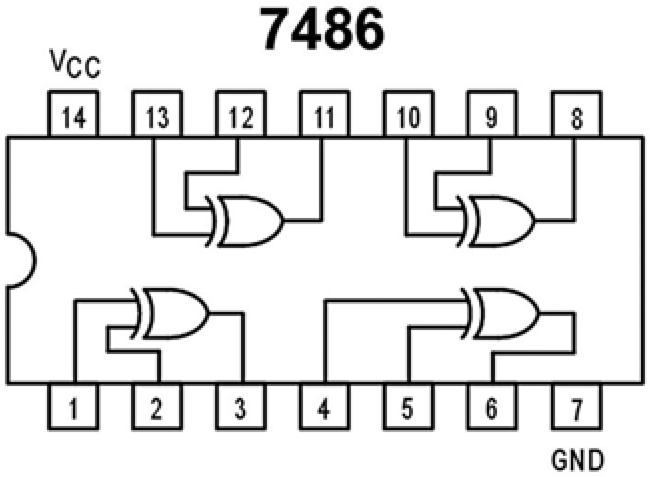


Illustration: https://en.wikipedia.org/wiki/Logic_gate

Logic gates

> *“Is a device which accepts one or more TRUE (1) or FALSE (0) inputs and produces a TRUE (1) or FALSE (0) output according to a logical rule.”*

- XOR Gate (Exclusive OR)
  - Outputs TRUE if its inputs are different.
    - one is FALSE and the other is TRUE

- **XNOR Gate (Exclusive NOR)**
  - The inverse of the XOR gate.
  - Outputs TRUE if its inputs are the same.
    - Both has to either FALSE or TRUE

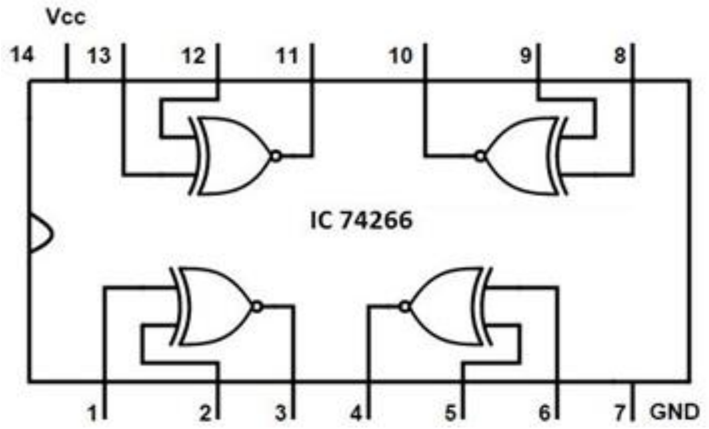
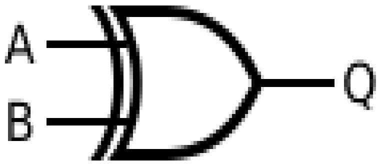
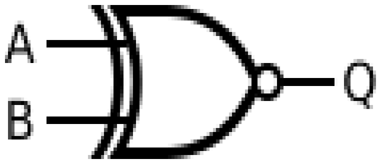


Illustration: https://en.wikipedia.org/wiki/Logic_gate



XOR		
Input		Output
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0



XNOR		
Input		Output
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

Combinational Logic Circuits

Combinational Logic Circuits

> *"Built using multiple basic logic gates like AND, OR, NOT, NAND, NOR, XOR, and XNOR."*

- **Combining logic gates** in various ways to implement specific functions.

The three main ways of specifying the function of a combinational logic circuit are:

Illustration: https://www.electronics-tutorials.ws/combination/comb_1.html

Combinational Logic Circuits

> *"Built using multiple basic logic gates like AND, OR, NOT, NAND, NOR, XOR, and XNOR."*

- **Combining logic gates** in various ways to implement specific functions.

The three main ways of specifying the function of a combinational logic circuit are:

- **Logic Diagram:** A graphical representation of a logic circuit.
  - Shows the wiring and connections of each individual logic gate, represented by a specific graphical symbol, that implements the logic circuit.

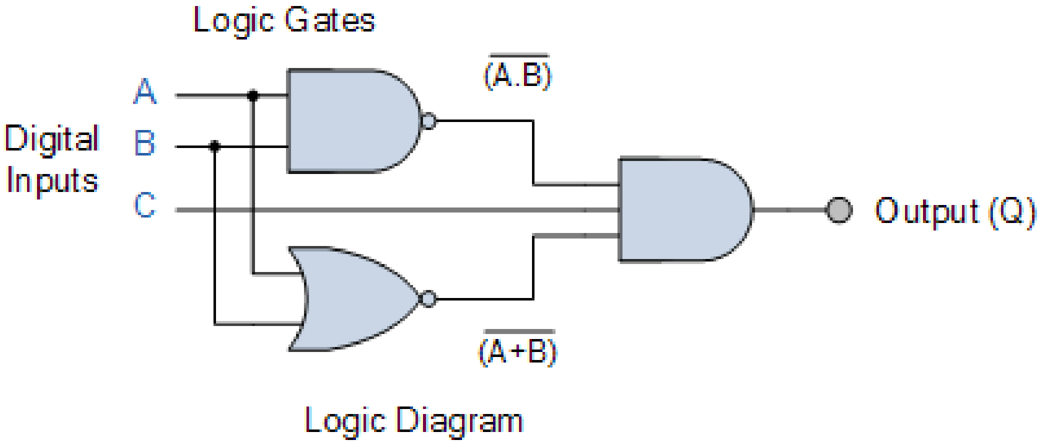


Illustration: https://www.electronics-tutorials.ws/combination/comb_1.html

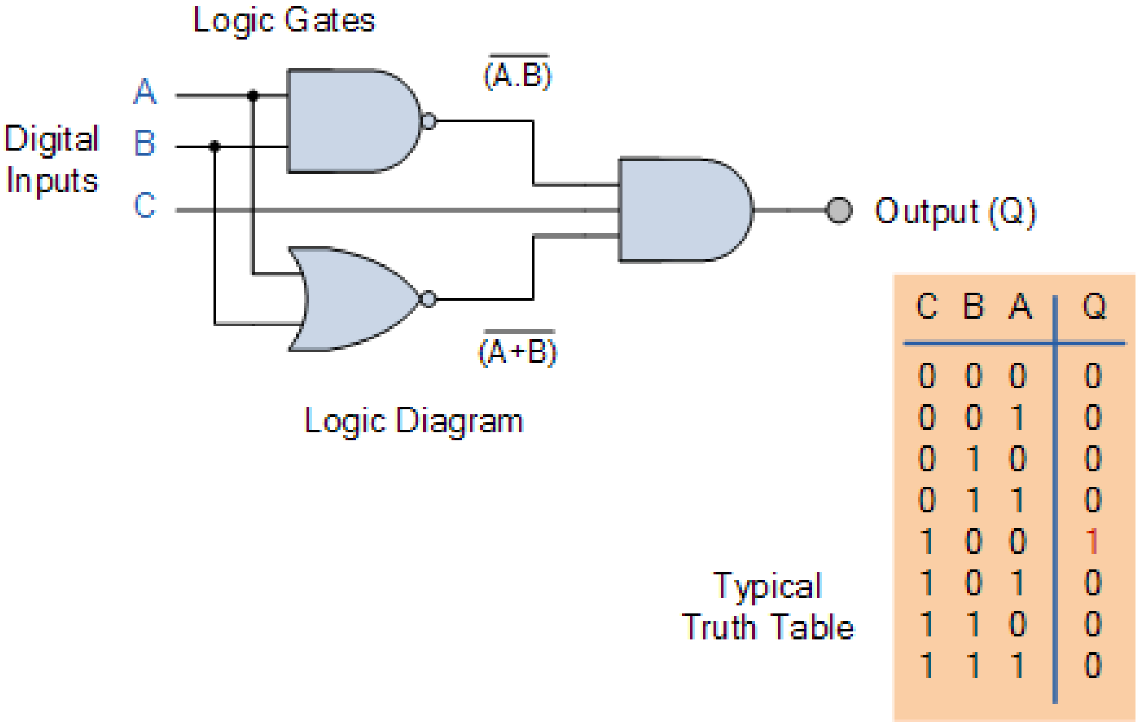
Combinational Logic Circuits

> *“Built using multiple basic logic gates like AND, OR, NOT, NAND, NOR, XOR, and XNOR.”*

- **Combining logic gates i**n various ways to implement specific functions.

The three main ways of specifying the function of a combinational logic circuit are:

- **Logic Diagram:** This is a graphical representation of a logic circuit.
- **Truth Table:** A concise list that defines all the output states in tabular form.
  - …for each possible combination of input variable that the gate could encounter.



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

Illustration: https://www.electronics-tutorials.ws/combination/comb_1.html

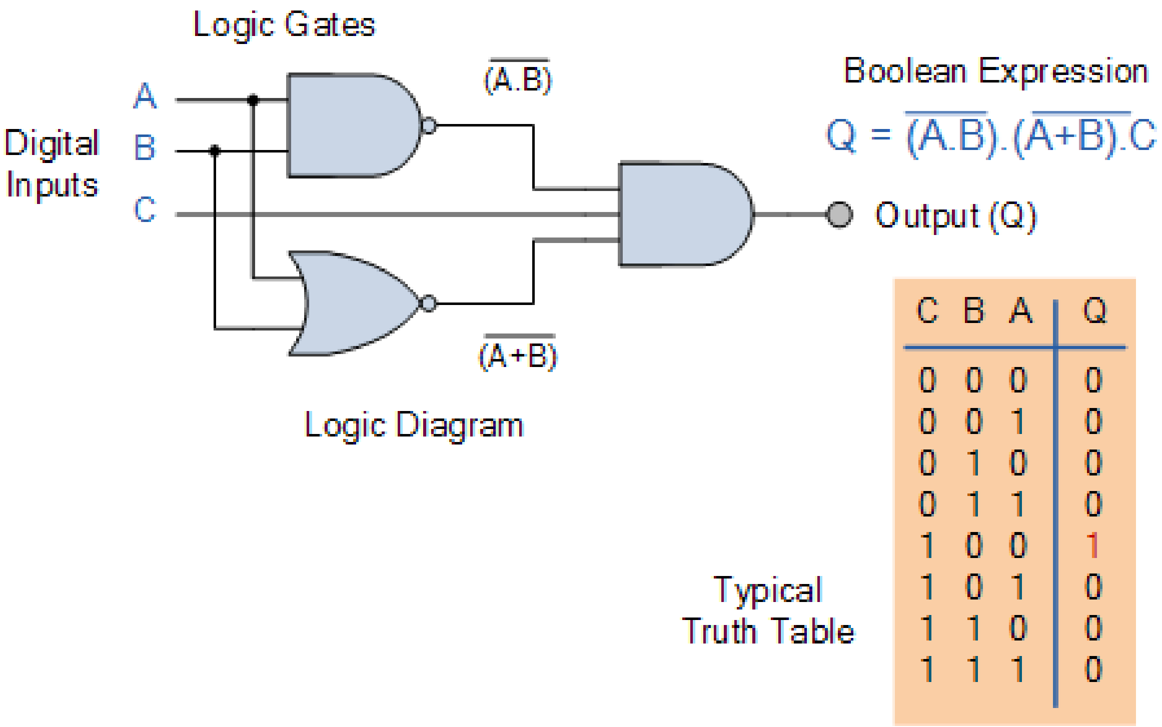
Combinational Logic Circuits

> *“Built using multiple basic logic gates like AND, OR, NOT, NAND, NOR, XOR, and XNOR.”*

- **Combining logic gates** in various ways to implement specific functions.

The three main ways of specifying the function of a combinational logic circuit are:

- **Logic Diagram:** This is a graphical representation of a logic circuit.
- **Truth Table:** A concise list that defines all the output states in tabular form.
- **Boolean expression:** A mathematical/algebraic expression showing the operation of the logic circuit for each input variable
  - Each variable is a logic value (either True or False)
  - …and results in a logic output (either True or False)



$Q = \overline{(A.B)}.\overline{(A+B)}.C$

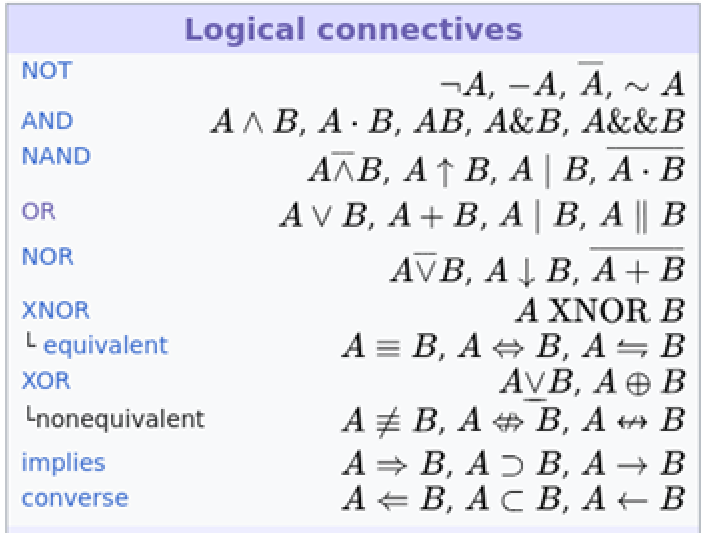
C	B	A	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Illustration: https://www.electronics-tutorials.ws/combination/comb_1.html

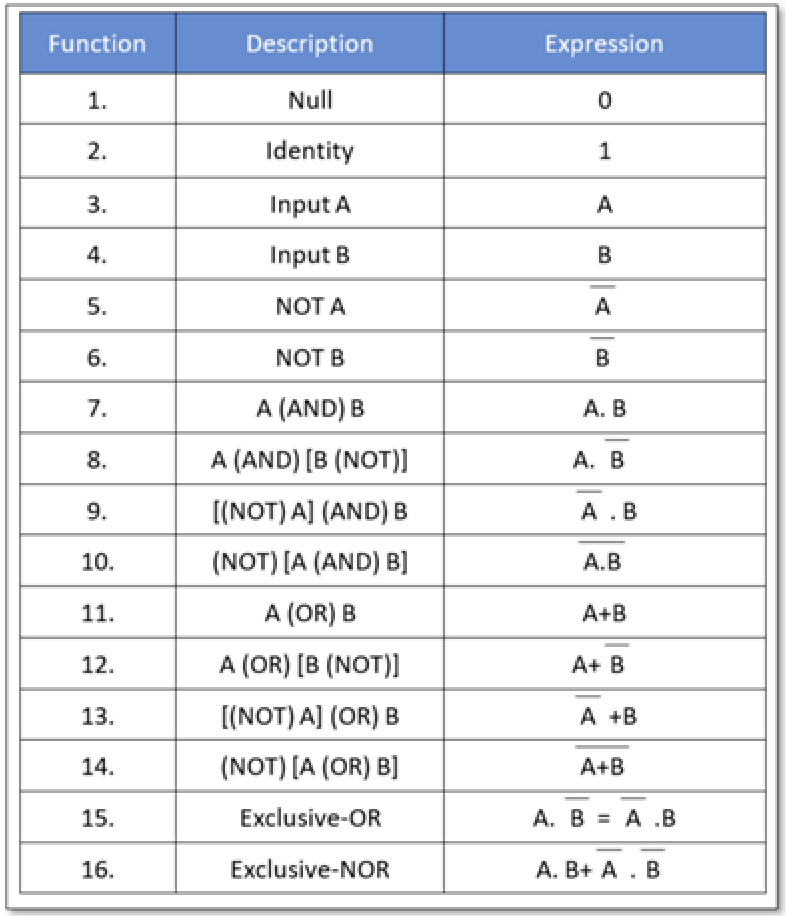
Combinational Logic Circuits

> *"Built using multiple basic logic gates like AND, OR, NOT, NAND, NOR, XOR, and XNOR."*

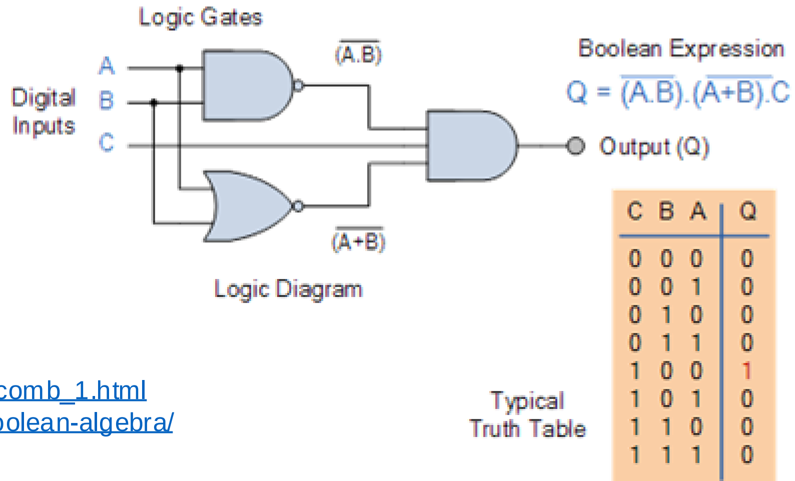
- **Boolean algebra expressions and functions**
  - Symbols / "Syntax"
    - …different symbols, same functionalities



Logical connectives	
NOT	$\neg A,\ -A,\ \overline{A},\ \sim A$
AND	$A \wedge B,\ A \cdot B,\ AB,\ A\&B,\ A\&\&B$
NAND	$A \overline{\wedge} B,\ A \uparrow B,\ A \mid B,\ \overline{A \cdot B}$
OR	$A \vee B,\ A + B,\ A \mid B,\ A \parallel B$
NOR	$A \overline{\vee} B,\ A \downarrow B,\ \overline{A + B}$
XNOR	$A\ \mathrm{XNOR}\ B$
└ equivalent	$A \equiv B,\ A \Leftrightarrow B,\ A \leftrightharpoons B$
XOR	$A \veebar B,\ A \oplus B$
└nonequivalent	$A \not\equiv B,\ A \nLeftrightarrow B,\ A \nleftrightarrow B$
implies	$A \Rightarrow B,\ A \supset B,\ A \rightarrow B$
converse	$A \Leftarrow B,\ A \subset B,\ A \leftarrow B$



Function	Description	Expression
1.	Null	0
2.	Identity	1
3.	Input A	A
4.	Input B	B
5.	NOT A	$\overline{A}$
6.	NOT B	$\overline{B}$
7.	A (AND) B	A. B
8.	A (AND) [B (NOT)]	$A.\ \overline{B}$
9.	[(NOT) A] (AND) B	$\overline{A}\ .B$
10.	(NOT) [A (AND) B]	$\overline{A.B}$
11.	A (OR) B	A+B
12.	A (OR) [B (NOT)]	$A+\ \overline{B}$
13.	[(NOT) A] (OR) B	$\overline{A}\ +B$
14.	(NOT) [A (OR) B]	$\overline{A+B}$
15.	Exclusive-OR	$A.\ \overline{B}\ =\ \overline{A}\ .B$
16.	Exclusive-NOR	$A.\ B+\ \overline{A}\ .\ \overline{B}$



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

Illustration: https://www.electronics-tutorials.ws/combination/comb_1.html
https://www.electronics-lab.com/article/laws-of-boolean-algebra/
https://en.wikipedia.org/wiki/Boolean_function

Example: Implementation of Boolean Algebra

- Can be implemented by interconnection of logic gates:
  - $F = \overline{AB} + \bar{B}C$

Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

Example: Implementation of Boolean Algebra

- Can be implemented by interconnection of logic gates:
  - $F = \overline{AB} + \bar{B}C$

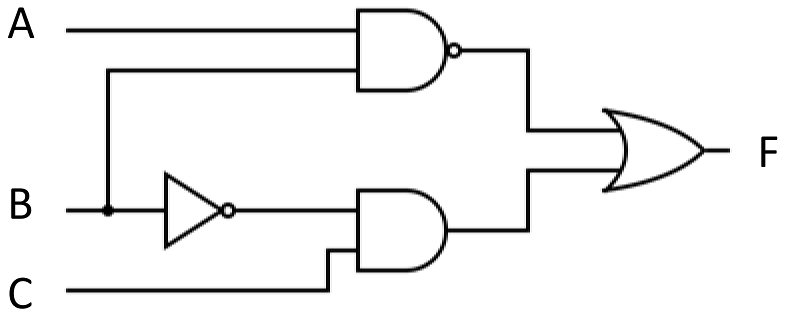


Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

Example: Implementation of Boolean Algebra

- Can be implemented by interconnection of logic gates:
  - $F = \overline{AB} + \bar{B}C$

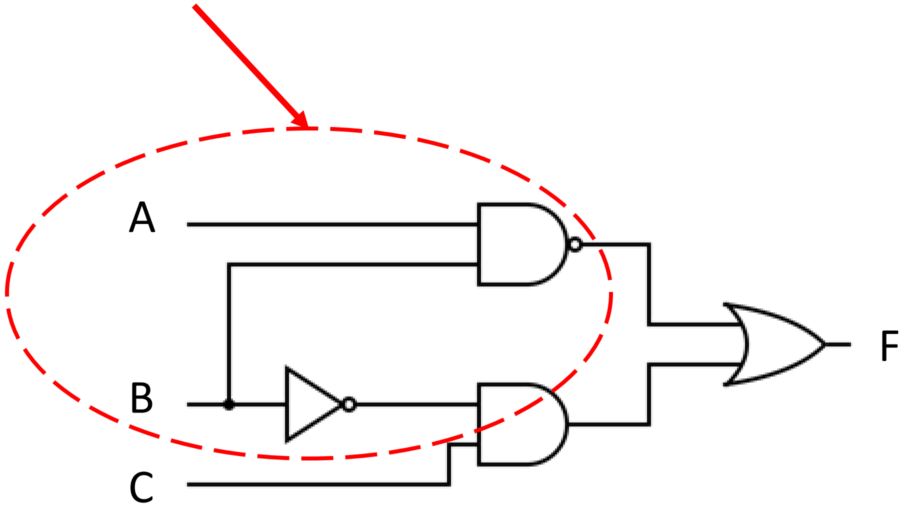


Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

Example: Implementation of Boolean Algebra

- Can be implemented by interconnection of logic gates:
  - $F = \overline{AB} + \bar{B}C$

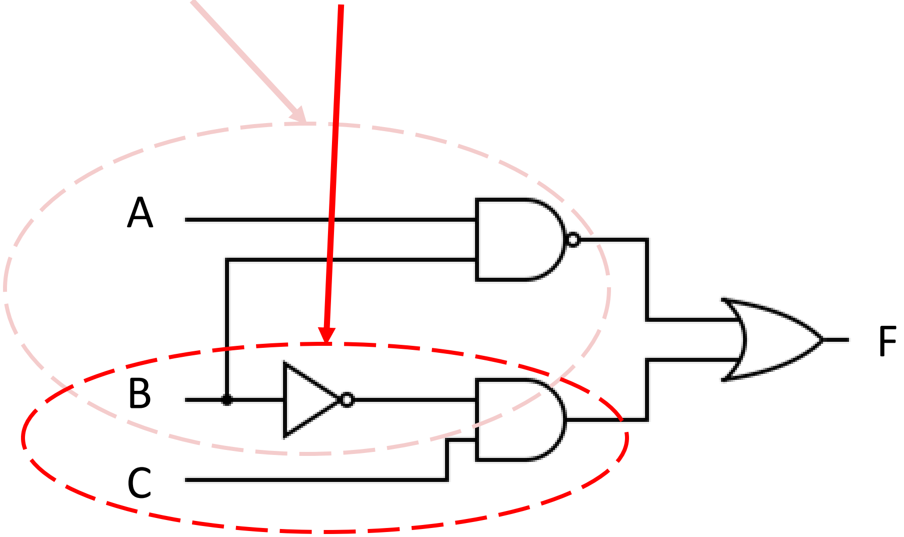


Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

Example: Implementation of Boolean Algebra

- Can be implemented by interconnection of logic gates:
  - $F = \overline{AB} + \bar{B}C$

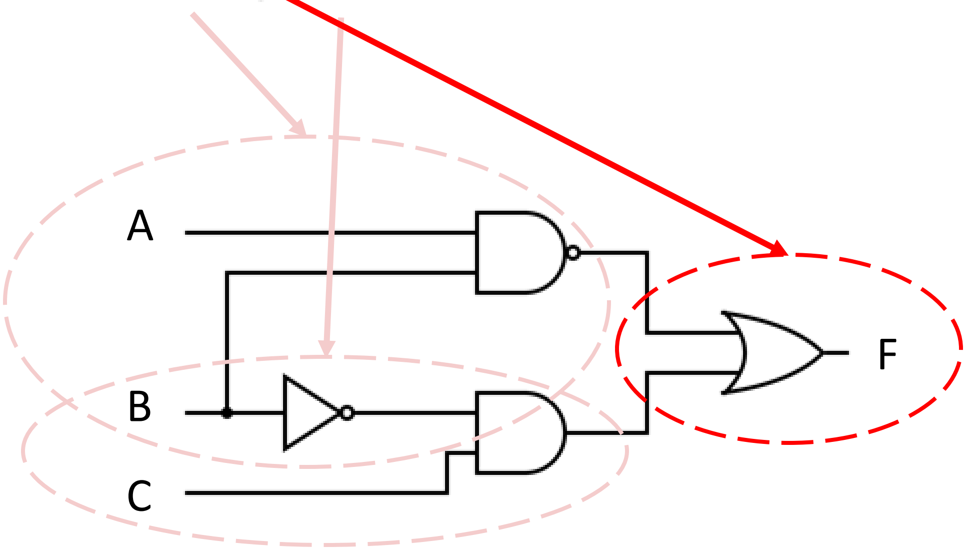
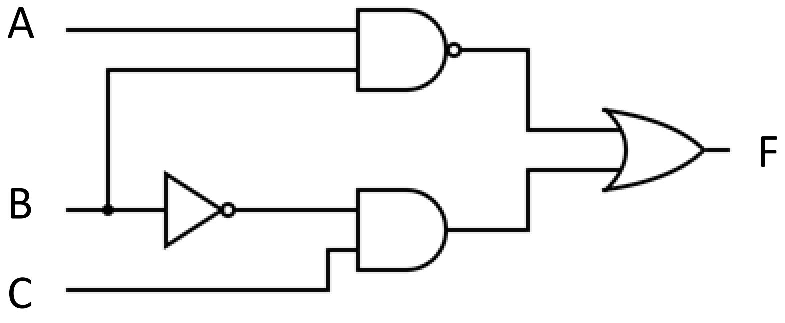


Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

Example: Implementation of Boolean Algebra

- Can be implemented by interconnection of logic gates:
  - $F = \overline{AB} + \bar{B}C$



Input

0 0 0

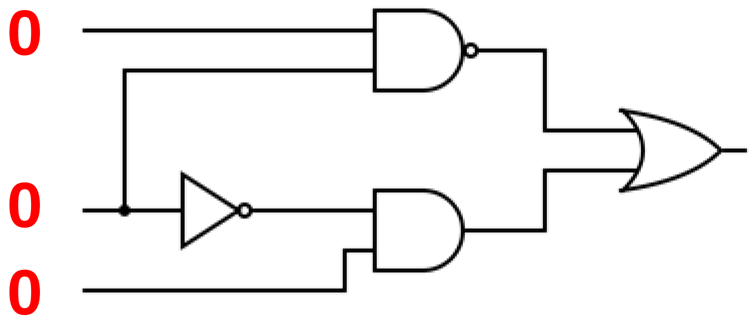
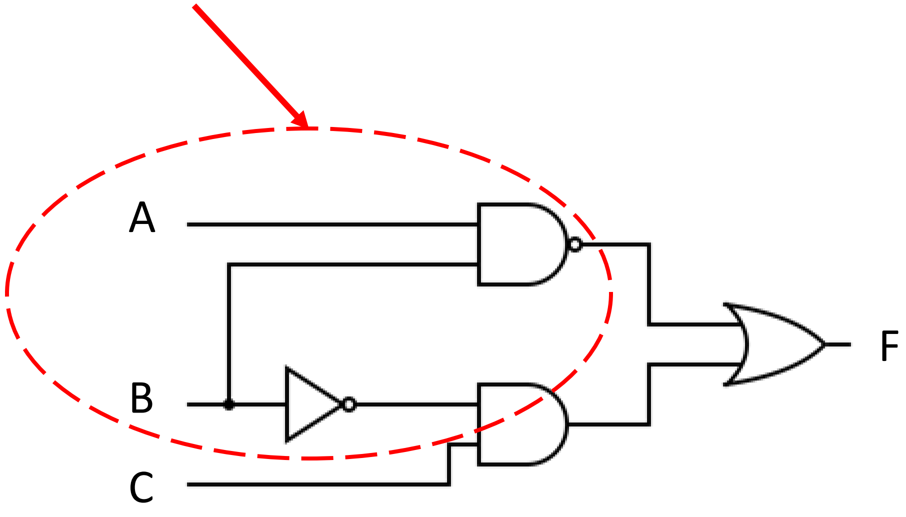


Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

Example: Implementation of Boolean Algebra

- Can be implemented by interconnection of logic gates:
  - $F = \overline{AB} + \bar{B}C$



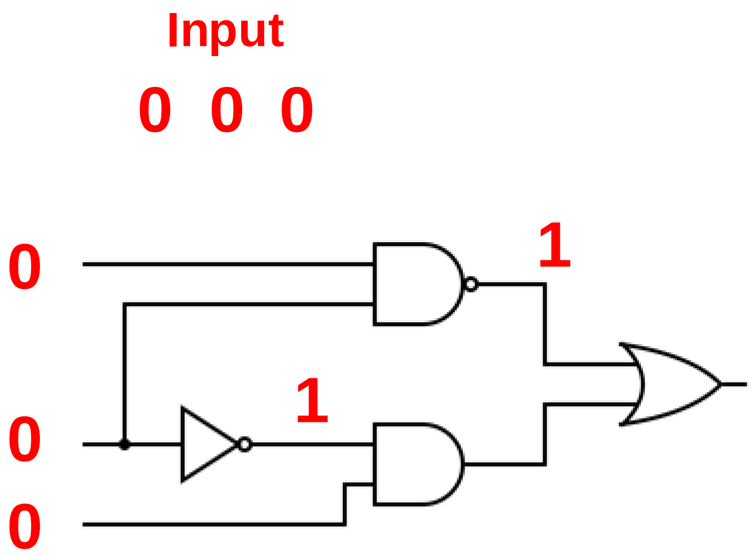
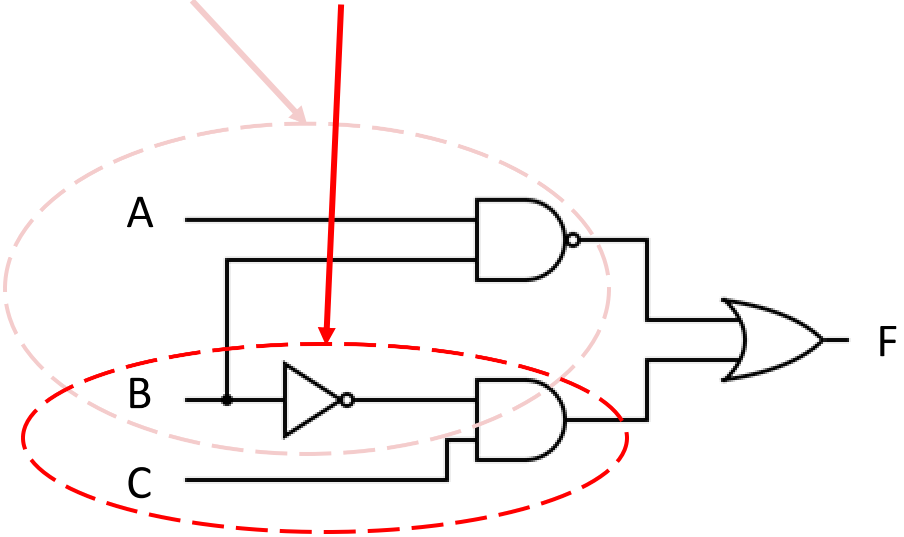


Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

Example: Implementation of Boolean Algebra

- Can be implemented by interconnection of logic gates:
  - $F = \overline{AB} + \bar{B}C$



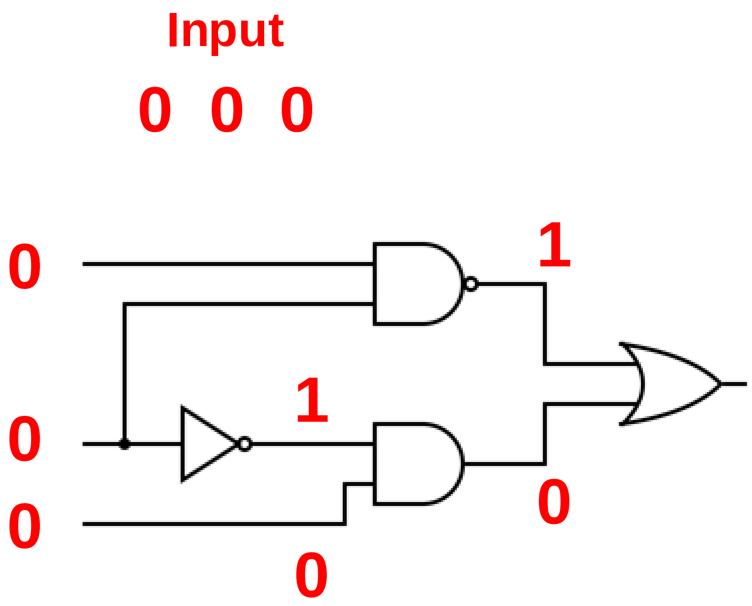
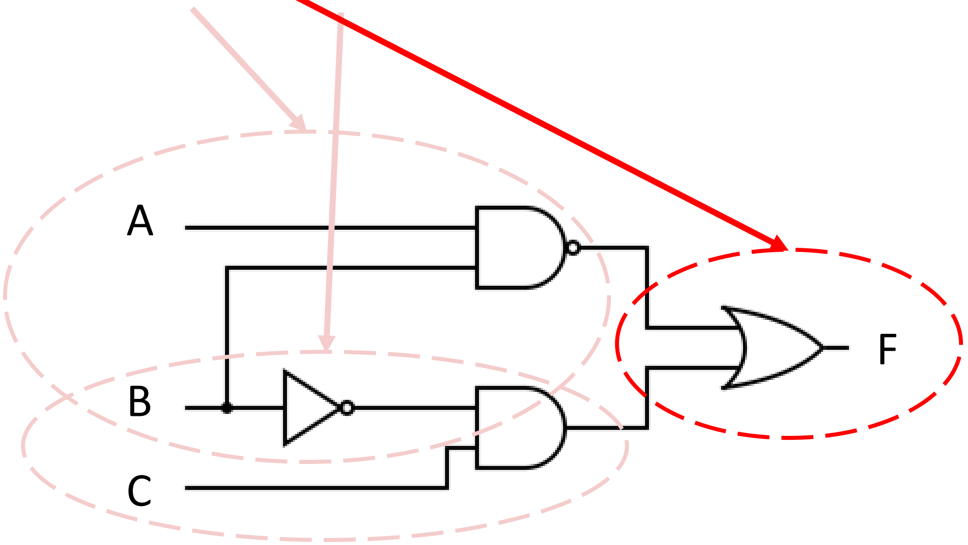


Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

Example: Implementation of Boolean Algebra

- Can be implemented by interconnection of logic gates:
  - $F = \overline{AB} + \bar{B}C$



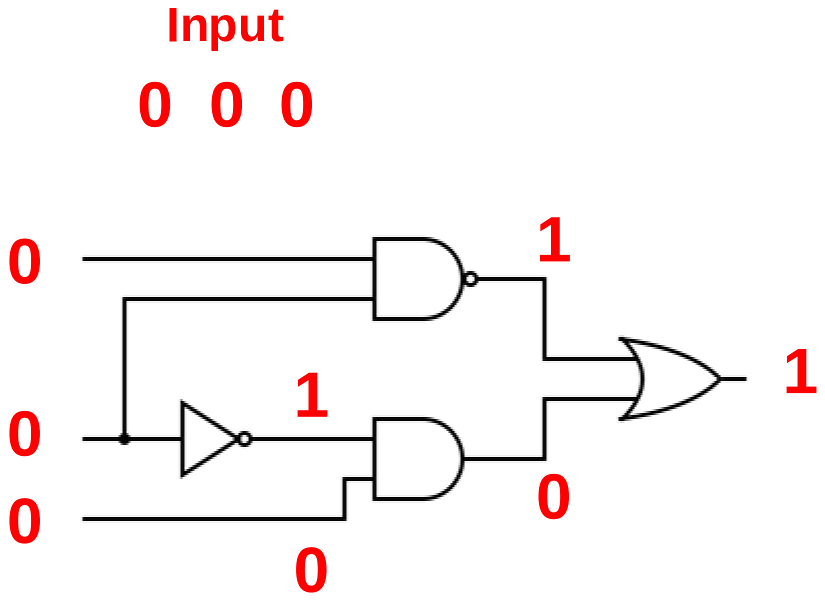
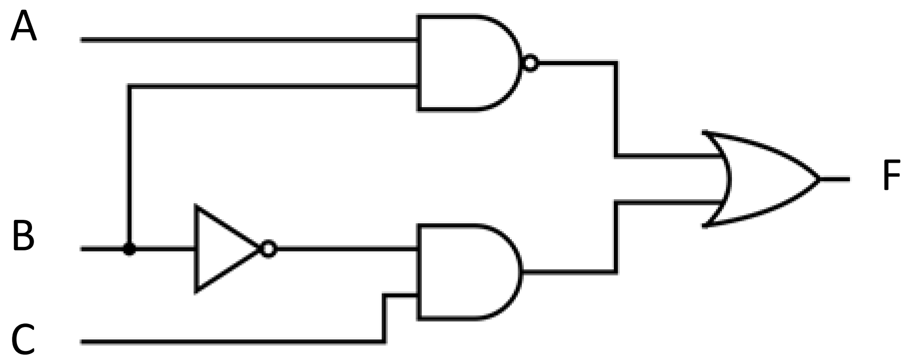


Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

Task: 15 min

- Go to https://logic.ly/demo/
- Draw the following circuit
- Simulate the following circuit and fill the truth table for that



A	B	C	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

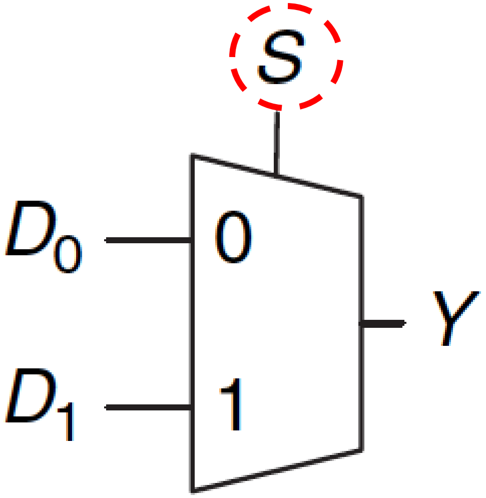
Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

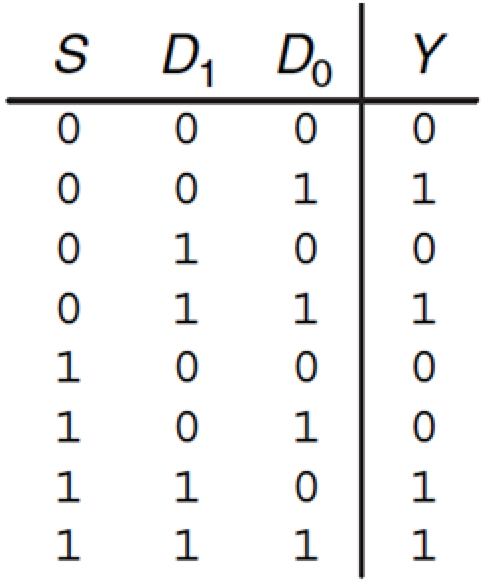
Combinational Logic Circuits
(common examples)

Combinational Logic Circuits (Common Examples)

- **Multiplexers (MUX):** a circuit that selects one of several input signals and forwards the selected input to a single output line. It uses control signals to determine which input to pass through.
  - **Example:** 2-to-1 MUX

S	*Y*
0	D_0
1	D_1



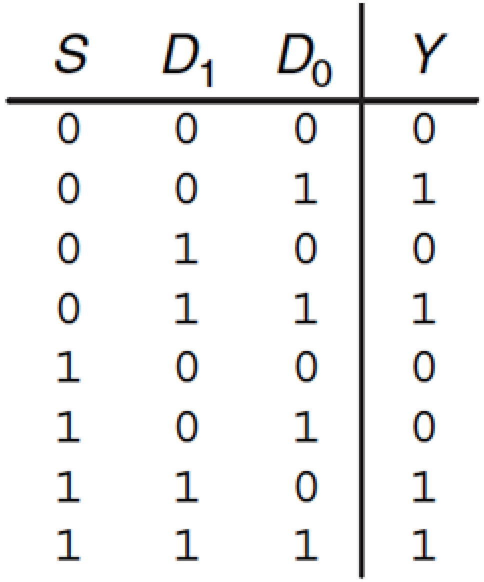


S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

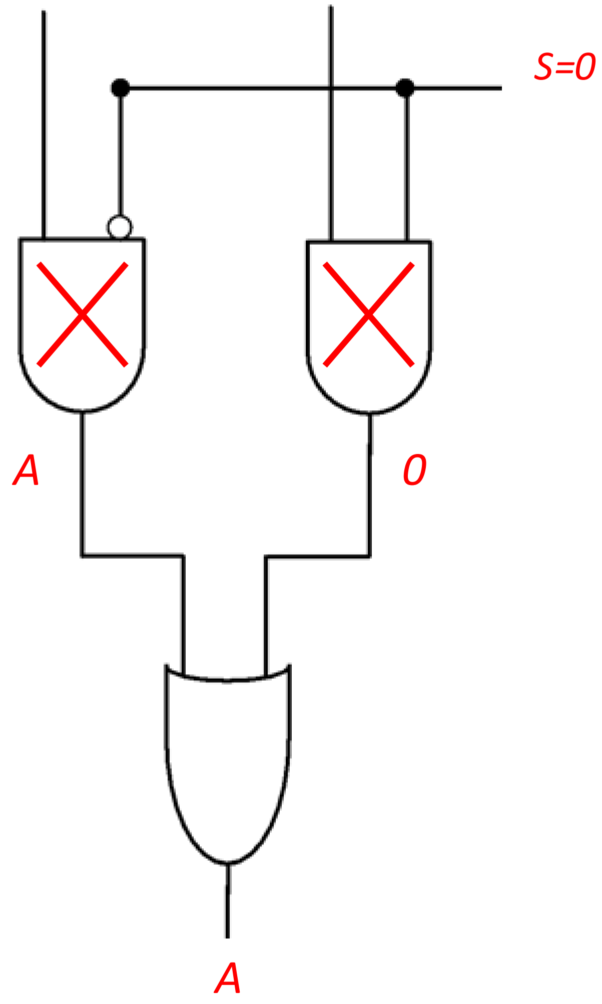
Image: https://www.electronics-tutorials.ws/wp-content/uploads/2018/05/logic-log43.gif

Combinational Logic Circuits (Common Examples)

- **Multiplexers (MUX):** a circuit that selects one of several input signals and forwards the selected input to a single output line. It uses control signals to determine which input to pass through.
  - **Example:** 2-to-1 MUX
    - S=0
      - A AND 1 = A
      - B AND 0 = 0
      - A OR 0 = A
    - S=1
      - A AND 0 = 0
      - B AND 1 = B
      - B OR 0 = B



S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Combinational Logic Circuits (Common Examples)

- **Decoder:** a combinational logic circuit that converts binary input data into a unique output pattern.
  - A decoder has n inputs and 2^n outputs

- **Encoder**: a combinational logic circuit that performs the reverse operation of a decoder.
  - A decoder has 2^n inputs and n outputs

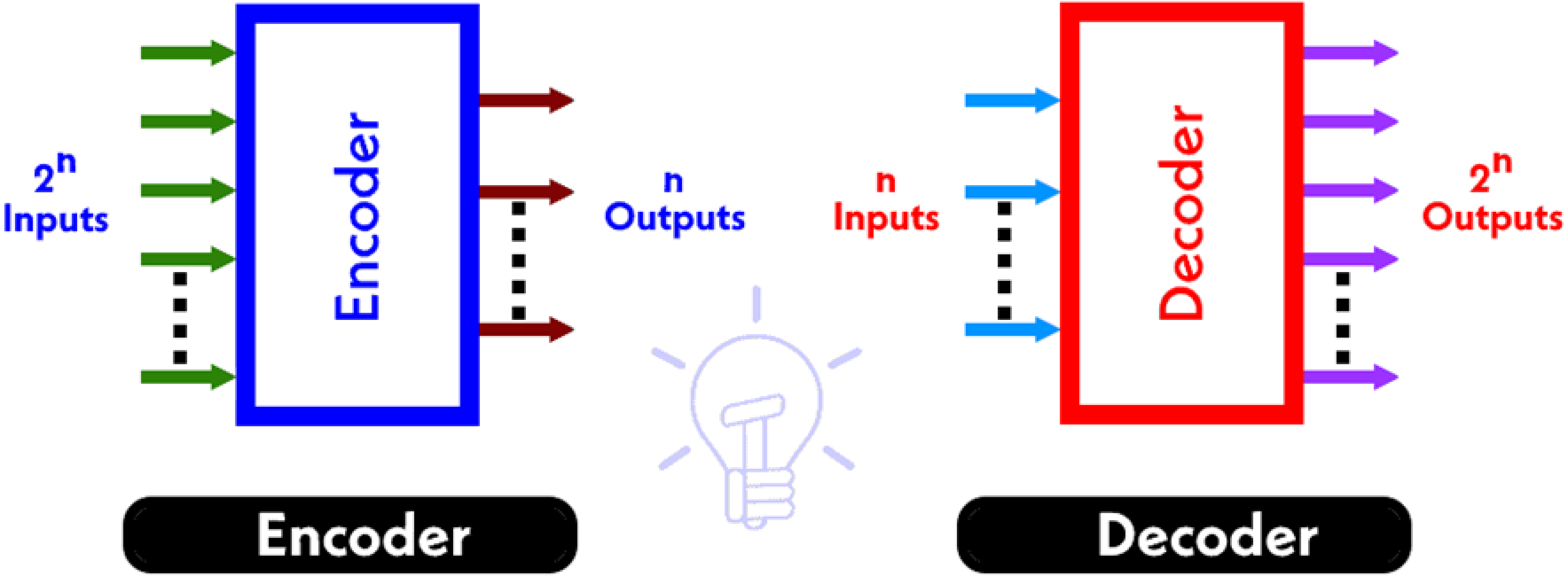
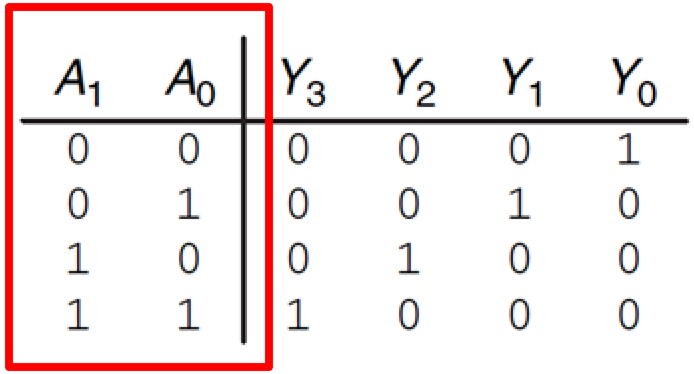


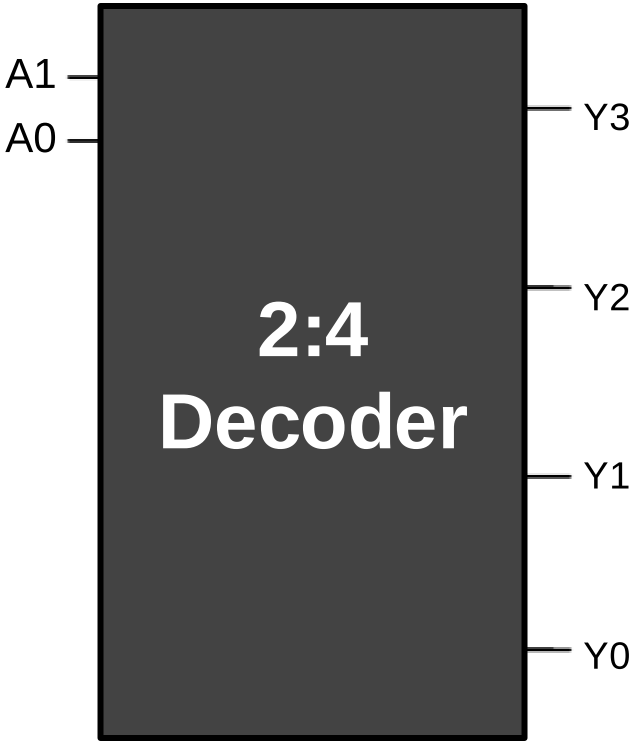
Illustration: https://www.electricaltechnology.org/2022/12/difference-between-encoder-decoder.html

Combinational Logic Circuits (Common Examples)

- **Decoder:** a combinational logic circuit that converts binary input data into a unique output pattern.
  - A decoder has n inputs and 2^n outputs
- **Encoder**: a combinational logic circuit that performs the reverse operation of a decoder.
  - A decoder has 2^n inputs and n outputs
- **Example: 2-to-4 decoder**
  - Two input lines: A0 and A1,

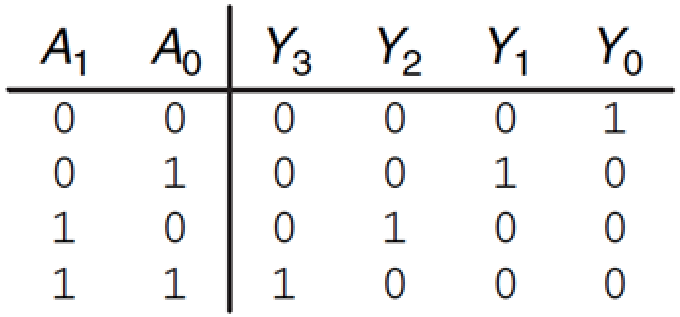


A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

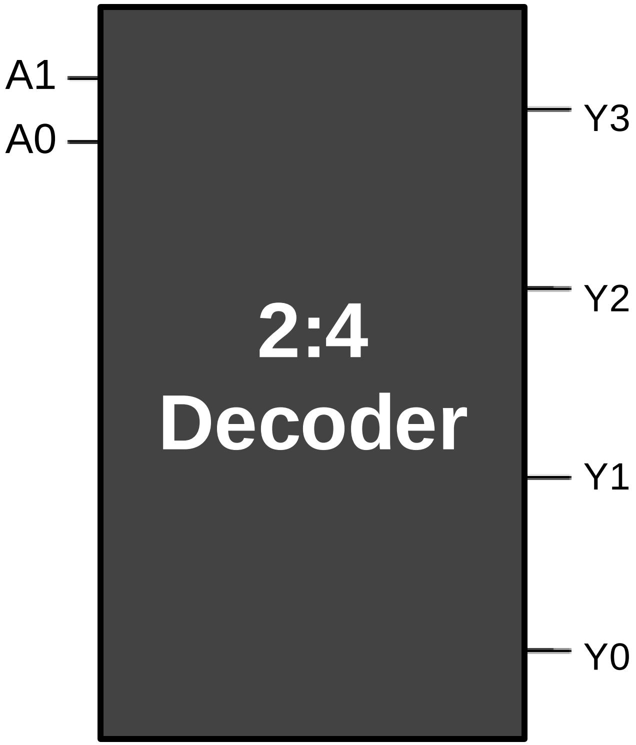


Combinational Logic Circuits (Common Examples)

- **Decoder:** a combinational logic circuit that converts binary input data into a unique output pattern.
    - A decoder has n inputs and 2^n outputs
- **Encoder**: a combinational logic circuit that performs the reverse operation of a decoder.
    - A decoder has 2^n inputs and n outputs
- **Example: 2-to-4 decoder**

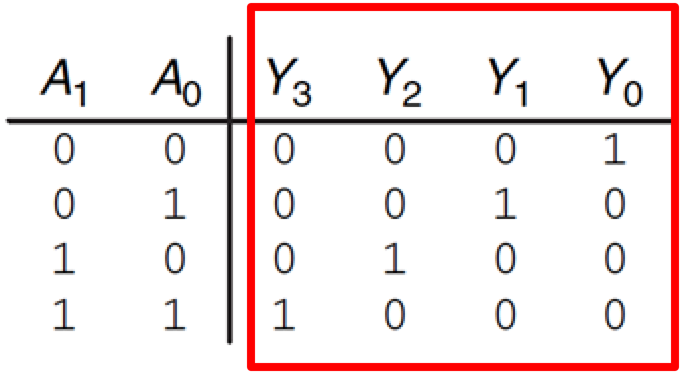


A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

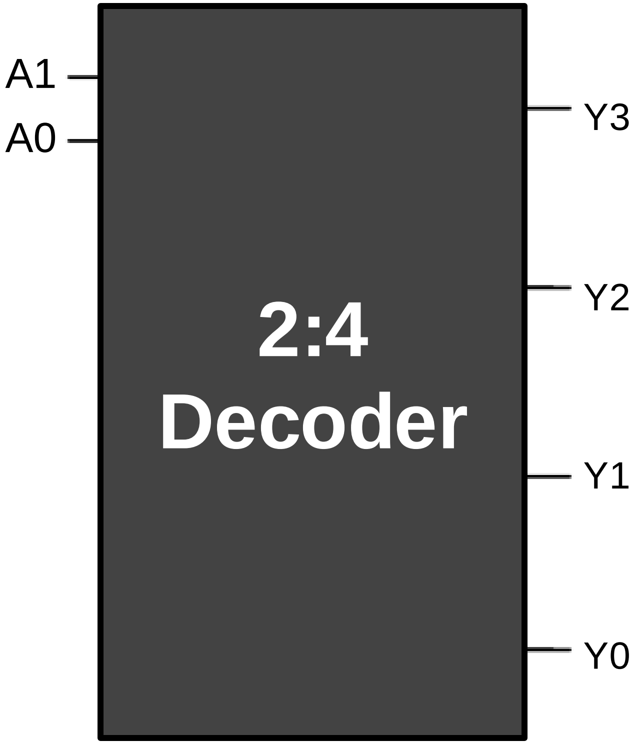


Combinational Logic Circuits (Common Examples)

- **Decoder:** a combinational logic circuit that converts binary input data into a unique output pattern.
  - A decoder has n inputs and 2^n outputs
- **Encoder**: a combinational logic circuit that performs the reverse operation of a decoder.
  - A decoder has 2^n inputs and n outputs
- **Example: 2-to-4 decoder**
  - Two input lines: A0 and A1,
  - Generates four output lines:Y0, Y1, Y2, and Y3
    - Corresponding to the truth table



A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

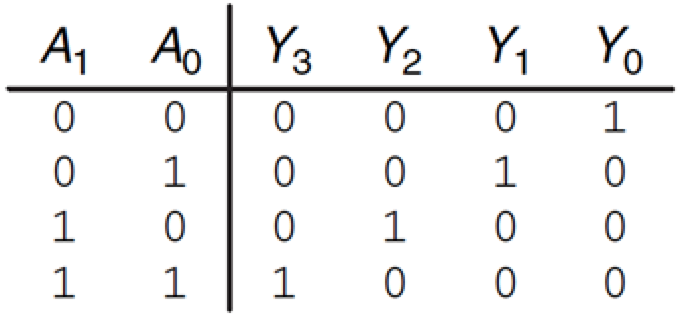


Combinational Logic Circuits (Common Examples)

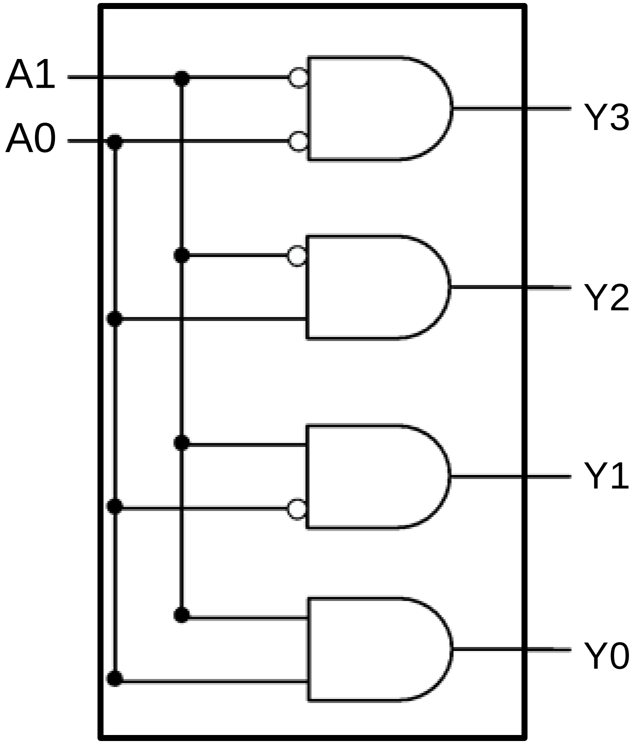
- **Decoder:** a combinational logic circuit that converts binary input data into a unique output pattern.
  - A decoder has n inputs and 2^n outputs

- **Encoder:** a combinational logic circuit that performs the reverse operation of a decoder.
  - A decoder has 2^n inputs and n outputs

- **Example: 2-to-4 decoder**
  - Two input lines: A0 and A1,
  - Generates four output lines:Y0, Y1, Y2, and Y3
    - Corresponding to the truth table
  - 4x AND gates
  - 4x NOT gates



A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

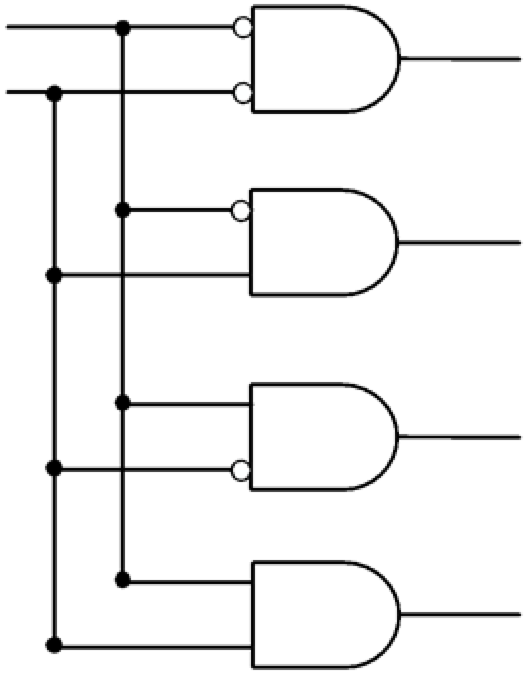


Task: 10 min

? Go to https://logic.ly/demo/

? Draw the 2 to 4 Decoder and simulate it

A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



Combinational Logic Circuits (Common Examples)

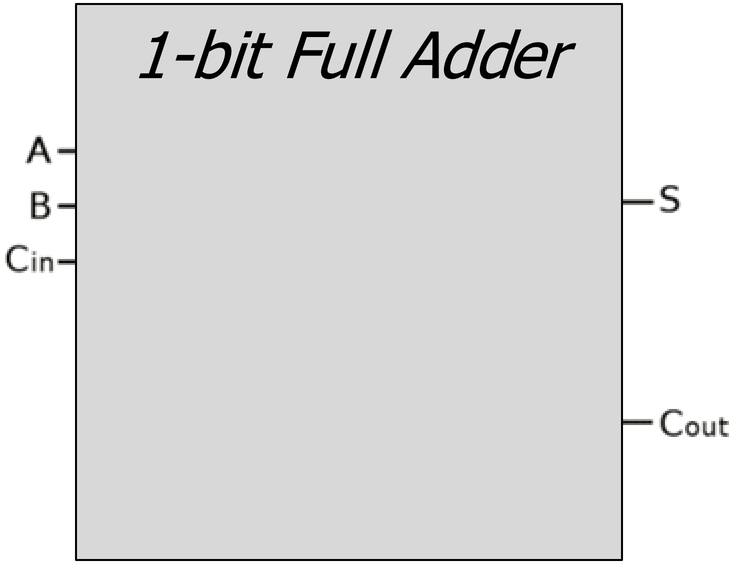
- **Full Adder:** a combinational logic circuit that adds three binary digits: two significant bits and an optional carry-in bit from a previous addition operation.

- **Inputs**:
  - `A`: First significant bit.
  - `B`: Second significant bit.
  - `Cin`: Carry-in bit from a previous addition.

- **Outputs**:
  - `Sum (S)`: The sum of the three input bits.
  - `Cout (Cout)`: The carry-out bit, which is the overflow bit from the sum.

  - **Example:** 1-bit Full Addition

$$
\begin{array}{r}
a_{n-1}a_{n-2}\dots a_1a_0 \\
b_{n-1}b_{n-2}\dots b_1b_0 \\
C_n\,C_{n-1}\quad\dots\quad C_1 \\
\hline
S_{n-1}\quad\dots\quad S_1S_0
\end{array}
$$



Combinational Logic Circuits (Common Examples)

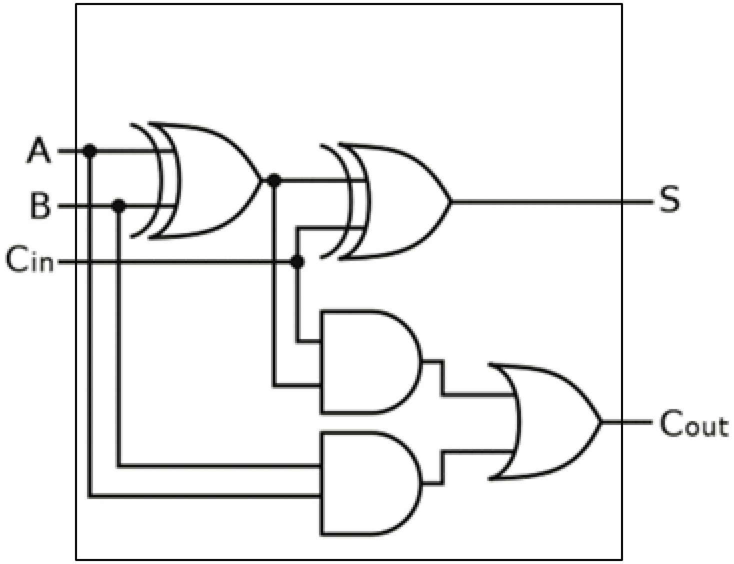
- **Full Adder:** a combinational logic circuit that adds three binary digits: two significant bits and an optional carry-in bit from a previous addition operation.

- **Inputs**:
  - A: First significant bit.
  - B: Second significant bit.
  - Cin: Carry-in bit from a previous addition.

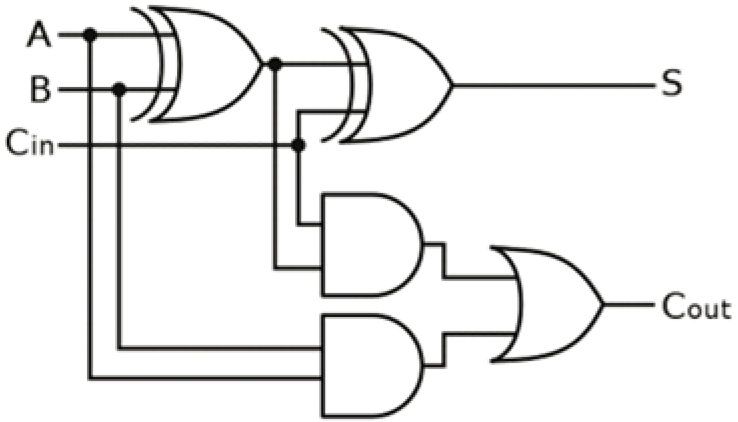
- **Outputs**:
  - Sum (S): The sum of the three input bits.
  - Cout (Cout): The carry-out bit, which is the overflow bit from the sum.

  - **Example:** 1-bit Full Addition
    - 2x XOR gate
    - 2x AND gate
    - 1x OR gate

$$\begin{array}{r} a_{n-1}a_{n-2}\dots a_1a_0 \\ b_{n-1}b_{n-2}\dots b_1b_0 \\ c_n\,c_{n-1}\quad\dots\quad c_1\quad \\ \hline s_{n-1}\quad\dots\quad s_1s_0 \end{array}$$



Combinational Logic Circuits (Common Examples)



- **Full Adder:** a combinational logic circuit that adds three binary digits: two significant bits and an optional carry-in bit from a previous addition operation.

- **Inputs**:
  - `A`: First significant bit.
  - `B`: Second significant bit.
  - `Cin`: Carry-in bit from a previous addition.

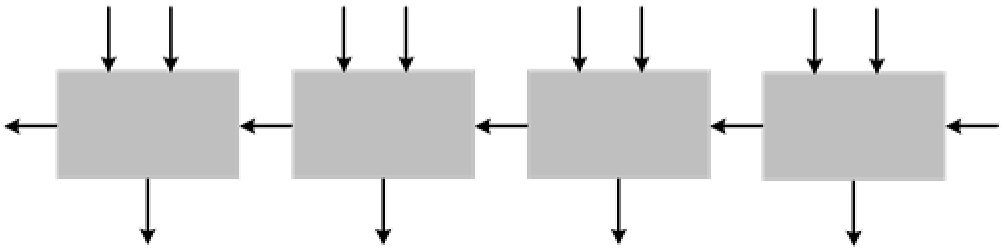
- **Outputs**:
  - `Sum (S)`: The sum of the three input bits.
  - `Cout (Cout)`: The carry-out bit, which is the overflow bit from the sum.

  - **Example:** 4-bit adder (4x 1-bit full adders)
    - Adds two binary numbers A and B
    - Example:

      11 (0b1011)

      + 9 (0b1001)

      = 20 (0b10100)



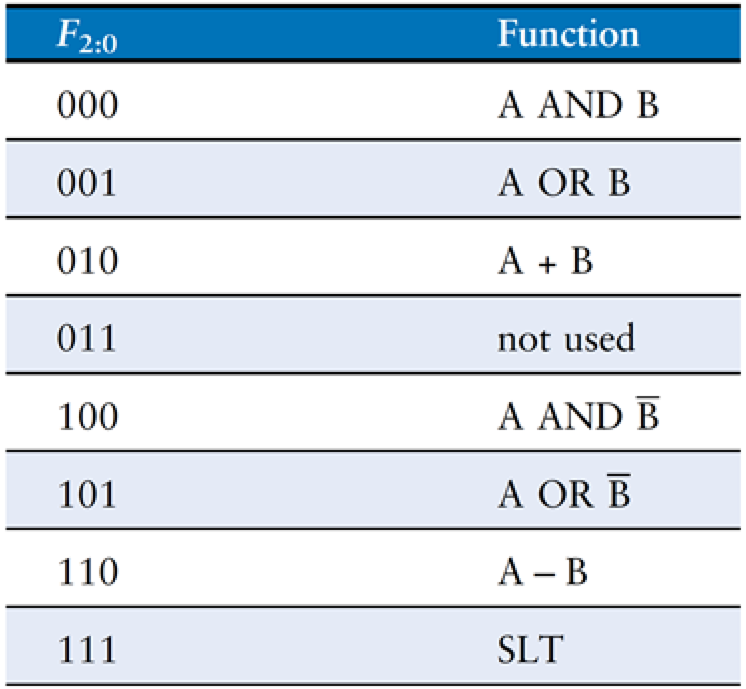
$$\begin{array}{ccccc} & a_3 & a_2 & a_1 & a_0 \\ + & b_3 & b_2 & b_1 & b_0 \\ c_4 & c_3 & c_2 & c_1 & \\ \hline & s_3 & s_2 & s_1 & s_0 \end{array}$$

$$\begin{array}{ccccc} & 1 & 0 & 1 & 1 \\ + & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & \\ \hline & 0 & 1 & 0 & 0 \end{array}$$

CPU (Central Processing Unit) [module 1]:

- The brain of the microcontroller, responsible for executing instructions.
  - **Control Unit**
  - **Arithmetic Logic Unit (ALU)**
    - Usually denoted with this symbol:
    - Example
  - **Registers**
  - **Memory**

  - …all build on logic gates..



$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{\text{B}}$
101	A OR $\overline{\text{B}}$
110	A – B
111	SLT

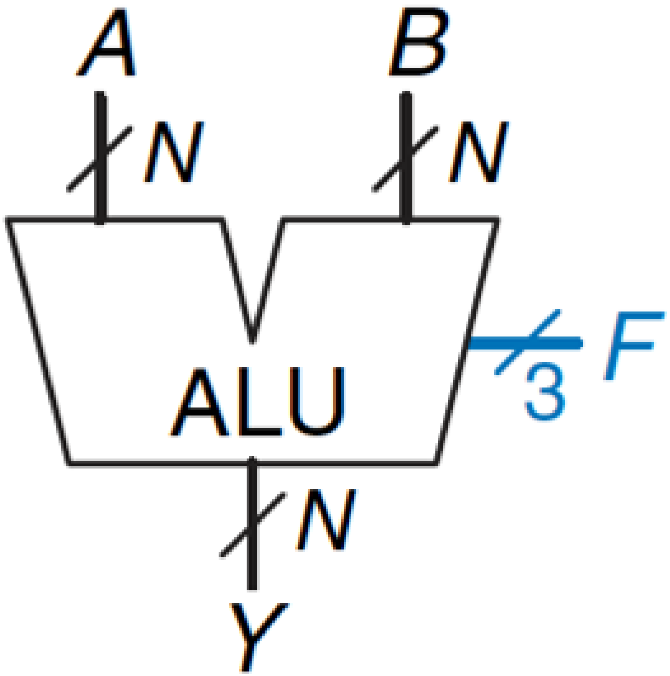
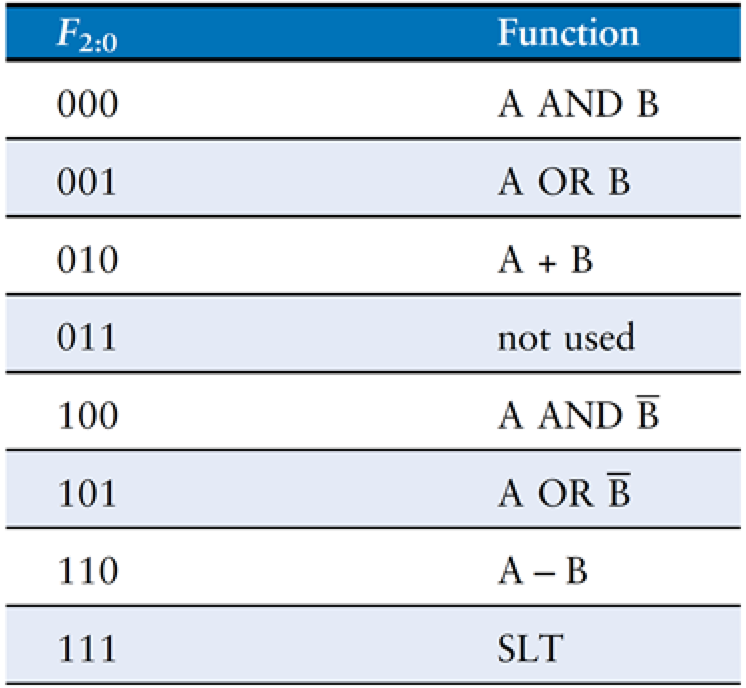


Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

CPU (Central Processing Unit) [module 1]:

- The brain of the microcontroller, responsible for executing instructions.
  - **Control Unit**
  - **Arithmetic Logic Unit (ALU)**
    - Usually denoted with this symbol:
    - Example
  - **Registers**
  - **Memory**

  - …all build on logic gates..



$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\bar{B}$
101	A OR $\bar{B}$
110	A – B
111	SLT

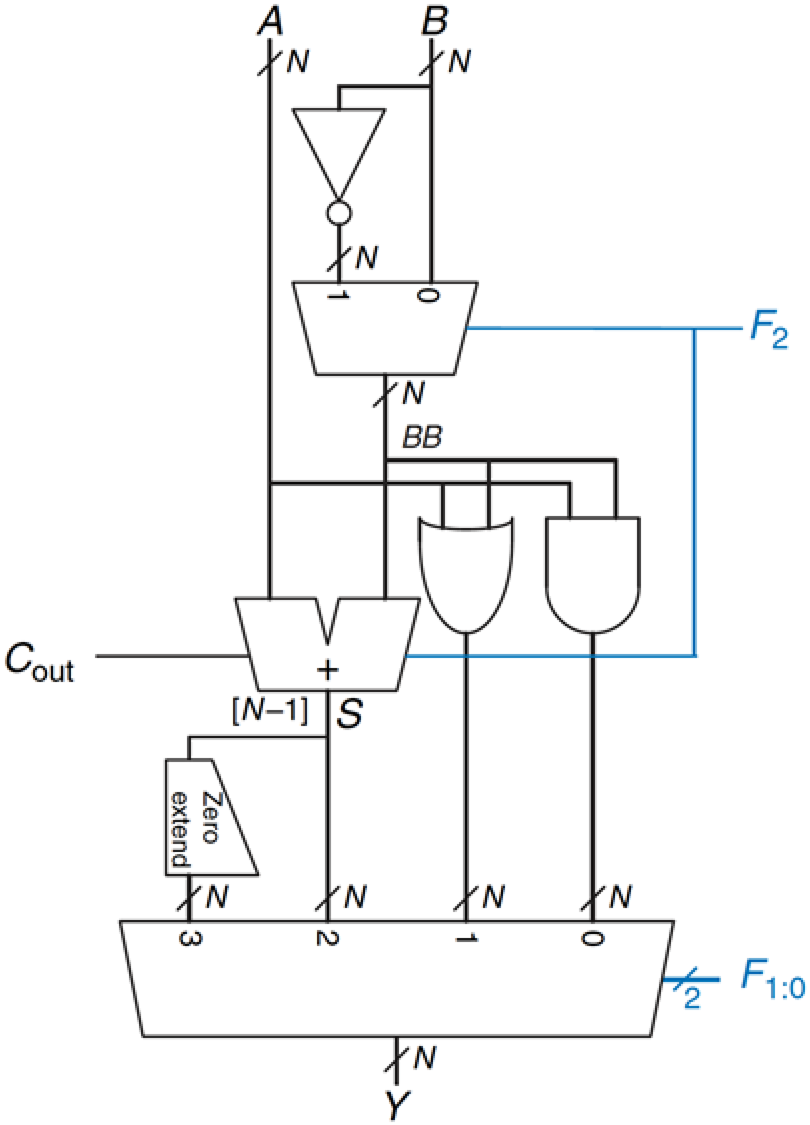


Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

CPU (Central Processing Unit) [module 1]:

- The brain of the microcontroller, responsible for executing instructions.
    - **Control Unit**
    - **Arithmetic Logic Unit (ALU)**
    - **Registers**
    - **Memory**

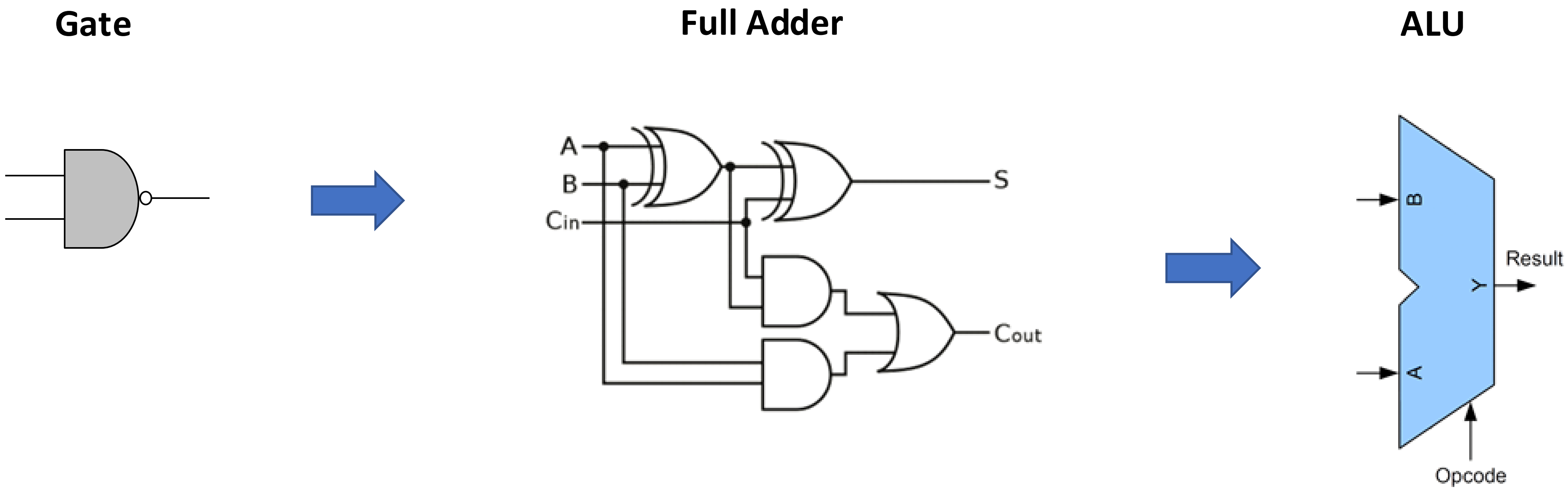
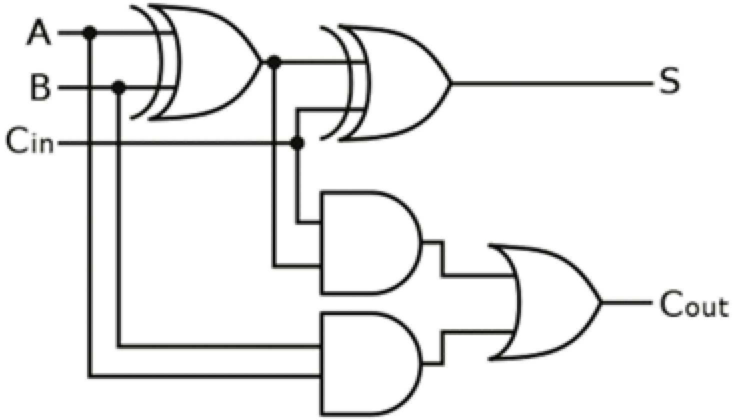


Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

Task: 15 min
Simulate a Full adder



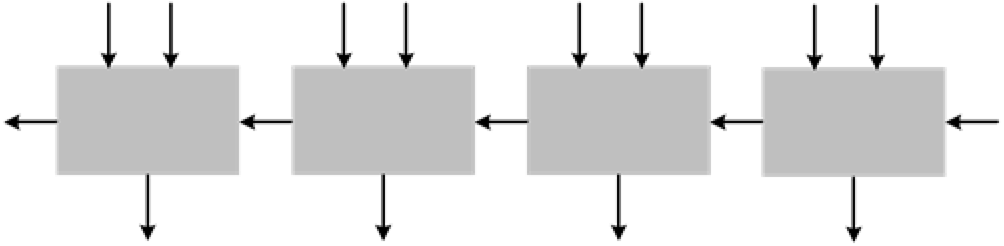
? Go to https://logic.ly/demo/

? Implement a one-bit full adder

? Fill out the truth table (on the right)

? Implement a 4-bit adder (as illustrated below)

A	B	C_{in}	C_{out}	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

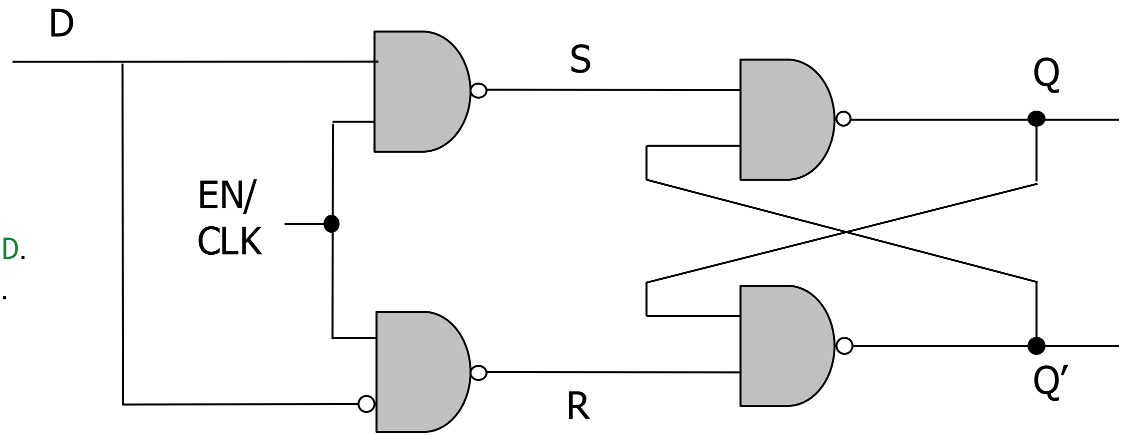


Combinational Logic Circuits (Common Examples)

- **Gated D Latch:** a type of digital storage device. It is a basic building block of flip-flops and is used to store one bit of data.

Input		Output
EN/CLK	D	Q
0	0	Q_{prev}
0	1	Q_{prev}
1	0	0
1	1	1

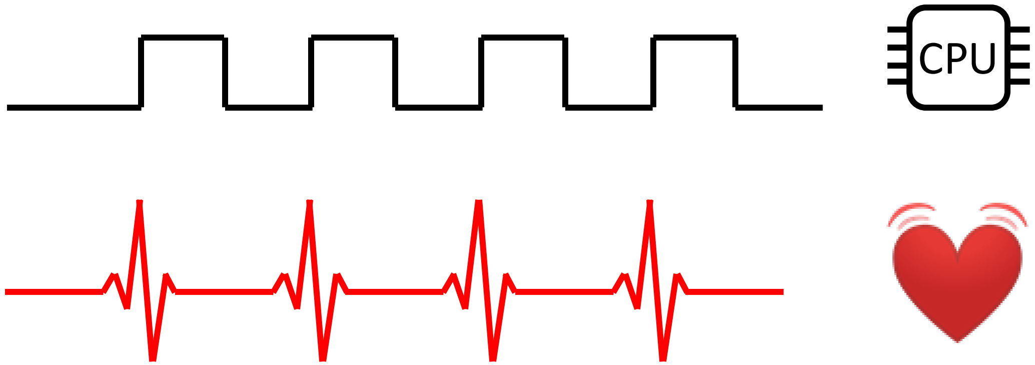
- **Inputs**:
  - **Data Input (D)**: This is the input signal that the latch will store.
  - **Enable/Clock Input (CLK or EN)**: This control signal determines when the data input is sampled and stored in the latch. **(Write enable)**
- **Outputs**
  - **Q Output**: The output of the latch, which represents the stored data.
  - **Q' (Q-bar) Output**: The complement of the Q output.
- **When Enable (EN/CLK) is High (1)**:
  - The latch is in the "*transparent*" mode,
    - …meaning the output Q follows the input D.
    - Whatever value is present on D is passed directly to Q.
  - If D is 1, Q will become 1.
  - If D is 0, Q will become 0.
- **When Enable (EN/CLK) is Low (0)**:
  - The latch is in the *"latched"* or *"hold"* mode.
    - The output Q retains its previous state, regardless of any changes in the input D.
  - The stored value remains the same until the enable signal (**EN/CLK**) goes high again.



CPU (Central Processing Unit) [module 1]:

- The brain of the microcontroller, responsible for executing instructions.
  - **Control Unit**
  - **Arithmetic Logic Unit (ALU)**
  - **Registers**
    - **Example:** a 4-bit register (data is referenced as Q[3:0])
    - A single Clock signal for all latches for simultaneous writes
  - **Memory**
  - …all build on logic gates..

Clock



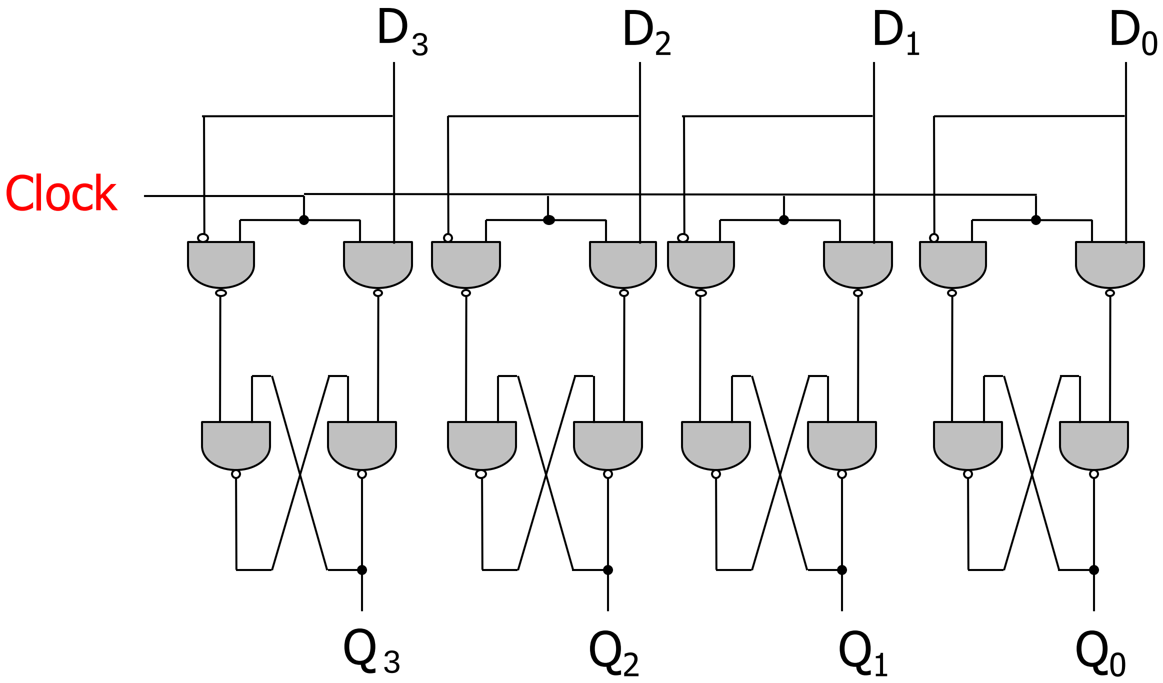


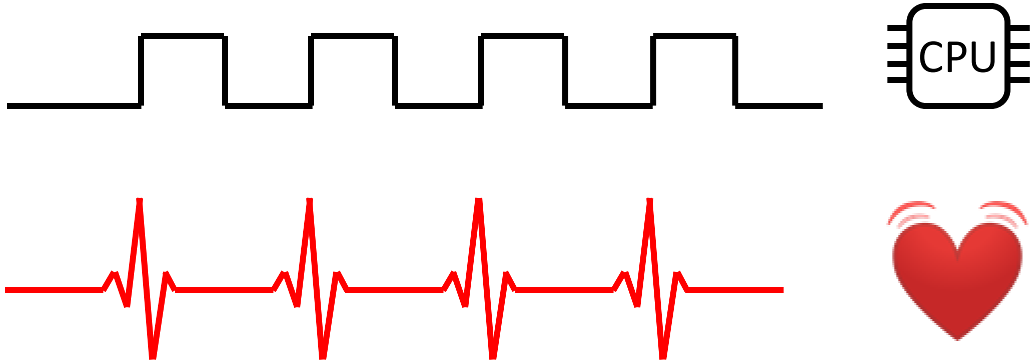
Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

CPU (Central Processing Unit) [module 1]:

- The brain of the microcontroller, responsible for executing instructions.
  - **Control Unit**
  - **Arithmetic Logic Unit (ALU)**
  - **Registers**
    - **Example:** a 4-bit register (data is referenced as Q[3:0])
    - A single Clock signal for all latches for simultaneous writes
  - **Memory**

  - …all build on logic gates..

Clock



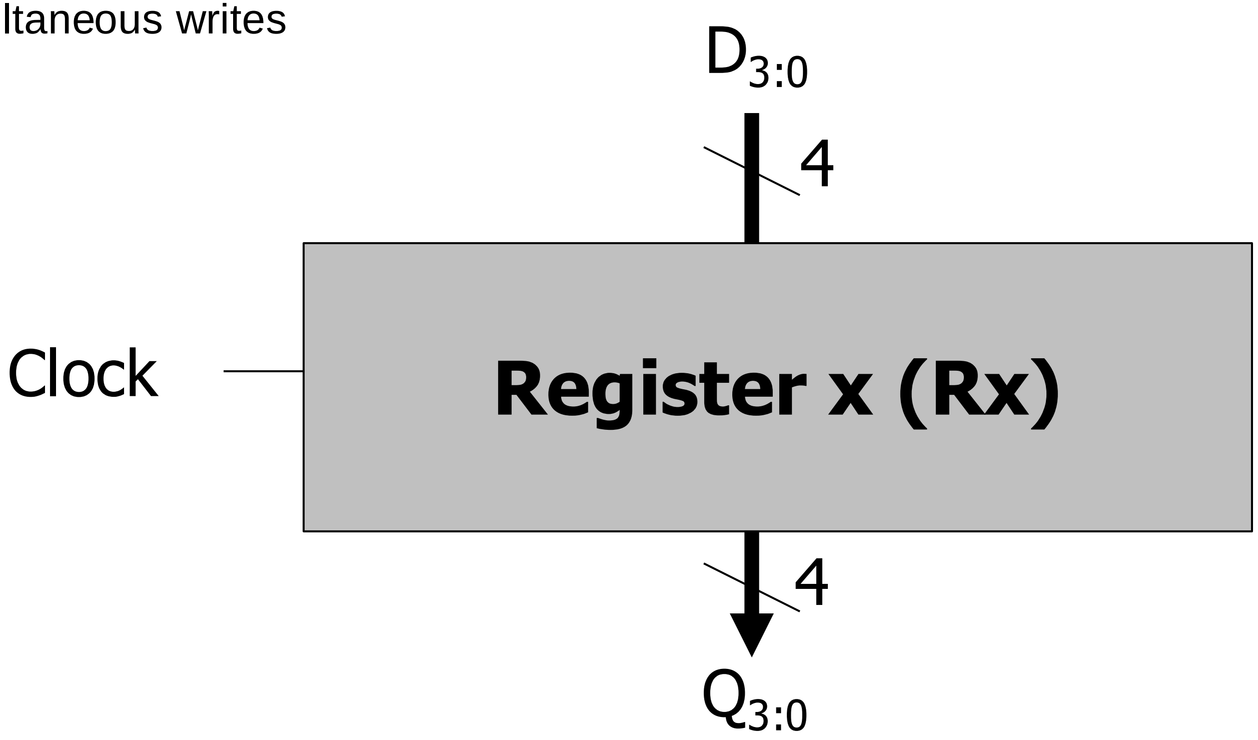
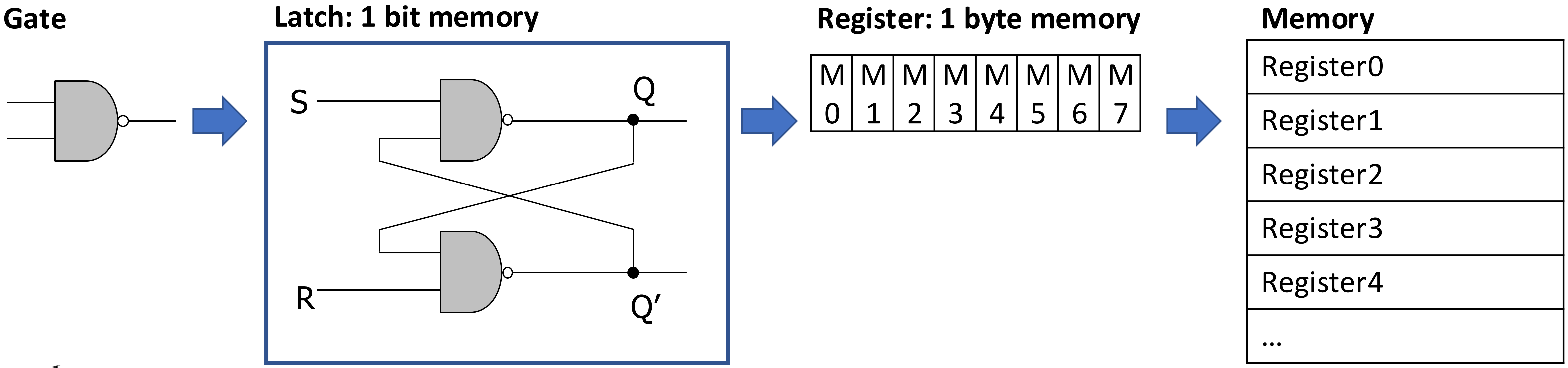


Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

CPU (Central Processing Unit) [module 1]:

- The brain of the microcontroller, responsible for executing instructions.
  - **Control Unit**
  - **Arithmetic Logic Unit (ALU)**
  - **Registers**
    - **Example:** a 4-bit register (data is referenced as Q[3:0])
    - A single Clock signal for all latches for simultaneous writes
  - **Memory**



Register: 1 byte memory

M 0	M 1	M 2	M 3	M 4	M 5	M 6	M 7

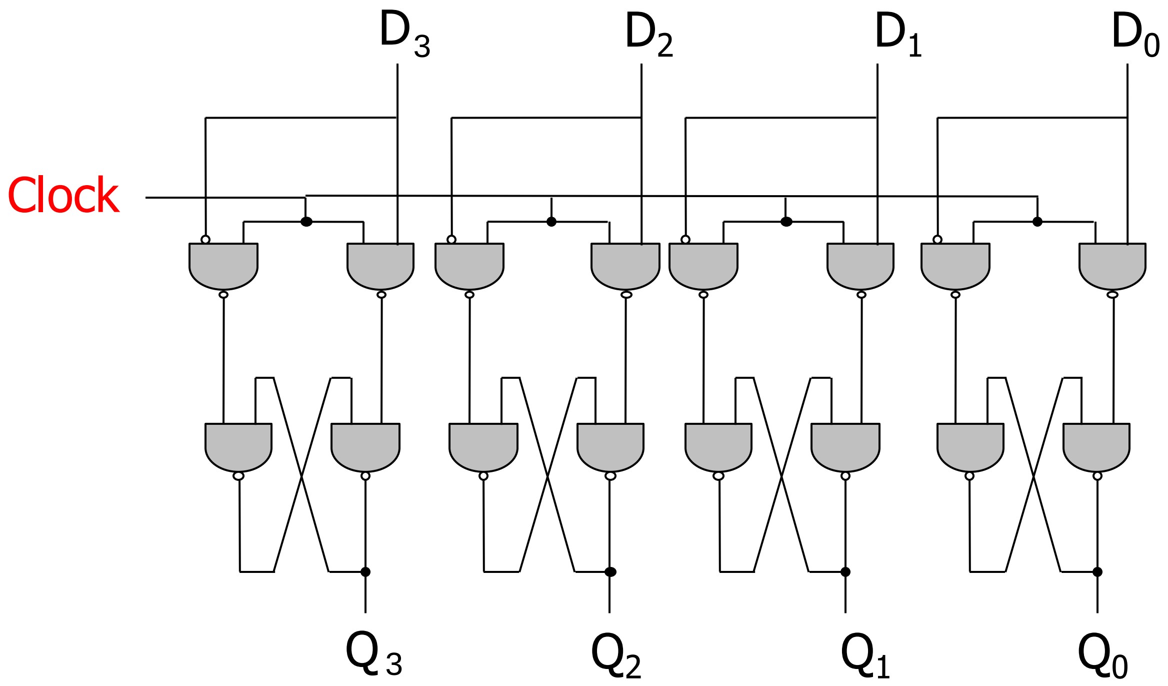
Memory

Register0
Register1
Register2
Register3
Register4
…

Illustration: Datateknik course, W2-W4 (T540050101, by Ali Sahafi)

Task (10 min): implement a D latch

- Go to https://logic.ly/demo/
- Implement and simulate one bit memory using a gated D latch

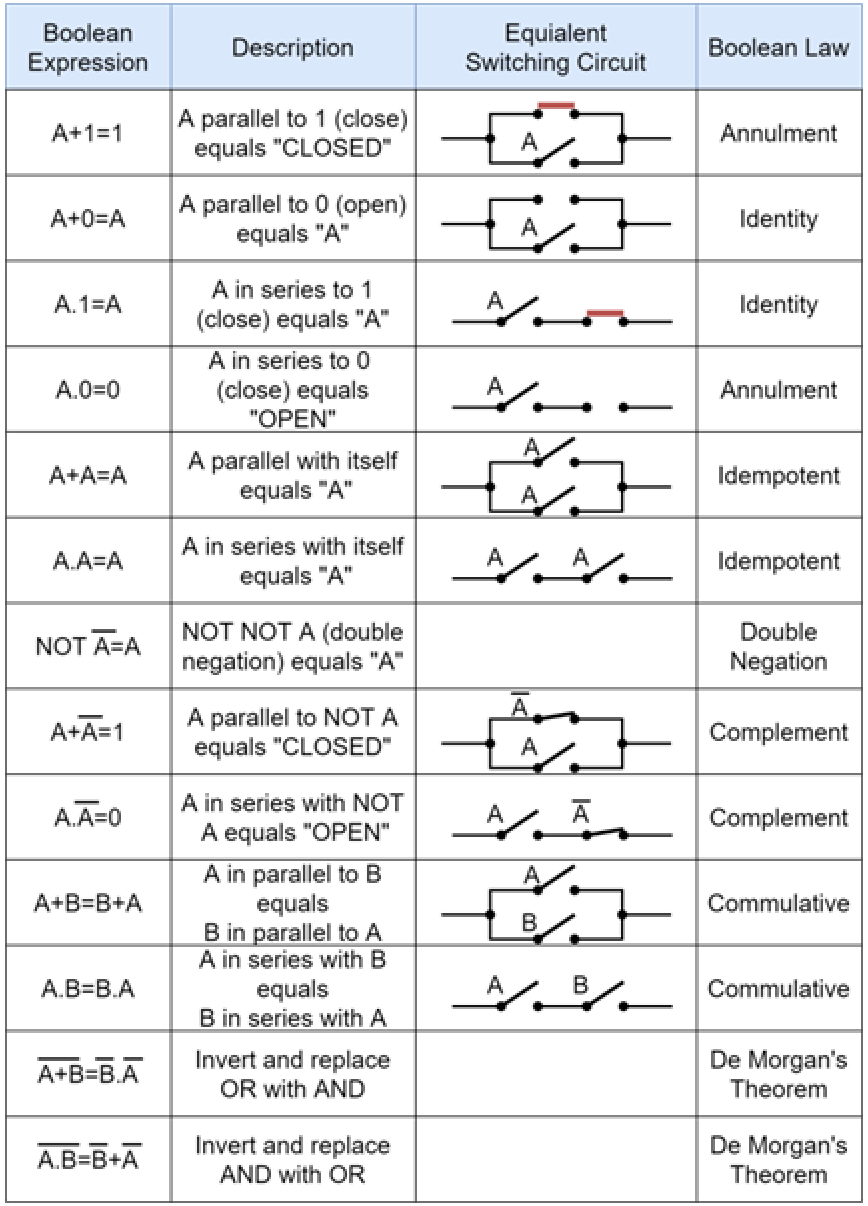


Laws of Boolean Algebra

Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit ”

- …provide the foundational rules for simplifying logic circuits
  - **Optimization of circuit design**
    - Large logic expressions can be minimized
    - …which translates to fewer gates in a circuit.
    - …less gates, less heat…
  - **Increased processing speed**
    - …fewer components, faster processing time
  - **Reliability**
    - Less components, less complexity
      - …meaning higher reliability



Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}=\overline{B}.\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}=\overline{B}+\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

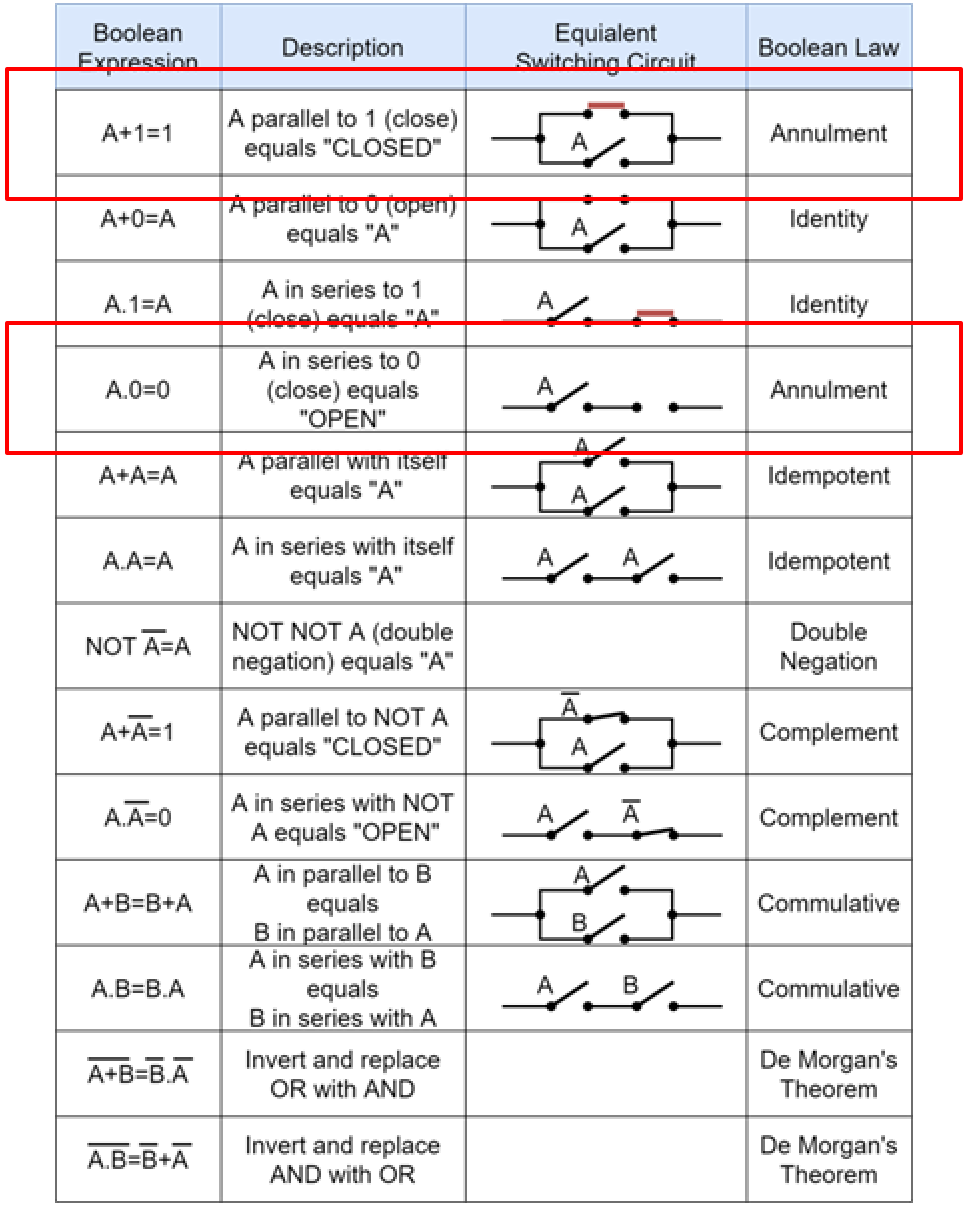
Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit.”

- **Annulment laws** (Null laws)
  - OR null law:
    - …anything OR’ed with 1 is 1.
  - AND null law:
    - …anything AND’ed with 0 is 0.

(Input A has no effect)



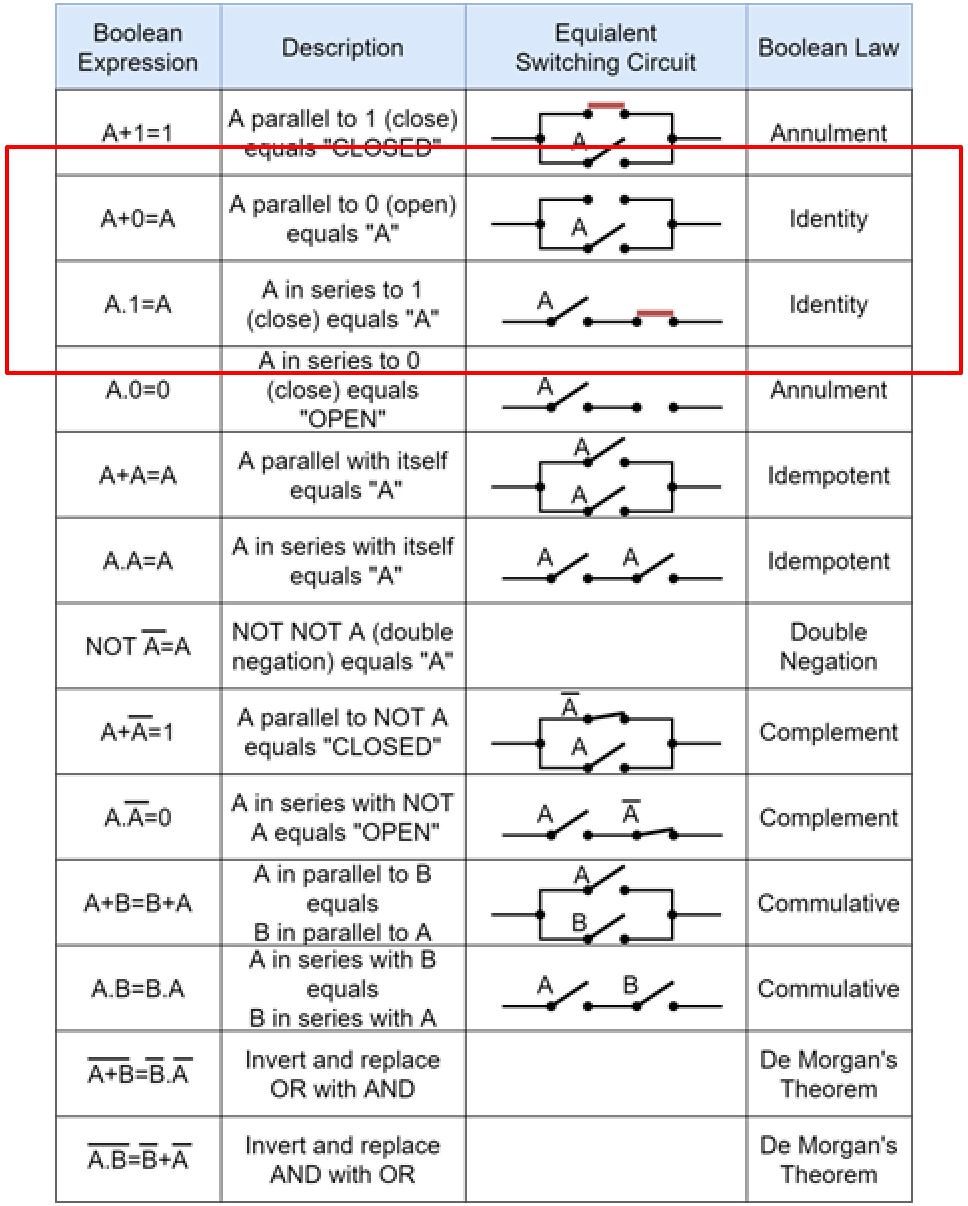
Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}=\overline{B}.\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}=\overline{B}+\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit.”

- **Identity laws:**
  - …a variable retains its identity
    - when
      - …a variable OR’ed to 0 will always yield the variable itself, or
      - …a variable AND’ed by 1 will always yield the variable itself



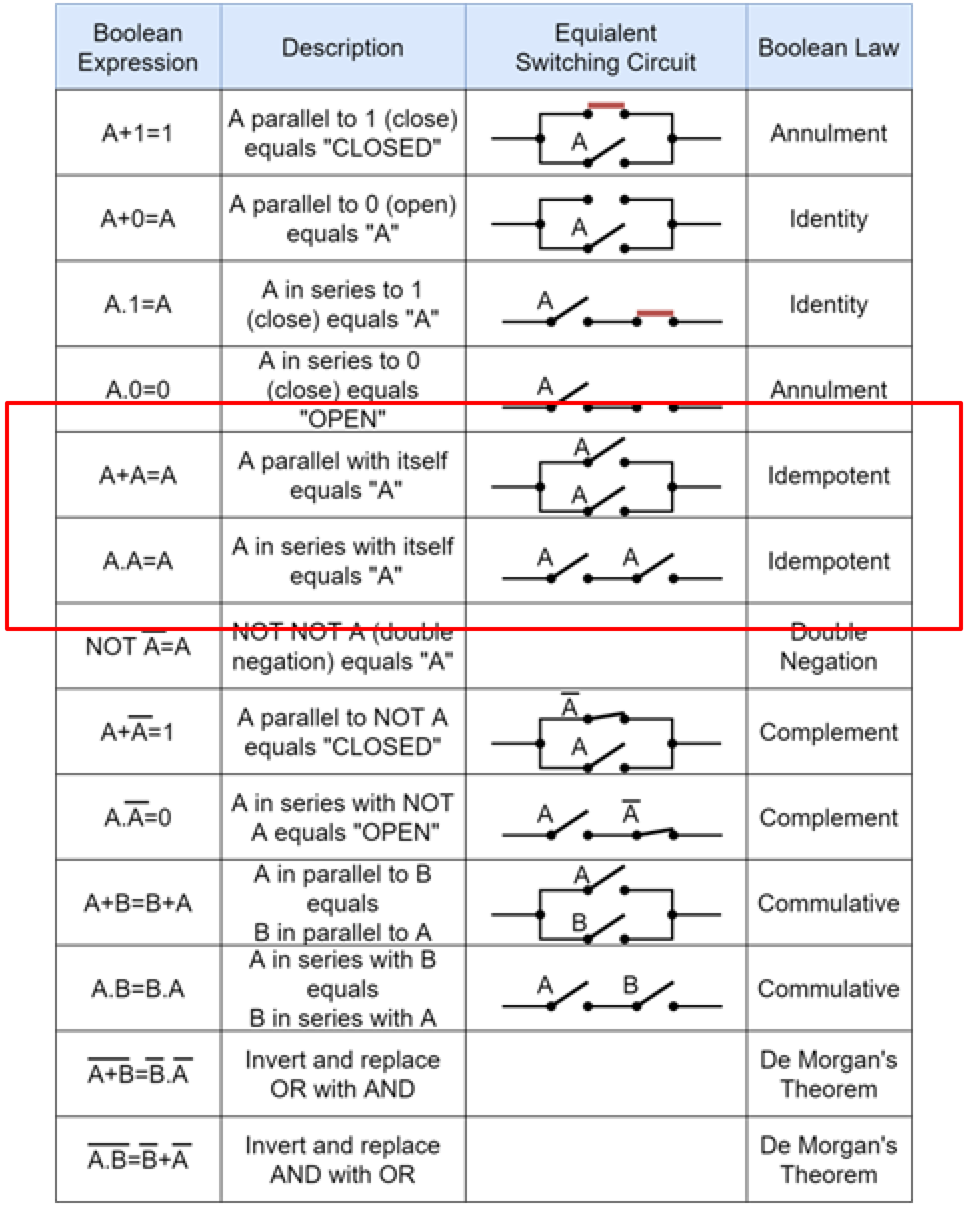
Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"	A	Annulment
A+0=A	A parallel to 0 (open) equals "A"	A	Identity
A.1=A	A in series to 1 (close) equals "A"	A	Identity
A.0=0	A in series to 0 (close) equals "OPEN"	A	Annulment
A+A=A	A parallel with itself equals "A"	A A	Idempotent
A.A=A	A in series with itself equals "A"	A A	Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"	$\overline{A}$ A	Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"	A $\overline{A}$	Complement
A+B=B+A	A in parallel to B equals B in parallel to A	A B	Commulative
A.B=B.A	A in series with B equals B in series with A	A B	Commulative
$\overline{A+B}$=$\overline{B}$.$\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}$=$\overline{B}$+$\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit.”

- **Identity laws:**
  - …a variable retains its identity
- **Idempotent laws**
  - …combining a variable with itself using an OR or AND operation yields the variable itself.



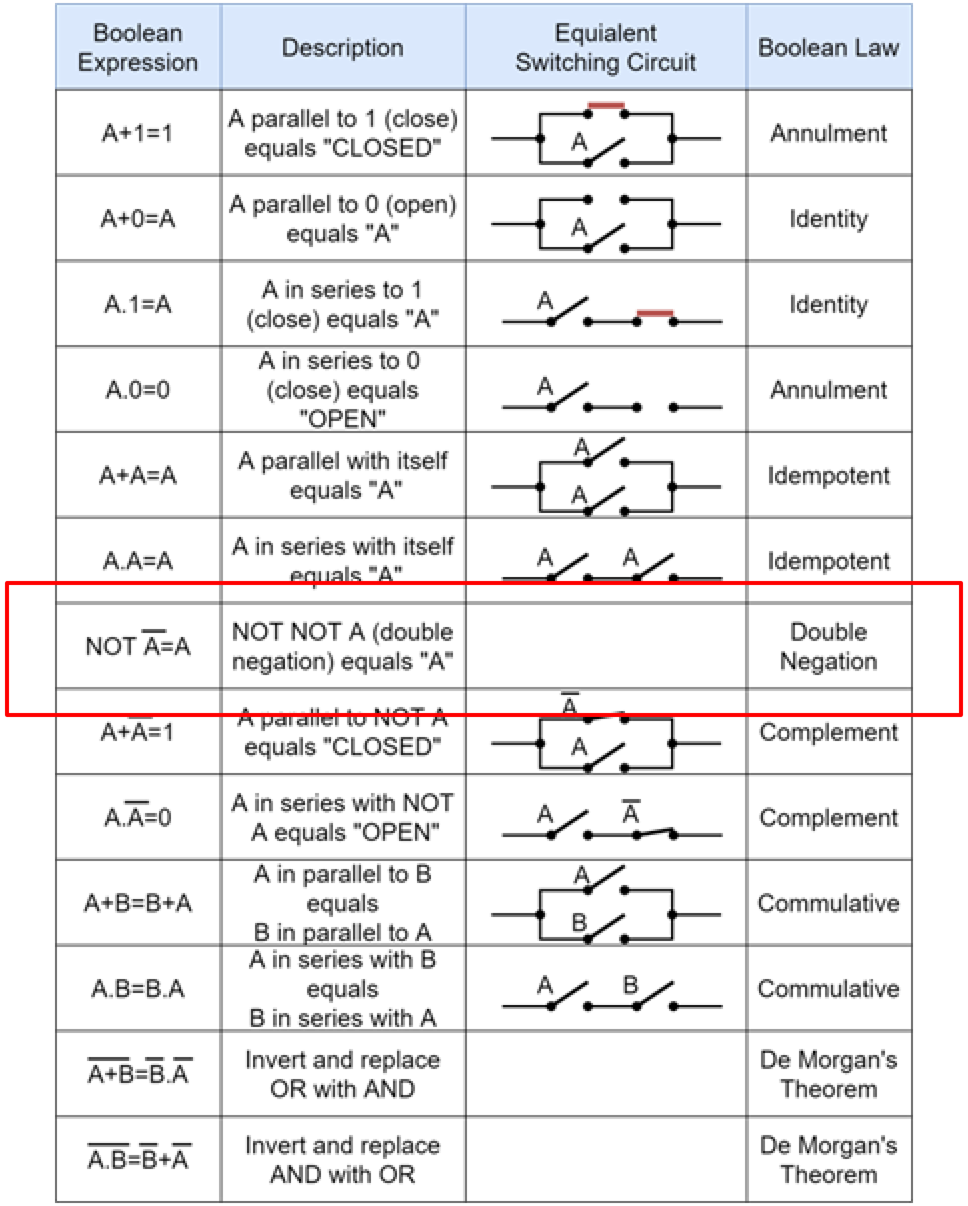
Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"	A	Annulment
A+0=A	A parallel to 0 (open) equals "A"	A	Identity
A.1=A	A in series to 1 (close) equals "A"	A	Identity
A.0=0	A in series to 0 (close) equals "OPEN"	A	Annulment
A+A=A	A parallel with itself equals "A"	A A	Idempotent
A.A=A	A in series with itself equals "A"	A A	Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
$A+\overline{A}=1$	A parallel to NOT A equals "CLOSED"	$\overline{A}$ A	Complement
$A.\overline{A}=0$	A in series with NOT A equals "OPEN"	A $\overline{A}$	Complement
A+B=B+A	A in parallel to B equals B in parallel to A	A B	Commulative
A.B=B.A	A in series with B equals B in series with A	A B	Commulative
$\overline{A+B}=\overline{B}.\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}=\overline{B}+\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

Laws of Boolean Algebra

"Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit."

- **Identity laws:**
  - …a variable retains its identity
- **Idempotent laws**
  - …combining a variable with itself using an OR or AND operation yields the variable itself.
- **Double Negation law**
  - …a variable negated twice will return to its original value



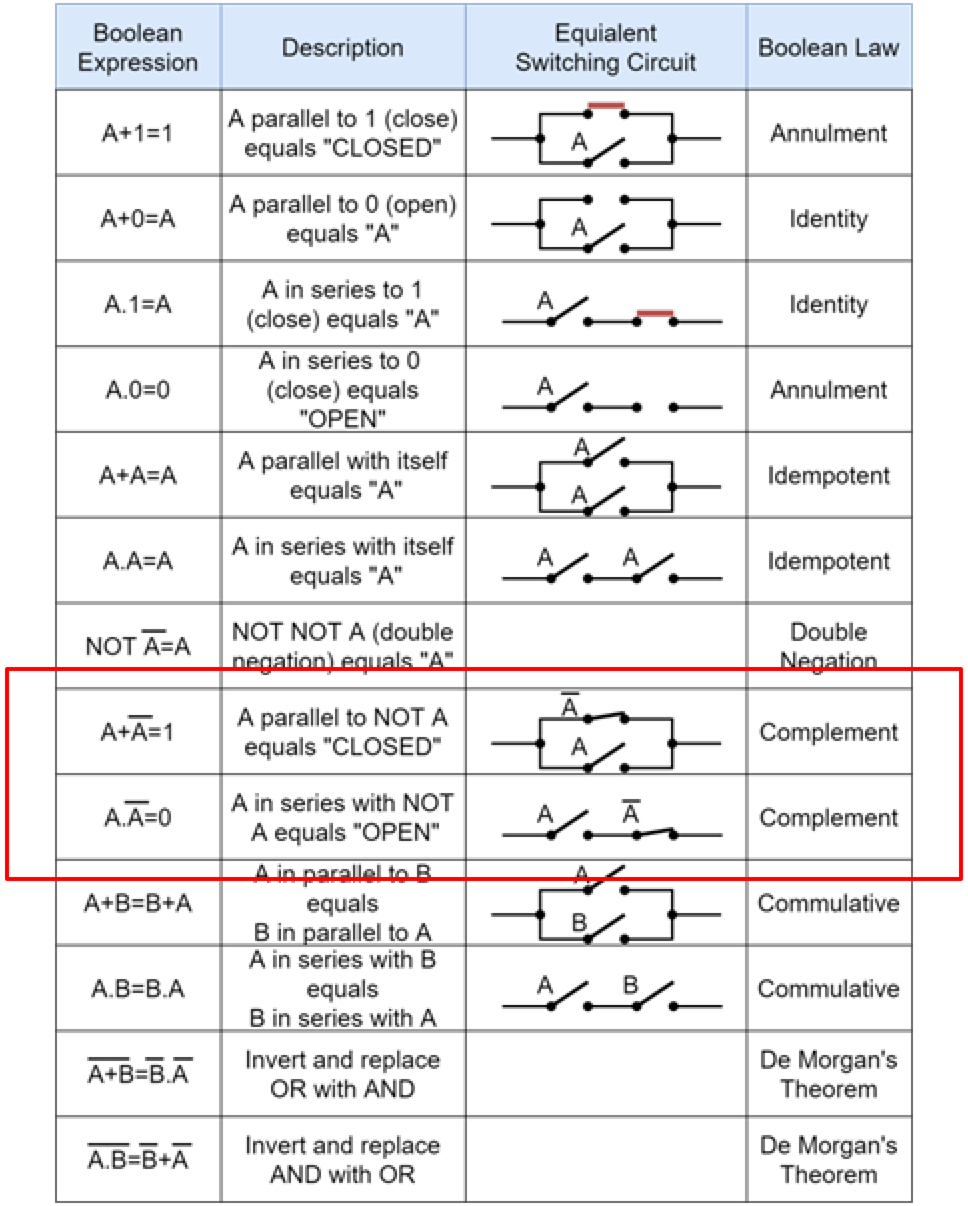
Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}=\overline{B}.\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}=\overline{B}+\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit.”

- **Identity laws:**
  - …a variable retains its identity
- **Idempotent laws**
  - …combining a variable with itself using an OR or AND operation yields the variable itself.
- **Double Negation law**
  - …a variable negated twice will return to its original value
- **Complement laws**
  - … a variable OR’ed with its complement (inverse) will always yield 1, or
  - …a variable AND’ed with its complement will always yield 0



Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}=\overline{B}.\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}=\overline{B}+\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

Laws of Boolean Algebra

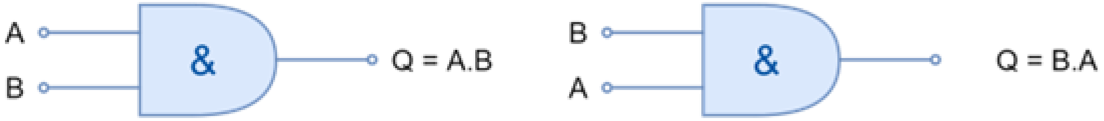
“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit.”

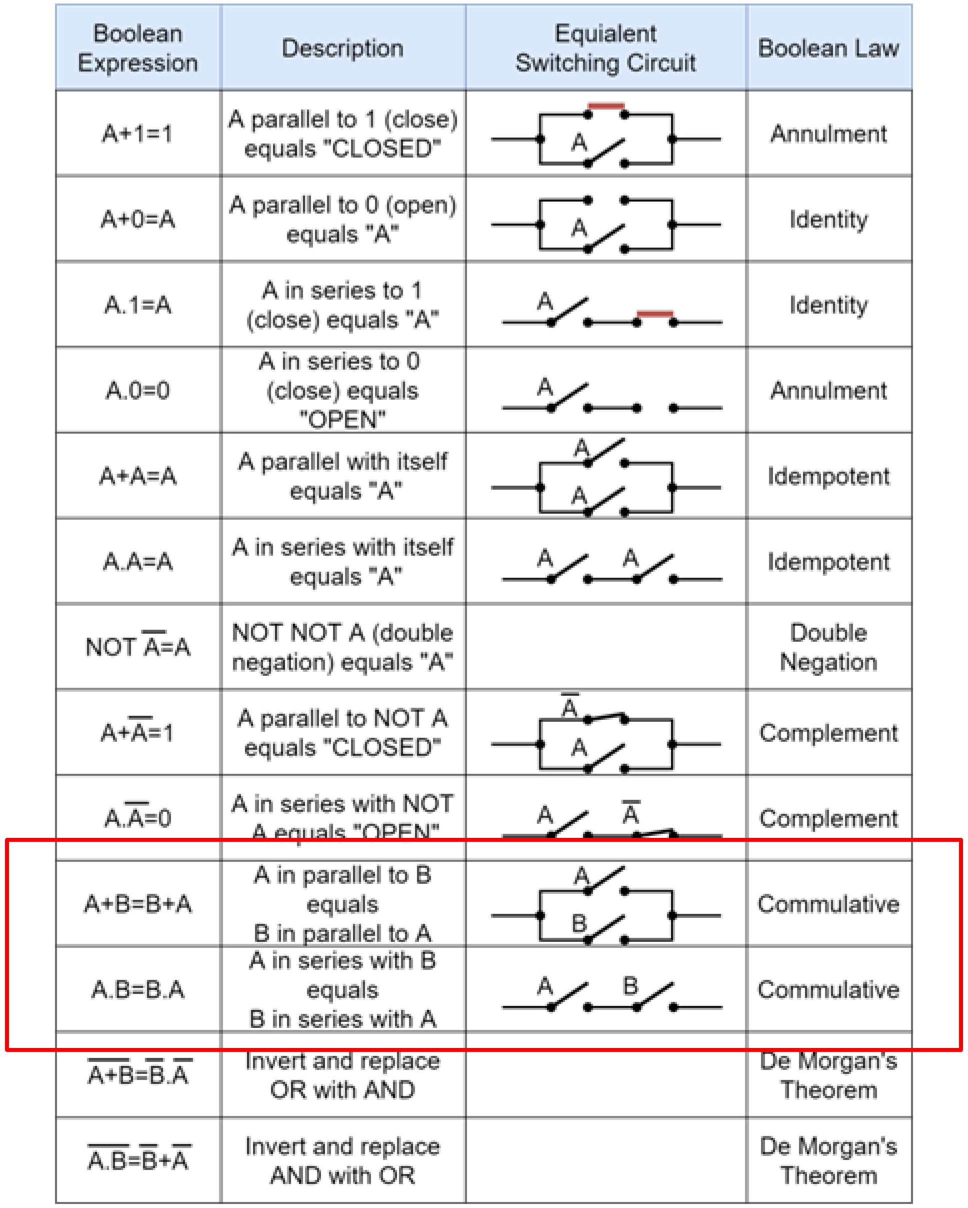
- **Commutative law**
  - ○ …the order of variables in an OR or AND operation does not affect the result.

Commutative Law

A.B = A.B

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





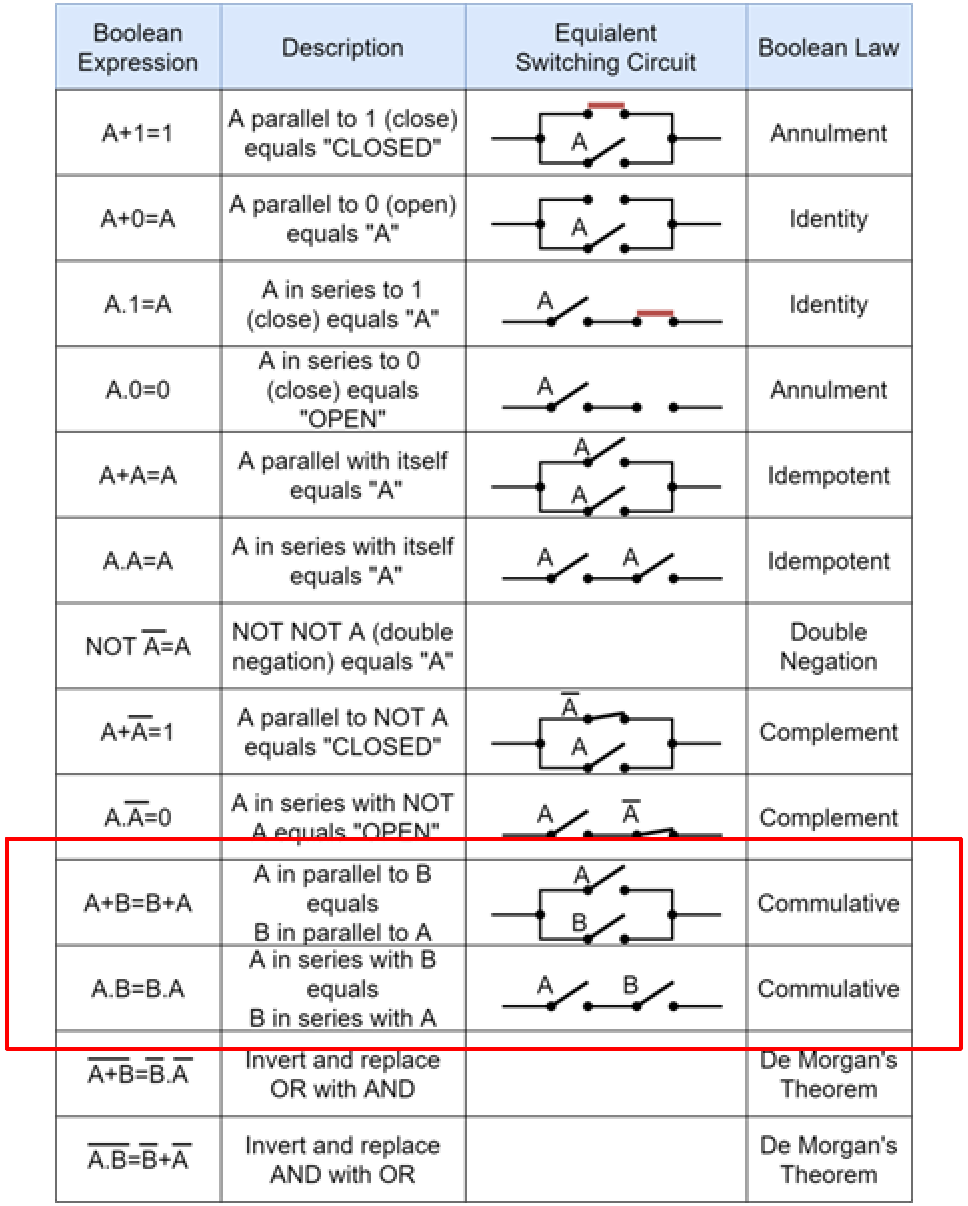
Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}=\overline{B}.\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}=\overline{B}+\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit.”

- **Commutative law**
  - …the order of variables in an OR or AND operation does not affect the result.
  - …meaning, your boolean expressions to be rearranged without changing the outcome
    - Useful for
      - Simplifying expressions
      - Designing flexible logic circuits.



Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}=\overline{B}.\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}=\overline{B}+\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

Commutative Law

A.B = A.B

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

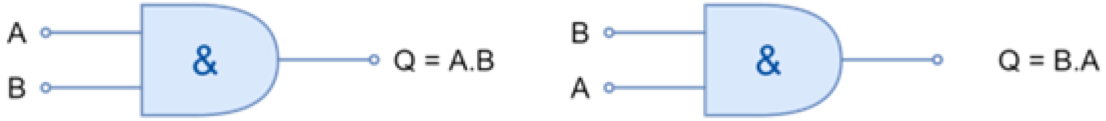
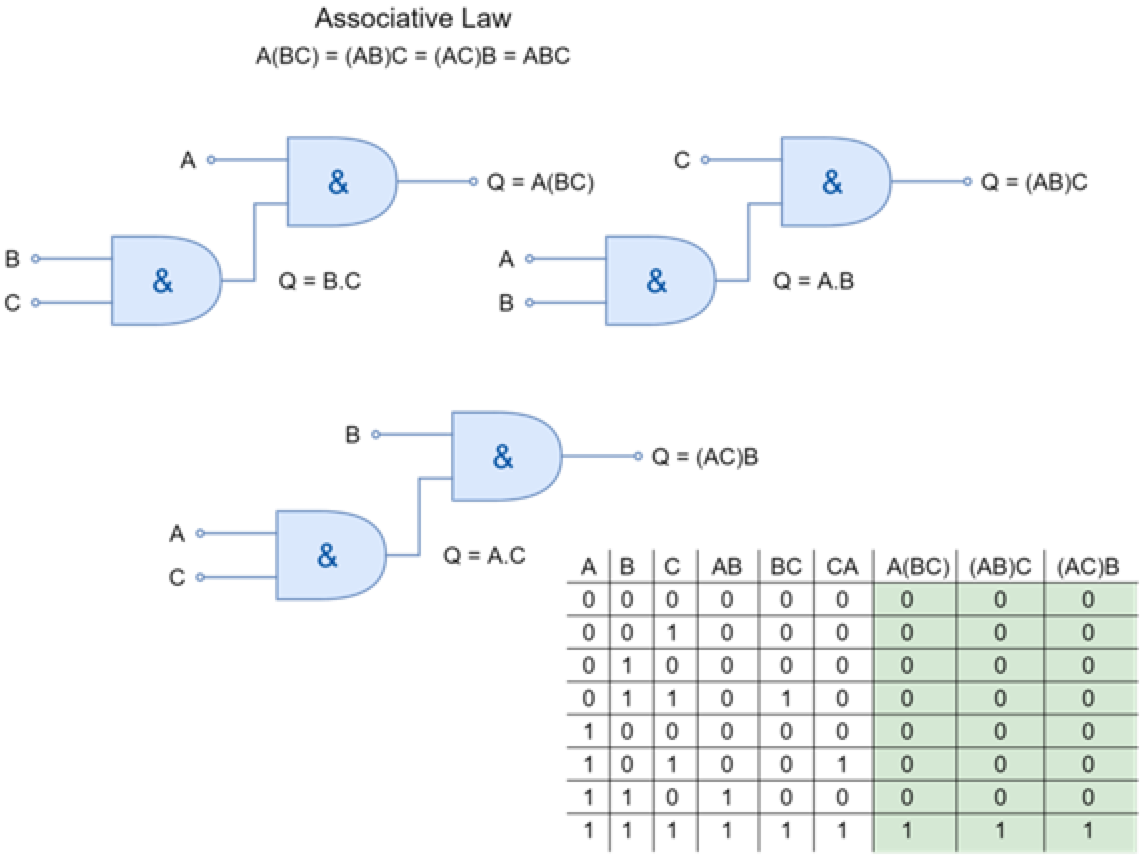


Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

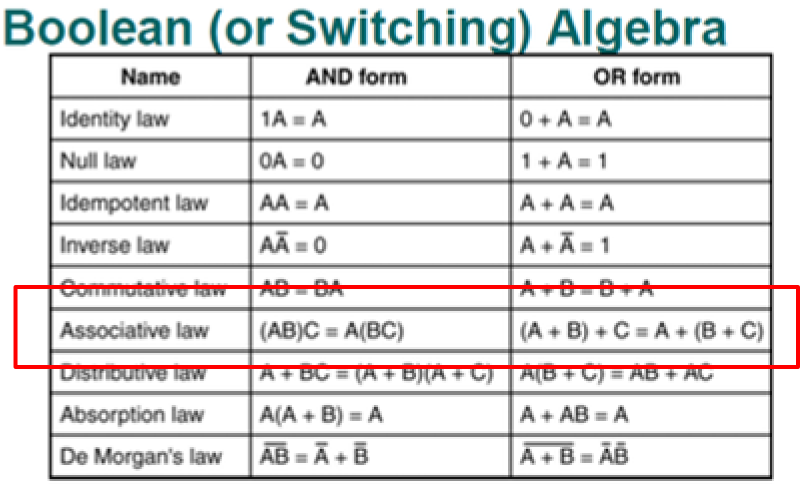
Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit.”

- **Commutative law**
  - …the order of variables in an OR or AND operation does not affect the result.
- **Associative law**
  - …grouping of variables in an OR or AND operation does not affect the result.



A	B	C	AB	BC	CA	A(BC)	(AB)C	(AC)B
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	1	1	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	1	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0
1	1	1	1	1	1	1	1	1



Boolean (or Switching) Algebra

Name	AND form	OR form
Identity law	1A = A	0 + A = A
Null law	0A = 0	1 + A = 1
Idempotent law	AA = A	A + A = A
Inverse law	$A\bar{A} = 0$	$A + \bar{A} = 1$
Commutative law	AB = BA	A + B = B + A
Associative law	(AB)C = A(BC)	(A + B) + C = A + (B + C)
Distributive law	A + BC = (A + B)(A + C)	A(B + C) = AB + AC
Absorption law	A(A + B) = A	A + AB = A
De Morgan's law	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}\bar{B}$

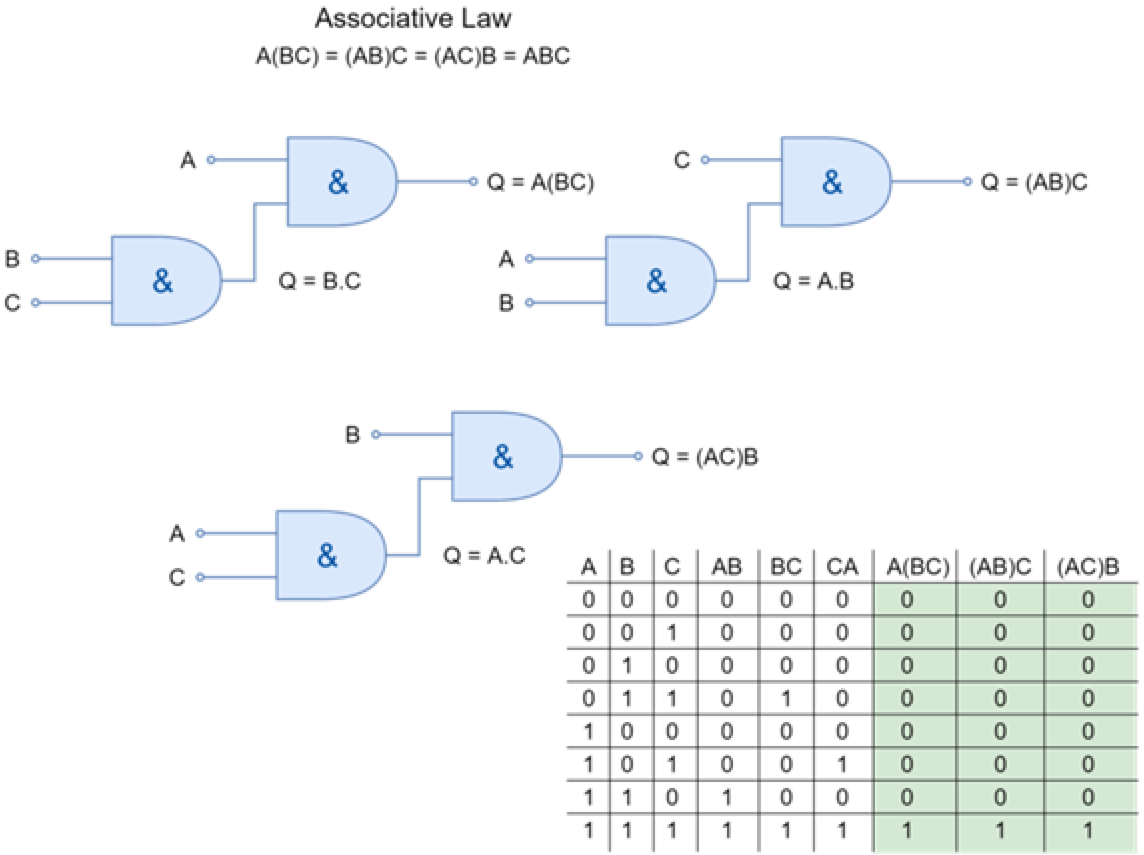
Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/
http://booleanalgebraforyou.weebly.com/laws.html

Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit.”

- **Commutative law**
  - …the order of variables in an OR or AND operation does not affect the result.

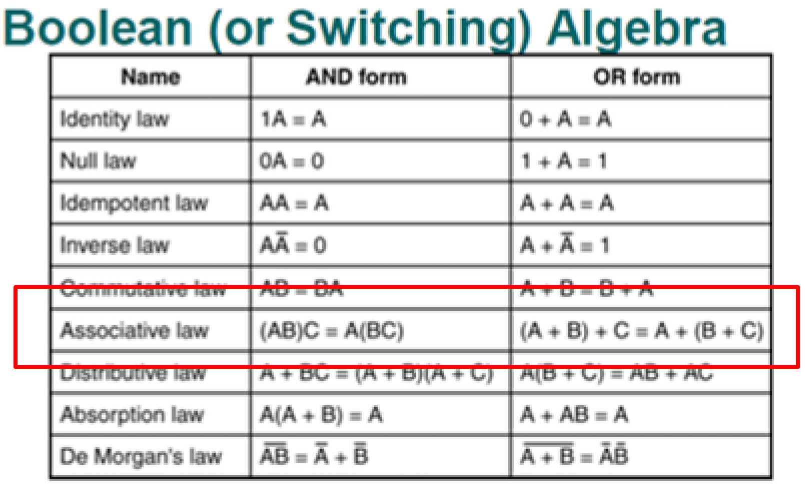
- **Associative law**
  - …grouping of variables in an OR or AND operation does not affect the result.
  - …meaning,
    - when multiple instances of the same operation are combined,
    - the order in which they are grouped can be rearranged without changing the outcome.



Associative Law

A(BC) = (AB)C = (AC)B = ABC

A	B	C	AB	BC	CA	A(BC)	(AB)C	(AC)B
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	1	1	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	1	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0
1	1	1	1	1	1	1	1	1



Boolean (or Switching) Algebra

Name	AND form	OR form
Identity law	1A = A	0 + A = A
Null law	0A = 0	1 + A = 1
Idempotent law	AA = A	A + A = A
Inverse law	$A\bar{A} = 0$	$A + \bar{A} = 1$
Commutative law	AB = BA	A + B = B + A
Associative law	(AB)C = A(BC)	(A + B) + C = A + (B + C)
Distributive law	A + BC = (A + B)(A + C)	A(B + C) = AB + AC
Absorption law	A(A + B) = A	A + AB = A
De Morgan's law	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}\bar{B}$

Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/
http://booleanalgebraforyou.weebly.com/laws.html

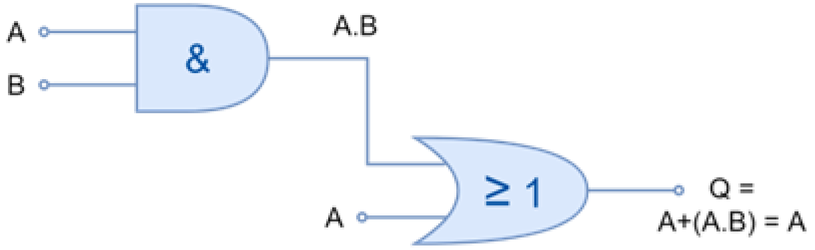
Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit.”

- Commutative law
  - …the order of variables in an OR or AND operation does not affect the result.
- Associative law
  - …grouping of variables in an OR or AND operation does not affect the result.
- **Absorption law**
  - …certain combinations of variables can be "absorbed" into simpler expressions.
  - …meaning, simplifying expressions by eliminating redundant terms.
- **Distributive Law**
  - …a variable to be distributed across an AND or OR operation.

Absorption Law

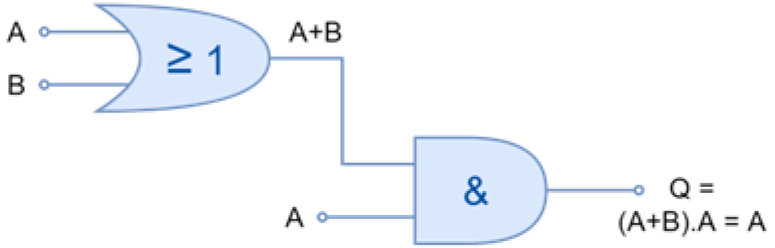
A+(A.B) = (A.1) + (A.B) = A(1+B) = A



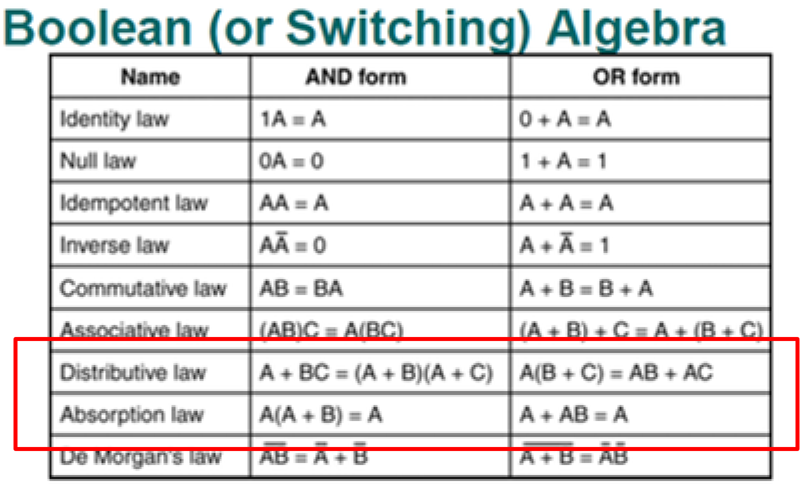
A	B	A.B	Q
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Absorption Law

A(A+B) = (A+0).(A+B) = A+(0.B) = A



A	B	A+B	Q
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1



Boolean (or Switching) Algebra

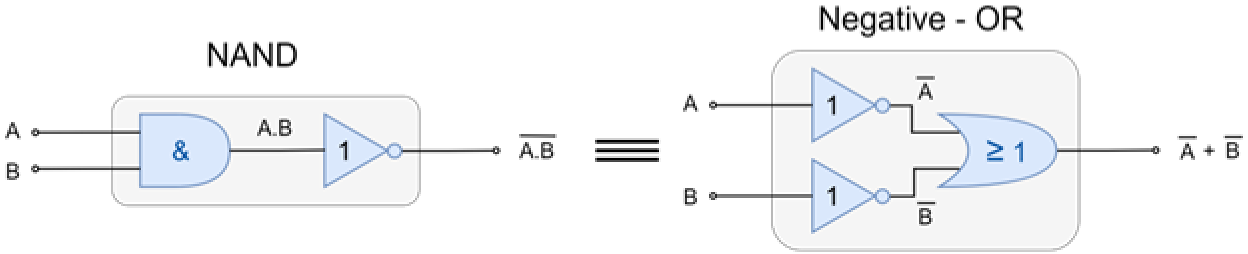
Name	AND form	OR form
Identity law	1A = A	0 + A = A
Null law	0A = 0	1 + A = 1
Idempotent law	AA = A	A + A = A
Inverse law	$A\bar{A} = 0$	$A + \bar{A} = 1$
Commutative law	AB = BA	A + B = B + A
Associative law	(AB)C = A(BC)	(A + B) + C = A + (B + C)
Distributive law	A + BC = (A + B)(A + C)	A(B + C) = AB + AC
Absorption law	A(A + B) = A	A + AB = A
De Morgan's law	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}\bar{B}$

Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/
http://booleanalgebraforyou.weebly.com/laws.html

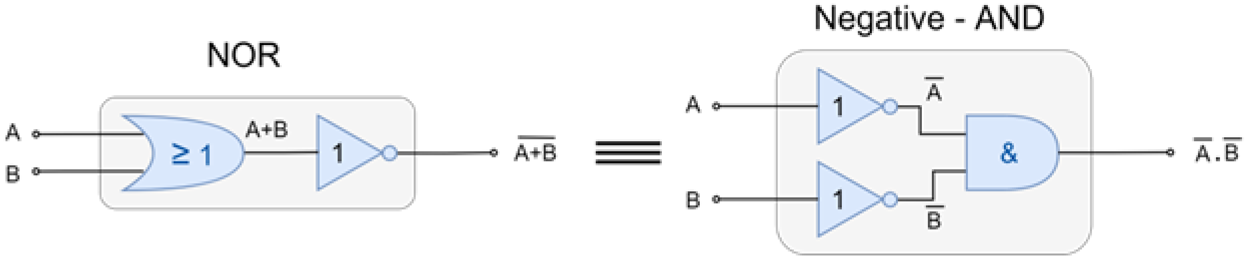
Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit.”

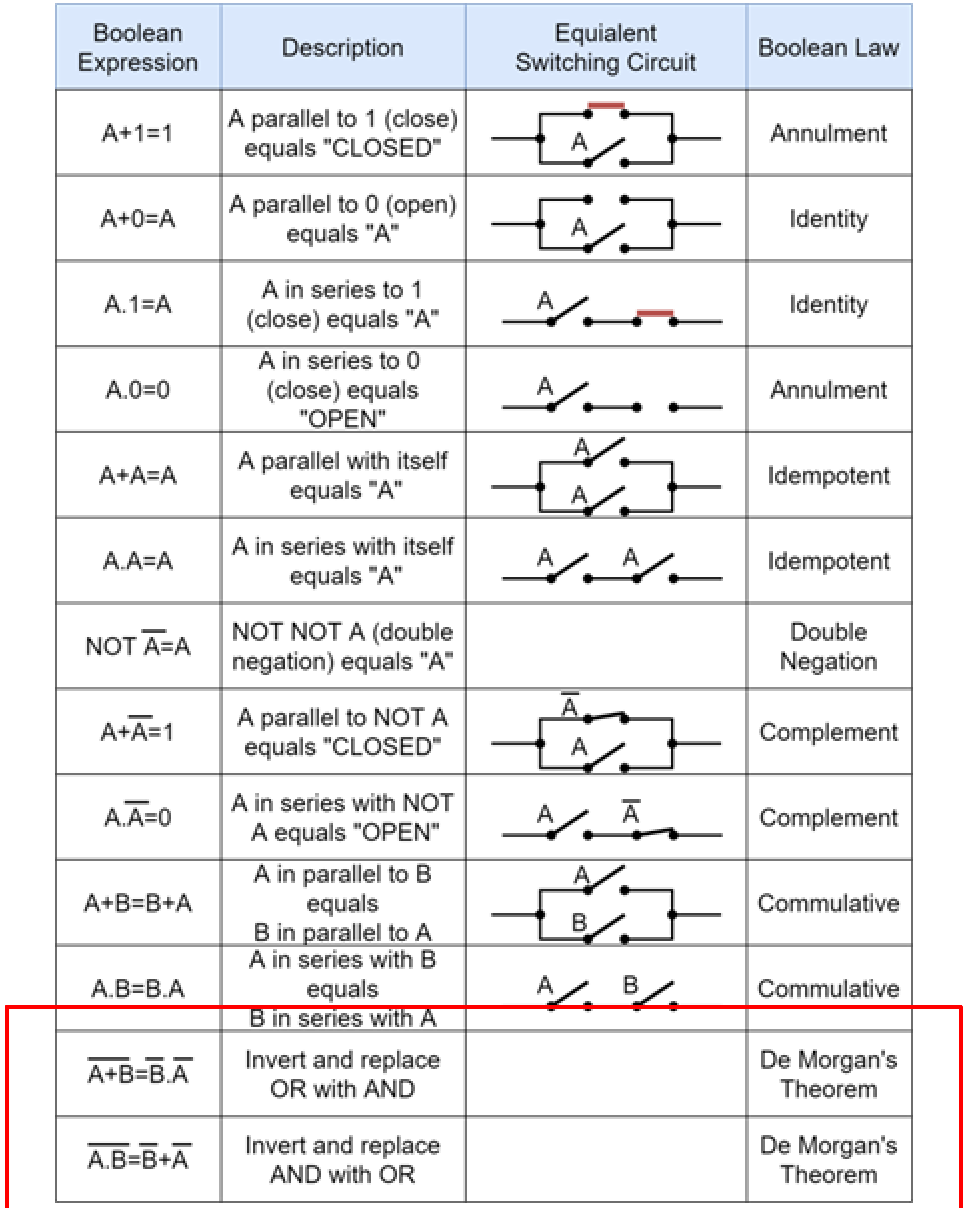
- **De Morgan’s Theorem** (Two fundamental principles)
    - First Theorem (Change an OR to an AND)



-
    - Second Theorem (Change an AND to an OR)



(Only ANDs or ORs makes it easier to simplify the expression)



Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commutative
A.B=B.A	A in series with B equals B in series with A		Commutative
$\overline{A+B}=\overline{B}.\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}=\overline{B}+\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

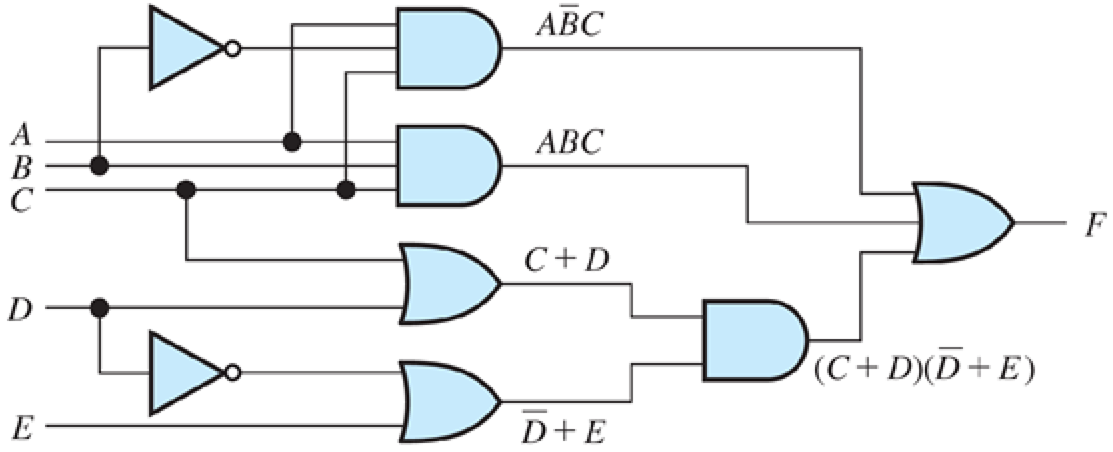
Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

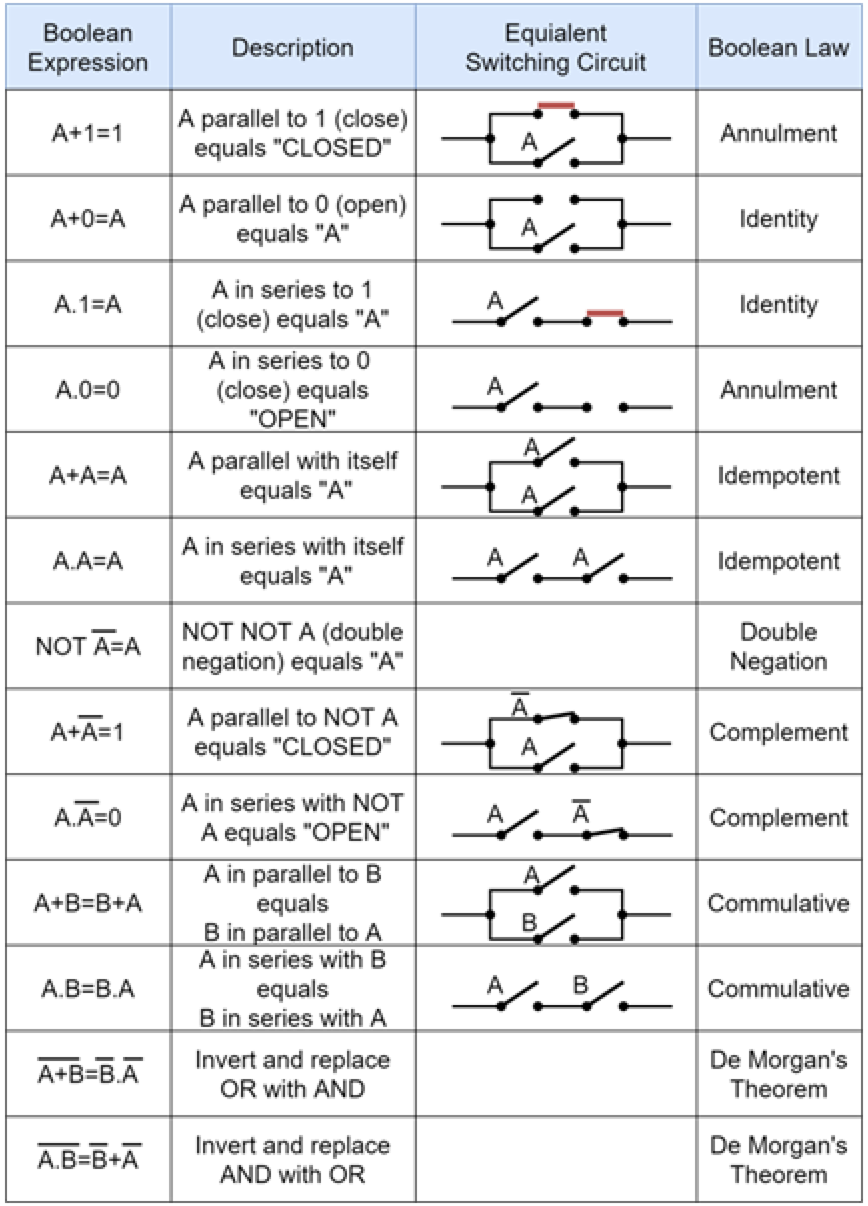
Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit ”

- Example (complex)

$$F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$$





Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"	A	Annulment
A+0=A	A parallel to 0 (open) equals "A"	A	Identity
A.1=A	A in series to 1 (close) equals "A"	A	Identity
A.0=0	A in series to 0 (close) equals "OPEN"	A	Annulment
A+A=A	A parallel with itself equals "A"	A A	Idempotent
A.A=A	A in series with itself equals "A"	A A	Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"	$\overline{A}$ A	Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"	A $\overline{A}$	Complement
A+B=B+A	A in parallel to B equals B in parallel to A	A B	Commulative
A.B=B.A	A in series with B equals B in series with A	A B	Commulative
$\overline{A+B}$=$\overline{B}$.$\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}$=$\overline{B}$+$\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

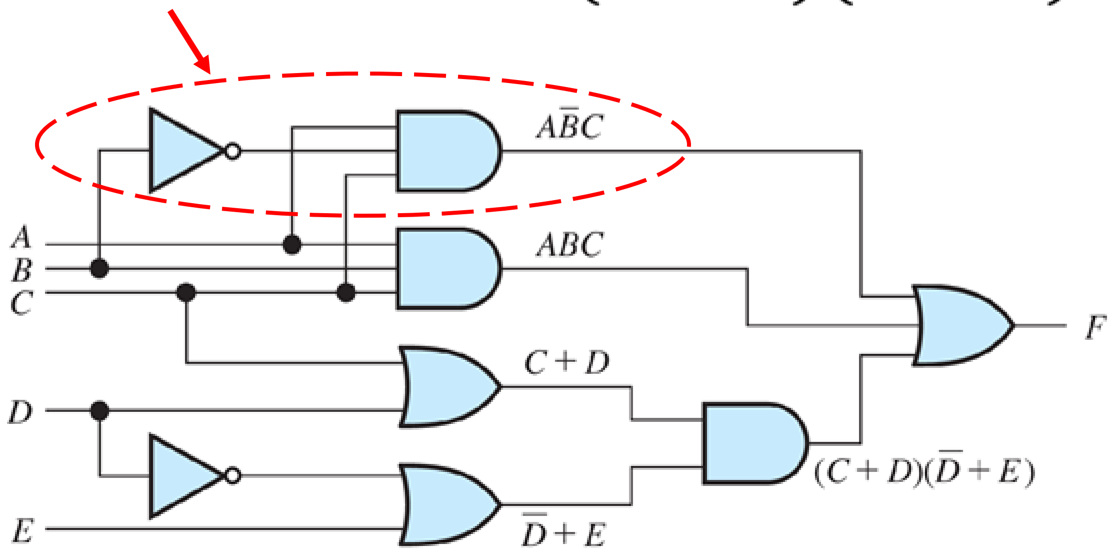
Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

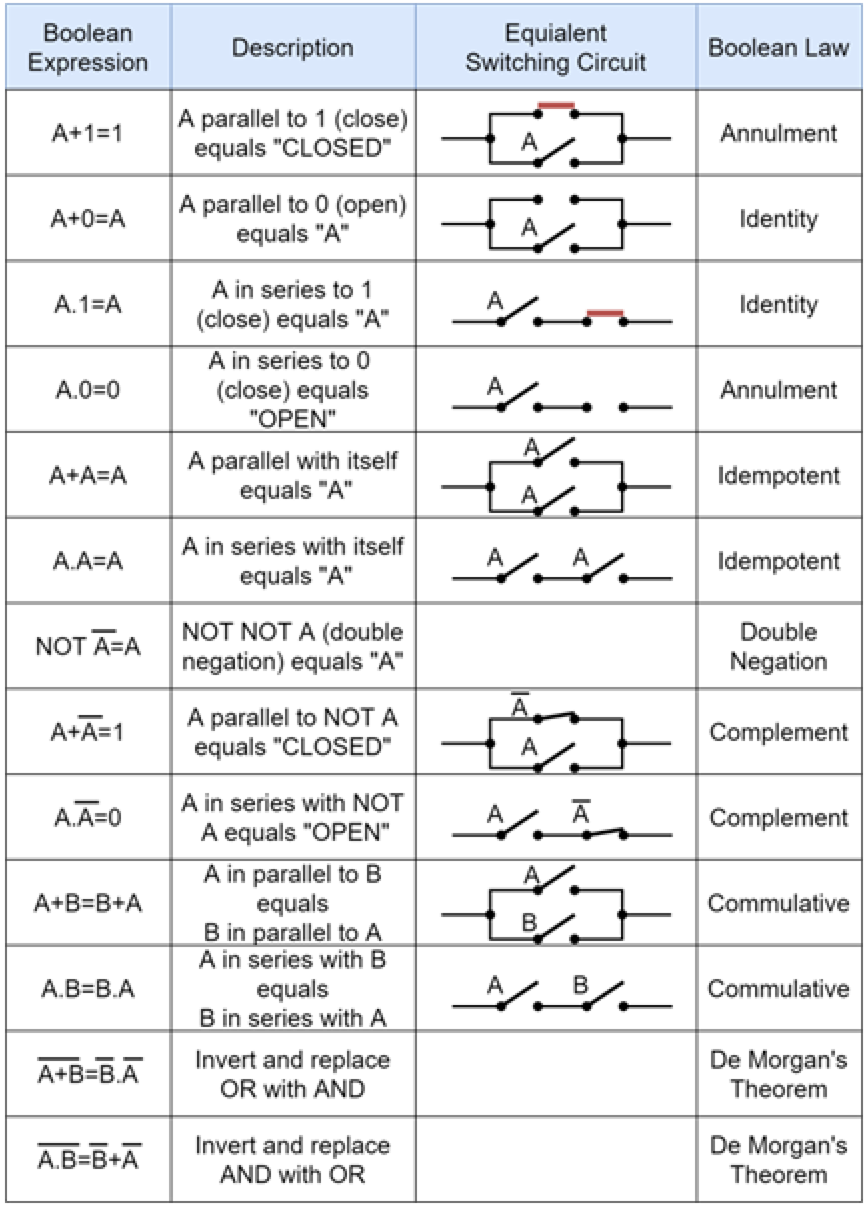
Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit ”

- Example (complex)

$$F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$$





Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
$A+\overline{A}=1$	A parallel to NOT A equals "CLOSED"		Complement
$A.\overline{A}=0$	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}=\overline{B}.\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}=\overline{B}+\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

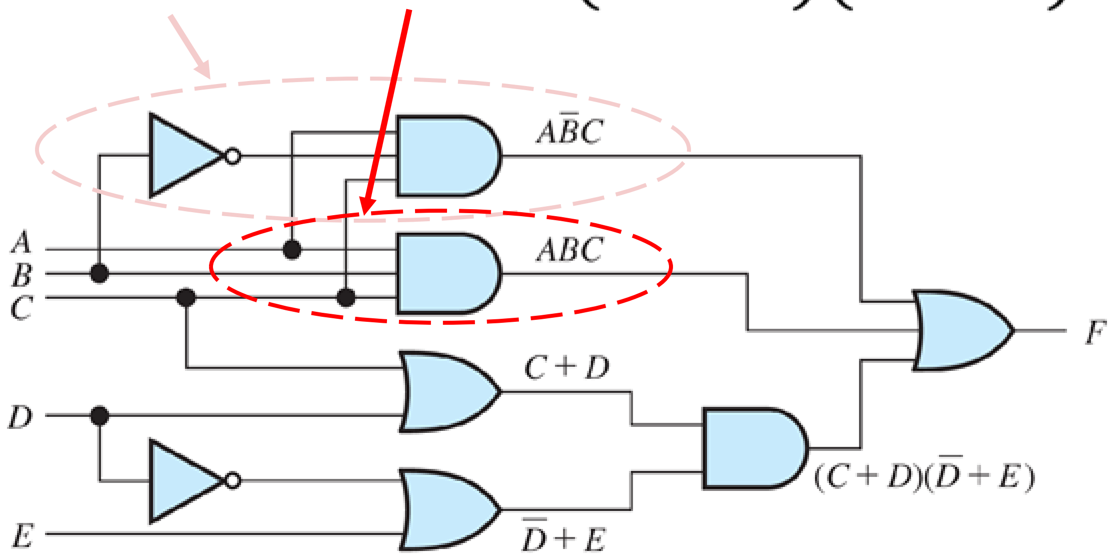
Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

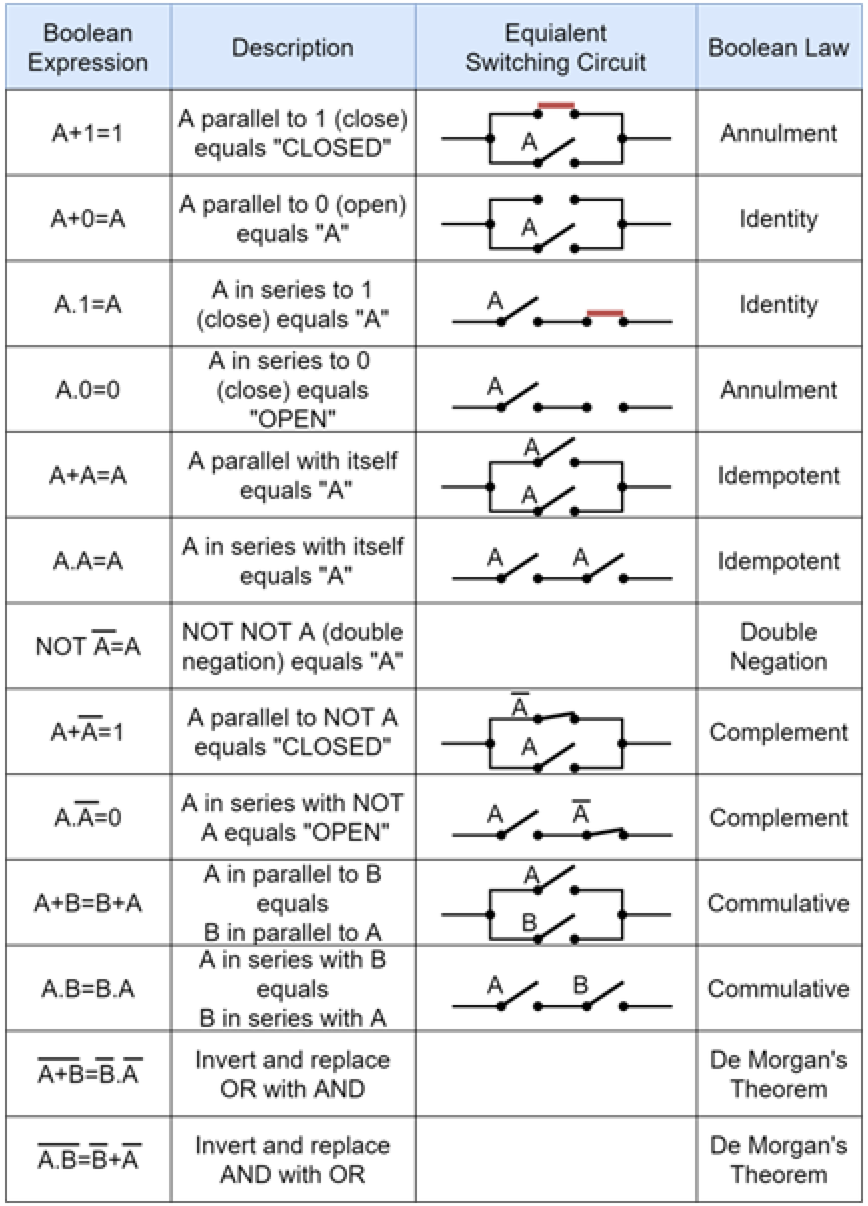
Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit ”

- Example (complex)

$$F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$$





Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}=\overline{B}.\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}=\overline{B}+\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

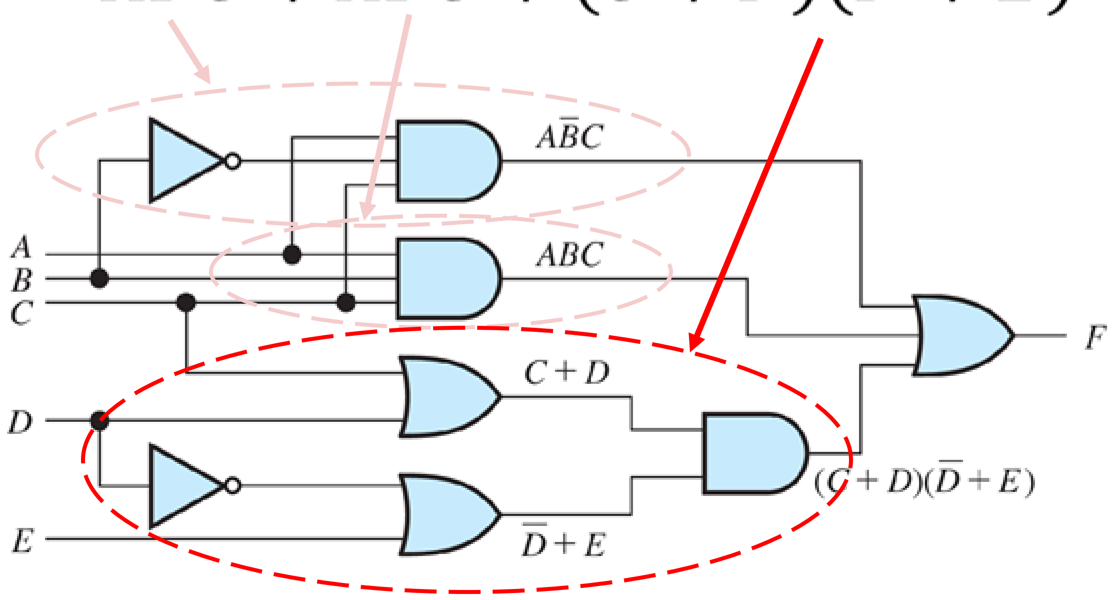
Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

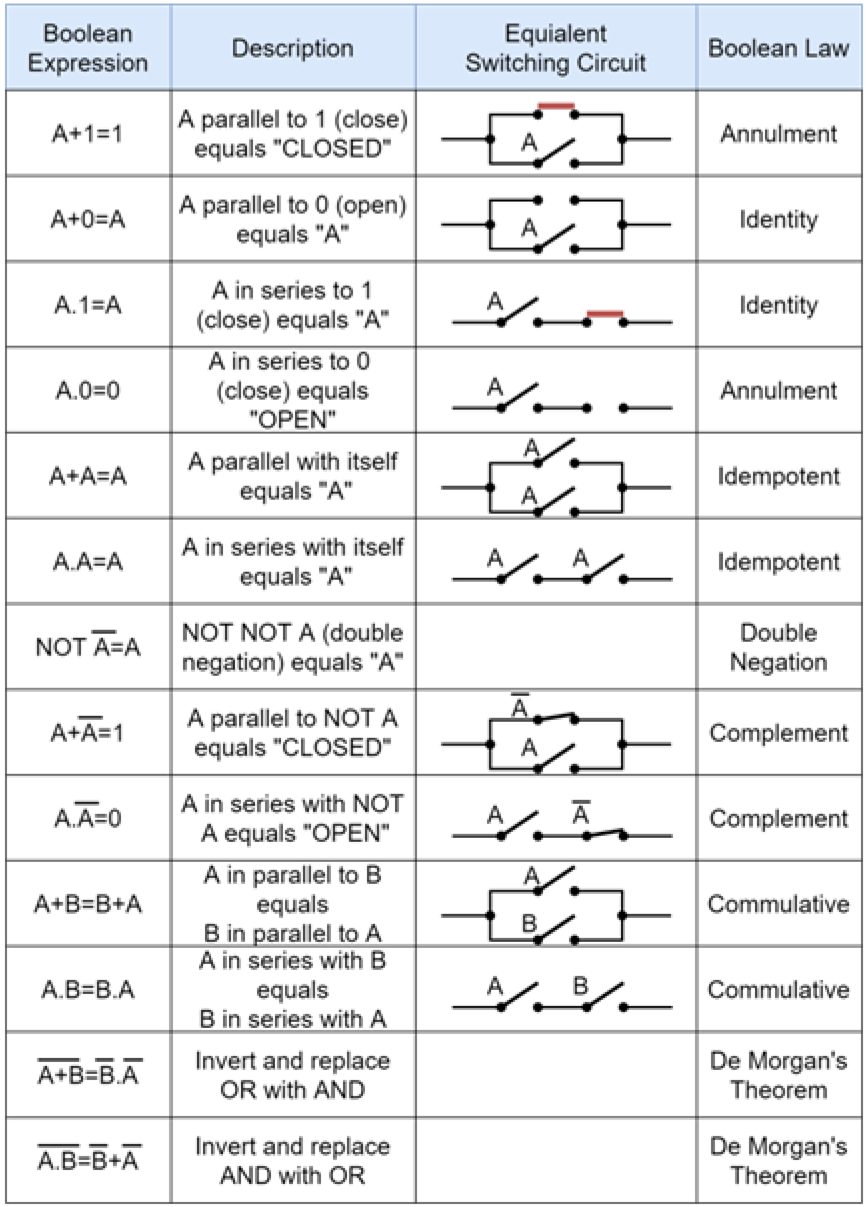
Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit ”

- Example (complex)

$$F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$$





Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"	A	Annulment
A+0=A	A parallel to 0 (open) equals "A"	A	Identity
A.1=A	A in series to 1 (close) equals "A"	A	Identity
A.0=0	A in series to 0 (close) equals "OPEN"	A	Annulment
A+A=A	A parallel with itself equals "A"	A A	Idempotent
A.A=A	A in series with itself equals "A"	A A	Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"	$\overline{A}$ A	Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"	A $\overline{A}$	Complement
A+B=B+A	A in parallel to B equals B in parallel to A	A B	Commulative
A.B=B.A	A in series with B equals B in series with A	A B	Commulative
$\overline{A+B}$=$\overline{B}$.$\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}$=$\overline{B}$+$\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

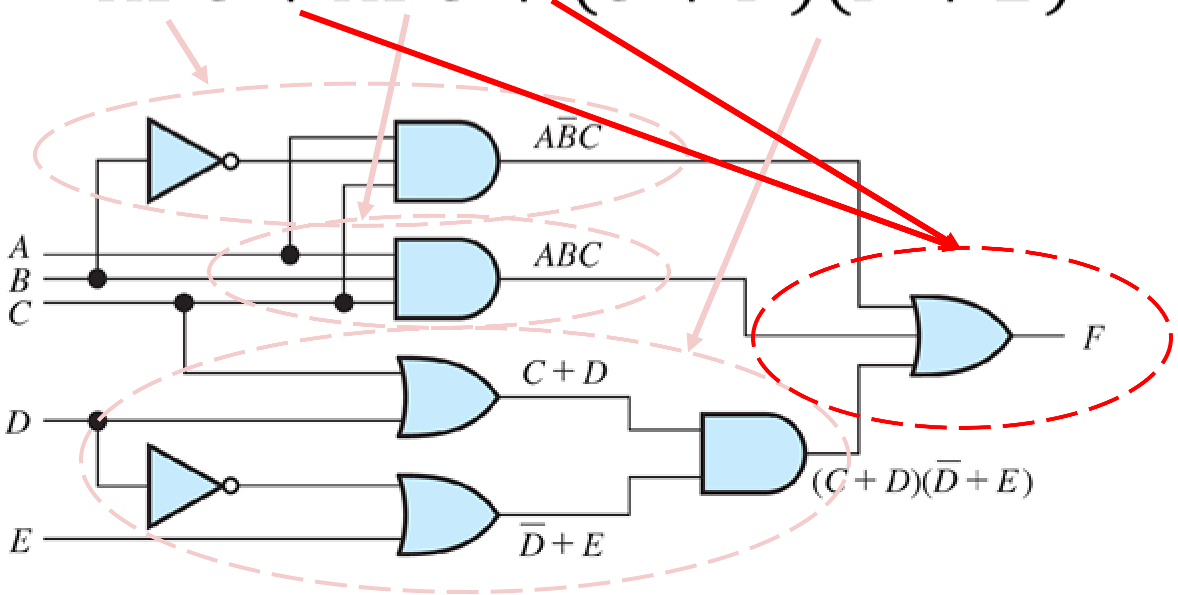
Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

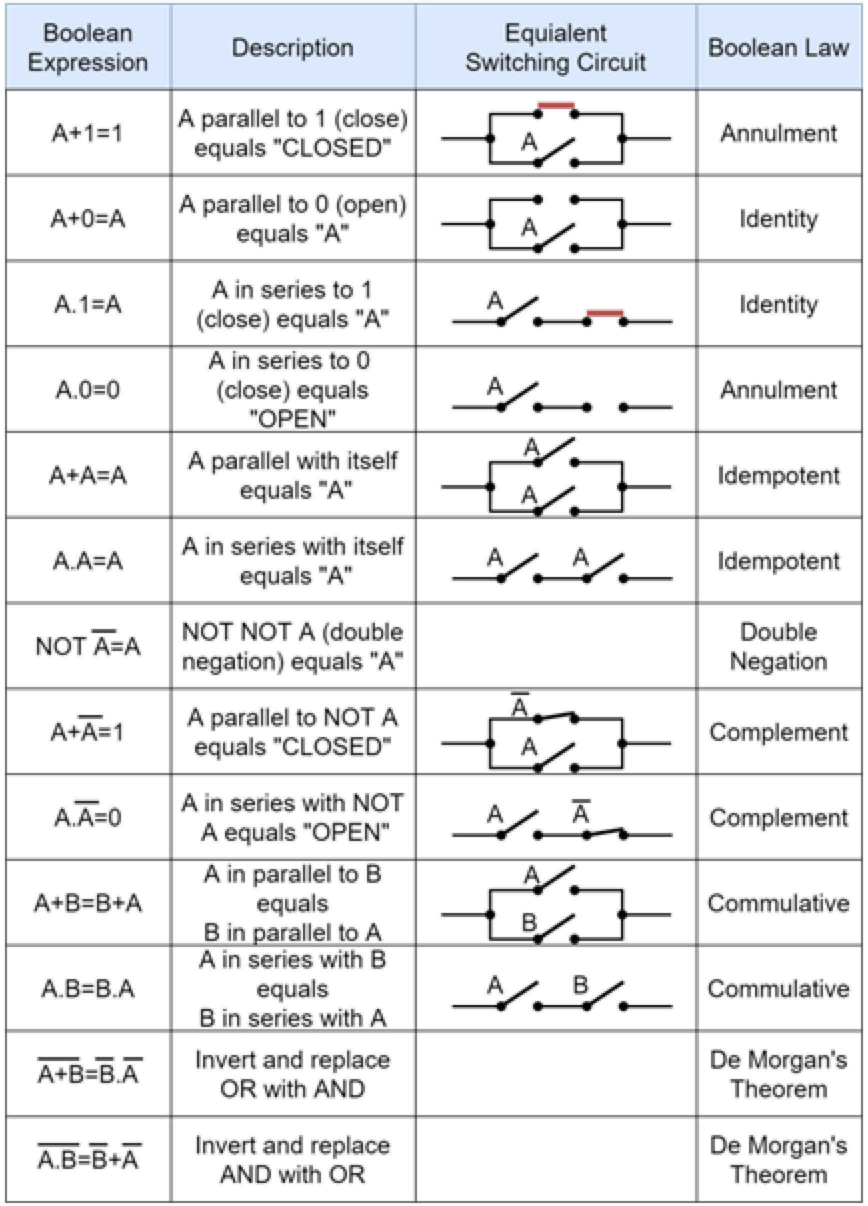
Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit ”

- Example (complex)

$$F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$$





Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}$=$\overline{B}$.$\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}$=$\overline{B}$+$\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

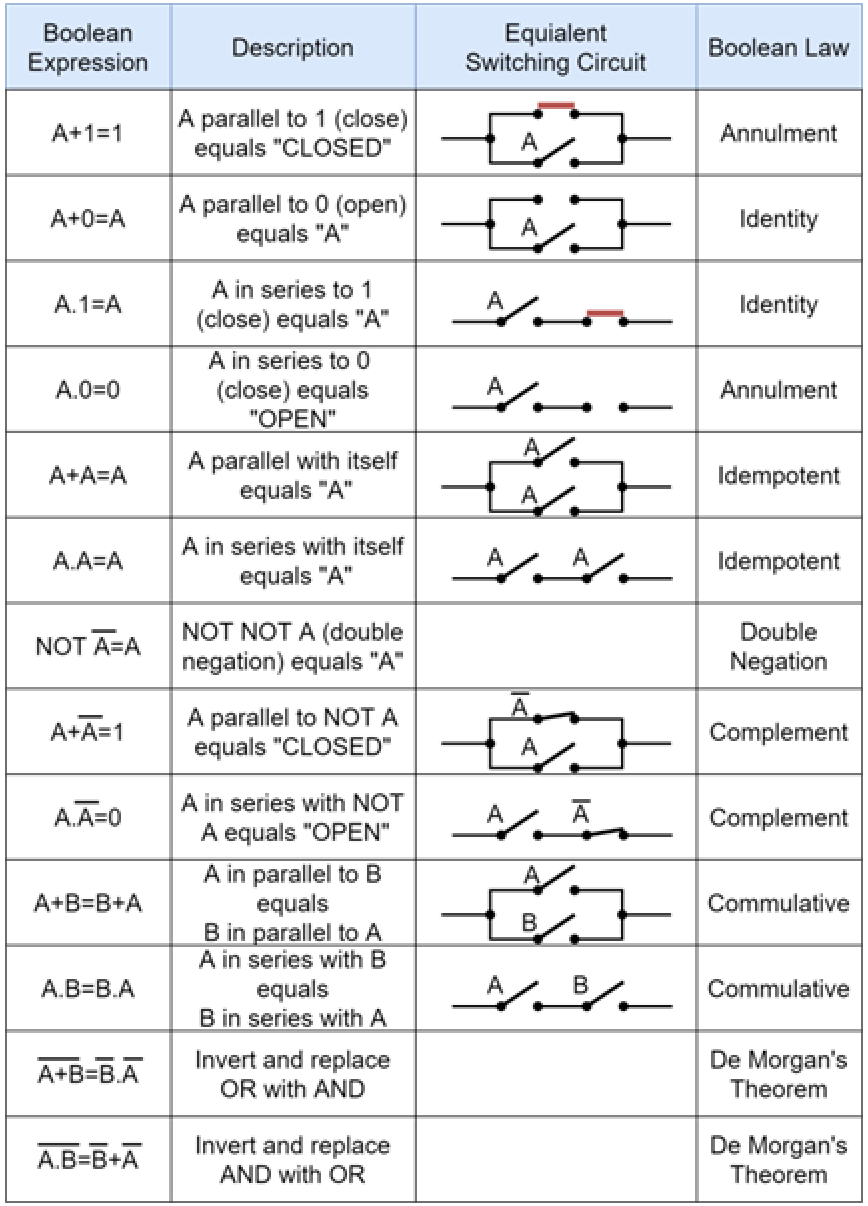
Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

Laws of Boolean Algebra

"Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit "

- Example (complex →simplified)

$$F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$$



Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}$=$\overline{B}$.$\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}$=$\overline{B}$+$\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

Laws of Boolean Algebra

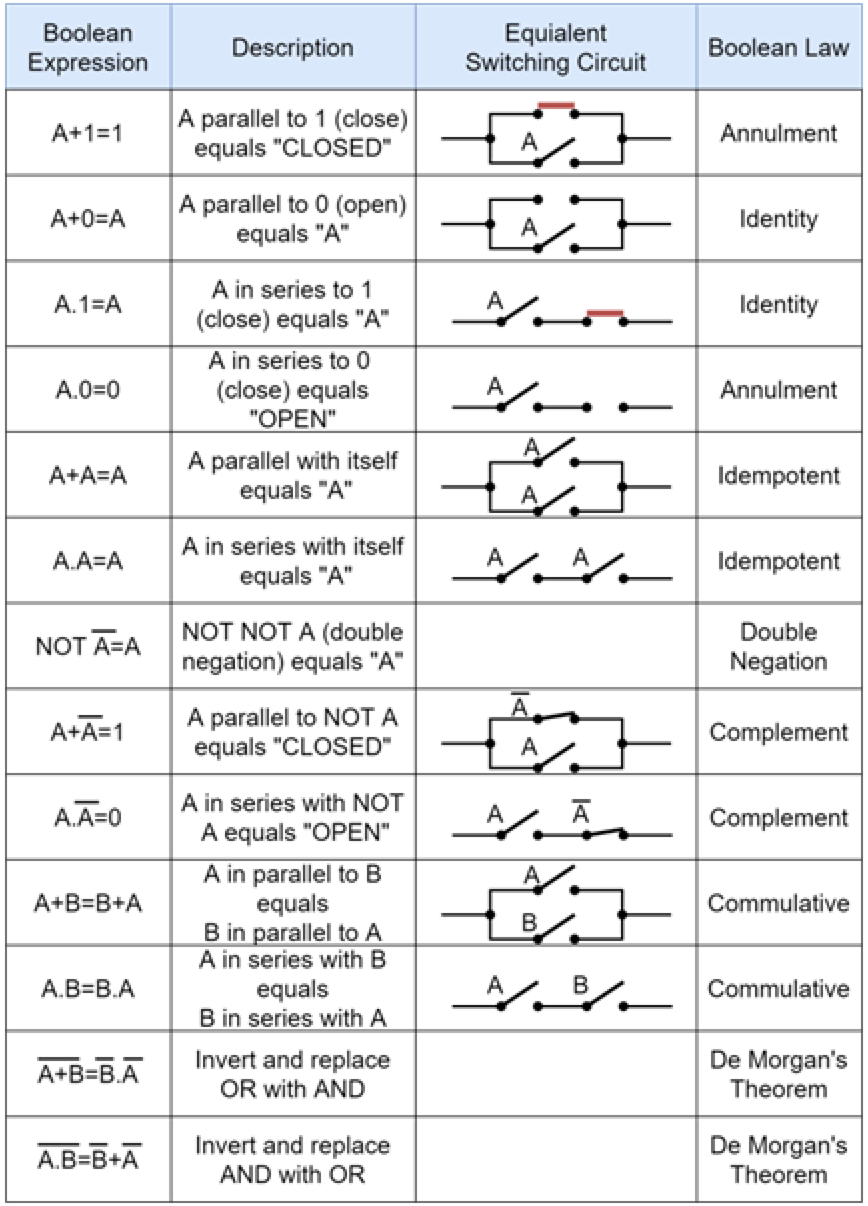
“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit ”

- Example (complex →simplified)

$$F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$$

$$F = AC(\bar{B} + B) + C\bar{D} + CE + D\bar{D} + DE$$

Expand by applying the Distributive Law



Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}$=$\overline{B}$.$\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}$=$\overline{B}$+$\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

Laws of Boolean Algebra

"Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit "

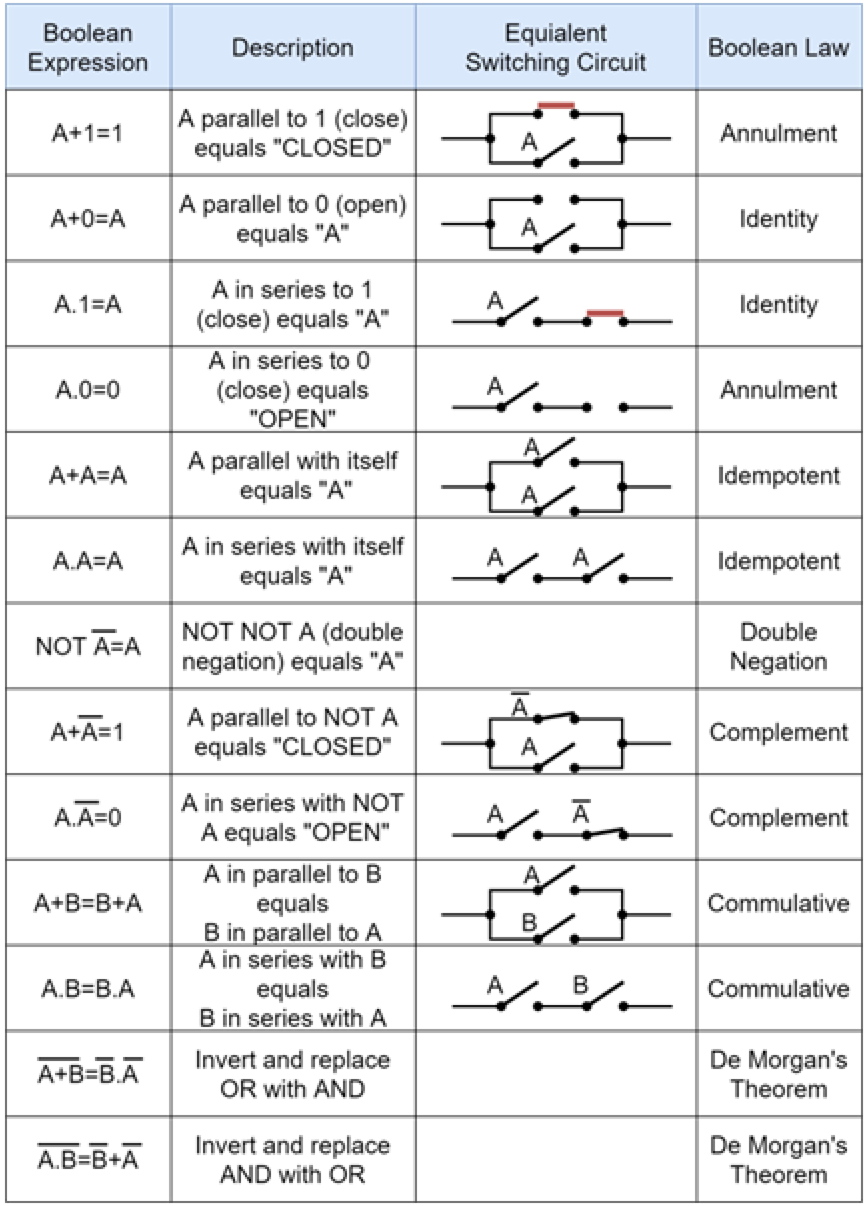
- Example (complex →simplified)

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

$$F = AC(\overline{B} + B) + C\overline{D} + CE + \boxed{1} + DE$$

$$F = AC + C\overline{D} + CE + DE$$

Applying the Complement Law



Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"	A	Annulment
A+0=A	A parallel to 0 (open) equals "A"	A	Identity
A.1=A	A in series to 1 (close) equals "A"	A	Identity
A.0=0	A in series to 0 (close) equals "OPEN"	A	Annulment
A+A=A	A parallel with itself equals "A"	A A	Idempotent
A.A=A	A in series with itself equals "A"	A A	Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"	$\overline{A}$ A	Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"	A $\overline{A}$	Complement
A+B=B+A	A in parallel to B equals B in parallel to A	A B	Commulative
A.B=B.A	A in series with B equals B in series with A	A B	Commulative
$\overline{A+B}$=$\overline{B}$.$\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}$=$\overline{B}$+$\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

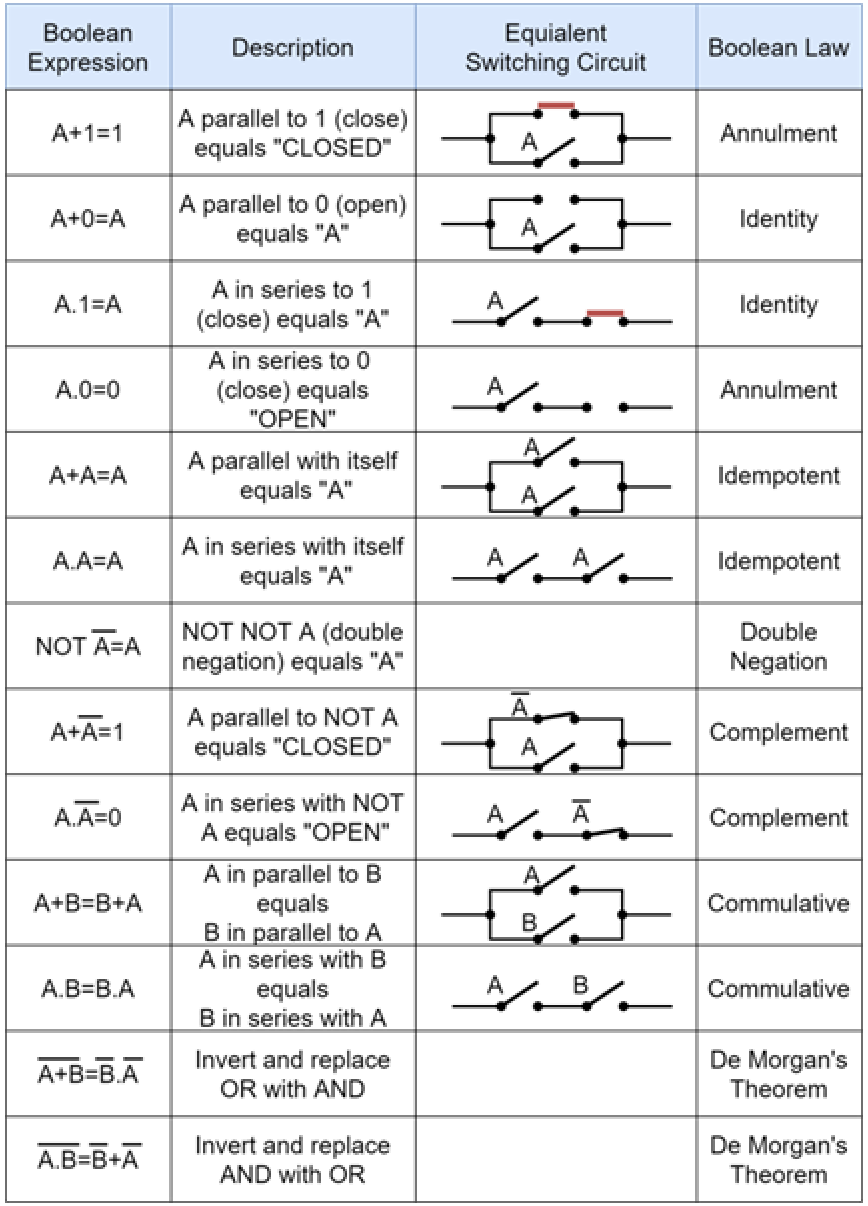
Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit ”

- Example (complex →simplified)

$$F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$$
$$F = AC(\bar{B} + B) + C\bar{D} + CE + D\bar{D} + DE$$
$$F = AC + C\bar{D} + CE + DE$$
$$F = C(A + \bar{D} + E) + DE$$

Factorise by applying the Distributive Law



Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}=\overline{B}.\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}=\overline{B}+\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

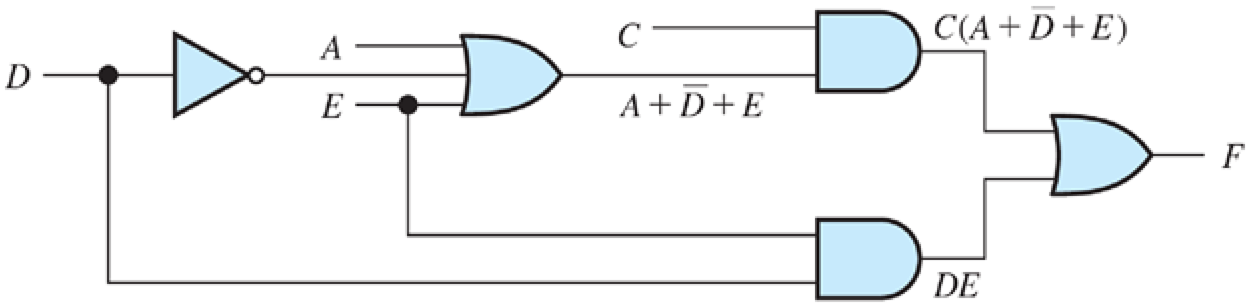
Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

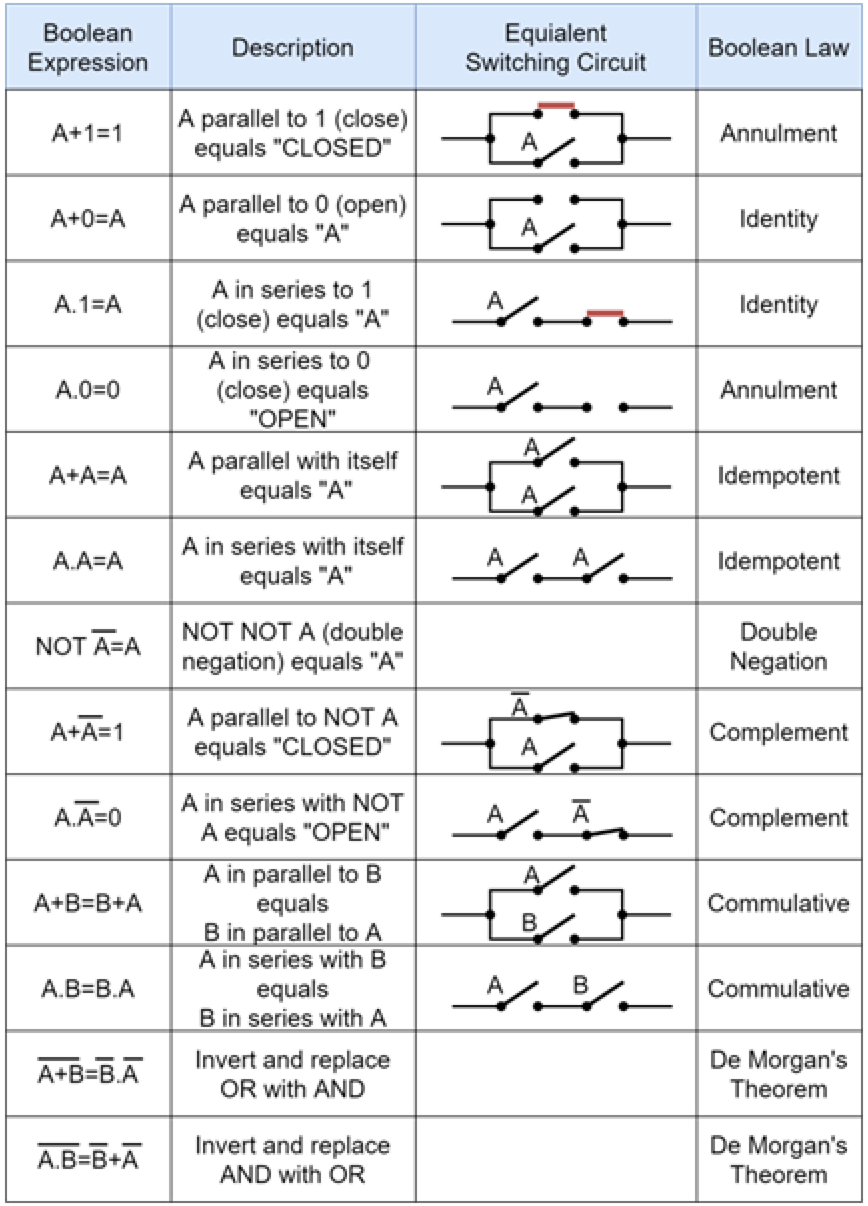
Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit ”

- Example (complex →simplified)

$$F = A\overline{B}C + ABC + (C + D)(\overline{D} + E)$$
$$F = AC(\overline{B} + B) + C\overline{D} + CE + D\overline{D} + DE$$
$$F = AC + C\overline{D} + CE + DE$$
$$F = C(A + \overline{D} + E) + DE$$





Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}$=$\overline{B}$.$\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}$=$\overline{B}$+$\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

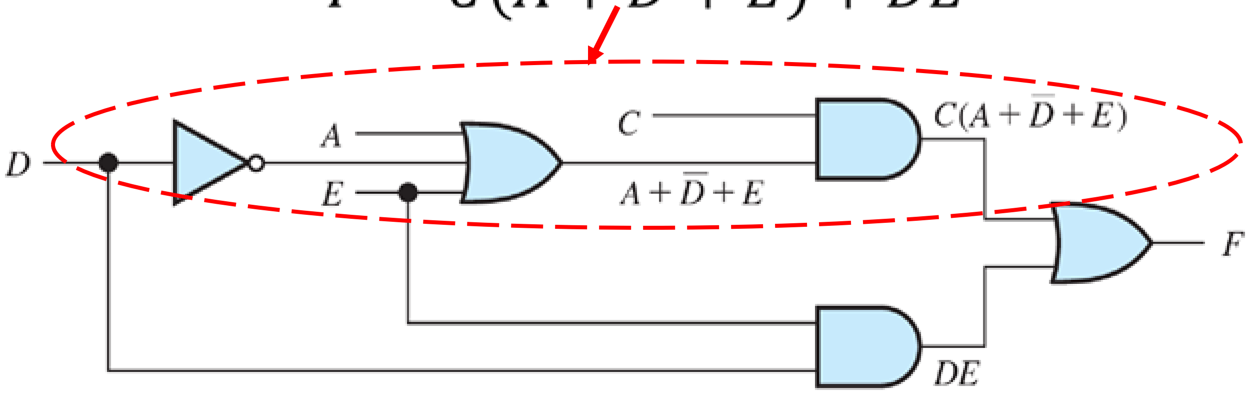
Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

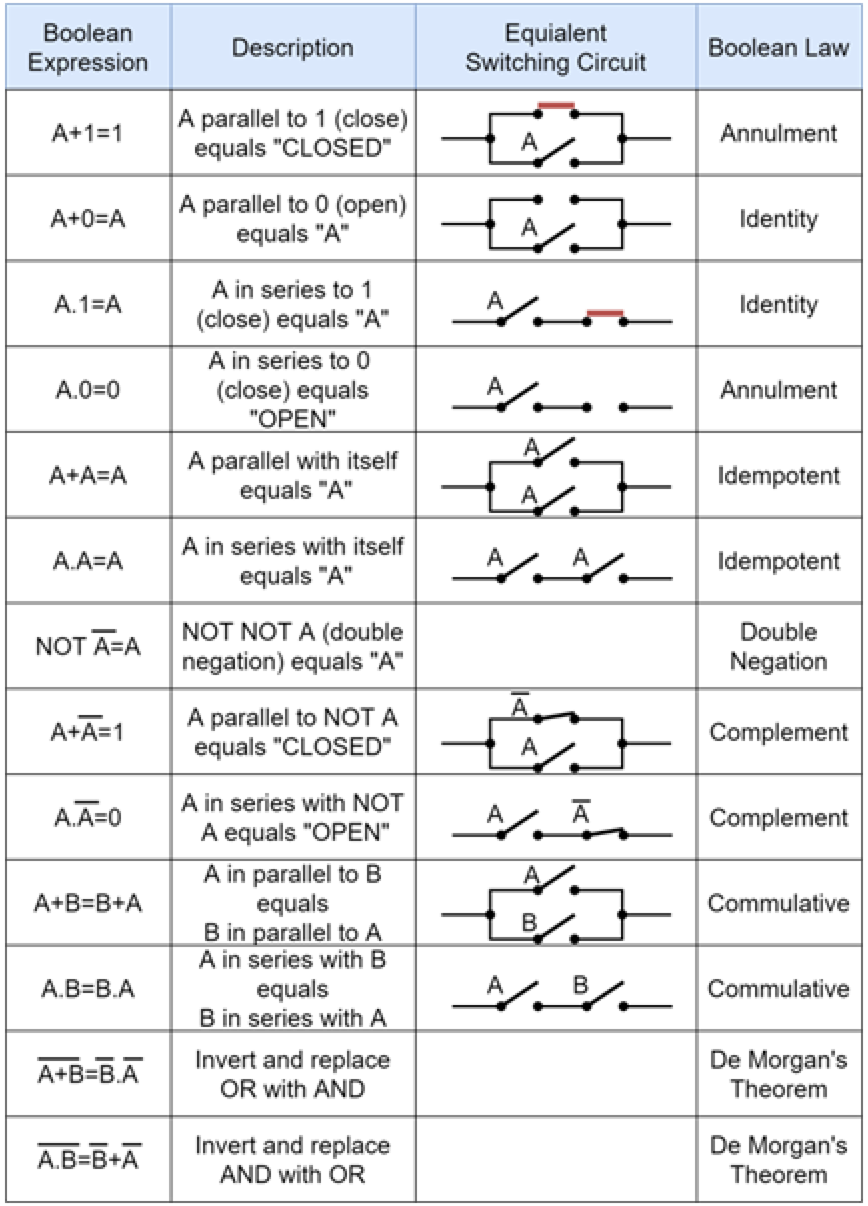
Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit ”

- Example (complex →simplified)

$$F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$$
$$F = AC(\bar{B} + B) + C\bar{D} + CE + D\bar{D} + DE$$
$$F = AC + C\bar{D} + CE + DE$$
$$F = C(A + \bar{D} + E) + DE$$





Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}$=$\overline{B}$.$\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}$=$\overline{B}$+$\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

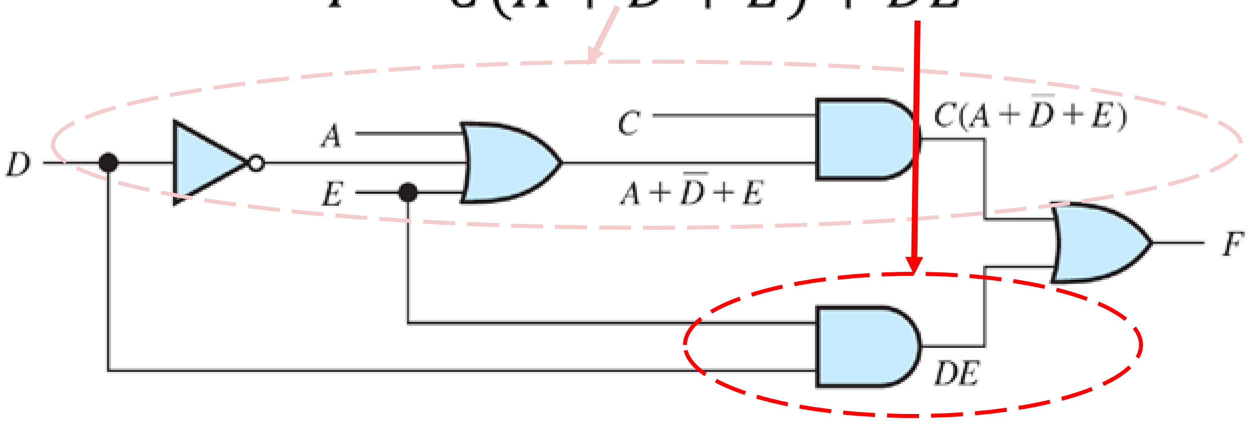
Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

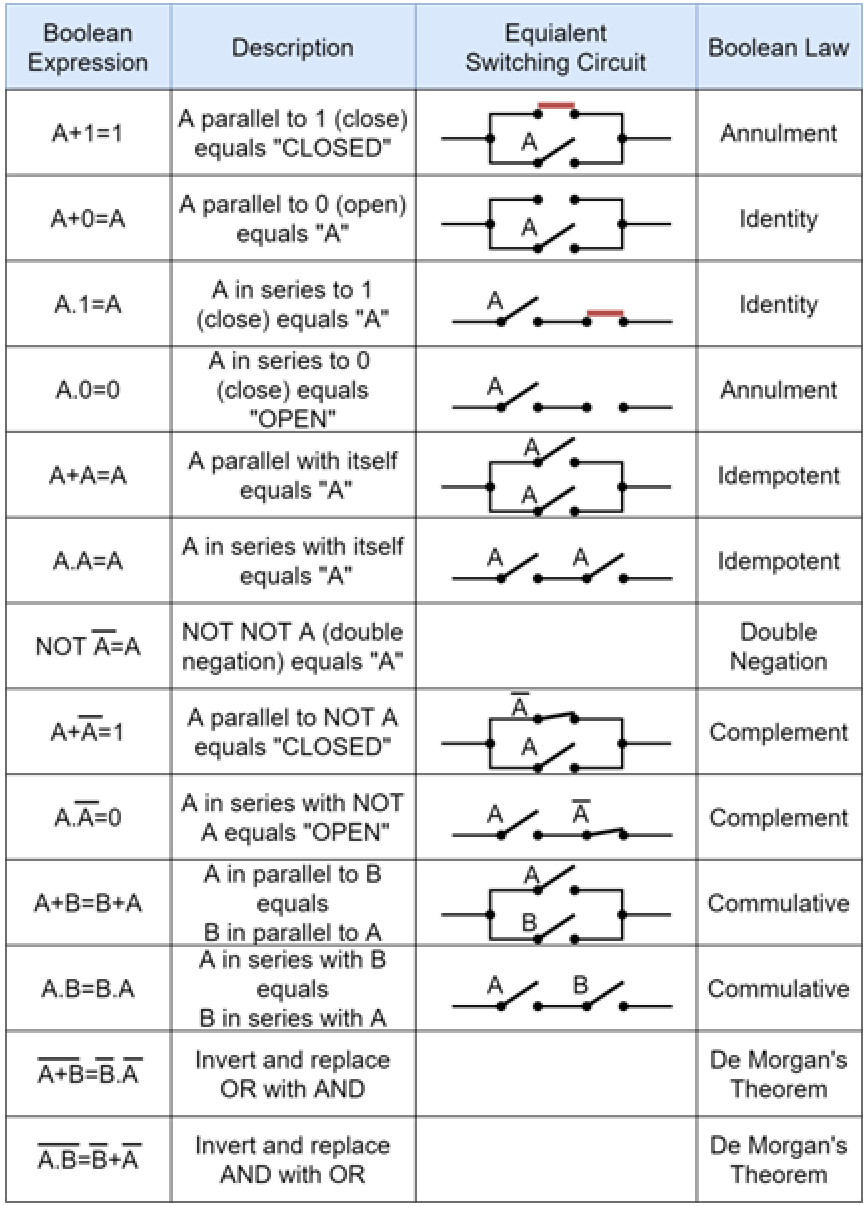
Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit ”

- Example (complex →simplified)

$$F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$$
$$F = AC(\bar{B} + B) + C\bar{D} + CE + D\bar{D} + DE$$
$$F = AC + C\bar{D} + CE + DE$$
$$F = C(A + \bar{D} + E) + DE$$





Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}$=$\overline{B}$.$\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}$=$\overline{B}$+$\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

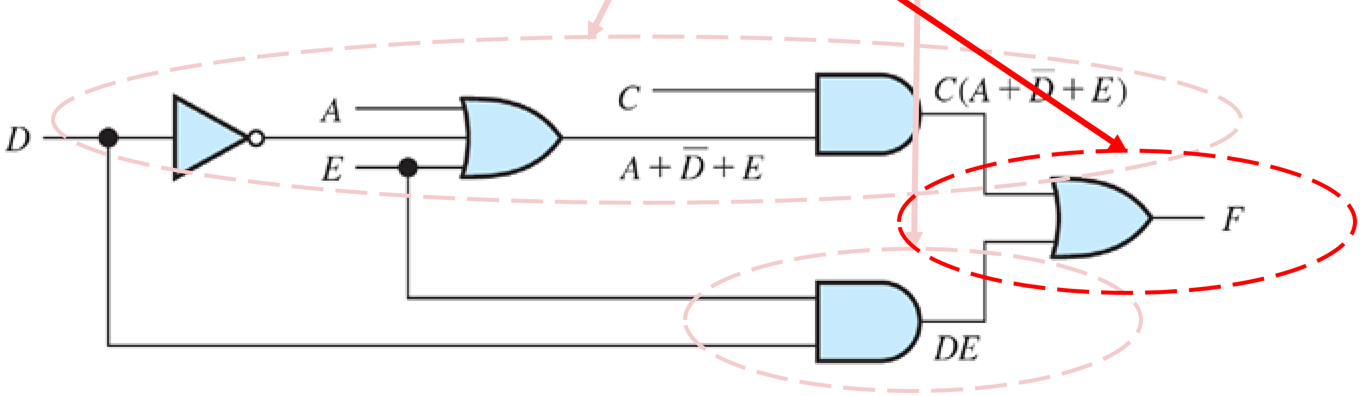
Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

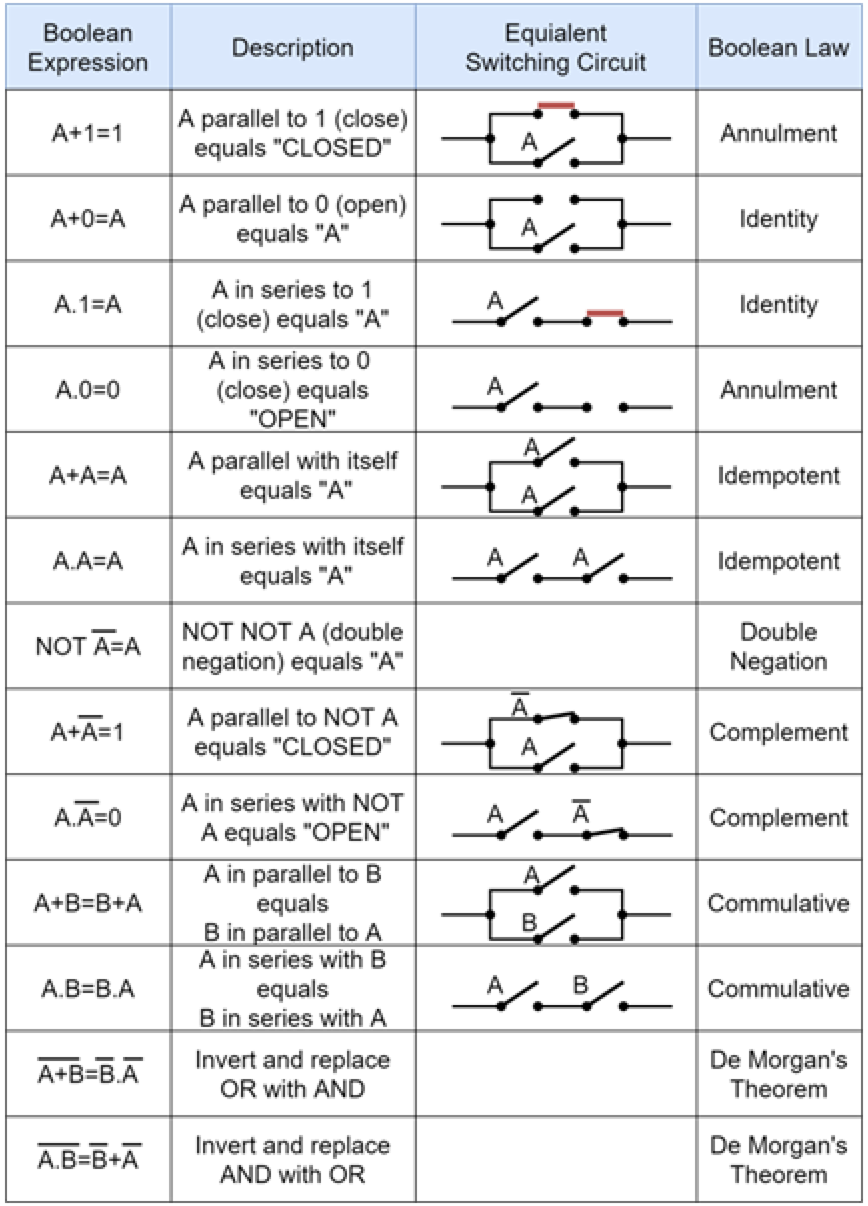
Laws of Boolean Algebra

“Used to simplify Boolean expressions and reduce the number of logic gates needed in a circuit ”

- Example (complex →simplified)

$$F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$$
$$F = AC(\bar{B} + B) + C\bar{D} + CE + D\bar{D} + DE$$
$$F = AC + C\bar{D} + CE + DE$$
$$F = C(A + \bar{D} + E) + DE$$





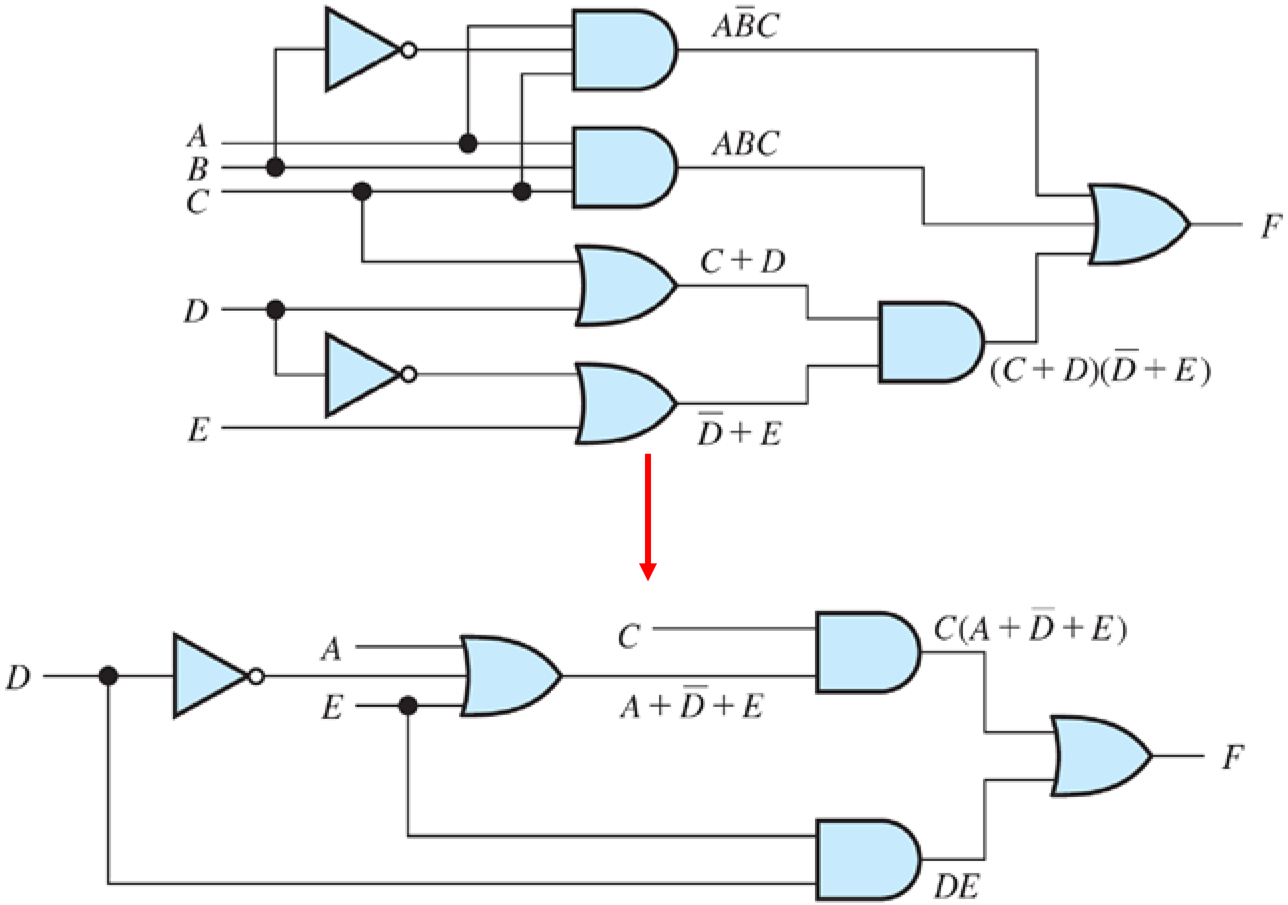
Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}$=$\overline{B}$.$\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}$=$\overline{B}$+$\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

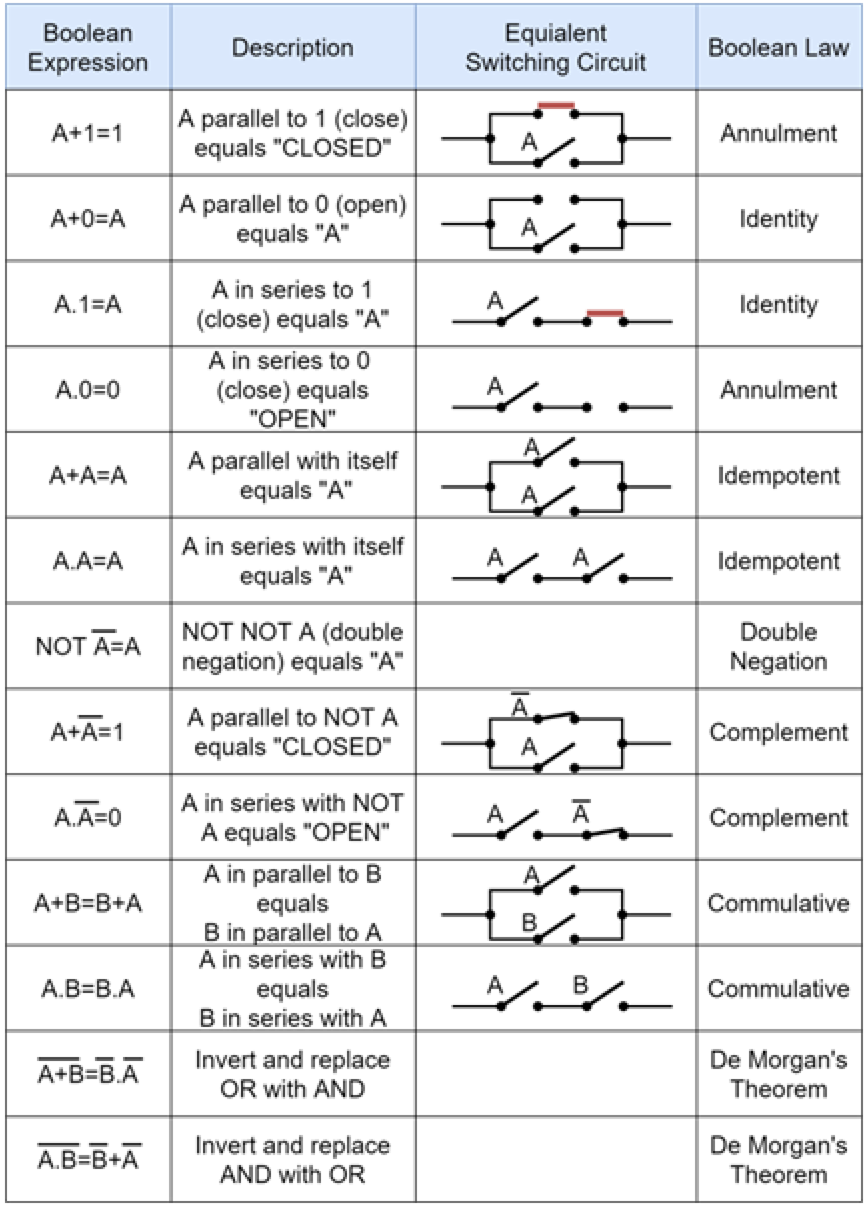
Laws of Boolean Algebra

- Example (complex →simplified)

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



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



Boolean Expression	Description	Equialent Switching Circuit	Boolean Law
A+1=1	A parallel to 1 (close) equals "CLOSED"		Annulment
A+0=A	A parallel to 0 (open) equals "A"		Identity
A.1=A	A in series to 1 (close) equals "A"		Identity
A.0=0	A in series to 0 (close) equals "OPEN"		Annulment
A+A=A	A parallel with itself equals "A"		Idempotent
A.A=A	A in series with itself equals "A"		Idempotent
NOT $\overline{A}$=A	NOT NOT A (double negation) equals "A"		Double Negation
A+$\overline{A}$=1	A parallel to NOT A equals "CLOSED"		Complement
A.$\overline{A}$=0	A in series with NOT A equals "OPEN"		Complement
A+B=B+A	A in parallel to B equals B in parallel to A		Commulative
A.B=B.A	A in series with B equals B in series with A		Commulative
$\overline{A+B}=\overline{B}.\overline{A}$	Invert and replace OR with AND		De Morgan's Theorem
$\overline{A.B}=\overline{B}+\overline{A}$	Invert and replace AND with OR		De Morgan's Theorem

Illustration: https://www.electronics-lab.com/article/laws-of-boolean-algebra/

Portfolio / Lab work