Faculty of Engineering

School of Electrical Engineering and Telecommunications

ELEC 1111 – Topic 10

Digital Logic Circuits

Dr. Inmaculada (Inma) Tomeo-Reyes

Lecturer

School of Electrical Engineering and Telecommunications, UNSW

Topic 10 Content

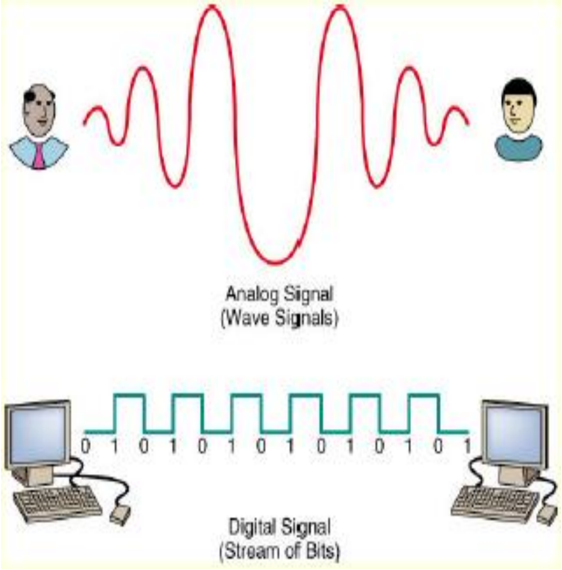
This lecture covers:

- Binary numbers
  - Conversion from decimal to binary and vice versa
  - Simple binary arithmetic (addition and subtraction)
- Digital representations
- Digital logic
  - Truth table
- Digital gates
  - AND, OR, NOT
  - NAND, NOR,
  - XOR, XNOR
- Boolean algebra
  - DeMorgan's Theorem

This topic is not covered in the prescribed textbook for this course

Analog versus digital

- **Analog signal** is an electric signal whose value varies **continuously** with time.
  - A sinusoid voltage $v(t) = V_m \cos(\omega t)$ can take **any value** between $-V_m$ and V_m.
- **Digital signals** can take only a **finite** number of values. They are also called **discrete-time signals** as they vary between a finite set of **digits.**
  - Systems that **work** with **digital signals** are called **digital systems.**
  - To understand how digital systems work, we need to get familiar with **binary numbers** first.
  - The most common digital signals are **binary signals.**
  - A binary signal can take **only two** discrete values.
  - These two values are represented in **binary** format using **0** and **1** digits.



Decimal vs binary numbers

- **Decimal** numbers system is **base 10** or **radix 10.**
  - In **decimal** system, there are **10 digits**, 0, 1, 2, …, 9.
  - A decimal number is represented by the **sum** of the **powers of 10.**

$$372.5 = (3 \times 10^2) + (7 \times 10^1) + (2 \times 10^0) + (5 \times 10^{-1})$$

- **Binary** number system is **base 2.**
  - For any system that operates in **two states** (like on or off), **binary number system** is a natural choice.
  - In **binary** system, there are **2 digits**, 0 and 1.
  - A binary number is represented by the **sum** of the **powers of 2.**

$$10110 = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$$

- We can use subscript to denote numbers in different **bases.**

$$(10110)_2 = 16 + 0 + 4 + 2 + 0 = (22)_{10}$$

To convert a **binary** number to **decimal**, just **expand** the binary number as the **sum** of the **powers of 2**.

Binary numbers

- Binary digits are also called **bits.**
- The **rightmost bit** is called **least significant bit (LSB).**
- The **leftmost bit** is called **most significate bit (MSB).**

$$(11100110)_2$$

MSB LSB

- Binary numbers require **large** number of bits to represent large numbers.
  - They are usually grouped in sets of 4, 8 or 16.
    - **Nibble**: group of **4/four** bits.
    - **Byte**: group of **8/eight** bits.
    - **Word**: group of **16/sixteen** bits.
  - $(11100110)_2$ is an **8-bit** number.

Conversion from decimal to binary

- Conversion from a decimal number to its binary equivalent is performed by **successive division** of the decimal number by **2/two** followed by the division of its quotients.
- The **remainders** constitute the binary number.
- This method is used to convert the **integer** part of decimal numbers **only**.

	Quotient	Remainder
$156 \div 2$	78	0
$78 \div 2$	39	0
$39 \div 2$	19	1
$19 \div 2$	9	1
$9 \div 2$	4	1
$4 \div 2$	2	0
$2 \div 2$	1	0
$1 \div 2$	0	1

LSB

MSB

$$(156)_{10} = (10011100)_2$$

Addition

- To add binary numbers, the simple rule shown in the table is used:

Addition Rule
$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$ (with a carry of 1)

Example:

$15 + 20 = 35$

$$\begin{array}{r} 1111 \\ \underline{+10100} \\ 100011 \end{array}$$

	Q	R
$15 \div 2$	7	1
$7 \div 2$	3	1
$3 \div 2$	1	1
$1 \div 2$	0	1

$(15)_{10} = (1111)_2$

	Q	R
$20 \div 2$	10	0
$10 \div 2$	5	0
$5 \div 2$	2	1
$2 \div 2$	1	0
$1 \div 2$	0	1

$(20)_{10} = (10100\,)_2$

$$100011 = (1 \times 2^5) + 0 + 0 + 0 + (1 \times 2^1) + (1 \times 2^0) = 32 + 2 + 1 = 35$$

Subtraction

- For **subtraction**, $A - B$ is replaced with $A + (-B)$ so that only addition of a positive number to a negative one is applied.
- **Three conventions** are used to represent a **negative number** in binary system:
  - **Sign-magnitude:** Sign bit 1 means minus sign and Sign bit 0 means plus sign.
  - **1's complement:** Replacing 0's with 1's and vice versa (inverting) in a binary number.
  - **2's complement:** Adding 1 to the 1's complement in a binary number.

Sign-magnitude convention

Sign bit b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
$0\ (+)$ $1\ (-)$	Actual binary number						

1's complement convention

Sign bit b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
$0\ (+)$	Actual binary number						
$1\ (-)$	**1's complement** of binary number						

2's complement convention

Sign bit b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
$0\ (+)$	Actual binary number						
$1\ (-)$	**2's complement** of binary number						

Subtraction

- **2's complement** is the most commonly used method in many digital computers for **subtraction.**
- In performing $A + (-B)$, $-B$ is the **2's complement** of B with **sign bit 1**, and we simply **add** the two numbers with their **sign bits** to obtain the **subtraction.**
- We need to know beforehand that how many bits are required to represent the largest number in the calculations.

Decimal $(+)$	4-bit 2's complement
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Sign bit

Decimal $(-)$	4-bit 2's complement
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001

Note: To represent -8 using 2's complement, at least 5 bits are needed to accommodate sign bit.

UNSW SYDNEY

Exercise

Perform the following subtraction in binary format:

$$(4)_{10} - (6)_{10} = (4)_{10} + (-6)_{10} = (-2)_{10}$$

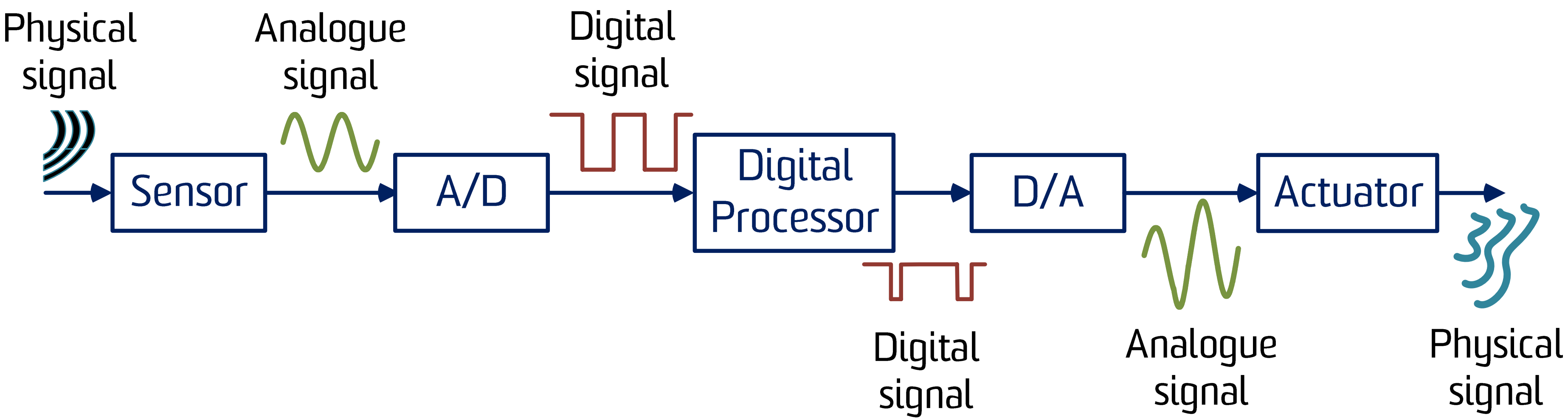
Exercise

Perform the following subtraction in binary format:

$$(7)_{10} - (5)_{10} = (7)_{10} + (-5)_{10} = (2)_{10}$$

Digital systems

A typical digital system would consist of the following parts:



Note:
A/D or ADC: Analog to Digital Converter.
D/A or DAC: Digital to Analog Converter.

Why use digital?

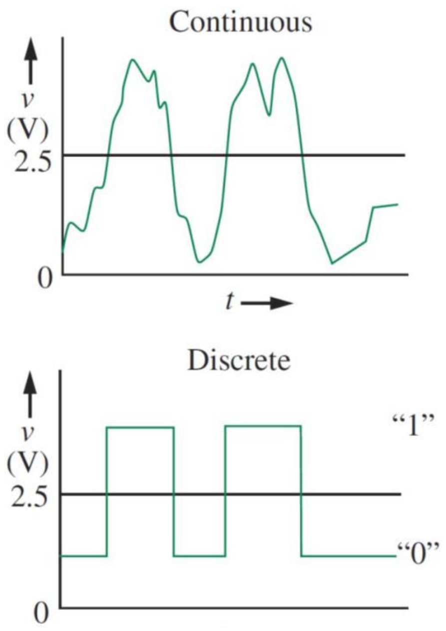
- Any physical quantity that is capable of switching between two values can be used to represent numerical quantities in the binary system.
- Numerical calculations can be used for manipulation of binary numbers.
- The benefits of using digital systems are numerous.
  - Benefits include: good noise rejection, high reliability, high accuracy, predictability, low power, ease of design.
  - There are some limitations too, such as noise generation and need for A/D & D/A interface.

Two-level logic for digital systems

- Binary digits can be represented with the use of a **voltage signal:**
  - **Presence of voltage** (“**high**”) denotes the **binary digit 1.**
  - **Absence of voltage** (“**low**”) denotes the **binary digit 0.**
- More specifically:
  - **5 V** represents **digit 1.**
  - **0 V** represents **digit 0.**

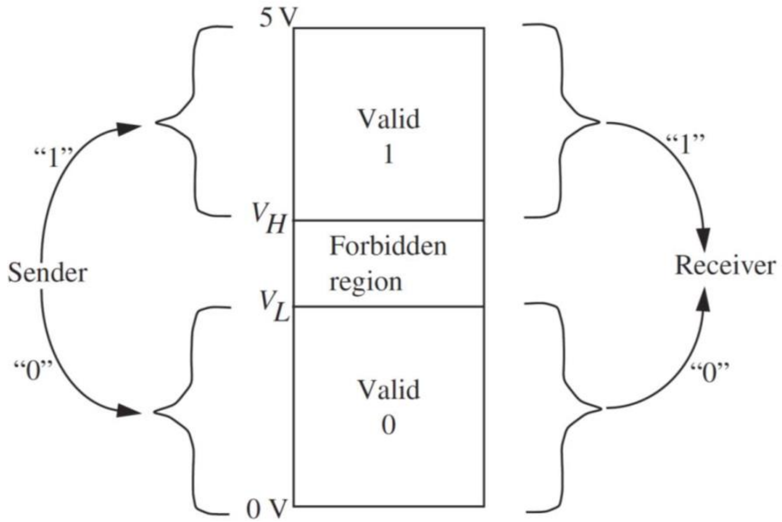
Two-level logic for digital systems

- In an **“ideal”** case:
  - The ‘sender’ of the digit outputs **exactly** either 5 V or 0 V as for 1 or 0.
  - The ‘receiver’ sees **exactly** the same voltages and translate 5 V to 1 and 0 V to 0.
- However, the world is not “ideal” or “perfect”:
  - Circuits **cannot** generate voltages with **high accuracy** (exactly 5 V).
  - There are **losses** in the circuits (voltage drops).
  - There is **noise**!
- One way to address this issue is the use of a **band of voltages** to define binary bits.
  - For instance, any voltage **between** 2.5 V and 5 V is interpreted as 1, and below this range as 0, as shown in the figure.
- However, the **noise** can affect the voltages around the **boundary** and it causes uncertainty in the accuracy of the received voltages.



Two-level logic for digital systems

- To resolve the issue with the noise, a **clear zone** is introduced to **separate** the voltage range used for digit 1 and digit 0.
- This zone is called **forbidden region.**
- We have to introduce a tolerance on **how much noise** is acceptable to be present in a system.
- Based on the **technology** used in building a digital system (e.g. TTL or CMOS), different **standards** can be defined for the range of voltages to be used for digits 1 and 0.

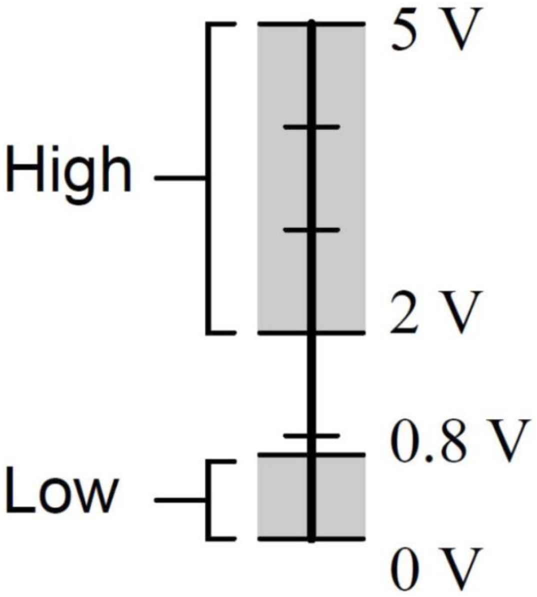


In the **forbidden region**, the voltages received are considered as **undefined** or **noise-affected.**

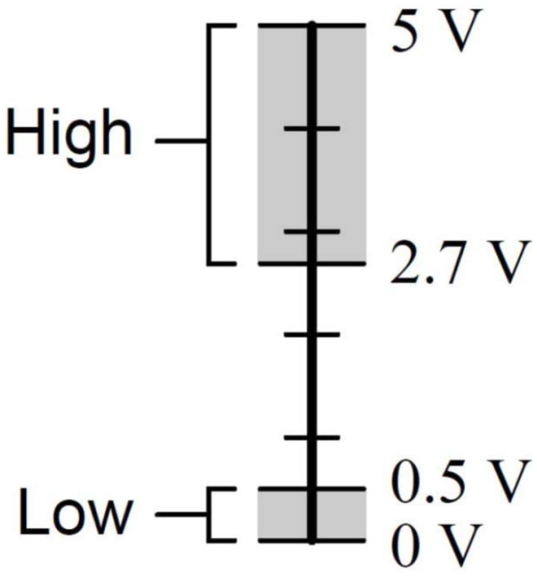
UNSW SYDNEY

Two-level logic for digital systems

- TTL (Transistor - Transistor Logic) uses the following voltages levels from 0 V to 5 V:



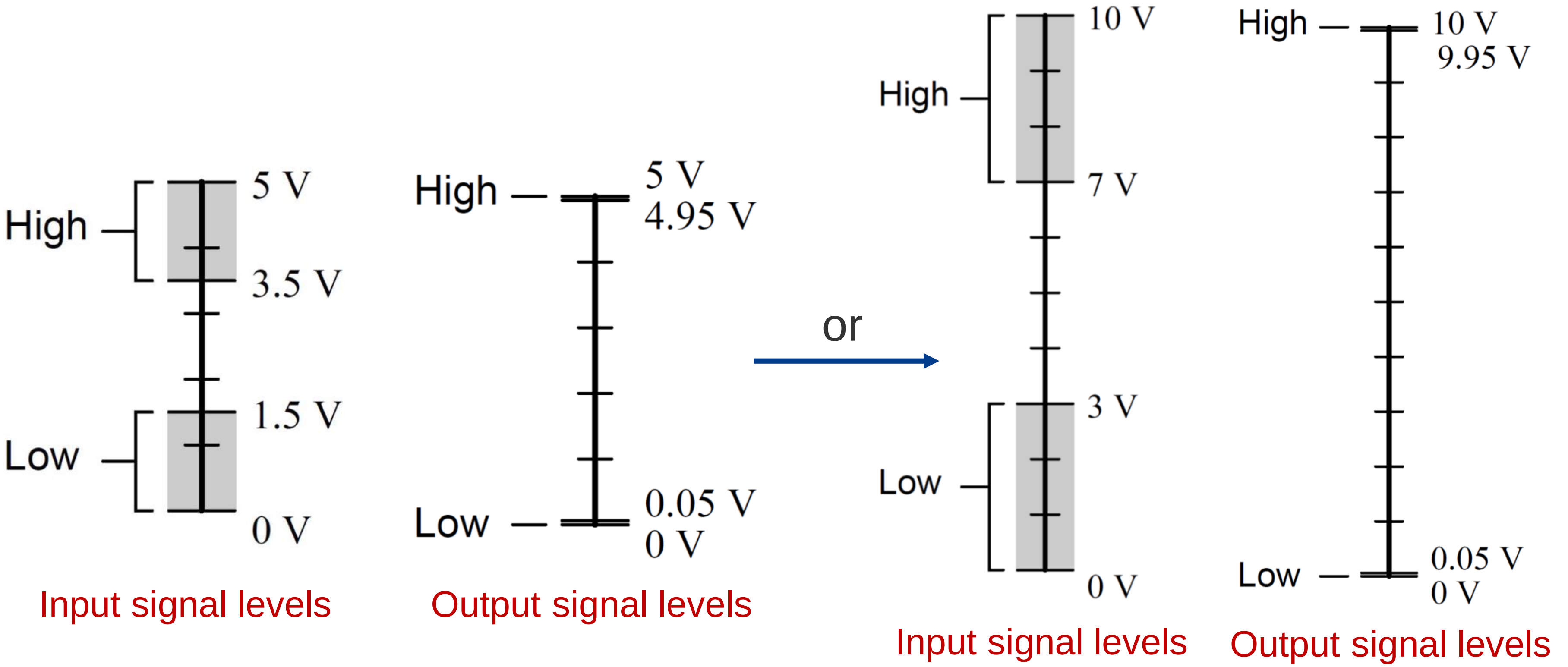
Input signal levels



Output signal levels

Two-level logic for digital systems

- CMOS (Complementary Metal Oxide Semiconductor) can uses higher voltages levels.



Logic circuits

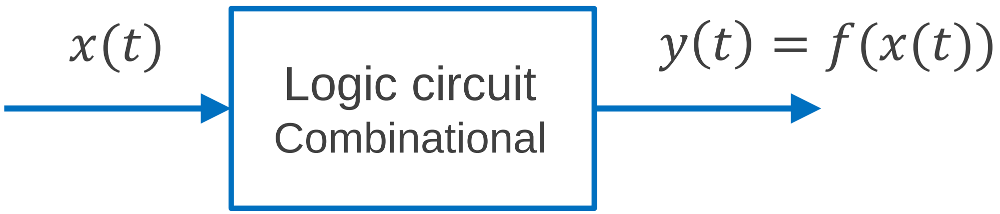
- Digital logic can be implemented in digital circuits. That's why they are also called **Digital Logic Circuits.**
- There are two types of logic circuits:
  1. **Combinational** logic.
  2. **Sequential** logic.

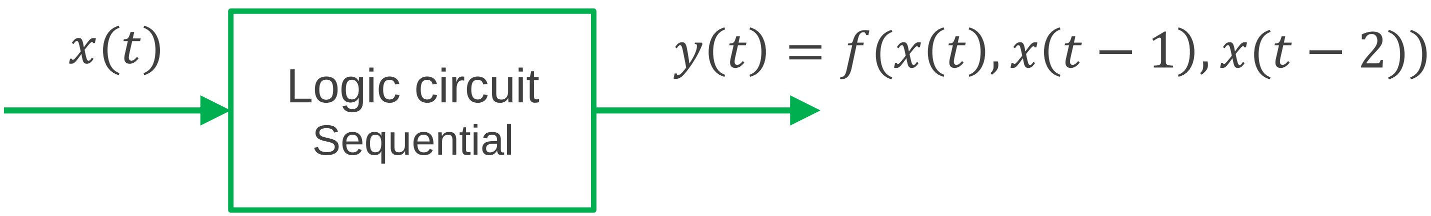
Combinational:

- **Output** depends **only** on the **current value** of the input.
- **No memory** is needed.

Sequential:

- Output depends on **both current** and **previous values** of the input.
- It **needs memory** to keep the previous values.





Truth table and logic variables

- **Truth table** is a method of tabulating and describing the output of a logic operation for all possible combinations of inputs.
- For example, the following logic statements can be tabulated as shown in the truth table:

Phone rings ($R = 1$) if the power is on ($P = 1$) and there is an incoming call ($C = 1$).

- In this example, R, P, and C are called **logic variables**.
- P and C are inputs and R is the output.
- They can be either **0** or **1**.
- There are $4 = 2^2$ combinations.

Truth Table

P	C	R
0	0	0
0	1	0
1	0	0
1	1	1

- In logic statements, assign a **logic variable** to **each precise statement.**
- If there are $\boldsymbol{n}$ inputs, there will be $\mathbf{2^n}$ combinations.

Exercise

Create the truth table for the following example describing how David would make a purchase.

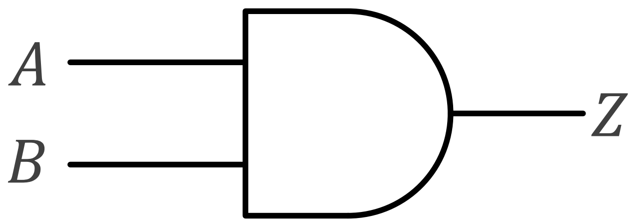
David buys if he wants an item **and** he has cash **or** if he needs the item **and** he has cash **or** EFTPOS card

Primitive logic gates

- A logic gate implements a combinational logic function.
- We can have 2^n possible functions of n variables.
- All these possible functions can be described using primitive gates:
  - AND
  - OR
  - NOT
  - NAND
  - NOR
  - XOR
  - XNOR

Logic AND gate

- **AND** gate is the **binary multiplication.**
- It can have **multiple** inputs.
- For a **2-input AND** gate, the **truth table** is:



AND gate

$$Z = A \text{ and } B$$
$$Z = A.B$$
$$Z = AB$$

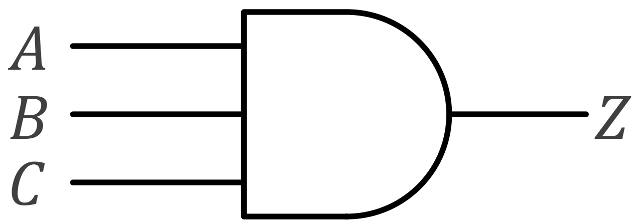
AND Truth Table

A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

$Z = 1$ if both A and B are 1

Logic AND gate

- For a **3-input AND** gate, the **truth table** is:



AND gate

$$Z = A.B.C$$

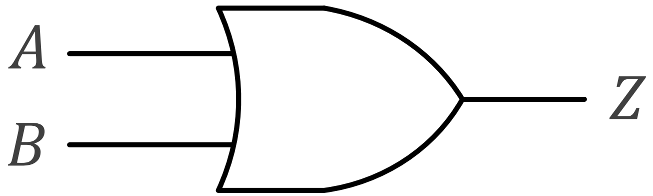
$Z = 1$ if all A and B and C are 1

AND Truth Table

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Logic OR gate

- **OR** gate is the **binary addition.**
- It can have **multiple** inputs.
- For a **2-input OR** gate, the truth table is:



OR gate

$$Z = A \; or \; B$$
$$Z = A + B$$

OR Truth Table

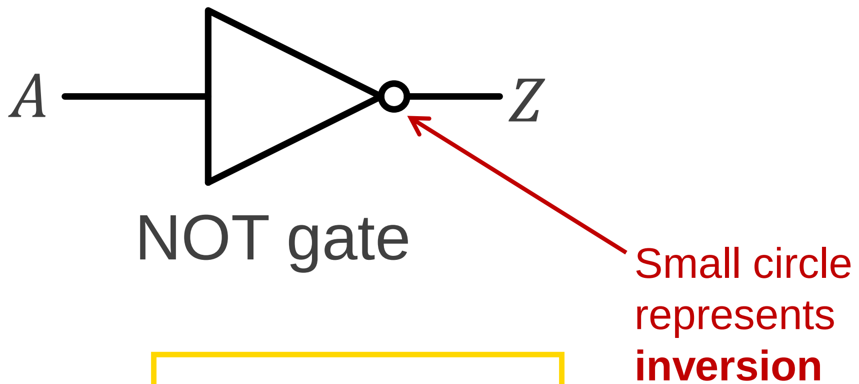
A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

$Z = 1$ if either A or B is 1

UNSW SYDNEY

Logic NOT gate

- **NOT** gate is the **binary negation, complement or inversion.**
- **NOT** gate simply **inverts** 1 to 0 and vice versa (similar to an amplifier with a **negative unit gain).** It is known as an **inverter.**
- The **truth table** of **NOT** is:



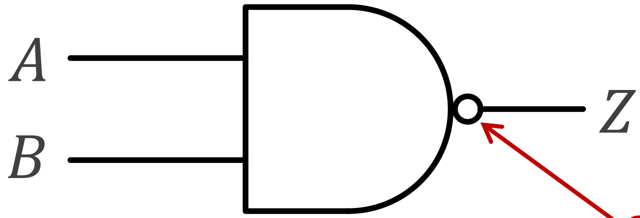
$$Z = not\ A$$
$$Z = \bar{A}$$
$$Z = A'$$

NOT Truth Table

A	Z
0	1
1	0

Logic NAND gate (NOT AND)

- **NAND** is the **inverted** of **AND** gate.
- It can have **multiple** inputs.
- For a **2-input NAND** gate, the **truth table** is:



NAND gate

$$Z = \overline{A.B}$$
$$Z = \overline{AB}$$
$$Z = (AB)'$$

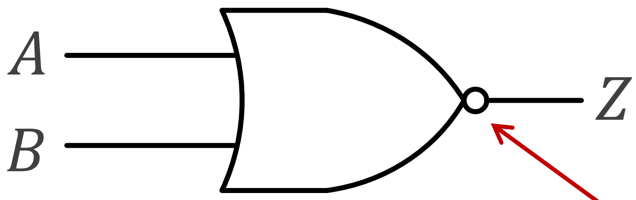
NAND Truth Table

A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

$Z = 1$ if either A or B is 0

Logic NOR gate (NOT OR)

- **NOR** is the **inverted** of **OR** gate.
- It can have **multiple** inputs.
- For a **2-input NOR** gate, the **truth table**:



NOR gate

Small circle represents **inversion**

$$Z = \overline{A + B}$$
$$Z = (A + B)'$$

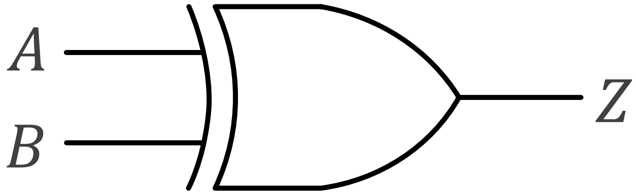
NOR Truth Table

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	0

$Z = 1$ if both A and B are 0

Logic XOR (Exclusive OR)

- **XOR** gate is a circuit whose output is **1 only** if **one** of its inputs is **1** but **not both.**
- It can have **multiple** inputs.
- For a **2-input XOR** gate, the truth table is:



XOR gate

$$Z = A \, xor \, B$$
$$Z = A \oplus B$$

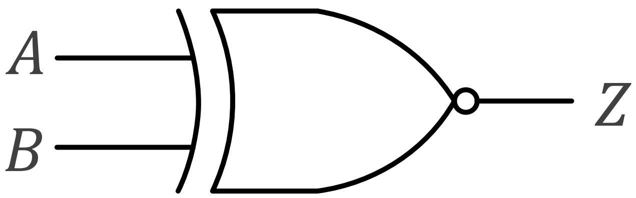
XOR Truth Table

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	0

$Z = 1$ if either A or B is 1 but **not both**

Logic XNOR (Exclusive NOR)

- **XNOR** gate is the **inverted** of **XOR.**
- It can have **multiple** inputs.
- For a **2-input XNOR** gate, the **truth table** is:



XNOR gate

$$Z = A\ xnor\ B$$
$$Z = \overline{A \oplus B}$$
$$Z = (A \oplus B)'$$

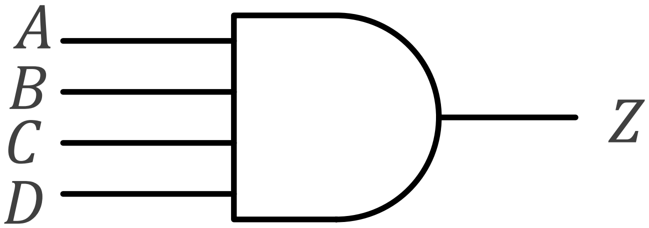
XNOR Truth Table

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	1

$Z = 1$ if A or B are the same

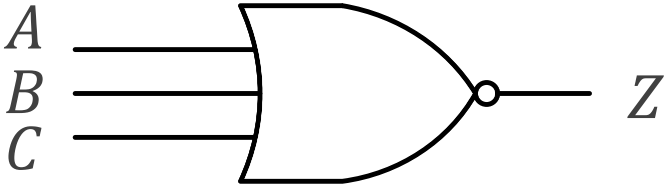
Logic gates with multiple inputs

- As mentioned before, we can have gates with multiple inputs.
- The output still follows the same rule of the 2-input logical gate.



4-input AND gate

$$Z = A.B.C.D$$

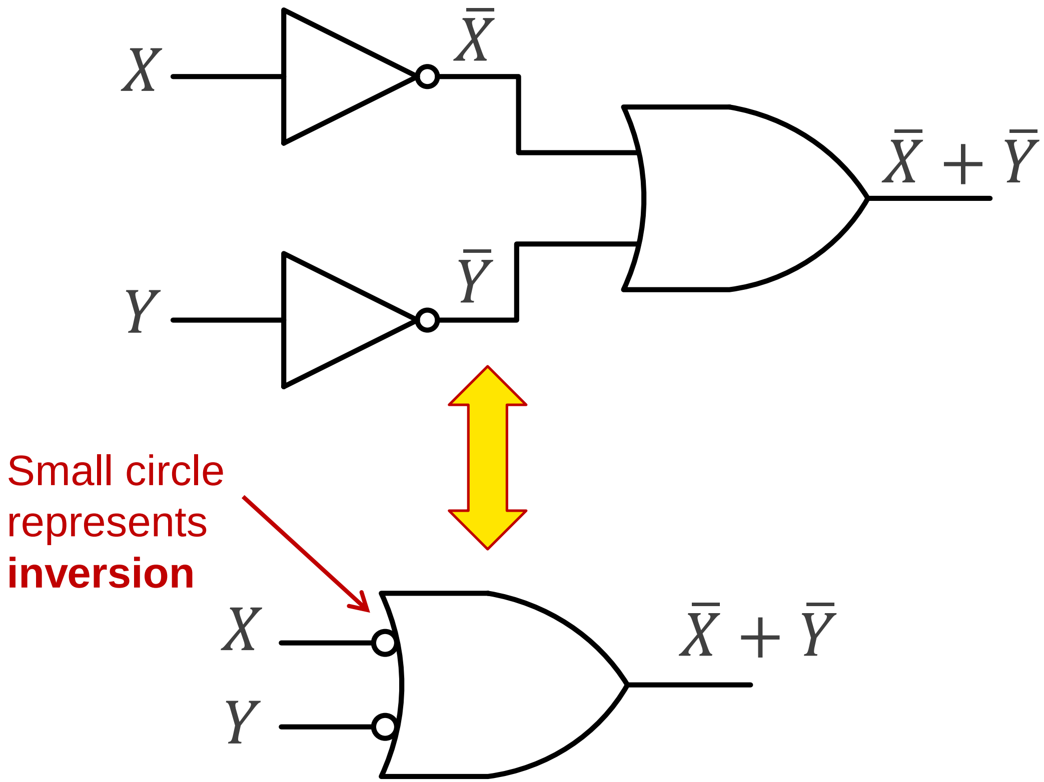


3-input NOR gate

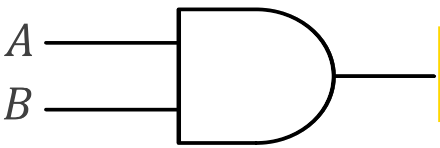
$$Z = \overline{A + B + C}$$

Drawing convention

- For the sake of simplicity, sometimes the NOT gates at the inputs can be combined to with gate in the form of **small circle** representing inverted input.



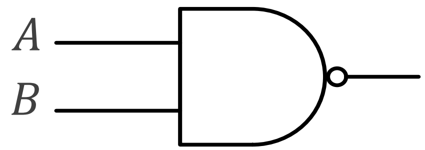
Logical gates summary



AND gate

$\boxed{A.B}$

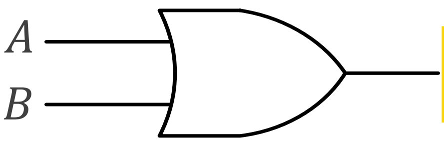
A	B	$A.B$
0	0	0
0	1	0
1	0	0
1	1	1



NAND gate

$\boxed{\overline{A.B}}$

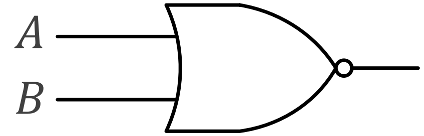
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0



OR gate

$\boxed{A+B}$

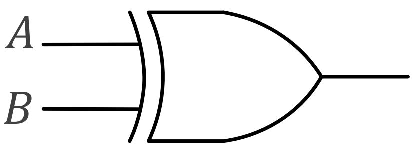
A	B	$A+B$
0	0	0
0	1	1
1	0	1
1	1	1



NOR gate

$\boxed{\overline{A+B}}$

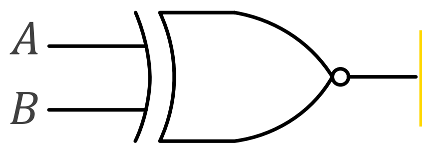
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0



XOR gate

$\boxed{A \oplus B}$

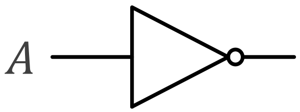
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



XNOR gate

$\boxed{\overline{A \oplus B}}$

A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1



NOT gate

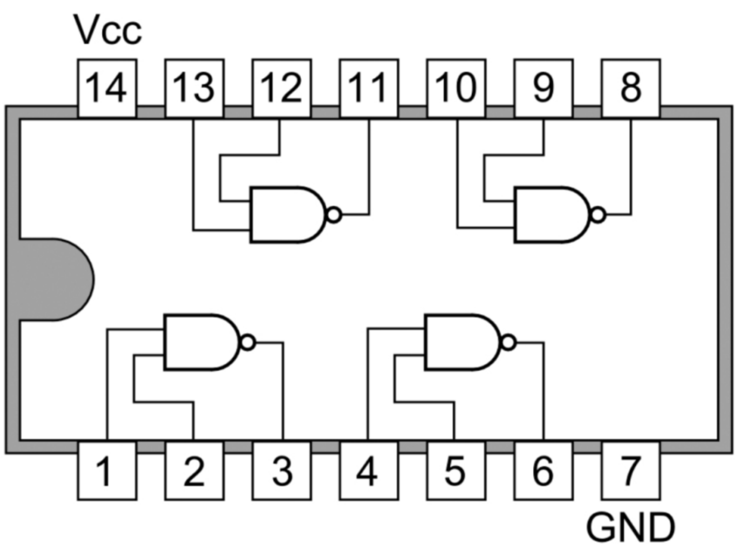
$\boxed{Z = \bar{A}}$

A	$\bar{A}$
0	1
1	0

Physical implementation on ICs

- Since logic gates are built using **transistors**, they can be easily implemented on **integrate circuits** (**ICs**) .
- For instance, **74LS00,** known as **Quad 2-input NAND gate,** contains **4 NAND** gates as seen below:

Connection Diagram

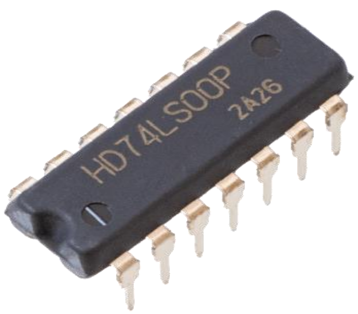


Function Table

$Y = \overline{AB}$

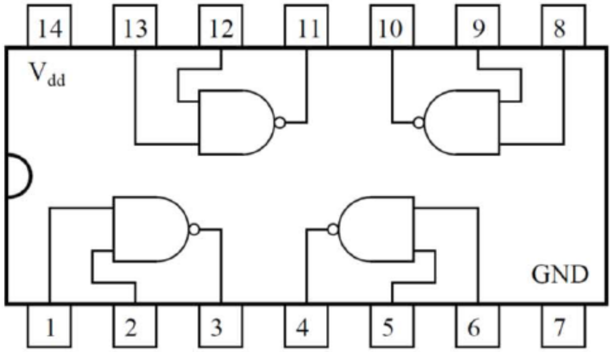
Inputs		Output
A	**B**	**Y**
L	L	H
L	H	H
H	L	H
H	H	L

H = HIGH Logic Level
L = LOW Logic Level

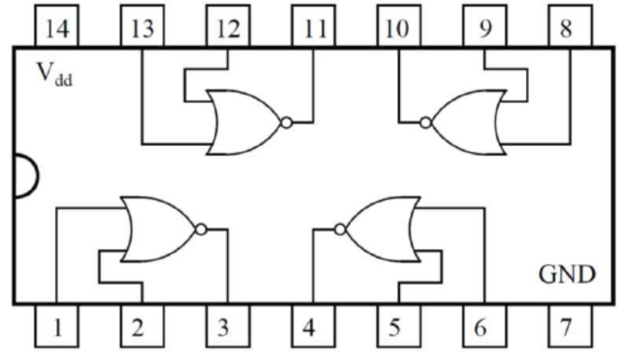


Physical implementation on ICs

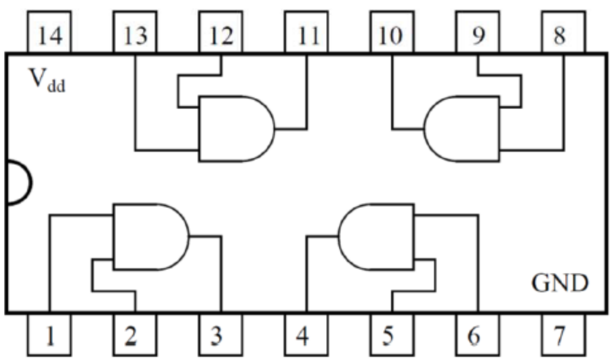
4011
Quad NAND gate



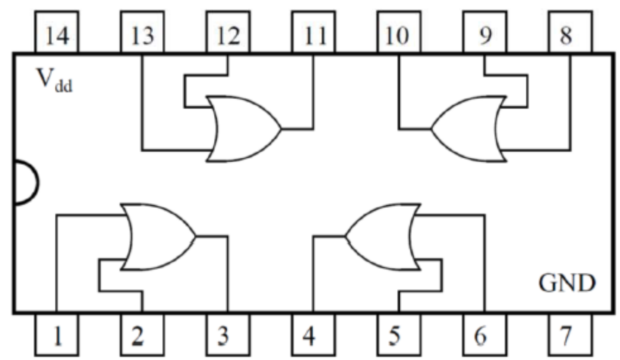
4001
Quad NOR gate



4081
Quad AND gate

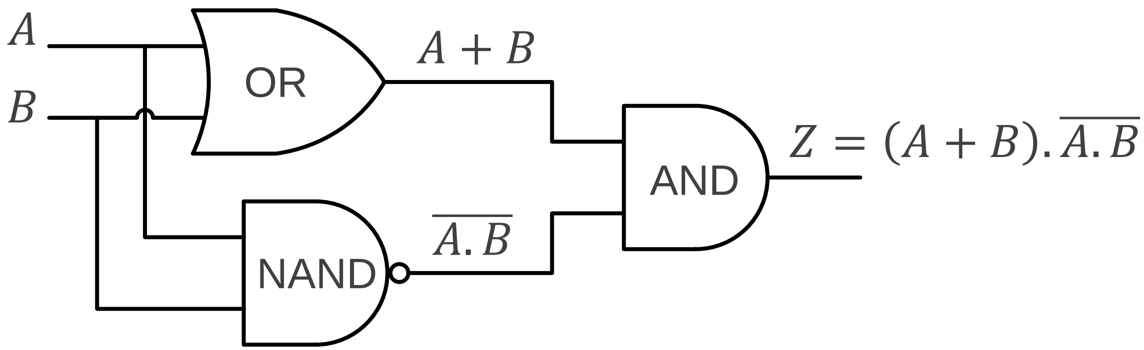


4071
Quad OR gate



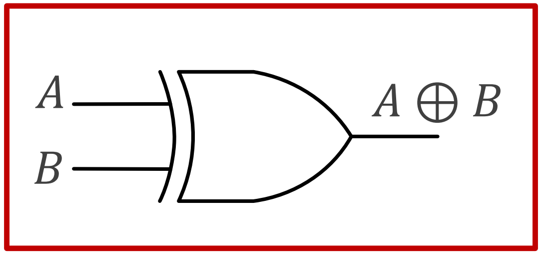
Combining primitive gates

- We can construct **truth table** for a digital circuit consisting of logical gates and find an **equivalent digital circuit** with **less** number of gates.
- Consider the following circuit:



Truth Table

A	B	$A + B$	$\overline{A.B}$	$(A + B).\overline{A.B}$	$A \oplus B$
0	0	0	1	0	0
0	1	1	1	1	1
1	0	1	1	1	1
1	1	1	0	0	0



Exercise

Using the logic given in the problem (and / or) write a logical expression describing the problem and draw the digital circuit.

David buys (B) if he wants an item (W) **and** he has cash (C) **or** if he needs the item (N) **and** he has cash (C) **or** EFTPOS card (E)

Truth Table
($2^4 = 16$ combinations)

N	W	C	E	B
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

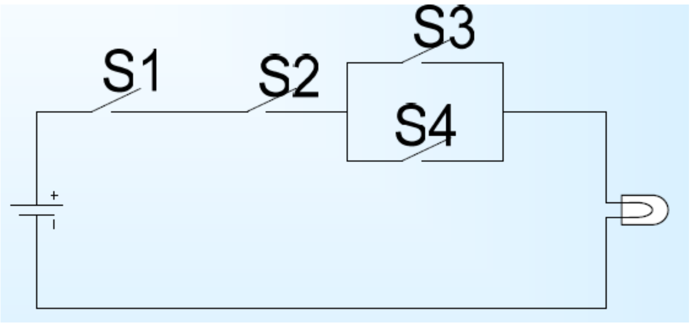
Exercise

For the following expression, draw the digital circuit using logical gates and construct the truth table.

$$Z = A + (B.\bar{C})$$

Boolean algebra

- In practice, we deal with **switching circuits** where signals can have two states. These are known as **switching signals.**
- These switching circuits can be easily **implemented** using digital logic gates once the output is obtained as a function of inputs in the form of a **logic expression.**
- **Boolean algebra** is a **set of rules or laws** that enable us to **simplify** the **digital logic/Boolean expressions** and **reduce** the **number of gates** needed to perform a particular logic operation in switching circuits.



In this circuit, **four** switches control the operation of the bulb. The bulb is switched on if the switches **S1 and S2** are closed, **and S3 or S4** is also closed, otherwise the bulb will not be switched on.

Boolean algebra – Single value theorems

- **Variables** in Boolean algebra only have **two values**, **0** or **1**.
- **Single value theorems** are a set of rules regarding different operations on a **single variable**. They are also known as **Boolean identities.**

AND operations

$\boxed{A.0 = 0}$

$\boxed{A.1 = A}$

$\boxed{A.A = A}$

$\boxed{A.\bar{A} = 0}$

OR operations

$\boxed{A + 0 = A}$

$\boxed{A + 1 = 1}$

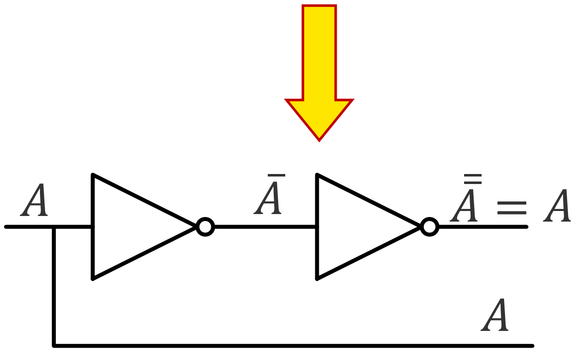
$\boxed{A + A = A}$

$\boxed{A + \bar{A} = 1}$

NOT operations

$\boxed{\overline{(\bar{A})} = \bar{\bar{A}} = A}$

Double negation

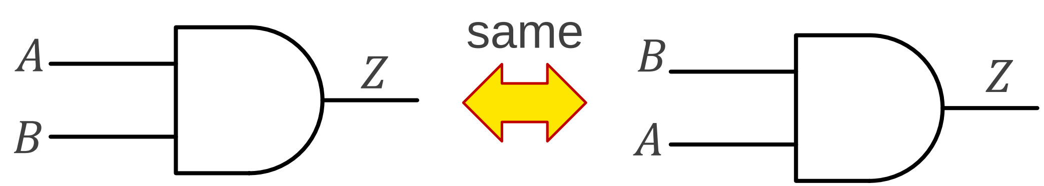


Boolean algebra – Commutative laws

- **Commutative** laws state that **input order does not matter** in **AND/OR** operations.

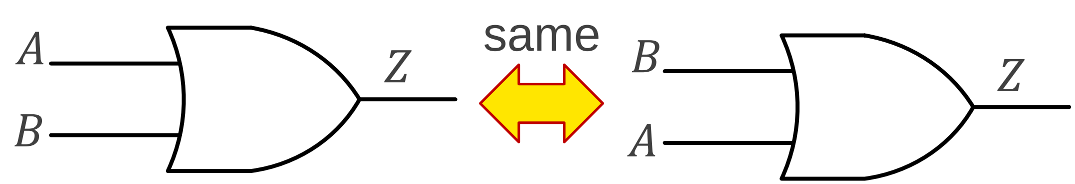
AND operation

$$\boxed{A.B = B.A}$$



OR operation

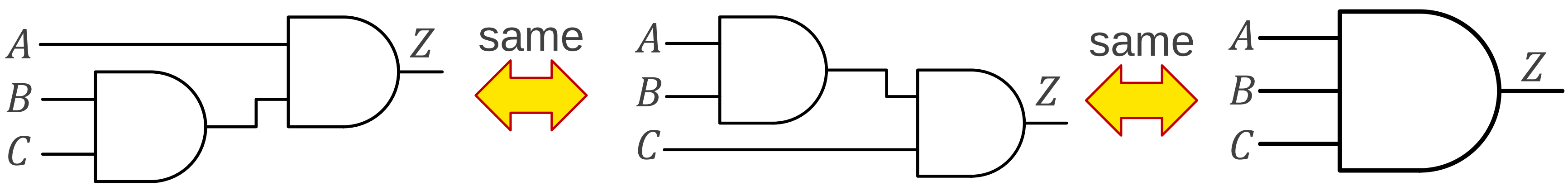
$$\boxed{A + B = B + A}$$



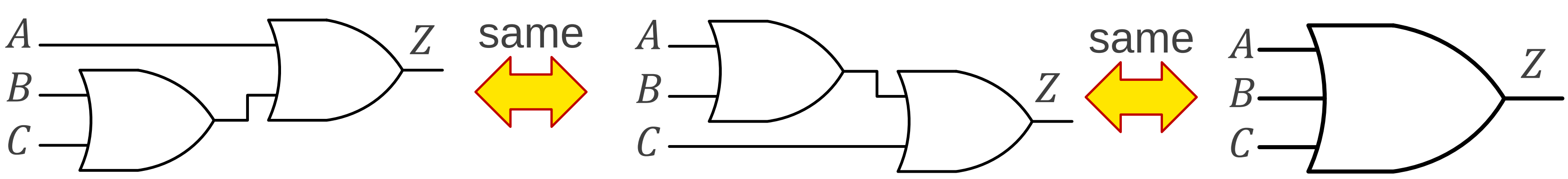
Boolean algebra – Associative laws

- **Associative** laws state that there is **no precedence** in **multiple OR operations** and in **multiple AND operations.**

AND operation: $\boxed{A.(B.C) = (A.B).C = A.B.C}$



OR operation: $\boxed{A + (B + C) = (A + B) + C = A + B + C}$



Boolean algebra – Distributive laws

- According to distributive laws, **AND** operation can be **distributed** over **OR** and **vice versa**.

$$A.(B + C) = (A.B) + (A.C)$$

$$A + (B.C) = (A + B).(A + C)$$

$A\ B\ C$	$B + C$	$A.B$	$A.C$	$A.(B + C)$	$(A.B) + (A.C)$
0 0 0	0	0	0	0	0
0 0 1	1	0	0	0	0
0 1 0	1	0	0	0	0
0 1 1	1	0	0	0	0
1 0 0	0	0	0	0	0
1 0 1	1	0	1	1	1
1 1 0	1	1	0	1	1
1 1 1	1	1	1	1	1

$A\ B\ C$	$B.C$	$A + B$	$A + C$	$A + (B.C)$	$(A + B).(A + C)$
0 0 0	0	0	0	0	0
0 0 1	0	0	1	0	0
0 1 0	0	1	0	0	0
0 1 1	1	1	1	1	1
1 0 0	0	1	1	1	1
1 0 1	0	1	1	1	1
1 1 0	0	1	1	1	1
1 1 1	1	1	1	1	1

NOTE: AND operation has precedence over OR operation.

Boolean algebra – Absorption laws

- Using **single value theorems**, some expressions can be **reduced** to a single variable. This is known as **absorption laws.**

$A + (A.B) = A$	Proof: Use the identity $A = A.1$ and $B + 1 = 1$ $A.1 + (A.B) = A(1 + B) = A.1 = A$
$A.(A + B) = A$	Proof: Use distribution and first absorption law $A.A + (A.B) = A + (A.B) = A$
$A.B + A.\bar{B} = A$	Proof: Factor out A and the identity $B + \bar{B} = 1$ $A.(B + \bar{B}) = A.1 = A$
$(A + B).(A + \bar{B}) = A$	Proof: Use distribution, second absorption law, and the identities $B\bar{B} = 0$, and $1 + \bar{B} = 1$ and factoring out (For your practice!)

Exercise

Prove the equivalency of the following expressions:

1. $(A + B).(A + C) = A + B.C$
2. $A + (\bar{A}.B) = A + B$

Exercise

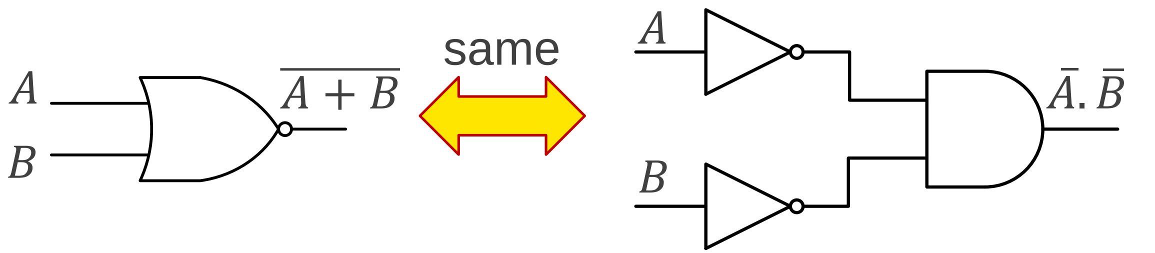
Write a logical expression for the output Z given the following truth table.

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

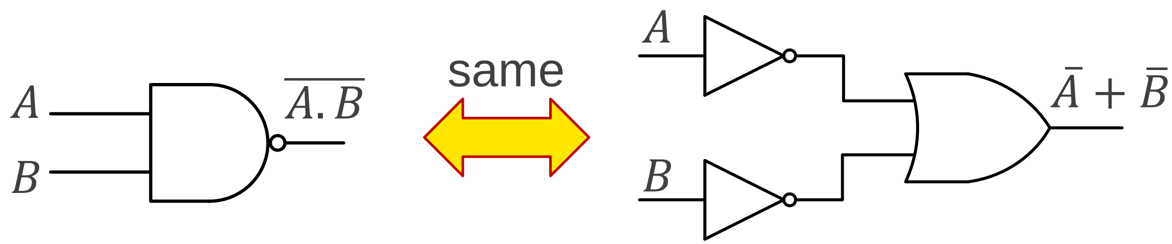
Boolean algebra – DeMorgan’s Theorems

- The **two** famous DeMorgan’s Theorems state the relationship in converting **NAND** operation into **OR of negated** inputs, and **NOR** operation into **AND of negated** inputs.

$$\boxed{\overline{A+B} = \bar{A}.\bar{B}}$$



$$\boxed{\overline{A.B} = \bar{A}+\bar{B}}$$



Boolean algebra – DeMorgan's Theorems

$$\boxed{\overline{A + B} = \bar{A}.\bar{B}}$$

Truth Table

A	B	$\overline{A + B}$	$\bar{A}.\bar{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

$$\boxed{\overline{A.B} = \bar{A} + \bar{B}}$$

Truth Table

A	B	$\overline{A.B}$	$\bar{A} + \bar{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Boolean algebra – DeMorgan’s Theorems

- DeMorgan’s Theorems describe the equivalent relationship between gates with **inverted inputs** and gates with **inverted outputs.**
- When **“breaking”** a **complementation bar** in a Boolean expression, the operation directly underneath the break (addition or multiplication) **reverses**, and the broken bar pieces remain over the respective terms.
  - A **NAND** gate is equivalent to a **“Negative Input”-OR** gate.
  - A **NOR** gate is equivalent to a **“Negative Input”-AND** gate.
- Other important points include:
  - It is often easier to approach a problem by **breaking the longest** (**uppermost**) **bar** before breaking any bars under it
  - **Never** attempt to break **two bars** in **one** step!
  - Complementation bars function as **grouping** symbols.
  - When a bar is broken, the terms **underneath** it must remain **grouped**.
  - **Parentheses** may be placed around these grouped terms as a help to **avoid** changing precedence.

Exercise

Prove the equivalency of the following expressions:

1. $\overline{A + \overline{BC}}$
2. $\overline{\overline{A\overline{B}} + \overline{A + BC}}$

Questions?