# CompSci Course

{CODENATION}

# Binary & Logic.

# Learning Objectives

Recognise Binary, and Binary Logic.

Convert Binary to Decimal.

Understand the function of Logic Gates.

First things first

# What is Binary?

Binary is a numbering system where there are only two values: 0 and 1.

The term Binary is also applied to other encoding/decoding systems where there are only two possible states.

In computing 1 refers to “on” or “true”, and 0 refers to “off” or “false”.

{CODENATION}

# For example

**Remember that last slide?**

**Here's the Binary representation...**

CODENATION

# 01000110 01101001 01110010 01110011

**01010111 01101000 01100001 01110100 00100000 01101001 01110011 00100000 01000010 01101001 01101110 01100001 01110010 01111001 00111111**

01000010 01101001 01101110 01100001 01110010 01111001 00100000 01101001 01110011 00100000 01100001 00100000 01101110 01110101
01101101 01100010 01100101 01110010 01101001 01101110 01100111 00100000 01110011 01111001 01110011 01110100 01100101 01101101
00100000 01110111 01101000 01100101 01110010 01100101 00100000 01110100 01101000 01100101 01110010 01100101 00100000 01100001
01110010 01100101 00100000 01101111 01101110 01101100 01111001 00100000 01110100 01110111 01101111 00100000 01110110 01100001
01101100 01110101 01100101 01110011 00111010 00100000 00110000 00100000 01100001 01101110 01100100 00100000 00110001 00101110
11100010 10000000 10101000 00001010 01010100 01101000 01100101 00100000 01110100 01100101 01110010 01101101 00100000
01000010 01101001 01101110 01100001 01110010 01111001 00100000 01101001 01110011 00100000 01100001 01101100 01110011 01101111
00100000 01100001 01110000 01110000 01101100 01101001 01100101 01100100 00100000 01110100 01101111 00100000 01101111 01110100
01101000 01100101 01110010 00100000 01100101 01101110 01100011 01101111 01100100 01101001 01101110 01100111 00101111 01100100
01100101 01100011 01101111 01100100 01101001 01101110 01100111 00100000 01110011 01111001 01110011 01110100 01100101 01101101 01110011
00100000 01110111 01101000 01100101 01110010 01100101 00100000 01110100 01101000 01100101 01110010 01100101 00100000 01100001
01110010 01100101 00100000 01101111 01101110 01101100 01111001 00100000 01110100 01110111 01101111 00100000 01110000 01101111
01110011 01110011 01101001 01100010 01101100 01100101 00100000 01110011 01110100 01100001 01110100 01100101 01110011 00101110
11100010 10000000 10101000 11100010 10000000 10101000 01001001 01101110 00100000 01100011 01101111 01101101 01110000 01110101
01110100 01101001 01101110 01100111 00100000 00110001 00100000 01110010 01100101 01100110 01100101 01110010 01110011 00100000
01110100 01101111 00100000 11100010 10000000 10011100 01101111 01101110 11100010 10000000 10011101 00100000 01101111 01110010
00100000 11100010 10000000 10011100 01110100 01110010 01110101 01100101 11100010 10000000 10011101 00101100 00100000
01100001 01101110 01100100 00100000 00110000 00100000 01110010 01100101 01100110 01100101 01110010 01110011 00100000 01110100

… please don’t make me highlight it all

# Why is binary important?

The binary numbering system is key to all operations, it enables devices to process, access, store, and manipulate data at the instruction of the CPU.

The schema is simple, and elegant, in principle but offers an incredibly efficient method of controlling logic circuits.

Without binary none of the things we know and love would be possible, from opening a webpage, playing a game, to running the calculator app because we've not recently had a coffee.

# Bases

## Let's keep it **simple!**

Think about our everyday decimal numbering system which uses a range of **0-9**.

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and these combined to make higher numbers, **ten** numbers in total.

This is base-10.

# Bases

$10^2$, $10^1$, $10^0$

**534**

**hundreds**, **tens**, **units**

Our base-10 decimal system is also comprised of units to the factor of ten: **units**, **tens**, **hundreds**, **thousands** etc.

# Bases

$2^2$, $2^1$, $2^0$

**101**

**fours**, **twos**, **units**

Binary numbers are **base-2**, meaning they only have two values - **0** and **1**.

Just like base-10, these numbers work in combinations but mathematically the rules are the same.

# Conversion

Can we **convert** Binary to Decimal?

## Think back to this

$2^2$, $2^1$, $2^0$

$\downarrow \quad \downarrow \quad \downarrow$

**101**

$\uparrow \quad \uparrow \quad \uparrow$

**fours, twos, units**

## And consider this

$$2^0 = 1$$
$$2^1 = 2$$
$$2^2 = 2 \text{ x } 2 = 4$$
$$2^3 = 2 \text{ x } 2 \text{ x } 2 = 8$$
$$2^4 = 2 \text{ x } 2 \text{ x } 2 \text{ x } 2 = 16$$

* numbers to the power of **0** are **1**
* numbers to the power of **1** are **themselves**

{CODENATION}

# Conversion

## That would mean...

**101** as a decimal would be:

$$(1 \times 1) + (0 \times 2) + (1 \times 4) = \mathbf{5}$$
$$1 + 0 + 4 = \mathbf{5}$$

{CN}®

# Conversion

## Let's try a few ourselves...

1011 = ?
1001 = ?
0110 = ?

{ CN }®

# Conversion

## Let's try a few ourselves...

1011 = **11**
1001 = **9**
0110 = **6**

# Conversion

Can we **convert** Decimal to Binary?

# Think back to this

$2^0 = 1$

$2^1 = 2$

$2^2 = 2 \times 2 = 4$

$2^3 = 2 \times 2 \times 2 = 8$

$2^4 = 2 \times 2 \times 2 \times 2 = 16$

* numbers to the power of **0** are **1**
* numbers to the power of **1** are **themselves**

# Conversion

## Let's keep keeping it simple...

Following a simple rule we can convert decimal to binary, keep in mind this is one of many approaches.

**The rule:** subtract the highest possible power until we have 0

# Conversion

## For example...

Let's take the decimal **20**.

The highest power to 20 is **16**

$2^4 = 2 \times 2 \times 2 \times 2 = \underline{\mathbf{16}}$

# Conversion

## So that would mean...

$20 - 16 = \mathbf{4}$

The next highest power is **4**

$2^2 = 2 \times 2 = \mathbf{4}$

# Conversion

## So that would mean...

4 - 4 = **0**

And now we reach **0**

So far we have used two numbers: **16**, **4**

## Think back to this

$2^2$, $2^1$, $2^0$

**101**

fours, twos, units

## And consider this

We used **16**, **4**.

**1** sixteens
**no** eights
**1** fours
**no** twos
**no** units

20 = **10100**

{CODENATION}

# Conversion

## Let's try a few ourselves...

19 = ?
14 = ?
11 = ?

{CN}®

# Conversion

## Let's try a few ourselves...

19 = **10011**
14 = **1110**
11 = **1011**

{CN}®

Compute logic

## Awesome, but what about logic operations?

I'm glad you asked...

To understand how binary conditions 'control' our computer, we need to understand logic gates.

This may be a good time for a coffee break ☕

{CN}®

# Logic gates...

Logic gates act as 'building blocks' for digital circuitry, and they are responsible for performing basic logical functions.

Logic gates make decisions based on the combination of digital signals coming from its inputs.

They mostly take two inputs to produce one output and use **boolean algebra**.

Compute logic

# Boolean Algebra...

This is where binary comes in, boolean algebra uses two binary conditions - **true** and **false**.

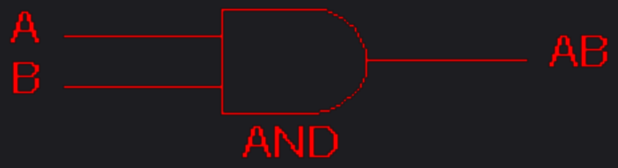
In boolean algebra true is represented by **1**, and false by **0**.

Think of a light switch: 1 = it's **on** and so it the light, 0 = it's **off** and so is the light.

{ CN }®

# Compute logic

## AND gate...




An AND gate can take two or more inputs and produces one output. It returns true if all inputs are true.

Think of it like this...

If input A **and** B are true, output true. If input A is true and input B is false, output false.

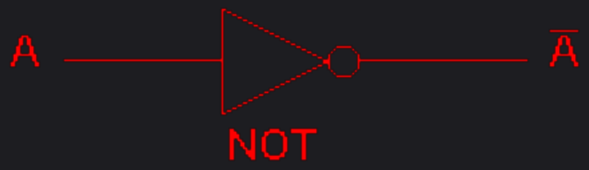
Truth table.

| Input A | Input B | Output |
|---|---|---|
| 0 | 0 | **0** |
| 0 | 1 | **0** |
| 1 | 0 | **0** |
| 1 | 1 | **1** |

{ CN }®

Compute logic

## NOT gate...




A NOT gate only takes one input and produces one output, it is also called an 'inverter'.

Think of it like this...

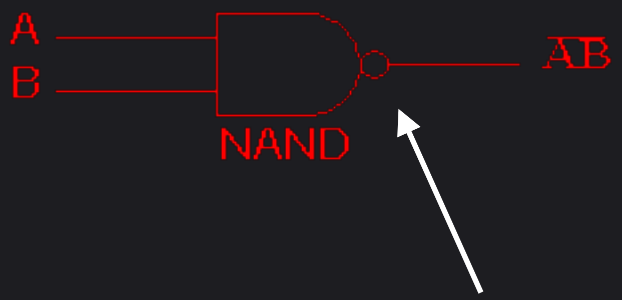
If the input is NOT true, output true.
If the input is true, output false.

| Input | Output |
|---|---|
| 0 | 1 |
| 1 | 0 |

{ CN }®

Compute logic

# **NAND** gate...




'**o**' represents the inversion.

A NAND gate only takes one input and produces one output, it is an AND gate with the output **inverted**.

Essentially think of this as an AND, and invert the output when considering the '**o**' – this symbol appears in multiple logic gates.

| Input A | Input B | Output |
|---|---|---|
| 0 | 0 | **1** |
| 0 | 1 | **1** |
| 1 | 0 | **1** |
| 1 | 1 | **0** |

Compute logic

## OR gate...


An OR gate can take two or more inputs and produces one output, it returns true if A OR B (or both) are true.

Think of it like this...

If A **OR** B is true true, output true.
If neither A **OR** B are true, output false.

| Input A | Input B | Output |
|---|---|---|
| 0 | 0 | **0** |
| 0 | 1 | **1** |
| 1 | 0 | **1** |
| 1 | 1 | **1** |

{ CN }®

Compute logic

## **NOR gate...**


A NOR gate can take two or more inputs and produces one output, as before it is an OR gate with the result **inverted**.

Notice the '**o**'?

The same logic as an OR applies, just inverted.

| Input A | Input B | Output |
|---|---|---|
| 0 | 0 | **1** |
| 0 | 1 | **0** |
| 1 | 0 | **0** |
| 1 | 1 | **0** |

{ CN }®

Compute logic

# EXOR gate...


')' represents the exclusion.

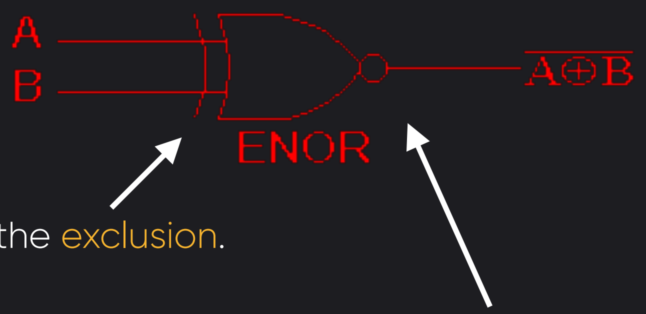
EXOR means **Ex**clusive-**OR**, it can take only two inputs and produces one output, similar to an OR gate but **excludes** both outputs being the same.

So we have an OR, an inverted OR (NOR), and a gate that **excludes** two values being true (**EXOR**).

| Input A | Input B | Output |
|---|---|---|
| 0 | 0 | **0** |
| 0 | 1 | **1** |
| 1 | 0 | **1** |
| 1 | 1 | **0** |

Compute logic

# EXNOR gate...




')' represents the exclusion.

'O' represents the inversion.

EXNOR means **Ex**clusive-**NOR**, it can take only two inputs and produces one output, similar to an EXOR gate which excludes both outputs being the same, but EXNOR also **inverts** the output too.

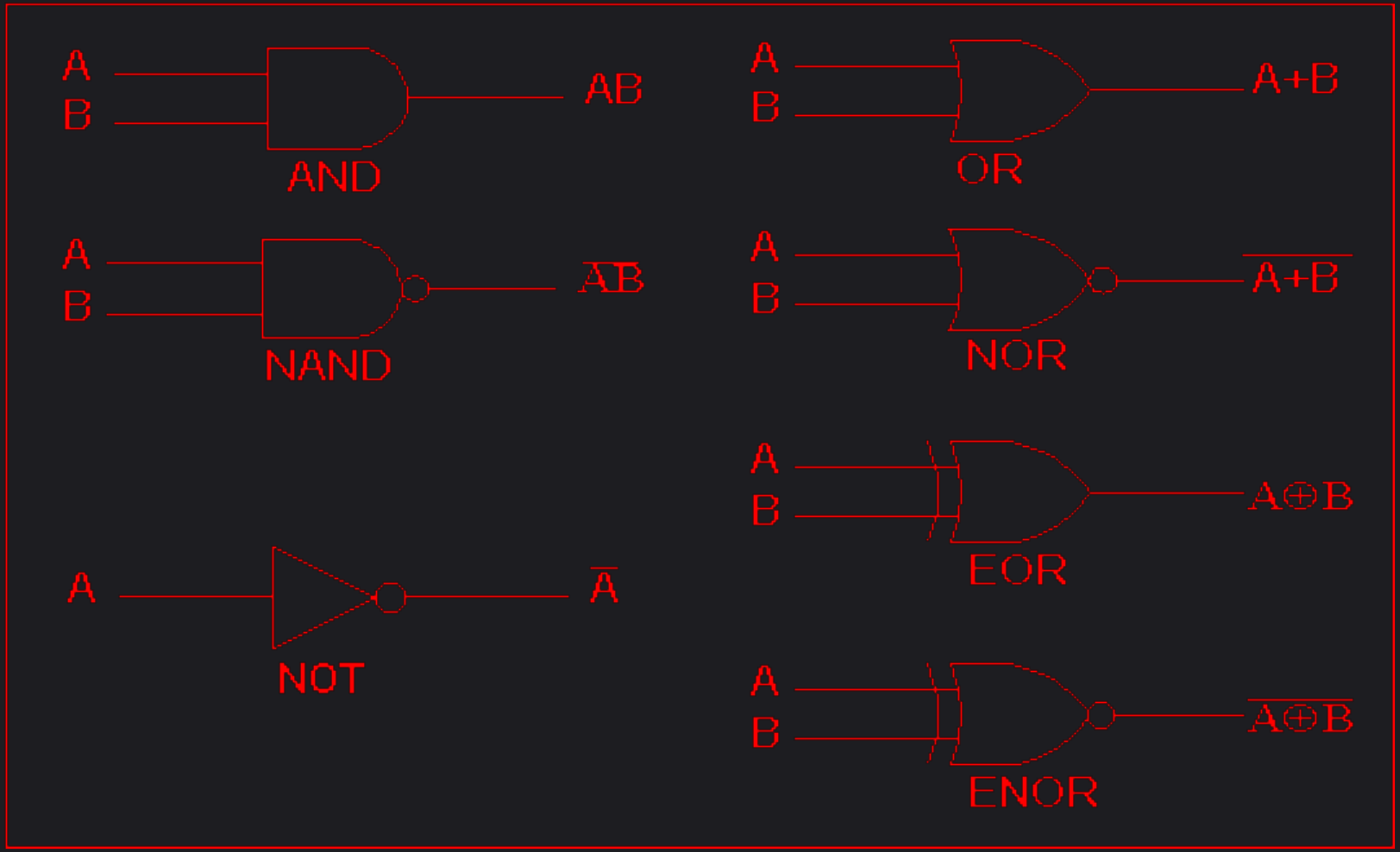
Finally we have an OR, an inverted OR (NOR), and a gate that excludes two values being true (EXOR), and a gate which **excludes** and **inverts** (**EXNOR**).

| Input A | Input B | Output |
|---|---|---|
| 0 | 0 | **1** |
| 0 | 1 | **0** |
| 1 | 0 | **0** |
| 1 | 1 | **1** |

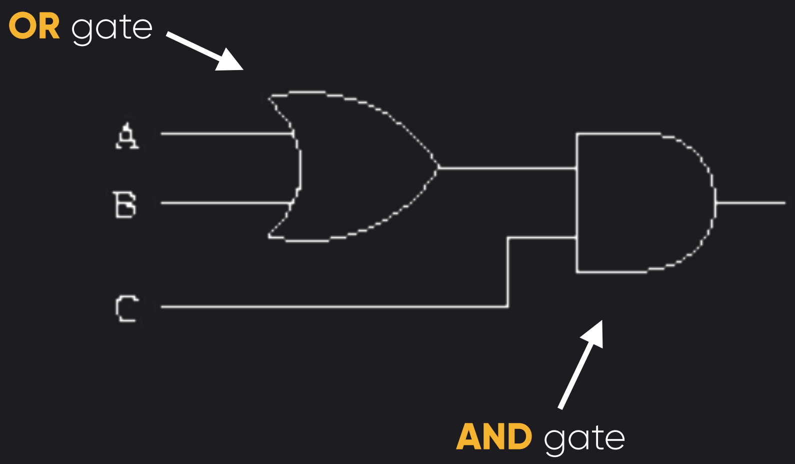
{ CN }®

# Cheatsheet...

Logic Gates



# Let's take this in...




| Input A | Input B | Input C | Output |
|---|---|---|---|
| 0 | 0 | 0 | **0** |
| 0 | 0 | 1 | **0** |
| 0 | 1 | 0 | **0** |
| 0 | 1 | 1 | **1** |
| 1 | 0 | 0 | **0** |
| 1 | 1 | 0 | **0** |
| 1 | 0 | 1 | **1** |
| 1 | 1 | 1 | **1** |

# Group Challenge

## Let's tackle a few...

In your group complete the truth table for both logic circuits.

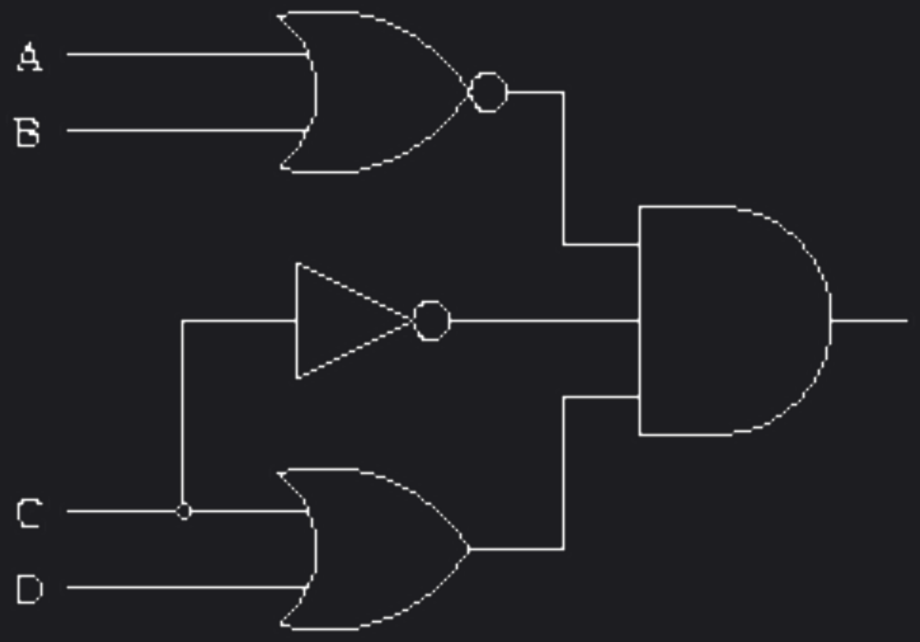
Use the cheatsheet, and identify the types of gate first.

Compute logic

## Circuit one...

Compute logic

# Circuit two: electric boogaloo...



# Revisiting Learning Objectives

Recognise Binary, and Binary Logic.

Convert Binary to Decimal.

Understand the function of Logic Gates.