ÇUKUROVA ÜNİVERSİTESİ

Department
of
Computer Engineering

CEN 263
Digital Design

Autumn 2024

Lecture 1

Administration

- Lecturer:
  Dr Yunus Emre Cogurcu
  ycogurcu@cu.edu.tr

- Lab Lead:
  Res. Asst. Fırat Kumru

Announcements

- **Website**:

- **No lecture for 28 October 2024**

Week 1 Outlines

- Number systems
- Binary numbers
- Logic levels
- Transistors
- Gates
- Boolean expressions

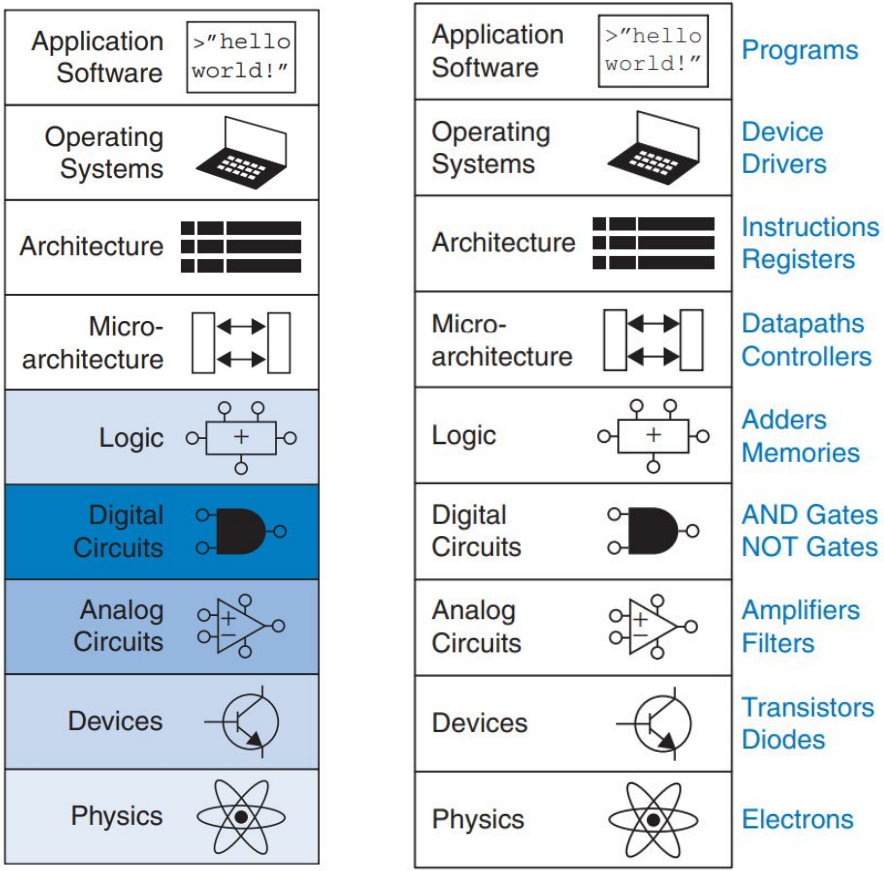
What is Digital Design?

- **Digital Design** is the creation of systems that use discrete (often binary) signals for processing information.

- It involves designing circuits that process data in the form of **0**s and **1**s, typically through logic gates.

- Examples of digital systems: Computers, smartphones, digital cameras, microcontrollers.

Levels of abstraction for an electronic computing system



Importance of Digital Design in Modern Electronics

- Digital systems form the backbone of modern electronics.
- Understanding how digital circuits work is crucial for designing hardware in computing, telecommunications, robotics, and more.
- Digital circuits are faster, more efficient, and more reliable than analog systems.

Comparing digital and analog signals

	Analog	Digital
Signal	Analog signal is a continuous signal which represents physical measurements.	Digital signals are discrete time signals generated by digital modulation.
Waves	Denoted by sine waves	Denoted by square waves
Representation	Uses continuous range of values to represent information	Uses discrete or discontinuous values to represent information
Example	Human voice in air, analog electronic devices	Computers, CDs, DVDs, and other digital electronic devices.
Data transmissions	Subjected to deterioration by noise during transmission and write/read cycle.	Can be noise-immune without deterioration during transmission and write/read cycle.

What are Number Systems?

- A system used to express quantities.

- In the decimal system (Base-10), we use the digits 0 through 9 to represent quantities like 25, 100, etc.

- In the binary system (Base-2), we use only the digits 0 and 1 to represent numbers, such as 1011 (which equals 11 in decimal).

What are Number Systems?

- Common types in digital design:
  - Decimal (Base-10): Standard system for humans.
  - Binary (Base-2): Used by computers and digital systems.
  - Octal (Base-8): Simplifies large binary numbers.
  - Hexadecimal (Base-16): Convenient for representing binary in compact form.

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

2's power	decimal
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024

Binary Numbers

Why Binary Numbers?

Binary Numbers

Why Binary Numbers?

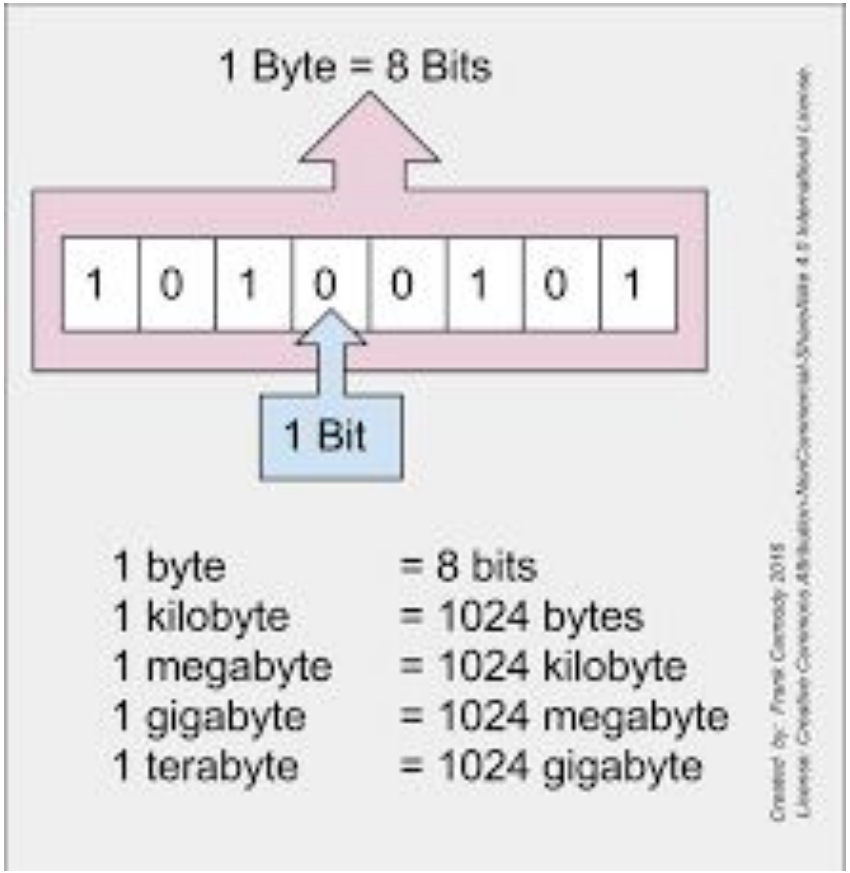
- Computers use electrical signals, represented by two states: On (1) and Off (0).
- Binary is the simplest number system that can be used to represent these two states.

Binary Numbers

- **Bits and Bytes:**
  - **A bit is a binary digit (either 0 or 1).**
  - **A byte is a group of 8 bits.**

Binary Numbers

- **Bits and Bytes:**
  - **A bit is a binary digit (either 0 or 1).**
  - **A byte is a group of 8 bits.**



Decimal Representation

Method:

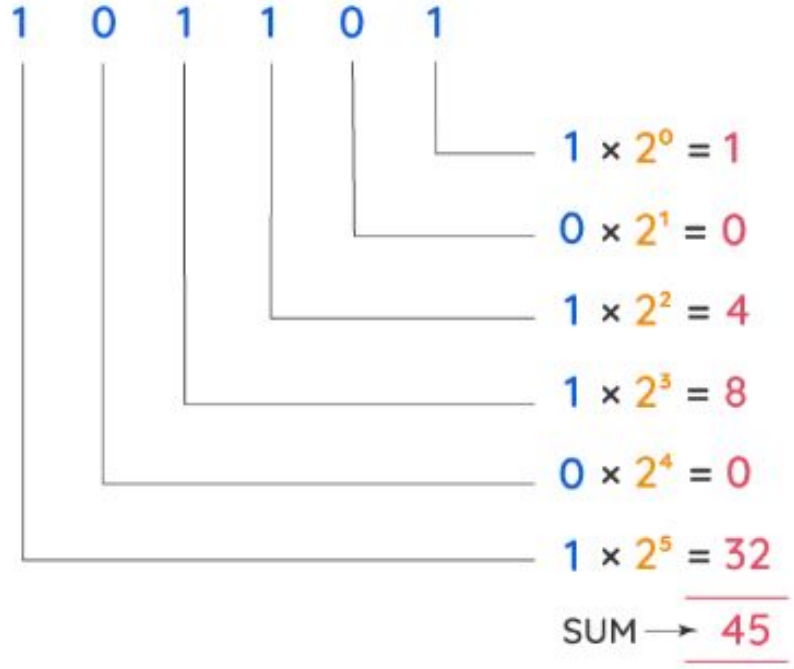
1's column
10's column
100's column
1000's column

- Multiply each bit by 10 raised to the power of its position (starting from 0 at the right).
- Add the results.

$$9742_{10} = 9 \times 10^3 + 7 \times 10^2 + 4 \times 10^1 + 2 \times 10^0$$

nine thousands | seven hundreds | four tens | two ones

Binary Representation



$(101101)_2 = (45)_{10}$

Binary to Decimal Conversion

16's column
8's column
4's column
2's column
1's column

$$10110_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 22_{10}$$

one sixteen
no eight
one four
one two
no one

Binary to Decimal Conversion

Binary System (Base 2) -> Decimal

- Digits: 0 1
- What is 1101?
- We prepend with “0b” to indicate that we’re representing a number in binary
- 0b01101 = $\mathbf{01101}_2$

$$= \mathbf{0} \times 2^4 + \mathbf{1} \times 2^3 + \mathbf{1} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{1} \times 2^0$$
$$= 8 + 4 + 1$$
$$= 13$$

Binary to Decimal Conversion

The prefix 0b is commonly used to explicitly indicate that the number following it is in binary (Base-2) format. This is helpful in programming and technical documentation, where it's important to differentiate between different number systems like binary, decimal, hexadecimal, etc.

For example:

0b01101 tells you that the number is binary, and the value is 1101 in decimal (which equals 13).

Binary to Decimal Conversion

So, 0b is a notation used to avoid ambiguity and make it clear that the number is in binary form.

Similarly:

0x is used to denote hexadecimal (e.g., 0xA is 10 in decimal).

Decimal to Binary Conversion

- **Conversion Method:**
  - **Repeatedly divide the decimal number by 2, recording the remainder.**
  - **The binary number is the remainders read from bottom to top.**

Decimal to Binary Conversion

Step 1: Divide the given number **13** repeatedly by 2 until you get '0' as the quotient

$13 \div 2 = 6$ (Remainder 1)
$6 \div 2 = 3$ (Remainder 0)
$3 \div 2 = 1$ (Remainder 1)
$1 \div 2 = 0$ (Remainder 1)

Step 2: Write the remainders in the reverse order **1 1 0 1**

$\therefore 13_{10} = 1101_{2}$

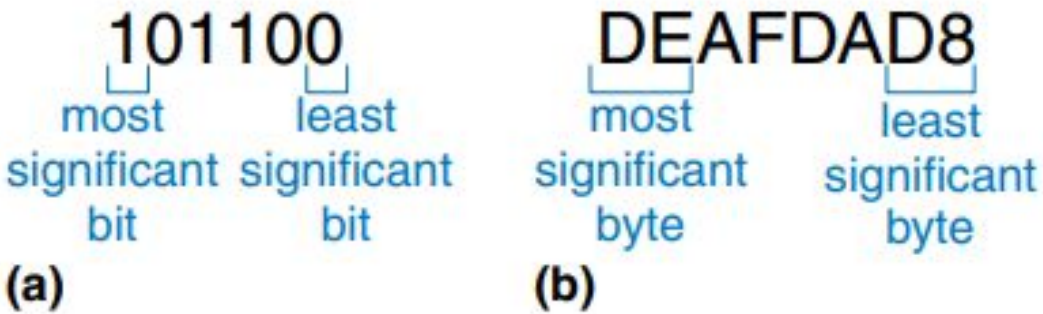
(Decimal) (Binary)

Decimal to Binary Conversion

Convert $174_{(10)}$ to binary?

Decimal to Binary Conversion

Division by 2	Quotient	Remainder
174 ÷ 2	87	0 (LSB)
87 ÷ 2	43	1
43 ÷ 2	21	1
21 ÷ 2	10	1
10 ÷ 2	5	0
5 ÷ 2	2	1
2 ÷ 2	1	0
1 ÷ 2	0	1 (MSB)


101100
most significant bit
least significant bit
(a)
DEAFDAD8
most significant byte
least significant byte
(b)

Hexadecimal number system

Table 1.2 Hexadecimal number system

Hexadecimal Digit	Decimal Equivalent	Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Conversion of a hexadecimal number to decimal

256's column
16's column
1's column

$$2ED_{16} = 2 \times 16^2 + E \times 16^1 + D \times 16^0 = 749_{10}$$

two
two hundred
fifty six's

fourteen
sixteens

thirteen
ones

Do some exercises with

- Base-2
- Base-8
- Base-16

with 4 operations

Logic Levels

Definition:

Logic Level 1 (High): Represents binary 1, typically a higher voltage (e.g., 5V or 3.3V).

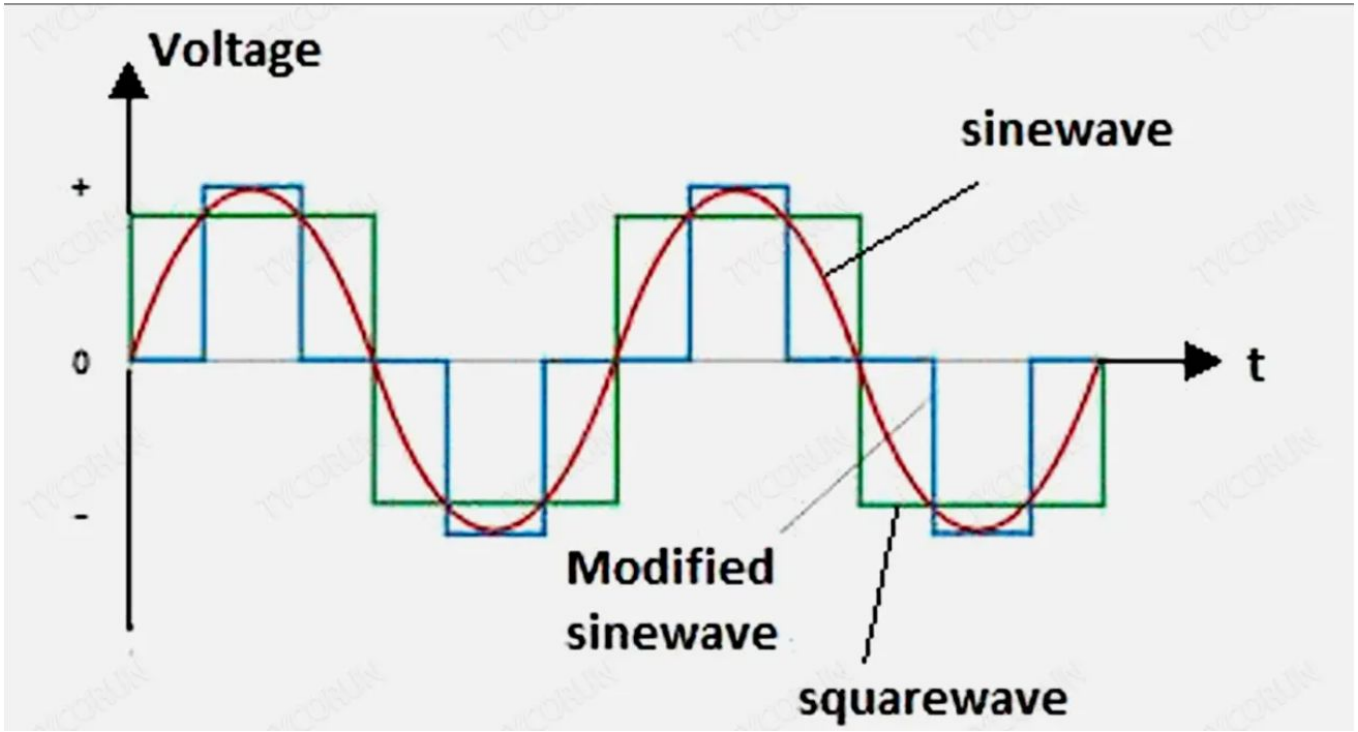
Logic Level 0 (Low): Represents binary 0, a lower voltage (e.g., 0V).

Logic Levels

Importance:

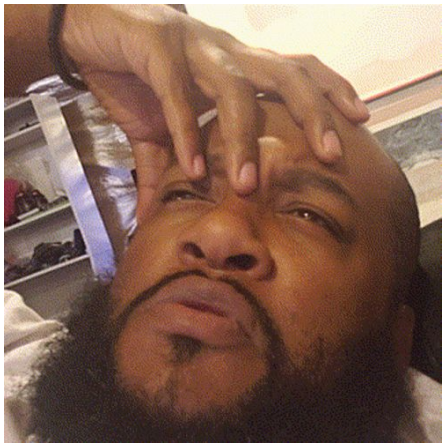
Digital circuits rely on these voltage levels to distinguish between 0s and 1s.

Logic Levels



Logic Levels

How to change square wave into sine wave?



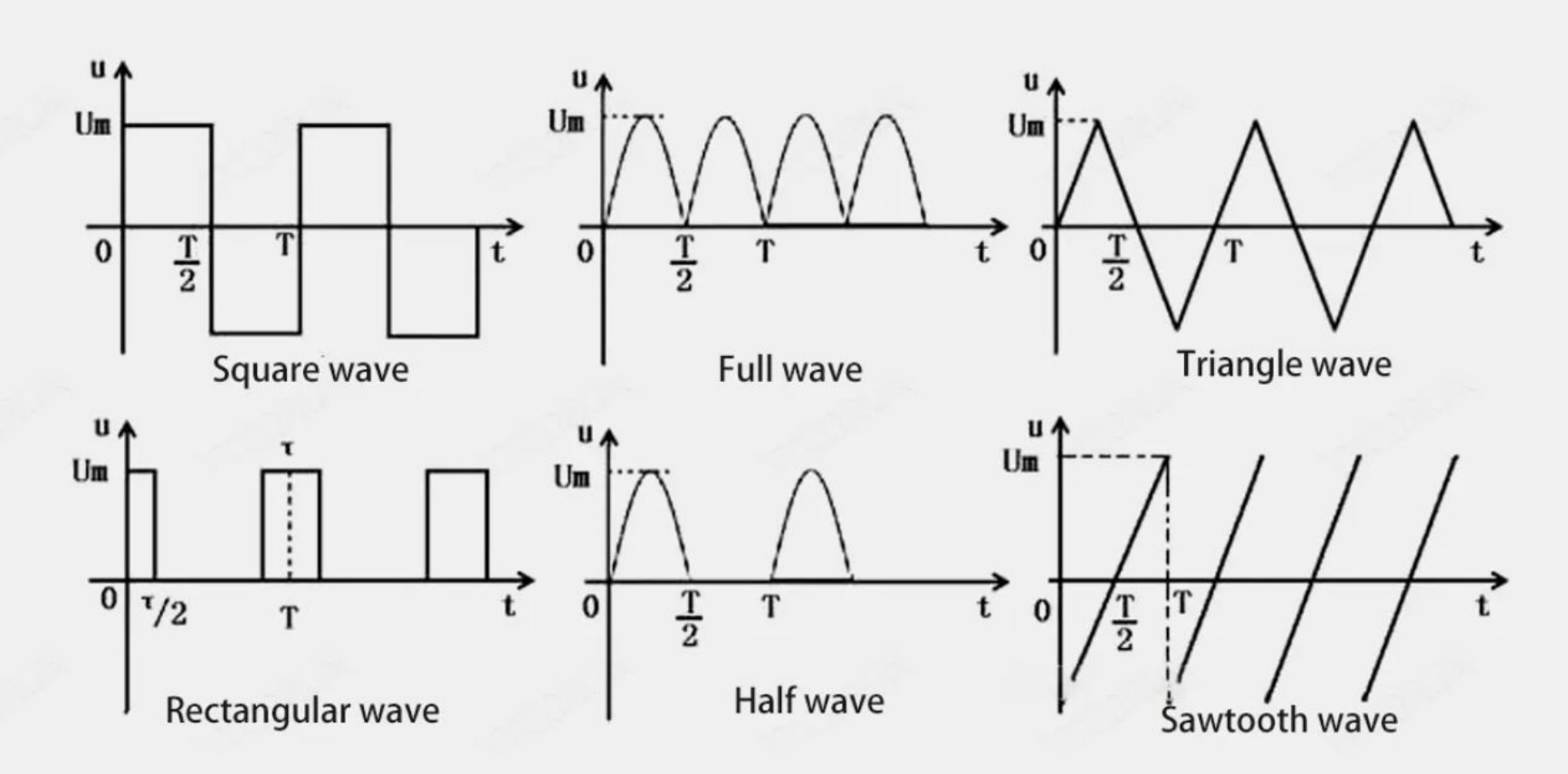
Logic Levels

How to change square wave into sine wave?

There are many ways to convert a square wave into a sine wave, including:

- **Use a D/A conversion chip to convert digital signals into analog signals.**
- **Use function generation chip to convert square wave into sine wave.**
- **Use the Wien bridge oscillation circuit to convert the square wave into a sine wave.**

Logic Levels



Decimal vs. Binary vs. Hexadecimal

- 8 bits
  - 1 “Byte”
  - 2 hex digits (= 256 values)
- 4 bits
  - 1 “Nibble”
  - 1 hex digit
- Conversions are usually time consuming, so it’s generally good to familiarize yourself with some powers of 2 and decimal->hex->binary conversion charts

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

2's power	decimal
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024


ME RIGHT NOW

Unsigned Integer

- `0b10110011`

1	0	1	1	0	0	1	1
$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$

- `0b10110011` = **128 + 32 + 16 + 2 + 1 = 179_{10}** for an **unsigned number**
- We can't represent negative numbers using unsigned representation!
- An n-bit unsigned integer can represent integers in the range [0, 2^n-1]
  - Smallest: `0b00…00`
  - Largest: `0b11…11`
- Often used in programs when you don't need negative numbers
  - Ex. size or length parameters of list-like structures since you can't really have negative length

What if we want to represent negative numbers?

- Decimal numbers: negative sign (e.g. -1234)
- One solution: define the leftmost bit to be the sign bit
  - Leftmost bit = 0 -> number is positive
  - Leftmost bit = 1 -> number is negative
- The rest of the number is the absolute value
- Consequence: We won't be able to represent as many positive numbers!
- For most signed systems, we want to be able to represent about as many negative numbers as positive numbers

Sign and Magnitude

- `0b10110011`

Sign	Magnitude						
1	0	1	1	0	0	1	1
negative	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$

- `0b10110010` = -1 x (32 + 16 + 2 + 1) = -51_{10}
- An n-bit sign and magnitude integer can represent integers in the range $[-(2^{n-1}-1), 2^{n-1}-1]$
  - Smallest: `0b11…11`
  - Largest: `0b01…11`

One’s Complement

1’s complement is a way to represent negative binary numbers by flipping all the bits of the positive version of the number (i.e., change 0s to 1s and 1s to 0s).

Example (for a 4-bit number):

+5 in binary (4 bits): 0101

-5 in 1’s complement:

Flip the bits of 0101 → 1010

(this is the 1’s complement representation of -5).

One’s Complement

Range: The range of numbers that can be represented (with 4 bits) is -7 to +7.

Problem: It has two representations for zero:
0000 is +0, and 1111 is -0
(this can cause issues in calculations).

One’s Complement

+0 Representation in 1’s Complement:

In any binary number system, +0 is naturally represented as all bits being 0.
For example:
In a 4-bit system, +0 is represented as 0000.

-0 Representation in 1’s Complement:

In 1’s complement, negative numbers are obtained by flipping all the bits of the corresponding positive number.

So, if we apply the 1’s complement operation to +0 (0000):
Flipping all the bits of 0000 gives us 1111, which represents -0 in 1’s complement.

Two’s Complement

2’s complement is considered an advanced and improved version of 1’s complement.

It builds on the idea of 1’s complement but solves its key problems, making it more practical and efficient for representing negative numbers in binary systems.

Two’s Complement

2’s complement is a more common and practical way of representing negative numbers in binary.

It allows for easy binary arithmetic (addition and subtraction) and only has one representation for zero.

Two’s Complement

How to compute 2’s complement:

Step 1: Start with the binary representation of the positive number.

Step 2: Flip all the bits (like in 1’s complement).

Step 3: Add 1 to the result.

Two’s Complement

Easy!

Two’s Complement

Example (for a 4-bit number):

+5 in binary: 0101 = 0b0101

Step 1: Flip the bits → 1010

Step 2: Add 1 → 1010 + 1 = 1011

-5 in 2’s complement is **1011**.

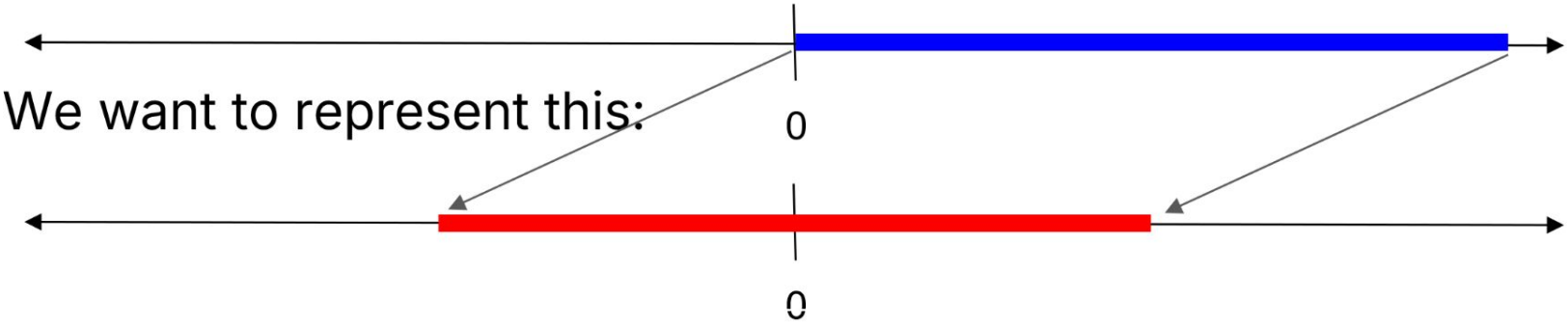
Two’s Complement

Properties of 2’s Complement:

- Positive numbers remain unchanged from their binary form.
- Negative numbers are represented by flipping the bits and adding 1.
- Range: The range of numbers (with 4 bits) is from -8 to +7.
- Single Zero: 2’s complement has only one representation of zero (0000).

Another try...

- Main idea: We have a system that can represent this:
- We want to represent this:



- “Shift” the numbers so they center on 0

BIG IDEA: Bits can represent everything

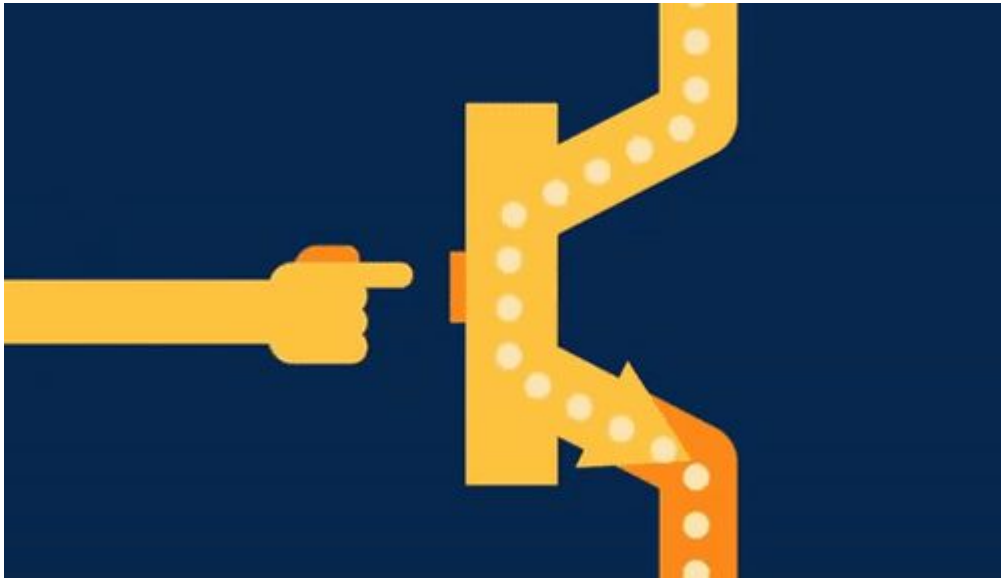
- Binary: a system of storing data using just 0 and 1
  - Everything in a computer is just bits (**bi**nary digi**ts**)
  - High voltage wire = 1, low voltage wire = 0
- Can represent just about everything
  - Numbers
  - Characters
    - 26 letters -> 5 bits (32 values)
    - All the English letters & punctuations are represented by ASCII
    - Unicode: 8, 16, 32 bits
  - Logical values
    - True -> 1; False -> 0

In summary…

- We represent “things” in computer science as particular bit patterns: n bits -> 2^n things
- The 4 different representations have their own benefits and uses
  - Unsigned
  - Sign and Magnitude: has most problems
  - Two’s complement: most widely used implementation for signed integers

Transistors

What is a Transistor?



Transistors

What is a Transistor?

A semiconductor device that acts as a switch or amplifier in circuits.

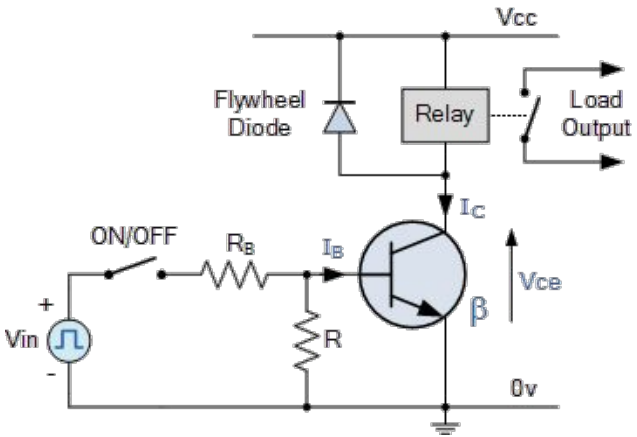
Fundamental building block for digital circuits (acts as an ON/OFF switch).

Transistors

Types of Transistors:

Bipolar Junction Transistor (BJT): Controlled by current.

Field Effect Transistor (FET): Controlled by voltage.



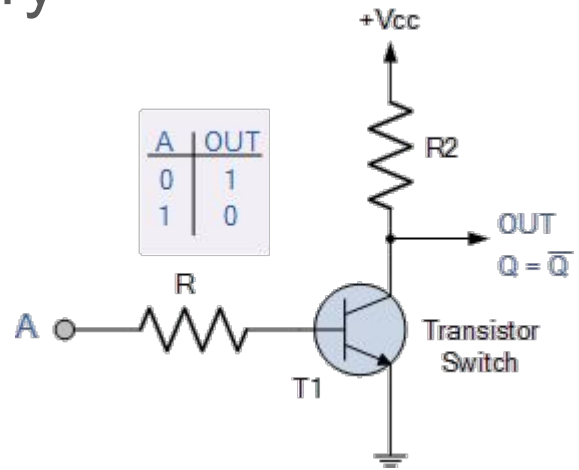
Transistors as Digital Switches

On/Off Behavior:

Transistors are used to implement binary logic by switching on (1) or off (0) depending on the input signal.

Example: A simple **NOT gate** using a transistor.

Input 0: Transistor is ON, output is 1.
Input 1: Transistor is OFF, output is 0.



Logic Gates

- Basic building blocks for digital circuits.
- Gates process binary inputs to produce a specific output.

Logic Gates

Common Gates:

NOT: Output is the opposite of the input.

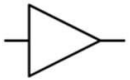
AND: Output is 1 if all inputs are 1.

OR: Output is 1 if any input is 1.

NAND, NOR, XOR, XNOR: Variations with different logic.

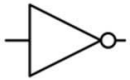
Logic Gates

Buffer



Input	Output
0	0
1	1

Inverter



Input	Output
0	1
1	0

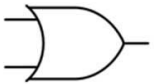
AND

A	B	Output
0	0	0
1	0	0
0	1	0
1	1	1

NAND

A	B	Output
0	0	1
1	0	1
0	1	1
1	1	0

OR

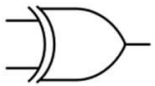


A	B	Output
0	0	0
1	0	1
0	1	1
1	1	1

NOR

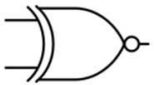
A	B	Output
0	0	1
1	0	0
0	1	0
1	1	0

XOR



A	B	Output
0	0	0
1	0	1
0	1	1
1	1	0

XNOR



A	B	Output
0	0	1
1	0	0
0	1	0
1	1	1

Boolean Algebra

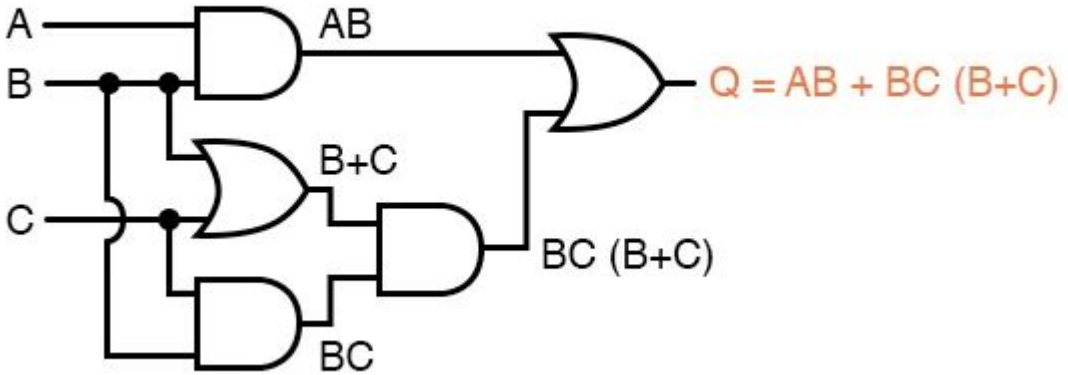
What is Boolean Algebra?

- A mathematical system for analyzing and simplifying logic circuits.

- Boolean variables take values of 0 or 1.

Boolean Algebra

- Operations:
- AND: $A \cdot B$
- OR: A+B
- NOT: $\overline{A}$

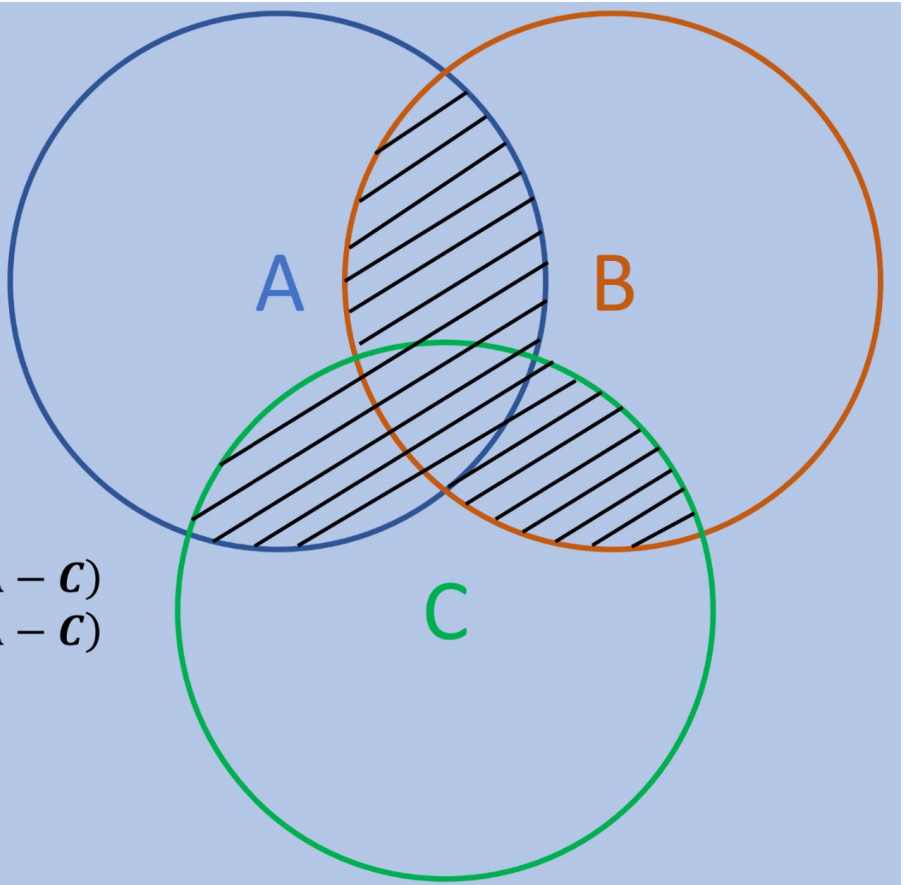


Simplifying Boolean Expressions

- De Morgan's Theorems:
  - $\overline{A \cdot B} = \overline{A} + \overline{B}$
  - $\overline{A + B} = \overline{A} \cdot \overline{B}$
- **Example**: Step-by-step simplification of an expression.
  - $A \cdot (B + C') = AB + AC'$

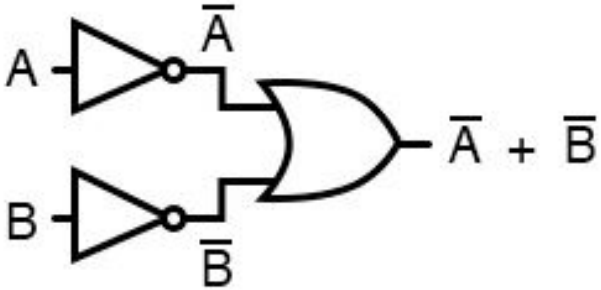
Simplifying Boolean Expressions

$$\mathbf{A} - (\mathbf{B} \cup \boldsymbol{C}) = (\mathbf{A} - \boldsymbol{B}) \cap (\mathbf{A} - \boldsymbol{C})$$
$$\mathbf{A} - (\mathbf{B} \cap \boldsymbol{C}) = (\mathbf{A} - \boldsymbol{B}) \cup (\mathbf{A} - \boldsymbol{C})$$



Simplifying Boolean Expressions

. . . is equivalent to . . .



$$\overline{AB} = \overline{A} + \overline{B}$$