

ARQCP Course

Arquitetura de Computadores
Licenciatura em Engenharia Informática

2025/26

Paulo Baltarejo Sousa

`pbs@isep.ipp.pt`

Material and Slides

Some of the material/slides are adapted from various:

- Presentations found on the internet;
- Books;
- Web sites;
- ...

- 1 **Number Systems**
- 2 **Binary Number Systems**
- 3 **Representing Data**
- 4 **Catastrophic Examples**

1 Number Systems

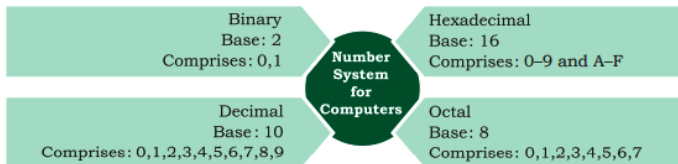
2 Binary Number Systems

3 Representing Data

4 Catastrophic Examples

What is?

- The **number system** is a way to represent or express (write) numbers.
- Every number system has a **set of unique characters or symbols or digits**.
 - The count of these digits is called the **radix** or **base** of the number system.
 - For example, if the number system has the base b , it has its digits in the $[0, b - 1]$ range.
- Number systems are also called **positional number system** because **the value of each digit in a number depends upon its position within the number**.
- The four different number systems used in the context of computer are:



Expressing numbers

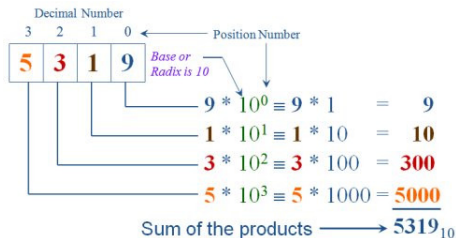
Binary $X_{(2)}$	Octal $X_{(8)}$	Decimal $X_{(10)}$	Hexadecimal $X_{(16)}$
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

■ $1101_{(2)} = 15_{(8)} = 13_{(10)} = D_{(16)}$

■ Recall, this is a way **to write numbers**, the value is the same.

Calculating a Number Value

- Each digit in a number system has **a weight value assigned**.
- Each digit also has **a position number**.
 - The first digit on the right is position zero. The next digit to the left is position one. To the left again is position two, etc.
- The weight of the digit in each position is the **base** of the number system **raised to the power of the position number**.



- The **value of the number** is then calculated by multiplying the value of the digit in each position by the base raised to the power of that position and then summing the products.

Converting to Decimal

■ Binary to decimal

$$\begin{aligned} \blacksquare \quad 1011010_{(2)} &= 1 * 2^6 + 0 * 2^5 + 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 = \\ &64 + 0 + 16 + 8 + 0 + 2 + 0 = 90_{(10)} \end{aligned}$$

■ Octal to decimal

$$\begin{aligned} \blacksquare \quad 16512_{(8)} &= 1 * 8^4 + 6 * 8^3 + 5 * 8^2 + 1 * 8^1 + 2 * 8^0 = \\ &4096 + 3072 + 320 + 8 + 2 = 7498_{(10)} \end{aligned}$$

■ Hexadecimal to decimal


$$\begin{aligned} \blacksquare \quad 13D1A_{(16)} &= 1 * 16^4 + 3 * 16^3 + 13 * 16^2 + 1 * 16^1 + 10 * 16^0 = \\ &65536 + 12288 + 3328 + 16 + 10 = 81178_{(10)} \end{aligned}$$

Converting from Decimal

Decimal to binary

$$27_{(10)} = 11011_{(2)}$$


Division	Result	Remainder
27/2	13	1
13/2	6	1
6/2	3	0
3/2	1	1
1/2	0	1



Decimal to octal

$$158_{(10)} = 236_{(8)}$$


Division	Result	Remainder
158/8	19	6
19/8	2	3
2/8	0	2



Decimal to hexadecimal

$$450_{(10)} = 1C2_{(16)}$$

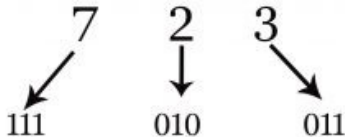
Division	Result	Remainder
450/16	28	2
28/16	1	12(C)
1/16	0	1



Converting to Binary

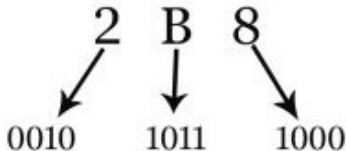
■ Octal to binary

$$723_{(8)} = 111010011_{(2)}$$



■ Hexadecimal to binary

$$2B8_{(16)} = 1010111000_{(2)}$$



Converting from Binary

■ Binary to octal

$$001110101011_{(2)} = 1653_{(8)}$$

$$\begin{array}{cccc} \underbrace{001} & \underbrace{110} & \underbrace{101} & \underbrace{011} \\ 1 & 6 & 5 & 3 \end{array}$$

■ Binary to hexadecimal

$$01011100111_{(2)} = 5E7_{(16)}$$

$$\begin{array}{ccc} \underbrace{0101} & \underbrace{1110} & \underbrace{0111} \\ 5 & E & 7 \end{array}$$

Arithmetic Operations

- The numbers are **represented in a unique manner** and help in performing different arithmetic operations like addition, subtraction, and division.
 - For **each number there is only one representation per base**.
- Arithmetic operations **could be performed in any base**:

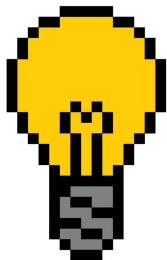
	Binary	Decimal	Hexadecimal
Addition	$\begin{array}{r} 1111100 \\ + \quad 10010 \\ \hline 10001110 \end{array}$	$\begin{array}{r} 124 \\ + \quad 18 \\ \hline 142 \end{array}$	$\begin{array}{r} 7C \\ + \quad 12 \\ \hline 8E \end{array}$

- Arithmetic operations with numbers in base r follow the same rules as decimal numbers.

- 1 Number Systems
- 2 Binary Number Systems**
- 3 Representing Data
- 4 Catastrophic Examples

- A **bit** (binary digit) is the smallest unit of data that a computer can process and store.
 - A bit is always in one of two physical states, similar to an on/off light switch.

ON



OFF



- Abstractly a bit has two values: 0 and 1.

Binary Codes

- A single bit **cannot convey much information**, so, we need a group of bits, called **binary codes**, to represent anything more complex.
- **Binary Code** is a representation format for the different types of data like numbers, text, image, video, and etc.
 - Binary codes for capital letters

A	100 0001	H	100 1000	O	100 1111	V	101 0110
B	100 0010	I	100 1001	P	101 0000	W	101 0111
C	100 0011	J	100 1010	Q	101 0001	X	101 1000
D	100 0100	K	100 1011	R	101 1010	Y	101 1001
E	100 0101	L	100 1100	S	101 0011	Z	101 1010
F	100 0110	M	100 1101	T	101 0100		
G	100 0111	N	100 1110	U	101 0101		

ARQCP

1000001 1010010 1010001 1000011 1010000

Boolean Algebra

- Boolean algebra is a type of algebra that is created by operating the binary system.
- Boolean algebra is concerned with **binary variables and logic operations**.

NOT	
x	x'
0	1
1	0

AND		
x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR		
x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

XOR		
x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

$$NOT(x) = \begin{cases} 1 & \text{if } x \text{ is } 0 \\ 0 & \text{if } x \text{ is } 1 \end{cases}$$

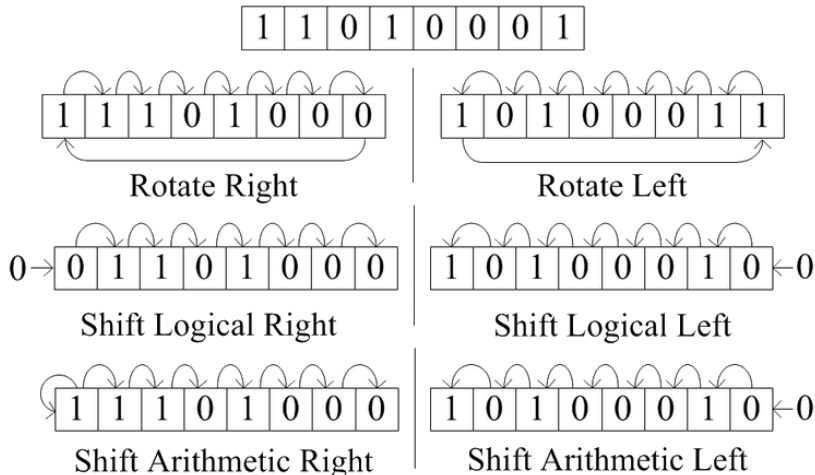
$$AND(x, y) = \begin{cases} 1 & \text{if both } x \text{ and } y \text{ are } 1 \\ 0 & \text{otherwise} \end{cases}$$

$$OR(x, y) = \begin{cases} 1 & \text{if either } x \text{ or } y \text{ (or both) is } 1 \\ 0 & \text{otherwise} \end{cases}$$

$$XOR(x, y) = \begin{cases} 1 & \text{if } x \text{ and } y \text{ are different} \\ 0 & \text{otherwise} \end{cases}$$

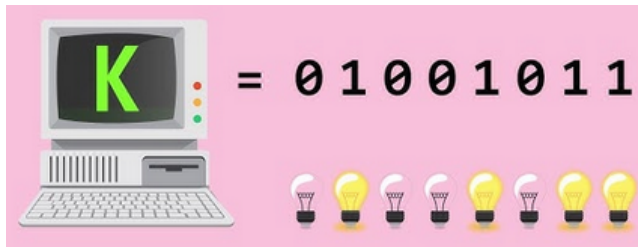
Bit Shift & Rotate

- A bit **shift** and **rotate** are operations where a **succession of bits is moved** either to the left or the right.



- 1 Number Systems
- 2 Binary Number Systems
- 3 Representing Data**
- 4 Catastrophic Examples

- **Data** refers to the symbols that represent people, events, things, ideas and so on.
- **Digitization** is the process of converting information, such as text, numbers, images, audio, and others into **digital data**.
- **Data representation** refers to the form in which data is stored, processed, and transmitted.
- Computers represent all **data as binary codes**: positive and negative numbers, fractional numbers, text, colors, images, audio, and video, memory addresses, and so on.



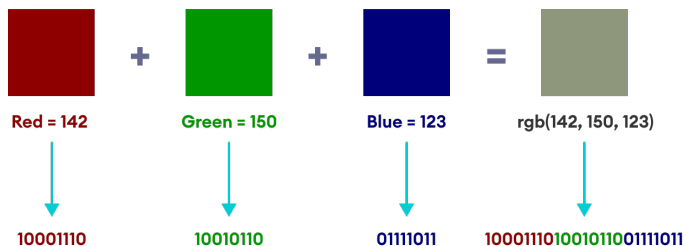
Representing Text

- **Character data** is composed of letters, symbols, and numerals that are not used in calculations.
- In order to represent text, **each individual letter or character** must be represented by a **unique binary pattern**.
 - The bit-pattern used for each character becomes a **numeric character code**.
- For computers to be able **to communicate and exchange text** between each other efficiently, they must have an agreed standard that defines which character code is used for which character.
 - A standardised collection of characters and the bit-patterns used to represent them is called a **character set**.

- **American Standard Code for Information Interchange (ASCII)** characters set is one of the most popular character sets.
 - It uses 7-bit codes to represent each character.
 - Concerning to letters, it only codes english letters.
- **Unicode** is a character set standard that defines all the characters needed for writing the majority of living languages in use on computers.
 - UTF-8 uses 1 byte to represent characters in the ASCII set, two bytes for characters in several more alphabetic blocks, and three bytes for the rest of the Basic Multilingual Plane (BMP). Supplementary characters use 4 bytes.
 - The first 65,536 code positions in the Unicode character set are said to constitute the Basic Multilingual Plane (BMP).
 - Character codes beyond BMPs are referred to as supplementary characters.
 - UTF-16 uses 2 bytes for any character in the BMP, and 4 bytes for supplementary characters.
 - UTF-32 uses 4 bytes for all characters.

Representing Colors

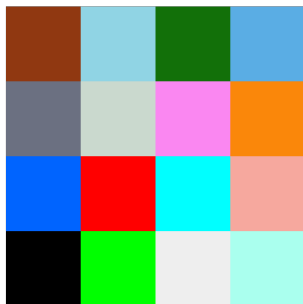
- Any computing device that has a color graphics display needs to have some **model for describing colors**.
- The most used is the RGB color model.
 - RGB uses three 8-bit codes (24 bits) to represent the intensity of Red, Green, and Blue that combine to make a single color.



Representing Images

- To represent an image, **one method is to store it as if it were a grid of colored squares.**
 - A bitmapped graphic (also called a bitmap image) is made up of a grid of pixels.
 - A pixel (short for 'picture element'), a square, is the smallest element in an image: a single color.

Original Image



RGB Codes of Pixels

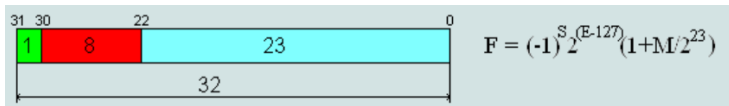
R-144 G-56 B-17	R-144 G-212 B-228	R-18 G-112 B-9	R-90 G-173 B-228
R-107 G-112 B-129	R-202 G-217 B-206	R-250 G-135 B-241	R-250 G-135 B-10
R-0 G-100 B-255	R-255 G-0 B-0	R-0 G-255 B-255	R-246 G-168 B-158
R-0 G-0 B-0	R-0 G-255 B-0	R-238 G-238 B-238	R-170 G-255 B-238

Representing Real Numbers (Floating-point numbers)

- It is represented using a fixed size bit pattern according to a specific format.
 - IEEE 754 Floating-Point format

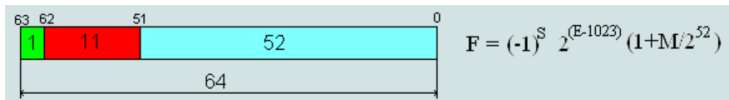


- Single precision (32 bits)



+105.625 = 01000010110100110100000000000000

- Double precision (64 bits)



Representing Machine Code

C

```
#include<stdio.h>
int main(){
    printf("Hello world!\n");
    return 0;
}
```

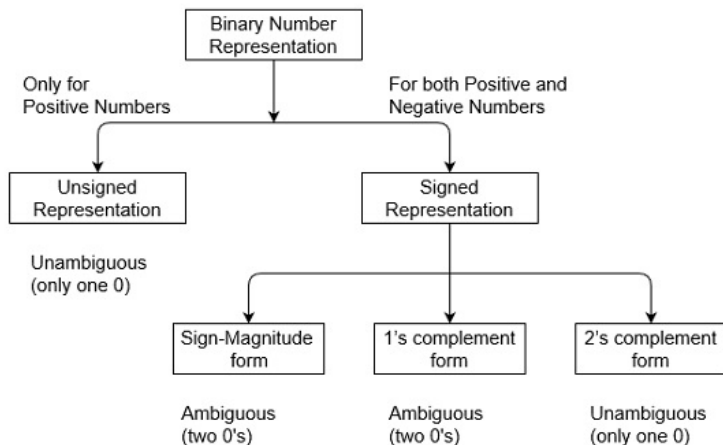
Binary

```
...
00000001 00000000 00000010 00000000 01001000 01100101
01101100 01101100 01101111 00100000 01110111 01101111
01110010 01101100 01100100 00100001 00000000 00000000
...
11111111 01101111 11101100 00000001 00000000 00000000
00000101 00000000 00000000 00000000 10001100 00000010
00000000 00000000 00000110 00000000 00000000 00000000
00001100 00000010 00000000 00000000 00001010 00000000
00000000 00000000 01111101 00000000 00000000 00000000
00001011 00000000 00000000 00000000 00010000 00000000
00000000 00000000 00010101 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000011 00000000
00000000 00000000 11011100 00011111 00000000 00000000
00000010 00000000 00000000 00000000 00011000 00000000
...
```

Assembly

```
.section .rodata
    .align 2
    .LC0:
        .string "Hello world!"
.text
    .align 2
    .globl main
main:
    addi sp,sp,-16
    sw ra,12(sp)
    sw s0,8(sp)
    addi s0,sp,16
    lla a0,.LC0
    call puts@plt
    li a5,0
    mv a0,a5
    lw ra,12(sp)
    lw s0,8(sp)
    addi sp,sp,16
    jr ra
```

Representing Integer Numbers



Representing Unsigned Integer Numbers

- Unsigned integers are a type of integer data type that can only **represent non-negative** whole numbers (positive numbers and zero).
- All **bits are used to represent the magnitude** of the integer value.
 - The range comprises values between 0 to $2^n - 1$, where n is the number of bits used to represent the integer.

8-bit unsigned integer

00000000 = 0

00000001 = $2^0 = 1$

00000101 = $2^0 + 2^2 = 5$

11111111 = $2^0 + 2^1 + \dots + 2^7 = 255$

16-bit unsigned integer

1111111111111111 = $2^0 + 2^1 + \dots + 2^{15} = 65535$

1000000000000000 = $2^{15} = 32768$

0000111111111111 = $2^0 + 2^1 + \dots + 2^{11} = 4095$

- **Memory addresses are unsigned numbers.**

Representing Signed Integer Numbers: Sign-Magnitude

- The Most Significant Bit (MSB) of a signed binary number holds **the sign** (positive, zero, or negative, 1, notation) and **the rest of the bits represent the magnitude** or value of the signed binary number.

$$+34 = \overset{\text{MSB} \leftarrow \text{Magnitude} \rightarrow}{0} 0 1 0 0 0 1 0$$

$$-34 = 1 0 1 0 0 0 1 0$$

- Since the magnitude of both numbers is the same, the first 7 bits in the representation are the same for both numbers.
 - For +34, the MSB is 0, and for -34, the MSB or sign bit is 1.
- The range comprises values between $-(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$, where n is the number of bits used to represent the integer.
- But, **there are two different representations for 0.**

$$+0 = 0 0 0 0 0 0 0 0$$

$$-0 = 1 0 0 0 0 0 0 0$$

Representing Signed Integer Numbers: 1's Complement

- The positive values are the **same as shown for Sign-Magnitude** positive values.
- The negative values are **one's complement that is inverted or negated**.

$$+34 = \begin{array}{ccccccc} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{array}$$

$$-34 = \begin{array}{ccccccc} 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{array}$$

- The range comprises values between $-(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$, where n is the number of bits used to represent the integer.
- But, **there are two different representations for 0**.

$$+0 = \begin{array}{ccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$-0 = \begin{array}{ccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}$$

Representing Signed Integer Numbers: 2's Complement (I)

- In 2's complement representation also, **the representation of the positive number is same as 1's complement and sign-magnitude form.**

$$+60 = 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0 \quad (\text{Sign Magnitude Representation})$$

$$+60 = 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0 \quad (1's\ Complement)$$

$$+60 = 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0 \quad (2's\ Complement)$$

- But the representation of the negative number is different.

$$-60 = 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0 \quad (\text{Sign Magnitude Representation})$$

$$-60 = 1\ 1\ 0\ 0\ 0\ 0\ 1\ 1 \quad (1's\ Complement)$$

$$-60 = 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0 \quad (2's\ Complement)$$

Representing Signed Integer Numbers: 2's Complement (II)

- A negative number in 2's complement is 1's complemented plus (+) "1" to the Least Significant Bit (LSB).

$$\begin{array}{r} \text{LSB} \\ + 34 = 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \\ \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \quad (\text{1's complement of } + 34) \\ + \qquad \qquad \qquad 1 \\ \hline - 34 = 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \quad (\text{2's complement of } + 34) \end{array}$$

- The range comprises values between $-(2^{n-1})$ to $+(2^{n-1} - 1)$, where n is the number of bits used to represent the integer.

Representing Signed Integer Numbers

Number	Sign magnitude	1's complement	2's complement
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
0	0000	0000	0000
(-0)	(1000)	(1111)	
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8			1000

Representing Data: Interpreting

- Binary codes **are ambiguous**, to get information from them an **use context is required**.

011000010110001001100011

abc

as ASCII

6,382,179

as a 32-bit integer



as an RGB color

- 01010101₍₂₎
 - As an ASCII character, it would be "U".
 - As a 2's complement number, it would be **85** (decimal).
 - As a machine code, it would be **push %ebp** instruction.

- 1 Number Systems
- 2 Binary Number Systems
- 3 Representing Data
- 4 Catastrophic Examples**

The Patriot Missile Failure (I)

- On February 25, 1991, during the Gulf War, an American Patriot Missile battery in Dharan, Saudi Arabia, failed to track and intercept an incoming Iraqi Scud missile.
- The Scud struck an American Army barracks, killing 28 soldiers and injuring around 100 other people.
- The report finds that **the failure to track the Scud missile was caused by a precision problem in the software.**



The Patriot Missile Failure (II)

- The computer used to control the Patriot missile is based on a 1970s design and uses 24-bit arithmetic.
- The Patriot system tracks its target by measuring the time it takes for radar pulses to bounce back from them.
- Time is recorded by the system clock in tenths of a second, but is stored as an integer then it was multiplied by $1/10$ to produce the time in seconds.
- **The number $1/10$ is a number that has no exact binary representation**, so every multiplication by $1/10$ necessarily causes some rounding error.
- The accumulated rounding error was sufficient to cause it from intercepting the incoming Scud missile **due to bad computer arithmetic**.
 - It had been in continuous operation for over 100 hours.

The Explosion of the Ariane 5

- On June 4, 1996 an unmanned Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after its lift-off from Kourou, French Guiana.
- A board of inquiry investigated the causes of the explosion and turned out that the cause of the failure was a **software error** in the inertial reference system.
- The **error was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value.**
- The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer.

