Sistemas Informáticos  
Unidad 02. Representación de la información

short line

Autores: Sergi García, Alfredo Oltra

Actualizado Septiembre 2025

Licencia

**Reconocimiento - No comercial - CompartirIgual** (BY-NC-SA): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se ha de hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán diferentes símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

📖 **Importante**

⚠️ **Atención**

💬 **Interesante**

**Índice**

[**1. Introducción 4**](#_dbh0n1vac4c8)

[1.1. Dato, pieza de información e información 4](#_bgvdmdd2qe46)

[1.2 Representación interna de los datos 4](#_gtbzwipk0t9b)

[1.3. El verdadero “conocimiento” de un ordenador 5](#_cz4ds4vnc79c)

[**2. Sistemas de numeración 5**](#_wz0jxvkvrah8)

[2.1. Descomposición de números en un sistema de numeración 5](#_4i7jqtvfxndi)

[2.2. Generalización a cualquier sistema de base B 5](#_wri44bqgnn7d)

[2.3. Código binario 6](#_50mmc223wnrq)

[2.4. Máximo valor representable con n bits 9](#_fsj6v77bub37)

[2.5. Operaciones con números binarios 9](#_v0scsfqpi1q)

[2.6. Representación de números negativos en binario 11](#_akjxv4pz9lk0)

[2.6.4. Resumen práctico 14](#_sotr8asnj2qg)

[2.7. Números reales en los ordenadores 15](#_myewc0sj88z0)

[**3. Álgebra booleana 17**](#_qrcz6wul1w0t)

[3.1. Operación NOT (negación) 17](#_l96u5yeqfaws)

[3.2. Operación AND (conjunción lógica) 17](#_uj8mdsxlak7d)

[3.3. Operación OR (disyunción lógica) 18](#_jtuoceya95dc)

[3.4. Operación XOR (disyunción exclusiva) 18](#_anjy4w7dhwq8)

[3.5. Propiedades del álgebra booleana 19](#_5zan576me3mh)

[**4. Sistemas octal y hexadecimal 19**](#_ouay4wjht5vs)

[4.1. Sistema octal 19](#_cug8dmy7t6gk)

[4.2. Conversión de binario a octal 20](#_4xl4pefhxmes)

[4.3. Conversión de octal a binario 20](#_6dp2ceuwuuwd)

[4.4. Sistema hexadecimal 20](#_offgayiaulnl)

[4.5. Conversión de binario a hexadecimal 21](#_ql1drfcure7)

[**5. Representación alfanumérica 22**](#_ms0q8wcbkshm)

[5.1. Datos numéricos y alfanuméricos 22](#_821l9oi6fj6i)

[5.2. Representación interna 23](#_wraicz83z0aa)

[**6. Sistema de unidades 23**](#_3onygjbyuwo2)

[**7. Bibliografía 24**](#_qxi5za5vk4zq)

Unidad 02. Representación de la información

# 1. Introducción

## 1.1. Dato, pieza de información e información

Los ordenadores (o, de manera más precisa, los sistemas de información) son máquinas diseñadas para **procesar información**, es decir, obtener resultados a partir de la aplicación de operaciones sobre un conjunto de datos.

Pero antes de continuar, conviene hacerse unas preguntas clave:

* ¿Qué entendemos por **información**?
* ¿Qué es exactamente una **pieza de información**?
* ¿Y qué significa **operación** en este contexto?

Veámoslo con un ejemplo sencillo:

* **Dato o pieza de información**: una representación formal y objetiva de un concepto. Por ejemplo: “30”.
* **Información**: el resultado de interpretar ese dato en un contexto. Ejemplo: “Hace calor”.
* **Operación**: la regla o el criterio que transforma el dato en información. Ejemplo: “Si la temperatura es superior a 23 ºC, entonces hace calor”.

👉 Conclusión: **el dato por sí solo carece de sentido**. Solo cuando lo interpretamos aplicando una regla, se convierte en información útil.

## 1.2 Representación interna de los datos

Si queremos que un ordenador trabaje con información, necesita almacenar y manipular **datos** y también las **operaciones** que permiten interpretarlos. Para lograrlo, utiliza un sistema universal: **el código binario**.

⚠️ **Atención:** todo tipo de datos, ya sean números, letras, imágenes o sonidos, se representan internamente con este sistema.

El sistema binario se basa en el uso exclusivo de dos dígitos: **0 y 1**, a diferencia del sistema decimal que emplea diez (0–9).  
Esto se debe a que los ordenadores no “piensan” en números como nosotros, sino que trabajan con señales eléctricas que solo pueden asumir **dos estados posibles**:

* **0** → cuando la tensión o potencial eléctrico está próxima a 0 voltios.
* **1** → cuando la tensión supera un umbral, normalmente alrededor de **3 o 5 voltios**.

💬 **Interesante**: en términos eléctricos, el “potencial” puede entenderse como la fuerza con la que circula la corriente a través de un conductor.

💬 **Interesante**: aunque los valores concretos varían según la tecnología, la idea es siempre la misma: un estado de “bajo” (0) y un estado de “alto” (1).

## 1.3. El verdadero “conocimiento” de un ordenador

Todos los componentes de un ordenador, desde la memoria hasta el procesador, utilizan este sistema binario para representar y procesar la información.  
 Por eso, puede afirmarse que en realidad **los ordenadores no “saben” nada**:

* Solo “entienden” 0 y 1.
* Solo saben aplicar operaciones básicas con ellos (sumar, restar, comparar…).
* La diferencia está en que lo hacen a una **velocidad extremadamente alta**, lo que les permite resolver problemas complejos y simular procesos del mundo real.

# 2. Sistemas de numeración

Un **sistema de numeración** es un conjunto ordenado de símbolos que se utilizan para representar cantidades. El número de símbolos que lo forman recibe el nombre de **base del sistema**.

En la vida cotidiana estamos acostumbrados al **sistema decimal** (base 10), cuyos símbolos son:  
 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9.

## 2.1. Descomposición de números en un sistema de numeración

Cualquier número, en cualquier sistema, puede descomponerse en **dígitos**.Por ejemplo:

* El número **128** puede descomponerse en los dígitos: 1, 2 y 8.
* El número **34,76** en los dígitos: 3, 4, 7 y 6.

A partir de estos dígitos, su **posición** y la **base** del sistema, es posible reconstruir el número original:

Un número decimal puede representarse como una suma de **potencias de base 10**.

## 2.2. Generalización a cualquier sistema de base B

De forma general, un número **N** expresado en un sistema de base **B** puede escribirse como:

Donde:

* **N**: número que se quiere representar
* **a**: cada uno de los dígitos del sistema, que siempre son enteros comprendidos entre 0 y (B – 1).
* Los dígitos **antes de la coma** representan la **parte entera**.
* Los dígitos **después de la coma** representan la **parte fraccionaria**.

💬 **Interesante**: en la cultura anglosajona, el separador entre parte entera y fraccionaria no es la coma (,) sino el **punto decimal** (.).

## 2.3. Código binario

El **código binario** es un sistema de numeración cuya base es **2**. Sus símbolos son únicamente: **0 y 1**.

📖 **Importante:** cada dígito de un número binario recibe el nombre de **bit** (*binary digit*).  
 El bit es la **unidad mínima de información**: lo más pequeño que puede representarse en un ordenador.

* 101(₁₀) significa 101 en base decimal.
* 101(₂) significa 101 en base binaria (que en decimal equivale a 5).

💬 **Interesante:** para evitar confusiones entre sistemas de numeración, es habitual indicar la base con un subíndice a la derecha:

### 2.3.1. Conversión de decimal a otra base

En general, para convertir un número decimal a otro sistema de numeración de base **B**, se sigue este procedimiento:

1. **Parte entera**:
   * Se realizan **divisiones sucesivas** del número entre la base.
   * En cada paso se guarda el **resto** de la división.
   * El proceso termina cuando el cociente es cero.
   * El número convertido se obtiene tomando los restos en **orden inverso** (del último al primero).
2. **Parte fraccionaria**:
   * Se multiplica la fracción sucesivamente por la base.
   * En cada paso se toma la **parte entera** del resultado.
   * El proceso se repite hasta obtener el nivel de precisión deseado o hasta que el resultado sea exacto.
   * En este caso, los dígitos se leen en el **orden directo** (del primero al último obtenido).

### **Ejemplo 1: conversión de número entero (45₁₀ → binario)**

1. 45 ÷ 2 = 22, resto 1
2. 22 ÷ 2 = 11, resto 0
3. 11 ÷ 2 = 5, resto 1
4. 5 ÷ 2 = 2, resto 1
5. 2 ÷ 2 = 1, resto 0
6. 1 ÷ 2 = 0, resto 1

Leyendo los restos en orden inverso: **45₁₀ = 101101₂**.

### **Ejemplo 2: conversión de número fraccionario (6,25₁₀ → binario)**

**Parte entera (6):**

1. 6 ÷ 2 = 3, resto 0
2. 3 ÷ 2 = 1, resto 1
3. 1 ÷ 2 = 0, resto 1

Parte entera en binario: **110**.

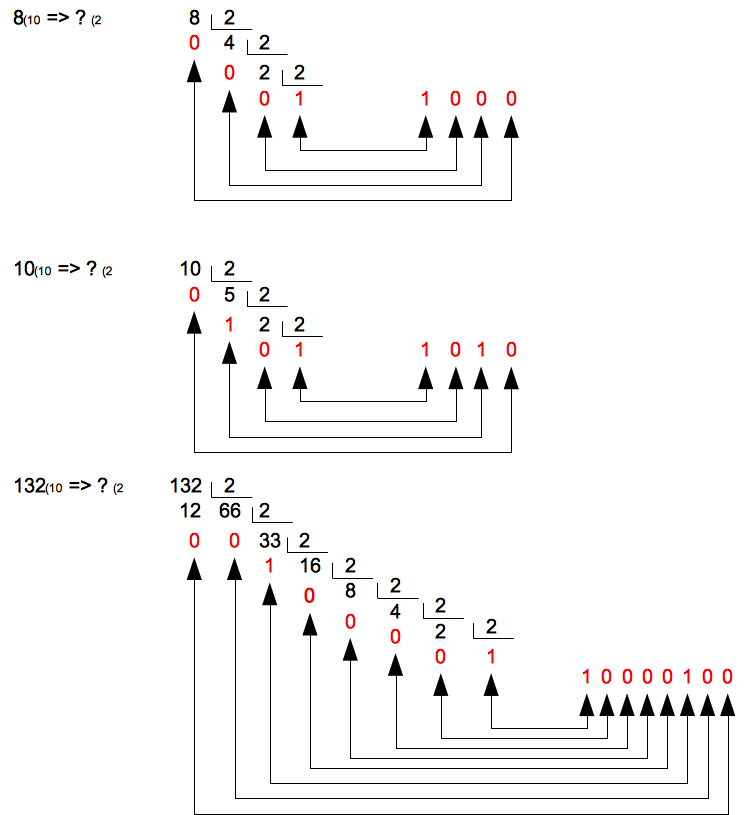
**Parte fraccionaria (0,25):**

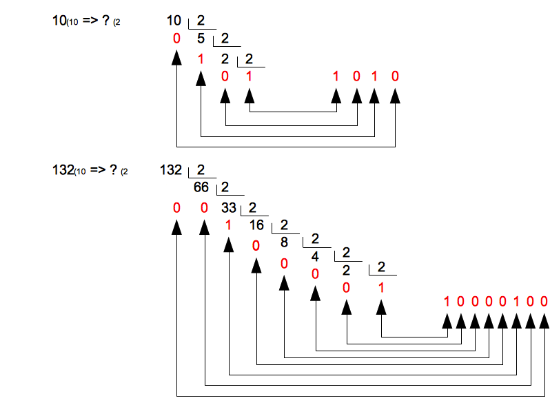
1. 0,25 × 2 = 0,50 → entero 0
2. 0,50 × 2 = 1,00 → entero 1

Parte fraccionaria en binario: **01**.

Resultado final: **6,25₁₀ = 110,01₂**.

**Vamos a verlo de forma gráfica**





📖 **Importante**:

* El **bit más a la izquierda** se denomina **bit más significativo (MSB, *Most Significant Bit*)**.
* El **bit más a la derecha** se denomina **bit menos significativo (LSB, *Least Significant Bit*)**.

### 2.3.2. Conversión de un número binario a decimal

El proceso de conversión de binario a decimal es muy sencillo.  
Como ya vimos, un número en cualquier sistema puede representarse como una **suma de potencias de la base**.

En general, para un número expresado en una base **B**, la fórmula es:

N =an-1Bn-1+an-2Bn-2+...+a1B1+a0B0+a-1B-1+...+a-pB-p = ∑ ai Bi

donde:

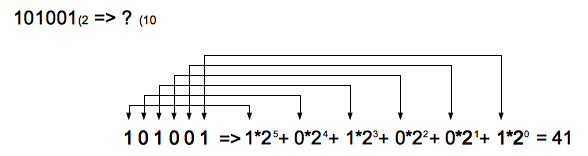
* **N** es el número representado.
* **B** es la base del sistema de numeración.
* **aᵢ** son los dígitos del número (de 0 hasta B – 1).

### 2.3.3. Pasos para convertir binario a decimal

1. Escribir las cifras del número binario.
2. Multiplicar cada cifra por **2 elevado al exponente correspondiente**.
   * El exponente empieza en 0 para el último dígito de la parte entera (a la derecha).
   * Hacia la izquierda aumenta de 1 en 1.
   * Hacia la derecha (parte fraccionaria) disminuye de 1 en 1.
3. Colocar un **signo +** entre cada producto.
4. Realizar la suma total.

### **Ejemplo 1: número entero**

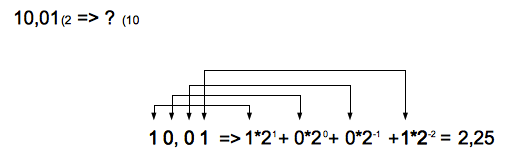
Convertir **101001₂** a decimal:



Por tanto: **101001₂ = 41₁₀**.

### **Ejemplo 2: número fraccionario**

Convertir **10,01₂** a decimal:



Por tanto: **10,01₂ = 2,25₁₀**.

⚠️ **Atención:** como veremos más adelante, B puede ser cualquier sistema de numeración.

## 2.4. Máximo valor representable con n bits

Un número binario de **n bits** puede representar exactamente **2ⁿ valores diferentes**.

Ejemplo:

* Con **4 bits** → 2⁴ = 16 valores.
* El rango es de **0** (0000₂) a **15** (1111₂).

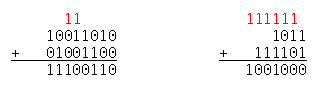
## 2.5. Operaciones con números binarios

### 2.5.1. Suma

Reglas básicas:

* 0 + 0 = 0
* 1 + 0 = 1
* 0 + 1 = 1
* 1 + 1 = 0 (con **acarreo** 1 a la siguiente posición)

Ejemplo:



⚠️ **Atención:** si el resultado excede el número de bits disponibles, se produce un **overflow (desbordamiento)**. Por ejemplo, en un sistema de 8 bits (máximo 255₁₀):

Por ejemplo, si queremos sumar () más (), tenemos un problema porque el resultado es () que es mayor que 255, ocurriendo desbordamiento al no caber en 8 bits.

## 

## 

### 2.5.2. Resta

Reglas básicas:

* 0 – 0 = 0
* 1 – 0 = 1
* 0 – 1 = 1 (con préstamo de 1 a la siguiente columna)
* 1 – 1 = 0

Ejemplo:



⚠️ **Atención:**  el préstamo no se añade al minuendo, sino al **sustraendo**.

### 2.5.3. Multiplicación

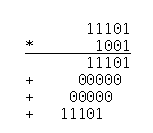
Reglas básicas:

* 0 × 0 = 0
* 1 × 0 = 0
* 0 × 1 = 0
* 1 × 1 = 1

Ejemplo:

1012×112101₂ \* 11₂1012​×112​

Se resuelve igual que en decimal, pero aplicando las reglas binarias:

****

Resultado: **11101 × 1001 = 100000101₂ = 261₁₀**.

⚠️ **Atención:** si en una columna aparecen más de dos unos, se van sumando en grupos de dos y generando acarreos.

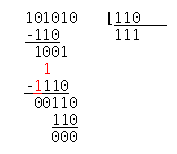
## 

### 2.5.4. División

Reglas básicas:

* 0 ÷ 1 = 0
* 1 ÷ 1 = 1
* 1 ÷ 0 = no definido (error).
* 0 ÷ 0 = indeterminado.

En binario:



⚠️ **Atención:** en la división binaria, se comienza comparando divisor y dividendo con el mismo número de cifras, y se va desplazando hasta poder “contener” el divisor dentro del dividendo.

## 2.6. Representación de números negativos en binario

Cuando necesitamos representar números negativos en binario, existen varias formas de hacerlo. Las más importantes son **cuatro**:

1. **Signo y magnitud.**
2. **Complemento a uno.**
3. **Complemento a dos.**
4. **Exceso-K.**

👉 La elección del método es una **convención** entre quién genera el número y quién lo interpreta.  
 Si no existe acuerdo, el valor real podría malinterpretarse.

### 2.6.1. Signo y magnitud

Es el método más sencillo de comprender.

* El **bit más significativo (MSB)** indica el signo:
  + 0 → positivo
  + 1 → negativo
* Los demás bits representan el valor absoluto del número.

Ejemplo con 4 bits:

| **Decimal** | **Binario puro** | **Positivo (signo-magnitud)** | **Negativo (signo-magnitud)** |
| --- | --- | --- | --- |
| +5 | 101 | 0101 | 1101 (–5) |

👉 Con 4 bits:

* En binario normal, el rango es [0, 15].
* En signo-magnitud, se dedica un bit al signo, por lo que el rango es de –7 a +7.

⚠️ **Atención:** existen dos formas de representar el 0 → 0000 (+0) y 1000 (–0). Esto complica operaciones aritméticas.

### 2.6.2. Complemento a uno

Este método también usa el **primer bit como signo**, pero los negativos se obtienen **invirtiendo todos los bits** del número positivo (cambiando 0 → 1 y 1 → 0).

Ejemplo con 4 bits:

| **Decimal** | **Binario puro** | **Positivo (comp. a 1)** | **Negativo (comp. a 1)** |
| --- | --- | --- | --- |
| +5 | 101 | 0101 | 1010 (–5) |

Ejemplo con 8 bits:

* +5 → 00000101
* –5 → invertir bits → 11111010

⚠️ **Atención:** también existen dos ceros posibles (0000 y 1111).

### 2.6.3. Complemento a dos

Es el sistema más utilizado en la práctica, porque **simplifica al máximo las operaciones aritméticas**.

El proceso es:

1. Tomar el número positivo en binario.
2. Aplicar el complemento a uno (invertir bits).
3. Sumar 1 al resultado.

**Ejemplo (–5 en 8 bits):**

1. +5 → 00000101
2. Complemento a 1 → 11111010
3. +1 → 11111011

Por tanto:

* +5 = 00000101
* –5 = 11111011

**Cómo recuperar el valor decimal en complemento a dos**

Ejemplo: 11111011₂

1. Detectamos que es negativo (MSB = 1).
2. Aplicamos complemento a 1 → 00000100.
3. Sumamos 1 → 00000101.
4. Resultado = –5.

**Ventaja clave**

En complemento a dos, la **resta** puede realizarse como una **suma con el negativo**.

Ejemplo:

Por ejemplo, restar **45₁₀ - 21₁₀,**  se suma **45₁₀ al número - 21₁₀ y da 24₁₀**

En binario (6 bits):

* 45 = 101101
* 21 = 010101

Obtenemos –21:

* 010101 → complemento a 1 = 101010 → +1 = 101011.

Ahora sumamos:

101101**₂**+101011**₂**=011000=​**24₁₀**

### 2.6.4. Exceso-K

Este método reparte el rango entre negativos y positivos desplazando el **cero al centro del rango**.

* Se define un valor de **exceso K**: K = 2n-1.
* El rango resultante será: [–K, K–1].

💬 **Interesante:** hay otra versión de este método con K = (2n-1 - 1).

Ejemplo con 3 bits (n = 3):

* Se pueden representar 2³ = 8 valores.
* K = 2² = 4.
* El rango será [–4, 3].

| **Decimal** | **Binario** |
| --- | --- |
| –4 | 000 |
| –3 | 001 |
| –2 | 010 |
| –1 | 011 |
| 0 | 100 |
| 1 | 101 |
| 2 | 110 |
| 3 | 111 |

**Ejemplo con 8 bits (n = 8):**

* K = 2⁷ = 128.
* Rango = [–128, 127].

**Conversión:**

* 11001100₂ = 204₁₀ → 204 – 128 = 76
* 00111100₂ = 60₁₀ → 60 – 128 = –68

## 2.6.4. Resumen práctico

* **Signo-magnitud**: simple, pero con dos ceros y operaciones complicadas.
* **Complemento a uno**: mejora, pero sigue teniendo dos ceros.
* **Complemento a dos**: el estándar actual (suma y resta unificadas).
* **Exceso-K**: muy usado en hardware (representación de exponentes en punto flotante, como en IEEE 754).

## 2.7. Números reales en los ordenadores

Cuando escribimos un número real en papel, utilizamos una **coma decimal** (o punto decimal, según la cultura) para separar la **parte entera** de la **parte fraccionaria**. En un ordenador, el espacio disponible para representarlo se divide en **dos campos**: uno para la parte entera y otro para la parte fraccionaria. Existen dos métodos principales para fijar el tamaño de estos campos y, por tanto, la posición de la coma:

* **Coma fijo.**
* **Coma flotante.**

### 2.7.1. Representación en coma fijo

En esta notación, se reserva un número **fijo de bits** para la parte entera y otro número fijo para la parte fraccionaria. Es decir, la posición de la coma no varía.

✅ **Ventaja**: Las operaciones básicas (suma, resta, multiplicación) se realizan igual que con números enteros.  
 ❌ **Inconveniente**: No aprovecha bien la capacidad de representación. El rango de valores es muy limitado.

📌 **Ejemplo**:  
 En un ordenador de **8 bits**, podemos usar:

* 5 bits para la parte entera
* 3 bits para la parte fraccionaria

**Formato: b7 b6 b5 b4 b3 , b2 b1 b0**

* Máximo representable: 01111,111₂ = 15,875₁₀
* Mínimo positivo: 00000,001₂ = 0,125₁₀

Si en lugar de fijar la coma, esta fuera “flotante”, podríamos representar:

* Desde 011111111₂ = 255
* Hasta 0,0000001₂ ≈ 0,0078

### 2.7.2. Representación en coma flotante

El rango de valores en **punto fijo** es insuficiente para aplicaciones científicas, donde aparecen números **muy grandes** o **muy pequeños**. Por eso se usa el **punto flotante**, inspirado en la **notación científica**:

📌 **Ejemplo en decimal**:  
 0,00000025 = 2,5\*

En general, cualquier número real puede expresarse como:

*N = M \* BE or N = (M;B;E)*

donde:

* **M**: mantisa (los dígitos significativos del número)
* **B**: base (por lo general 2 en informática)
* **E**: exponente

**📌 Ejemplo en binario:**

Por ejemplo en decimal (B=10) = 0,25975\* o (0,25975;10;3) o, en código binario

*→ → 0,10000001111 \* → 0,10000001111 \* → (0,10000001111;1001)*

### 2.7.3. Normalización

Un mismo número puede expresarse de infinitas formas en notación científica.  
 Ejemplo:

*2,5 representado en una forma normalizada es 0,25\**

*(0,000011101 ; 2 ; 0111) → (0,11101 ; 0011) → Exponente en exceso-k (0,11101 ; 1011)*

Para **evitar ambigüedad**, se elige la **forma normalizada**, que asegura **máxima precisión**.

👉 En binario, esto significa que el **punto binario** se coloca inmediatamente a la izquierda del primer dígito significativo (el primer 1).

**Más ejemplos:**

*(100,11110 ; 2 ; 0010) → (0,10011110 ; 2 ; 0101) → Exponente exceso-k (0,10011110 ; 1101)*

*(101,001 ; 2 ; 0100) → (0,1010010 ; 2 ; 0111) → Exponente exceso-k (0,10011110 ; 1111)*

### 2.7.4. Estándar IEEE 754

El formato más usado en informática para representar números en **punto flotante binario** es el estándar **IEEE 754**, creado por el **Institute of Electrical and Electronics Engineers**.

Además de números normales, este formato define:

* +∞ y –∞
* NaN (Not a Number) para resultados indefinidos

**📌 Tres formatos principales:**

1. **Half precision (precisión reducida)** – 16 bits
   * 1 bit: signo
   * 5 bits: exponente
   * 10 bits: mantisa

****

1. **Single precision (precisión simple)** – 32 bits
   * 1 bit: signo
   * 8 bits: exponente
   * 23 bits: mantisa

****

1. **Double precision (precisión doble)** – 64 bits
   * 1 bit: signo
   * 11 bits: exponente
   * 52 bits: mantisa

****

**📖 Detalles importantes:**

* En **mantisa normalizada**, el primer dígito significativo siempre es **1** en binario.
  + 👉 Este **1 no se almacena**, solo los bits a la derecha de la coma. Esto ahorra espacio y da **un bit extra de precisión**.
* El **exponente** se guarda en **exceso-K, donde** K = (2n-1 - 1)  
   y donde n = número de bits del exponente.  
   Ejemplo:
  + Con 8 bits → K=127.
  + Con 11 bits → K=1023.

# 3. Álgebra booleana

Además de las operaciones matemáticas habituales (+, −, ×, ÷) que se pueden aplicar a números binarios, también existen operaciones **booleanas o lógicas**: **AND, OR, XOR, NOT**, entre otras.

📖 **Importante:** para comprender mejor estas operaciones, es útil pensar en el **1 como “verdadero” (true)** y el **0 como “falso” (false)**.

## 3.1. Operación NOT (negación)

* Se representa de varias formas: NOT, ¬, !.
* Su función es invertir el valor lógico:
  + NOT 0 = 1
  + NOT 1 = 0

👉 Es decir, cambia un valor verdadero en falso y viceversa.

## 3.2. Operación AND (conjunción lógica)

* Se representa como AND, Y, ^, ·.
* Solo devuelve **1 (verdadero)** si **los dos operandos son 1**.
* Tabla de verdad:

| **A** | **B** | **A AND B** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

👉 Equivale a la **multiplicación** en aritmética (1×1=1, todo lo demás da 0).

## 3.3. Operación OR (disyunción lógica)

* Se representa como OR, O, v, +.
* Devuelve **1 (verdadero)** si **al menos uno de los operandos es 1**.
* Tabla de verdad:

| **A** | **B** | **A OR B** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

👉 Es suficiente con que uno de los valores sea verdadero para que el resultado sea verdadero.

## 3.4. Operación XOR (disyunción exclusiva)

* Se representa como XOR, ⊕.
* Devuelve **1 (verdadero)** únicamente cuando **los dos operandos son distintos**.
* Tabla de verdad:

| **A** | **B** | **A XOR B** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

👉 Es la operación del “o bien… o bien… pero no ambos”.

## 3.5. Propiedades del álgebra booleana

Estas operaciones cumplen reglas similares a la aritmética, pero adaptadas al sistema binario. Algunas propiedades importantes:

* **Conmutatividad**:  
   A AND B = B AND A  
   A OR B = B OR A
* **Asociatividad**:  
   (A AND B) AND C = A AND (B AND C)  
   (A OR B) OR C = A OR (B OR C)
* **Distributividad**:  
   A AND (B OR C) = (A AND B) OR (A AND C)
* **Identidad**:  
   A AND 1 = A  
   A OR 0 = A
* **Complemento**:  
   A AND NOT A = 0  
   A OR NOT A = 1

# 4. Sistemas octal y hexadecimal

Además del sistema binario, existen otros dos sistemas numéricos muy utilizados en informática: **el octal** y **el hexadecimal**. La razón es que permiten realizar conversiones muy fáciles hacia y desde el binario, ya que sus bases son potencias exactas de 2.

## 4.1. Sistema octal

El **sistema octal** tiene base **8**, lo que significa que utiliza los símbolos: **0, 1, 2, 3, 4, 5, 6, 7**. Su base es una potencia del sistema binario: es decir, con **3 bits** (3 dígitos binarios) se puede representar exactamente un dígito octal.

| **Decimal** | **Binario** | **Octal** |
| --- | --- | --- |
| 0 | 000 | 0 |
| 1 | 001 | 1 |
| 2 | 010 | 2 |
| 3 | 011 | 3 |
| 4 | 100 | 4 |
| 5 | 101 | 5 |
| 6 | 110 | 6 |
| 7 | 111 | 7 |

## 4.2. Conversión de binario a octal

Se agrupan los dígitos binarios en bloques de **3 bits**, comenzando por la derecha, y se sustituyen por el valor octal correspondiente.

Ejemplo:

*=> Se hacen grupos de tres elementos 1 101 011 =>*

## 4.3. Conversión de octal a binario

El proceso inverso consiste en convertir **cada dígito octal en su equivalente binario de 3 bits**.

Ejemplo:

*=> Se hacen grupos de tres elementos 111 100 000 010 =>*

## 4.4. Sistema hexadecimal

El **sistema hexadecimal** tiene base **16**.Como se necesitan más de diez símbolos, se utilizan las letras de la A a la F para representar los valores del 10 al 15.

El conjunto de símbolos es: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F**.

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

## 4.5. Conversión de binario a hexadecimal

Se agrupan los dígitos binarios en bloques de **4 bits**, comenzando por la derecha.

Ejemplo:

*=> Se hacen grupos de 4 elementos 110 1011 =>*

### **Conversión de hexadecimal a binario**

Cada dígito hexadecimal se convierte en su valor binario de **4 bits**.

Ejemplo:

*=> Se hacen grupos de 4 elementos 0111 1111 0000 1010 =*

### **Conversión de hexadecimal a octal**

Se suele pasar primero por binario:

**Ejemplo:**

*=> Se hacen grupos de 4 elementos 110 1011 => =>*

*=> Se hacen grupos de 3 elementos 1 101 011 =>*

**Nota importante 📖**

Las conversiones entre **octal/hexadecimal y decimal** (o viceversa) también se pueden hacer usando los métodos binario-decimal habituales, pero adaptados:

* Multiplicando cada dígito por una potencia de **8** (en octal) o **16** (en hexadecimal).
* O bien dividiendo sucesivamente entre 8 o 16, tomando los restos.

En el caso del hexadecimal, si el resto es mayor que 9, se sustituye por las letras **A–F**.

👉 Sin embargo, en la práctica suele ser **más rápido y sencillo convertir primero a binario** y desde ahí pasar al sistema deseado.

# 5. Representación alfanumérica

## 5.1. Datos numéricos y alfanuméricos

Un dato se considera **numérico** cuando es posible realizar operaciones matemáticas con él. Por el contrario, un dato es **alfanumérico** cuando **no se pueden realizar operaciones matemáticas**:

* Numérico: ¿Cuántos años tienes? → 45
* Alfanumérico: ¿Cómo te llamas? → "Roberto"

Para diferenciar claramente los datos alfanuméricos de los numéricos, es habitual representarlos entre **comillas simples o dobles**.

Un error común es pensar que los datos numéricos son solo números y que los alfanuméricos son únicamente letras. Esto no es correcto. Veamos ejemplos:

* ¿Cuál es tu dirección? → "Avenida de las Palmeras, 34"
* ¿Cuál es tu número de móvil? → "555341273"

En el primer caso, la información contiene **letras y números**. En el segundo caso, aunque solo hay cifras, **no puede considerarse un número** porque no tiene sentido sumar o multiplicar números de teléfono. Por tanto, ambos ejemplos son **cadenas de caracteres** (strings).

## 5.2. Representación interna

Los ordenadores necesitan una forma estándar de representar caracteres alfanuméricos. Para ello utilizan **tablas de codificación**, donde cada número entero corresponde a un carácter.

Uno de los estándares más antiguos y conocidos es la **tabla ASCII (American Standard Code for Information Interchange)**.

* Utiliza 7 bits, por lo que puede representar 2⁷ = 128 caracteres.
* Entre ellos se incluyen letras mayúsculas y minúsculas del alfabeto inglés, números, signos de puntuación y algunos caracteres de control (no visibles), como el tabulador (código 9) o el retorno de carro (código 13).
* Incluso el espacio " " es un carácter (código 32).

Ejemplos:

* 73 (10) → "I"
* 105 (10) → "i"
* 50 (10) → "2"

El problema del **ASCII original** es que no incluye caracteres propios de otros idiomas, como la **ñ**, la **ç** o las vocales acentuadas. Para solventar esto, se creó el **ASCII extendido** de **8 bits** (256 caracteres), que incorpora símbolos gráficos y variantes del alfabeto latino.

📖 **Importante:** Hoy en día, tanto ASCII como ASCII extendido son insuficientes. Con la globalización e Internet, se requieren sistemas que soporten **todos los alfabetos del mundo** (chino, árabe, ruso, hebreo, etc.). Por ello se utiliza el estándar **Unicode**, siendo **UTF-8** la codificación más extendida, capaz de representar millones de caracteres distintos de forma eficiente.

# 6. Sistema de unidades

Como ya se explicó, el **bit** es la unidad mínima de información. Sin embargo, en la práctica se trabaja con **agrupaciones de bits**:

* 8 bits = 1 byte

Y a partir de ahí se definen múltiplos:

* Kilobyte (kB)
* Megabyte (MB)
* Gigabyte (GB)
* Terabyte (TB)
* … y así sucesivamente.

Aquí surge una **doble notación**:

* En el **Sistema Internacional (SI)**, los múltiplos son potencias de 10.
  + 1 kB = 1000 bytes
* En **informática tradicional**, se han usado potencias de 2.
  + 1 KiB = 1024 bytes

Para evitar confusiones, se acordó el uso de prefijos especiales en binario (IEC 1998):

* **kB (kilobyte, decimal):** 1000 bytes
* **KiB (kibibyte, binario):** 1024 bytes
* **MB (megabyte):** 1 000 000 bytes
* **MiB (mebibyte):** 1 048 576 bytes
* … y así sucesivamente (GiB, TiB, PiB, EiB, ZiB).

📖 **Importante:** Aunque en la práctica se suele usar el **Sistema Internacional (kB, MB, GB...)**, los valores **no son exactamente los mismos** que en el sistema binario (KiB, MiB, GiB...).

Por ejemplo:

* 1 MB = 1 000 000 bytes
* 1 MiB = 1 048 576 bytes

❕ **Atención:** No confundir:

* **kB** = kilobyte
* **kb** = kilobit (8 veces más pequeño)

💬 **Interesante:** Todos los múltiplos (Mega, Giga, Tera, etc.) se escriben con inicial **mayúscula**, excepto el prefijo **kilo**, que por convención se escribe con minúscula (kB).

# 7. Bibliografía

[1] Wikipedia. Signed Number Representations

<https://en.wikipedia.org/wiki/Signed_number_representations>

[2] Wikipedia. Signed Number Representations

<https://en.wikipedia.org/wiki/Signed_number_representations>