



## UNIDAD 1.

# REPRESENTACIÓN DE LA INFORMACIÓN

Sistemas informáticos  
PRESA CFGS

Alfredo Oltra

[alfredo.oltra@ceedcv.es](mailto:alfredo.oltra@ceedcv.es)

2019/2020

Versión:220919.1509

## Licenciado



**Reconocimiento – NoComercial – 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 debe hacer con una licencia igual a la que regula la obra original.

## nomenclatura

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

-Importante

-atencion

-interesante

# ÍNDICE DE CONTENIDO

<b>1. Introducción.....</b>	<b>4</b>
1.1 Pieza de información e información .....	4
1.2 Representación interna de los datos .....	4
<b>2. Sistemas numéricos .....</b>	<b>5</b>
2.1 Código binario .....	5
2.1.1 Cómo convertir un número decimal en un número binario.....	5
2.1.2 Cómo convertir un número binario en un número decimal.....	7
2.1.3 Número máximo de valores a representar.....	8
2.1.4 Operaciones con números binarios.....	8
2.1.5 Números negativos.....	9
2.1.6 Números reales .....	11
2.1.7 Álgebra booleana.....	14
2.2 octales.....	15
2.2.1 Cómo convertir un número binario en octal.....	15
2.2.2 Cómo convertir un número octal en binario .....	dieciséis
2.3 Hexadecimales.....	dieciséis
2.3.1 Cómo convertir números binarios a hexadecimales.....	dieciséis
2.3.2 Cómo convertir números hexadecimales a binarios.....	17
2.3.3 Cómo convertir números hexadecimales en números octales.....	17
<b>3. Representación Alfanumérica.....</b>	<b>17</b>
3.1 Datos numéricos y alfanuméricos .....	17
3.2 Representación interna.....	18
<b>4. Sistema de unidades .....</b>	<b>19</b>
<b>5. Material adicional .....</b>	<b>20</b>
<b>6. Bibliografía.....</b>	<b>20</b>

## UD01. REPRESENTACIÓN DE LA INFORMACIÓN

### 1. INTRODUCCIÓN

#### 1.1 Pieza de información e información

Las computadoras (o más correctamente los sistemas de información) son máquinas diseñadas para procesar información o, en otras palabras, para obtener resultados de la aplicación de operaciones sobre un conjunto de datos. Pero, ¿qué es la información? ¿Qué es un dato? ¿Y qué es una operación?. Tome un ejemplo:

*La temperatura es de 30°*

- Pieza de información: representación formal de un concepto, en este caso: "30"
- Información: el resultado de la interpretación de los datos: "Hace calor"
- Operación: regla aplicada para obtener información: "As la temperatura es superior a 23, hace calor"

#### 1.2 Representación interna de datos

Por lo tanto, necesitamos almacenar y manejar datos y operaciones en las computadoras. Y para eso, necesitan usar el código binario.

-Todo tipo de datos, tanto números como letras, se almacenan mediante este sistema.

Este sistema se basa en el uso de solo dos dígitos, 0 y 1, a diferencia del sistema decimal que usa diez (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Esto se debe a que las computadoras solo conocen estos dos valores numéricos resultantes de la detección o no de algún potencial, de una cantidad de voltios. Así, una computadora sabe que hay un 0 cuando el potencial medido en un miembro interno tiene un valor cercano a 0 voltios. De lo contrario, detecta un 1.

-En términos eléctricos, el potencial podría asimilarse a la fuerza con la que pasa la corriente eléctrica a través de un alambre.

-En general, los valores de 1 suelen corresponder a un potencial en torno a los 3 o 5 voltios.

Todos los elementos informáticos manejan este sistema de numeración e interpretación de

información. Se podría decir que las computadoras en realidad no saben nada en absoluto. Solo conocen los 0 y los 1 y cómo realizar algunas operaciones básicas con ellos (+, -, \*...), aunque más rápido.

## 2. SISTEMAS NUMERALES

Un sistema numérico es un conjunto de **símbolos ordenados** se utiliza para representar cantidades. El número de símbolos se llama **base del sistema**.

En el mundo real, estamos acostumbrados a usar el sistema decimal (base 10) cuyo conjunto de símbolos ordenados son 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Cualquier número, representado en cualquier sistema numérico, se puede dividir en dígitos. Por ejemplo, 128 se puede dividir en 1, 2, 8 o 34,76 en 3, 4, 7, 6. A partir de estos dígitos y con su posición y la base del sistema es posible obtener nuevamente el número:

$$128 = 1 * 10_2 + 2 * 10_1 + 8 * 10_0$$

$$34,76 = 3 * 10_1 + 4 * 10_0 + 7 * 10_{-1} + 6 * 10_{-2}$$

Podemos ver que un número decimal se puede representar como sumas de potencias de 10 (la base del sistema decimal).

Si generalizamos, un número **norte** expresado en un sistema numeral **B** sería como:

$$\text{norte} = \text{un}_{n-1} \text{a}_{n-2} \dots \text{a}_1 \text{a}_0, \text{a}_{-1} \text{a}_{-2} \dots \text{a}_{-p+1} \quad \text{a-pags}$$

dónde:

N: número a representar

a: los símbolos que incluye nuestro sistema numérico (enteros del 0 al B-1)

Los dígitos antes de la coma (,) son la parte entera.

Los dígitos después de la coma (,) son la parte fraccionaria.

### 2.1 Código binario

El código binario es un sistema numérico cuya base del sistema es 2 y sus símbolos son 0 y 1.

-Cada dígito de un número binario se llama **un poco** y es la unidad de información más pequeña, es decir, es la mínima que se puede representar

-Para evitar confusiones, es habitual indicar el número de sistema base a representar mediante un subíndice a la derecha. Por ejemplo  $101_{(10)}$  o  $101_{(2)}$

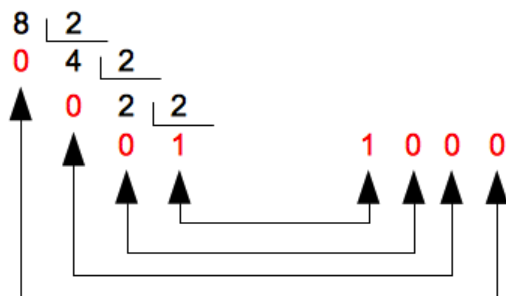
1 En la cultura inglesa, la separación entre la parte decimal y la parte fraccionaria es un punto decimal (.)

### 2.1.1 Cómo convertir un número decimal en un número binario

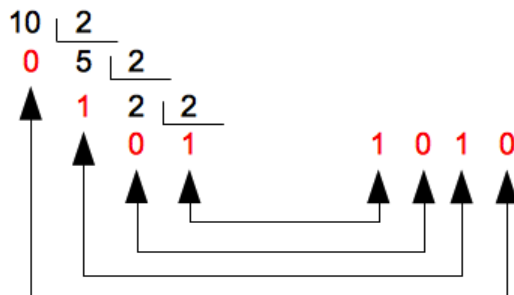
En general, para convertir un número decimal en otra base, tenemos que realizar sucesivas divisiones del número por la base. Al final, tenemos que obtener los residuos y el último cociente y ordenarlos en la dirección opuesta.

Considere el caso de convertir un decimal en binario con algunos ejemplos:

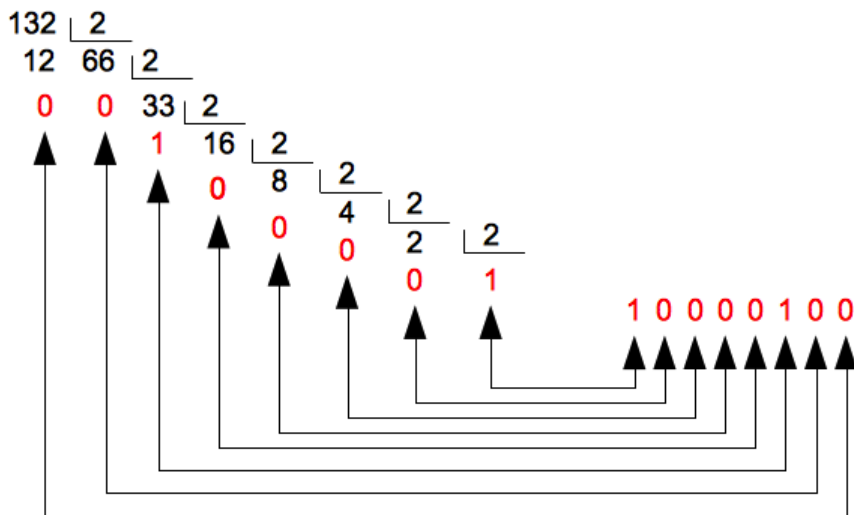
$$8_{(10)} \Rightarrow ?_{(2)}$$



$$10_{(10)} \Rightarrow ?_{(2)}$$

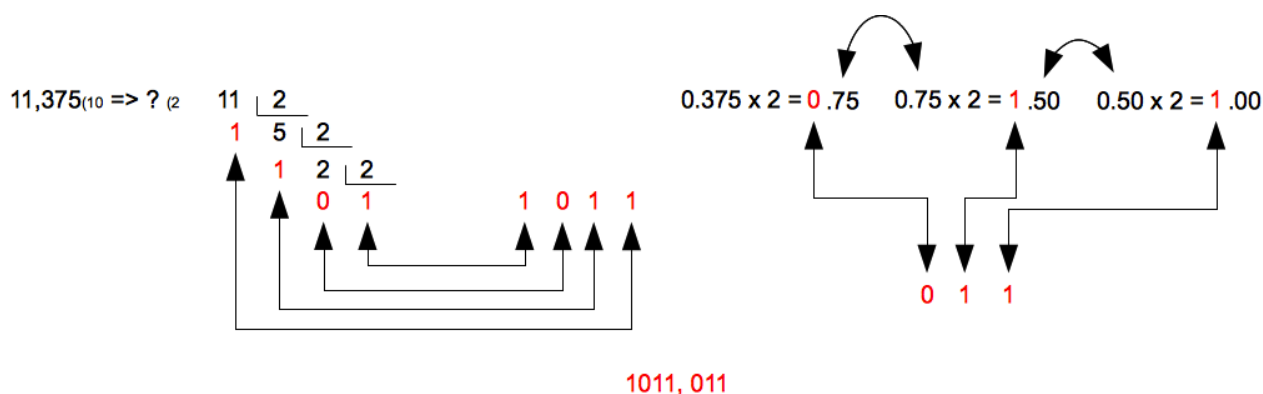


$$132_{(10)} \Rightarrow ?_{(2)}$$



En números con parte fraccionaria, el proceso es el mismo para la parte entera, pero la parte fraccionaria se calcula multiplicando por 2 sucesivamente y para sacar la parte entera (en este caso en el orden correcto).

-El bit más a la izquierda se llama bit más significativo (MSB) y el bit más a la derecha se llama bit menos significativo (LSB).



### 2.1.2 Cómo convertir un número binario en un número decimal


En este caso el proceso es muy fácil. Como se explicó anteriormente, un número decimal se puede representar como sumas de potencias de diez.

En general, puede convertir el valor de un número representado en un sistema numérico  $B_2$  al sistema decimal usando la siguiente fórmula:

$$n = u_{n-1}B_{n-1} + a_{n-2}B_{n-2} + \dots + a_1B_1 + a_0B_0 + a_{-1}B_{-1} + \dots + a_{-p}B_{-p} = \sum_{i=-p}^{n-1} a_i B_i$$

Lo vamos a usar para convertir en base 2

$101001_{(2)} \Rightarrow ?_{(10)}$


 $101001 \Rightarrow 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 41$

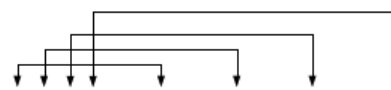
El proceso consta de cuatro pasos.

1. Escribir las cifras de los números binarios multiplicados por 2.
2. Escribir un signo más (+) entre cada uno de los productos.
3. Escribir un exponente en cada 2, comenzando desde cero y desde el último número de la parte entera (en el extremo derecho si no hay parte fraccionaria) y aumentándolo de uno en uno hacia la izquierda y decreciendo hacia la derecha.
4. Para realizar la operación

2 Como veremos más adelante, B puede ser cualquier sistema numérico



$$10,01_{(2)} \Rightarrow ?_{(10)}$$



$$10,01 \Rightarrow 1 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 2,25$$

### 2.1.3 Número máximo de valores a representar

Una de las preguntas típicas cuando se maneja un número binario es saber cuál es el máximo valor decimal que puede representar un determinado número de bits. La respuesta es fácil:  $2_n$ , donde n es el número de bit. Por ejemplo, con 4 bits podemos representar 16 valores, del 0 al 15 (0000-1111)

### 2.1.4 Operaciones con números binarios

La suma y resta binaria siguen las siguientes reglas:

#### Suma:

$$\begin{aligned} 0 + 0 &= 0 \\ 1 + 0 &= 1 \\ 0 + 1 &= 1 \\ 1 + 1 &= 0 \text{ (llevar 1)} \end{aligned}$$

#### Sustracción:

$$\begin{aligned} 0 - 0 &= 0 \\ 1 - 0 &= 1 \\ 0 - 1 &= 1 \text{ (llevar 1 a sustraendo)} \\ 1 - 1 &= 0 \end{aligned}$$

El resultado de las operaciones es el mismo que sus operaciones decimales relacionadas, excepto en los casos en que el resultado no tiene un valor en el sistema binario, es decir,  $1+1$ , que no puede representarse por 2 y  $0-1$ , que puede no representado por  $-1$ . Aquí es donde la transferencia es importante.

Algunos ejemplos:

$$\begin{array}{r} \textcolor{red}{11} \\ 10011010 \\ + 01001100 \\ \hline 11100110 \end{array}$$

$$\begin{array}{r} \textcolor{red}{111111} \\ 1011 \\ + 111101 \\ \hline 1001000 \end{array}$$

-Si queremos sumar dos números binarios cuya suma es mayor que el número máximo a representar la computadora arroja un *Desbordamiento* advertencia. Por ejemplo, si tenemos una computadora que trabaja con 8 bits,

puede representar desde  $0_{(10)}$  a  $255_{(10)}$ . Si queremos sumar  $10000000_2(128_{(10)})$  más  $10000000_2(128_{(10)})$  tenemos un problema porque el resultado es  $100000000_2(256_{(10)})$  mayor que 255. Así que un *Desbordamiento* ocurre.

$$\begin{array}{r} \phantom{+} \phantom{1} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ + \phantom{1} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ \hline \phantom{+} \phantom{1} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ \phantom{+} \phantom{1} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \end{array}$$

$$\begin{array}{r} \phantom{-} \phantom{1} \phantom{0} \phantom{1} \phantom{0} \phantom{1} \\ - \phantom{1} \phantom{0} \phantom{1} \phantom{0} \phantom{1} \\ \hline \phantom{-} \phantom{1} \phantom{0} \phantom{1} \phantom{0} \phantom{1} \phantom{0} \end{array}$$

$$\begin{array}{r} \phantom{-} \phantom{1} \phantom{1} \phantom{0} \phantom{1} \\ - \phantom{1} \phantom{0} \phantom{0} \phantom{1} \phantom{1} \\ \hline \phantom{-} \phantom{1} \phantom{1} \phantom{0} \end{array}$$

-En la resta, el remanente no se suma al minuendo, sino al sustraendo.

### Multiplicación:

$$0 * 0 = 0$$

$$1 * 0 = 0$$

$$0 * 1 = 0$$

$$1 * 1 = 1$$

### División:

$$0 / 0 = \text{Indefinido}$$

$$1 / 0 = \text{Ilimitado}$$

$$1 / 1 = 1$$

$$0 / 1 = 0$$

Tanto la multiplicación como la división no presentan diferencia con las operaciones relacionadas en decimal, a menos que las operaciones auxiliares se realicen en binario.

-En la multiplicación, cuando sumamos, puede ser que tengamos en la misma columna más de dos unos. En este caso, realizamos las sumas en grupos de dos y vamos a llevar los 1 en la siguiente columna.

-En la división, empezamos a obtener en dividendo y divisor el mismo número de cifras. Si no se puede dividir, intentamos sacar una cifra más en el dividendo.

Si la división es posible, entonces, el divisor sólo puede estar contenido una vez en el dividendo, es decir, la primera cifra del cociente es 1. En este caso, el resultado de multiplicar el divisor por 1 es el propio divisor (el valor que restará).

$$\begin{array}{r}
 101010 \quad \overline{)1110} \\
 \underline{-110} \phantom{00} \\
 1001 \phantom{00} \\
 \underline{-1110} \phantom{00} \\
 00110 \phantom{00} \\
 \underline{110} \phantom{00} \\
 000
 \end{array}$$

### 2.1.5 Números negativos

Cuando necesitamos representar un número binario negativo, tenemos varias opciones aunque tres son las más importantes. Este rango indica que la forma de expresarlo debe ser un acuerdo entre dos lados: el que genera el número y el que lo lee. De no ser así, el valor real a expresar sería erróneo.

#### *Magnitud con signo*

Tal vez sea el enfoque más fácil de entender. La idea es mantener el MSB para indicar el signo del número: 0 positivo, 1 negativo. Los bits restantes indican el valor numérico en valor absoluto. Por ejemplo:

Decimal	Binario	binario positivo	binario negativo
5	101	0101	1101

Como puede verse, necesitamos un bit para indicar el signo, de modo que lo que en valores normales de representación sería de 0 a 15, en este caso, para usar el signo, pasa a ser de -7 a +7 (1111 - 0111).

Este sistema es sencillo de entender pero complejo de utilizar a la hora de realizar operaciones matemáticas. Además tiene un problema: hay dos formas de definir el 0<sub>(10)</sub>: 0000<sub>(2)</sub> y 1000<sub>(2)</sub>

#### *complemento de uno*

La segunda opción también utiliza el primer bit como indicador de signo, pero en este caso el número negativo se consigue complementado con el número positivo (cambiando unos por ceros y viceversa).

Decimal	Binario	binario positivo	binario negativo
5	0101	0101	1010

En esta opción se requiere dar la cantidad de bits a codificar, de tal forma que si en el ejemplo anterior usamos 8 bits para codificar:

Decimal	Binario	binario positivo	binario negativo
5	101	00000101	11111010

Este método tiene el mismo problema que la magnitud con signo: hay dos formas de definir el 0<sub>(10)</sub>: 0000<sub>(2)</sub> y 1111<sub>(2)</sub>

*complemento a dos*

Aunque el complemento a uno simplifica las operaciones matemáticas, hacen mucho más con el uso del complemento a dos. Por eso es el método más utilizado.

El complemento a dos consiste en aplicar un complemento a unos y luego sumar 1. Por ejemplo, el complemento a dos de 5 codificado con 8 bits es:

$$5_{(10)} \rightarrow 101_{(2)} \rightarrow (\text{codificado en 8 bits}) 00000101_{(2)} \rightarrow (\text{complemento a 1}) 11111010_{(2)} \rightarrow (+1) 11111011$$

Decimal	Binario	binario positivo	binario negativo
5	101	00000101	11111011

¿Qué número decimal representa un número en complemento a dos?. Fácil. Tenemos que realizar el mismo proceso:

$$11111011_{(2)} \rightarrow (\text{complemento a 1}) \rightarrow 00000100_{(2)} \rightarrow (+1) 00000101_{(2)} \rightarrow 5_{(10)}$$

La gran ventaja del método del complemento a dos es que permite restar como si fueran sumas. Esto se debe a que restar dos números binarios es lo mismo que sumar al minuendo el complemento del sustraendo.

$$101101_{(2)} (45_{(10)}) - 010101_{(2)} (21_{(10)}) - 010101_{(2)} (\text{complemento a 1}) \rightarrow 101010_{(2)} \rightarrow (+1) 101011 - 101101 + 101011$$

$$\begin{array}{r} 101101 \\ + 101011 \\ \hline \end{array}$$

$$1011000_{(2)} \quad (24_{(10)}) \text{ el último traspaso } 1 \text{ se rechaza}$$

*Exceso-K o binario compensado*

Dependiendo de la cantidad de bits disponibles, el rango medio se dedica a los números negativos y la otra mitad (menos 1) a los positivos (el valor cero está en el medio).

El nuevo rango será  $[-K, K-1]$ , donde podemos calcular por  $K = 2^{n-1}$ . Una vez que tenemos el rango permitido, el número más pequeño es el que tiene todos sus bits a 0. Veamos un ejemplo:

Tenemos 3 bits para representar el número por lo que podemos representar  $2^3$  números, el rango  $[0, 7]$ . En este caso,  $K$  será  $2^{3-1} = 2^2 = 4$ , por lo que el rango con números negativos será  $[-4, 3]$ . El número más pequeño -4 será 000 y el 3 más grande será 111. El tablero completo será:

- 4	- 3	- 2	- 1	0	1	2	3
000	001	010	011	100	101	110	111

Si tenemos un número en *Exceso-K* sabemos su valor decimal, debemos restar el valor del exceso al valor decimal. Por ejemplo, si  $n = 8$  y es  $K = 2^{n-1} = 128$ ,

$$11001100 \rightarrow 204; 204 - 128 = 76 \quad \text{o} \quad 00111100 \rightarrow 60; 60 - 128 = -68$$

### 2.1.6 Números reales

Cuando escribimos un número real en un papel usamos una coma decimal (o punto decimal, depende de la cultura) para distinguir entre parte entera y parte fraccionaria. En una computadora, el espacio para representar este tipo de números se divide en dos áreas: una para la parte entera y otra para la parte fraccionaria. Hay dos formas de indicar el tamaño de estas áreas (campos), y por lo tanto, la posición de coma: *punto fijo* y *punto flotante*.

#### *Punto fijo*

En esta notación, asignamos un tamaño fijo a la parte entera y la parte fraccionaria del número, en otras palabras, un lugar fijo a la coma.

La ventaja es que el proceso para realizar operaciones básicas es el mismo que con números enteros. Sin embargo, este método no aprovecha la capacidad de representación del formato utilizado. Por ejemplo, una computadora con 8 bits para representar números, podría usar 5 bits para la parte entera y 3 para la parte fraccionaria  $b_7b_6b_5b_4b_3, b_2b_1b_0$ .

En este caso el número máximo a representar será 01111,111 y el mínimo (positivo) 00000,001. Si la coma decimal estuviera en posición flotante, el rango de números positivos a representar sería 011111111 – 0,0000001

#### *Punto flotante*

El rango de números que se pueden representar en el formato de punto fijo es insuficiente para muchas aplicaciones, particularmente para aplicaciones científicas que a menudo usan números muy grandes y muy pequeños. Representar una amplia gama de números usando relativamente pocos dígitos, es bien conocido en el sistema decimal, la representación científica o notación exponencial. Por ejemplo,  $0,00000025 = 2,5 \cdot 10^{-7}$ . en general, para cualquier sistema de numeración, un

<sup>3</sup> Hay otra versión de este método con  $K = 2^{n-1}-1$

número real se puede expresar como:

$$\text{norte} = \text{metro} * \text{segundo}_{\text{mio}} \quad N = (M;B;E)$$

dónde:

M: mantisa

B: fondo

E: exponente

La representación interna que hace este formato en las computadoras se conoce como punto flotante.

Por ejemplo en decimal (B=10)  $259,75_{(10)} = 0,25975 * 10_3$  o  $(0,25975;10;3)$  o, en código binario (B=2)

$$259,75_{(10)} \rightarrow 100000011,11_{(2)} \rightarrow 0,10000001111 * 2_9 \quad (2 \rightarrow 0,10000001111 * 2_{1001} \quad (2 \rightarrow (0,10000001111;1001)$$

El rango de números representable para un valor dado de B, está fijado por el número de bits del exponente E, mientras que la precisión está determinada por el número de bits de M.

### Normalización

El mismo valor real se puede representar de infinitas formas mediante notación exponencial. Por ejemplo, 2,5 se puede representar como  $0,25 * 10_1$ ,  $0,025 * 10_2$ ,  $250 * 10_{-2}$ ,... Para evitar confusiones, deberíamos elegir uno de estos formatos como el estándar para la representación de punto flotante de un número real. La forma elegida se llama *forma normalizada* y es el que mantiene la mayor precisión en la representación de los números. Esto se logra cuando el punto binario se ubica inmediatamente a la izquierda del primer dígito significativo, de modo que no se desperdicia espacio representando dígitos no significativos.

Por ejemplo:

- 2,5 representado en forma normalizada es  $0,25 * 10_1$
- $(0,000011101 ; 2; 0111) \rightarrow (0,11101; 0011)_4 \rightarrow \text{Exceso de exponente-k}(0,11101; 1011)$

- En general, para representar exponentes negativos *exceso-k* se utiliza el método. Por otro lado, para representar negativo *mantisa* método de magnitud con signo

- $(100,11110; 2; 0010) \rightarrow (0,10011110; 2; 0101) \rightarrow \text{Exceso de exponente-k}(0,10011110; 1101)$
- $(101,001; 2; 0100) \rightarrow (0,1010010; 2; 0111) \rightarrow \text{Exceso de exponente-k}(0,10011110; 1111)$

### IEEE754

El formato más popular para representar puntos flotantes en binario fue desarrollado por

4 Eliminamos el valor base porque asumimos que la base del sistema es 2

la *Instituto de Ingenieros Eléctricos y Electrónicos* (IEEE) y se llama IEEE754. Este formato puede representar casos especiales como valores infinitos y resultados indefinidos, *Yaya* o *No un número* resultados. Propone 3 formatos:

Media precisión. Utiliza 16 bits



Utiliza 16 bits: un bit para el signo, 5 para el exponente, 10 para la mantisa

Precisión sencilla.



Utiliza 32 bits: un bit para el signo, 8 para el exponente, 23 para la mantisa

Precisión doble.



Utiliza 64 bits: un bit para el signo, 11 para el exponente, 55 para la mantisa



-Los tres formatos usan mantisa normalizada, por lo que el primer bit a la izquierda de la mantisa será un 1 (el primer dígito significativo debe ser un 1). Por ello, tres formatos no codifican este 1 en la mantisa, aunque se tiene en cuenta a la hora de operar con el número. En otras palabras, en estos formatos, el MSB está a la izquierda de la coma decimal y solo guardan los bits del lado derecho.

Para representar el exponente, el estándar utiliza el método Excess-K con  $K = 2^{n-1} - 1$

-Para entender mejor esta operación es muy recomendable ver la pastilla 02 [Convertir número real a URL de código binario](#)

### 2.1.7 Álgebra booleana

Además de las operaciones matemáticas (+, -, \*/,), sobre números binarios se pueden aplicar operaciones booleanas o lógicas: and, or, xor, not...

-Para entender mejor estas operaciones vale la pena nombrar 1 como *verdadero* y 0 como *falso*

## NO:

Se puede representar de varias formas: NOT,  $\neg$

$$\text{NO } 0 = 1$$

$$\text{NO } 1 = 0$$

## Y:

Se puede representar de varias formas: AND, Y,  $\wedge$

$$0 \text{ Y } 0 = 0$$

$$1 \text{ Y } 0 = 0$$

$$0 \text{ Y } 1 = 0$$

$$1 \text{ Y } 1 = 1$$

En otras palabras, el resultado será verdadero (1) solo cuando ambos dígitos fueron *verdadero*. Como se puede ver, el resultado es el mismo que la multiplicación.

$$\begin{array}{r} \text{Y} \quad 10011010 \\ \quad \underline{01001100} \\ \quad 00001000 \end{array}$$

$$\begin{array}{r} \text{Y} \quad 1011 \\ \quad \underline{111101} \\ \quad 001001 \end{array}$$

## O:

Se puede representar de varias formas: OR, O,  $\vee$

$$0 \text{ O } 0 = 0$$

$$1 \text{ O } 0 = 1$$

$$0 \text{ O } 1 = 1$$

$$1 \text{ O } 1 = 1$$

En este caso, el resultado será *verdadero* tan pronto como uno de los dígitos era *verdadero*.

$$\begin{array}{r} \text{O} \quad 10011010 \\ \quad \underline{01001100} \\ \quad 11011110 \end{array}$$

$$\begin{array}{r} \text{O} \quad 1011 \\ \quad \underline{111101} \\ \quad 111111 \end{array}$$

## XOR:

$$0 \text{ X } 0 = 0$$

$$1 \text{ X } 0 = 1$$



$$0 \text{ X } 0 \text{ } 1 = 1$$

$$1 \text{ X } 0 \text{ } 1 = 0$$

En este caso, el resultado será *verdadero* cuando uno y sólo uno de los dígitos eran *verdadero*.

$$\begin{array}{r} 10011010 \\ \text{XOR } 01001100 \\ \hline 11010110 \end{array}$$

$$\begin{array}{r} 1011 \\ \text{XOR } 111101 \\ \hline 110110 \end{array}$$

## 2.2 octales

Además del binario, existen otros dos sistemas de numeración interesantes a la hora de trabajar temas relacionados con las tecnologías de la información: el octal y el hexadecimal. Esto se debe a que son fáciles de convertir a binario.

El octal es un sistema numeral. Con una base del sistema igual a 8 (símbolos 0,1 ,2 ,3 ,4 ,5 ,6 ,7) . Su base es una potencia exacta del sistema binario base  $2_3=8$  o lo que es lo mismo, con tres dígitos binarios (con tres bits) podemos representar todos los dígitos octales.

Binario	octales
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

### 2.2.1 Cómo convertir un número binario en octal

El proceso consiste en crear grupos de bits de tres, comenzando por la mano derecha, y reemplazarlos por el valor octal relacionado

$$1101011_{(2)} \Rightarrow 1101011 \Rightarrow 153_{(8)}$$

### 2.2.2 Cómo convertir un número octal en binario

El proceso es inverso al anterior: se convierte a binario cada uno de los números del número octal

$$7402_{(8)} \Rightarrow 11110000010_{(2)} = 11110000010_{(2)}$$

## 2.3 hexadecimales

Su base del sistema es 16. Como el número de símbolos utilizados en el sistema es mayor que 10, se deben utilizar 6 caracteres, en este caso de la A a la F. Así, el conjunto ordenado 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

### 2.3.1 Cómo convertir números binarios a hexadecimales

El proceso es similar al proceso binario-octal, excepto que en este caso los grupos son de cuatro en cuatro.

$$1101011_{(2)} \Rightarrow 1101011_{(2)} \Rightarrow 6B_{(16)}$$

### 2.3.2 Cómo convertir números hexadecimales a binarios

El proceso es inverso al anterior: se convierte a binario cada uno de los números del número octal

$$7F0A_{(16)} \Rightarrow 0111111100001010_{(2)} = 111111100001010_{(2)}$$

### 2.3.3 Cómo convertir números hexadecimales en números octales

Convertimos a binario y agrupamos los bits en grupos de cuatro o tres, cualquiera que sea el sistema numérico

destino

6B<sub>(dieciséis)</sub> => 110 1011 => 1101011<sub>(2)</sub> => 1 101 011 => 153<sub>(8)</sub>

-La conversión entre octal o hexadecimal y decimal o viceversa, se puede realizar siguiendo los métodos para convertir entre binario y decimal, pero multiplicando por potencia de 8 o 16 o dividiendo por estos números y obtener los restos (en hexadecimal si el resto es mayor que 9 obtenemos sus valores relacionados A..F).

Sin embargo, suele ser más práctico realizar directamente la conversión a binario y luego convertir al sistema solicitado.

### 3.REPRESENTACIÓN ALFANUMÉRICA

#### 3.1 Datos numéricos y alfanuméricos.

Un dato es numérico si es posible realizar operaciones matemáticas. En cambio, un dato es alfanumérico si NO se pueden realizar operaciones matemáticas sobre él.

numérico:¿Cuántos años tienes?45 alfanumérico:

¿Cuál es su nombre?"roberto"

-Para diferenciar claramente entre los dos tipos de datos, es común usar comillas simples o dobles para indicar que los datos son alfanuméricos.

Es habitual pensar que los datos numéricos son números y los datos alfanuméricos son solo letras. Pero esto no es correcto. Por ejemplo:

¿Cual es tu dirección?"Avenida de las Palmeras 34" ¿Cuál es su número de móvil?"555341273"

En el primer caso,**Avenida de las Palmeras 34**,se compone de letras y números, y en el segundo, **555341273**,solo por números, pero no operables (no tiene sentido sumar o multiplicar dos números de teléfono).

#### 3.2 Representación interna

Los caracteres alfanuméricos para representar computadoras se basan en tablas, de modo que cada una de las entradas de la tabla (cada número) corresponde a un símbolo alfanumérico.

A lo largo de la historia de la informática han existido varias tablas que siempre se han caracterizado por la cantidad de bits utilizados para representar cada carácter.

Uno de los mejores ejemplos en la tabla ASCII. El número de bits es 7, lo que deja espacio para 128 caracteres ( $2^7=128$ )

Como se puede observar en la siguiente tabla, cada número está relacionado con un carácter. Por ejemplo, 73<sub>(10)</sub> es un "yo", 105<sub>(10)</sub> es una "i" o 50<sub>(10)</sub> es un "2". Las primeras entradas están reservadas para caracteres no imprimibles, aquellos que no son visibles, como tabulador(9<sub>(10)</sub>) o retorno de carro (15<sub>(10)</sub>).

-El espacio también es un personaje: 32<sub>(10)</sub>

El problema de esta mesa es su espacio limitado. Como ves, tiene cabida para todas las grafías latinas que se utilizan en las lenguas anglosajonas, pero no encontramos grafías como la ñ, la ç o las vocales acentuadas. Por lo tanto, se creó la tabla ASCII extendida de 8 bits (256 caracteres). Esta nueva tabla puede incorporar todas las grafías latinas más algunos símbolos gráficos.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	\$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: [www.LookupTables.com](http://www.LookupTables.com)

-Hoy en día las tablas ASCII están casi obsoletas. La expansión de Internet y la globalización hacen necesarias tablas que incorporen no solo caracteres latinos, sino chinos, árabes, coreanos, rusos, hebreos...

A lo largo de la historia de la informática, ha habido varios t

### 3.3 Código único

Unicode es un estándar que puede usar más de un byte para cada carácter, lo que permite un rango de representación mucho más amplio. Los elementos más importantes de Unicode son:

- **Punto de código:** Un "punto de código" representa un solo carácter. Se escriben como U+nnnnnn. Unicode admite más de un millón, y actualmente se asignan poco menos de 100 mil.
- **Glifo:** El glifo indica cómo se debe pintar un código en la pantalla. Pueden existir diferentes glifos para el mismo carácter (por ejemplo, diferentes tipos de fuentes)
- **Codificación:** Hay diferentes formas de representar un personaje:
  - UTF8: Tiene una longitud variable, usa de uno a seis bytes para cada carácter (los más comunes usan solo un byte)
  - UTF16: Es similar a UTF8, pero se utilizan un mínimo de 2 bytes por carácter.
  - UTF32: Tiene una longitud fija. Se utilizan 4 bytes para cada carácter.

El algoritmo de intercalación de Unicode (UCA) es un algoritmo definido en el Informe técnico de Unicode n.º 10, que es un método personalizable para producir claves binarias a partir de cadenas que representan texto en cualquier sistema de escritura e idioma que se pueda representar con Unicode. Estas claves se pueden comparar de manera eficiente byte por byte para recopilarlas u ordenarlas de acuerdo con las reglas del idioma, con opciones para ignorar mayúsculas y minúsculas, acentos, etc.

## 4. OTROS TIPOS DE INFORMACIÓN

Las imágenes y el sonido son señales continuas, con un nivel infinito de precisión. Para representar este tipo de señales, deben transformarse en señales discretas, es decir, con precisión limitada. Este proceso se llama codificación y la información se perderá en él. Con mayor nivel de precisión, más señal parecerá la original, pero tendrá un tamaño mayor.

### 4.1 Imágenes

La forma habitual de representar imágenes es proyectarlas en una cuadrícula de un tamaño determinado y guardar el "color" de cada uno de los cuadrados de esa cuadrícula. Cada uno de estos cuadrados se llama píxel. El tamaño de esa cuadrícula es la resolución de la imagen.

Para cada uno de los píxeles, es necesario almacenar su color. Aquí entran en juego dos conceptos fundamentales:

- **Espacio de color:** Indica qué colores se pueden representar. Hay un espacio de color.

llamado CIELab, que contiene todos los colores que el ojo humano puede ver. Los dispositivos informáticos no pueden representar todos esos colores, sino solo un subconjunto de ellos. Los espacios de color más comunes utilizados en informática son sRGB y AdobeRGB.

- **modelo de color:** definir cómo se representarán los colores del espacio de color, utilizando tuplas de números. Los modelos más comunes son RGB (que indica los niveles de rojo, verde (G) y azul (B) en la imagen y se usa para colores aditivos como la luz), CYMK (que se usa para colores sustractivos como la tinta) HSV y HSL .

Para saber cuál es el color a mostrar, debemos saber cuál es el modelo y también qué espacio de color se está definiendo. En los monitores, es común usar RGB sobre el espacio sRGB, pero para imprimir es más común usar CYMK sobre AdobeRGB.

También existen formatos vectoriales para representar imágenes geométricas. Estos formatos no almacenan píxeles, sino una descripción de lo que se debe dibujar (por ejemplo, un círculo centrado en 0,10 en rojo). Hay muchos formatos vectoriales, como ai, svg, etc. Los formatos vectoriales se pueden escalar sin problemas, ya que contienen una descripción de la imagen y no los píxeles que la componen.

## 4.2 Vídeos

Los videos están representados por secuencias de imágenes ligeramente modificadas. Esto se puede hacer así ya que nuestro cerebro, si las proyectamos lo suficientemente rápido, las interpretará como una señal continua y las transformará en movimiento (no es el caso de otras especies animales).

Cada imagen se llama marco. Un video se puede renderizar con una cantidad diferente de fotogramas por segundo, normalmente de 23,56 a 60.

Los videos generalmente se comprimen utilizando diferentes algoritmos llamados códecs, como mpeg-2, h.s64 xvid o divx. Dado que las imágenes entre cuadros normalmente cambian poco, los códecs de video pueden tener una eficiencia muy alta.

Los videos se almacenan en archivos con diferentes formatos. El formato del archivo de video se llama contenedor (por ejemplo: avi, divx o matroska) Se identifican comúnmente por su extensión. Un contenedor no tiene por qué contener sólo vídeo, habitualmente contienen vídeos y sonidos. Asimismo, puede contener información codificada con diferentes códecs.

## 4.3 Sonidos

Un sonido es una onda continua de vibraciones. El muestreo se utiliza para representar el sonido: la medida de la intensidad de la onda se toma a intervalos regulares. A partir de estas intensidades se puede volver a reproducir la señal original.

El espectro audible del oído humano oscila entre 20 Hz y 20 000 Hz, dependiendo de la edad del individuo. Por tanto, y según el teorema de Nyquist-Shannon, se necesita un muestreo del doble de la frecuencia de la señal, en este caso 40Khz (es divertido comprobarlo haciendo fotos con alta exposición a pantallas) Se suele utilizar un muestreo de 44,1 Khz en formatos digitales.

El número de bits de muestra necesarios para almacenar el sonido depende de la relación señal-ruido de la muestra. Por lo general, 16 bits por muestra son suficientes.

A la hora de representar digitalmente los sonidos se utilizan tres tipos de formatos:

- **Formatos PCM o RAW:**contener la muestra tal como se recibió. Ocupan un gran tamaño.
- **Formatos comprimidos:**pueden ser con pérdida (mp3, ogg) o sin pérdida (flac) Los formatos con pérdida intentan eliminar información que no es perceptible para el oído humano y pueden ser muy eficientes. Los formatos sin pérdida le permiten reproducir la señal tal como fue muestreada.
- **Formatos descriptivos:**sería el equivalente a los formatos vectoriales en imágenes: contiene las instrucciones para que un sintetizador interprete una serie de notas: es similar a una partitura que contiene las notas, los instrumentos y los tiempos.

## 5. SISTEMA DE UNIDADES

Como se discutió anteriormente, el bit es la unidad de información más pequeña. Hoy no trabajamos a nivel de bits, sino por grupos de bits (en el apartado anterior hemos visto que un carácter se codifica con 7 u 8 bits).

Debido a que dentro de la computadora todo está en código binario, una manera fácil de manejar grupos es usar algún valor que sea una potencia de 2, siendo el 2 el más básico. $2^3 = 8$ . Un grupo de 8 bits se llama **byte**.

Hoy en día no es habitual utilizar el grupo de bytes, sino algunos múltiplos del mismo: *Kilobyte*(kB), *Megabyte*(MEGABYTE), *Gigabyte*(GB), *Terabyte*(TB)... . En el *Sistema Internacional* estos múltiplos son potencias de 10 (se basan en el sistema decimal), pero en el cálculo se utilizan potencias de 2. Sin embargo, la tendencia es utilizar el Sistema Internacional, aunque cabe señalar que los valores son similares pero no iguales.

Usamos dos tipos diferentes de palabras para diferenciar entre las unidades del sistema. Cuando hablamos de kilobyte nos referimos al sistema decimal y cuando hablamos de *kibibytes* al sistema binario.

Las equivalencias se pueden ver en la siguiente tabla:

Nombre SI	SI	binario	Nombre binario
kilobyte (kB)	$10^3$ bytes = 1000 bytes	$2^{10}$ bytes = 1024 bytes	Kibibyte (kiB)

Megabyte (MB)	$10^6 \text{bytes} = 1000 \text{ kB}$	$2^{20} \text{bytes} = 1024^2 \text{bytes}$	Mebibyte (MiB)
Gigabyte (GB)	$10^9 \text{bytes} = 1000 \text{ MB}$	$2^{30} \text{bytes} = 1024^3 \text{bytes}$	Gibibyte (GiB)
Terabyte (TB)	$10^{12} \text{bytes} = 1000 \text{ GB}$	$2^{40} \text{bytes} = 1024^4 \text{bytes}$	Tebibyte (TiB)
Petabyte (PB)	$10^{15} \text{bytes} = 1000 \text{ TB}$	$2^{50} \text{bytes} = 1024^5 \text{bytes}$	Pebibyte (PiB)
Exabyte (EB)	$10^{18} \text{bytes} = 1000 \text{ PB}$	$2^{60} \text{bytes} = 1024^6 \text{bytes}$	Exbibyte (EiB)
Zetabyte (ZB)	$10^{21} \text{bytes} = 1000 \text{ EB}$	$2^{70} \text{bytes} = 1024^7 \text{bytes}$	Zebibyte (ZiB)

-Aunque suele usarse indistintamente y, aún hoy en día, es más común *Sistema Internacional*, los valores que se representan son diferentes: 1 MB son 1.000.000 bytes (un millón de bytes), mientras que 1 MiB son 1.048.576 bytes

-Es importante diferenciar entre kB y kb. El primero se refiere a kilobyte, mientras que el segundo a kilobit, 8 veces menos.

-Aunque la mayoría de los nombres y abreviaturas de múltiplos tienen mayúsculas, el kilo se define con minúsculas.

## 6.MATERIAL ADICIONAL

[1] Glosario.

[2] Vídeos sobre operaciones binarias.

[3] Ejercicios.

## 7.BIBLIOGRAFÍA

[1]Wikipedia. Representaciones de números con signo

[https://en.wikipedia.org/wiki/Signed\\_number\\_representations](https://en.wikipedia.org/wiki/Signed_number_representations)

[2]Wikipedia. Representaciones de números con signo

[https://en.wikipedia.org/wiki/Signed\\_number\\_representations](https://en.wikipedia.org/wiki/Signed_number_representations)

[3] Sistemas Informáticos. Isabel Mª Jiménez Cumbreras. *Garceta*. 2012