


# REPRESENTACIÓN DE LA INFORMACIÓN

## 1. Sistemas de numeración

Un sistema de numeración es aquel conjunto de símbolos y reglas que sirven para la representación de cualquier cantidad abstracta.

Lenguaje de signos	Sistema Morse	Sistema romano	Sistema decimal	Cantidad
	• - - - -	I	1	Uno
	• • - - -	II	2	Dos
	- - • • •	VII	7	Siete
	• - - - - • • • •	XV	15	Quince

Cada sistema de numeración tiene una cantidad concreta de símbolos que lo compone. Es lo que llamamos **base**. El sistema de numeración que utilizamos normalmente es el **sistema decimal** que se compone de diez símbolos (dígitos): 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9... por tanto es un sistema de base 10.

El mismo dígito tiene un significado diferente dependiendo de su posición en la representación completa (que desde ahora llamaremos cifra).

Dígitos (Símbolos)	Cifra	Cantidad
0, 5	505	Quinientos cinco

En el sistema decimal, cualquier cantidad se puede expresar como la suma de potencias de **base 10**.

$$13 = 10 + 3 = 1 \cdot 10^1 + 3 \cdot 10^0$$

$$6879 = 6000 + 800 + 70 + 9 = 6 \cdot 10^3 + 8 \cdot 10^2 + 7 \cdot 10^1 + 9 \cdot 10^0$$

$$13004 = 10000 + 3000 + 0 + 0 + 4 = 1 \cdot 10^4 + 3 \cdot 10^3 + 0 \cdot 10^2 + 0 \cdot 10^1 + 4 \cdot 10^0$$

### a. El sistema binario

El sistema de numeración binario utiliza dos dígitos (0 y 1) para representar las cantidades abstractas. Por tanto este sistema es **base 2**. En informática cada dígito en esta representación se le llama **bit**.

En los ordenadores hay una cantidad fija de bits para representar un número. Si el número es menor que la cantidad más grande representable, se rellena poniendo ceros a la izquierda. Si el número es mayor que la cantidad más grande representable,

estamos en un caso de desbordamiento y hay que truncar la representación. El número más grande representable en binario depende del número de bits.

$$N_{max}|_{10} = 2^n - 1$$

$$N_{max}(5 \text{ bits}) = 2^5 - 1 = 11111 = 31$$

Decimal	Binario	Representación 4 bits	Decimal	Binario	Representación 4 bits
0	0	0000	11	1011	1011
1	1	0001	12	1100	1100
2	10	0010	13	1101	1101
3	11	0011	14	1110	1110
4	100	0100	15	1111	1111
5	101	0101	20	10100	0100
6	110	0110	30	11110	1110
7	111	0111	40	101000	1000
8	1000	1000	50	110010	0010
9	1001	1001	100	1100100	0100
10	1010	1010	150	10010110	0110

Desbordamiento

Análogamente al sistema decimal, en el sistema binario podemos expresar cualquier cantidad como una suma de potencias de **base 2**.

$$1011|_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 0 + 2 + 1 = 11|_{10}$$

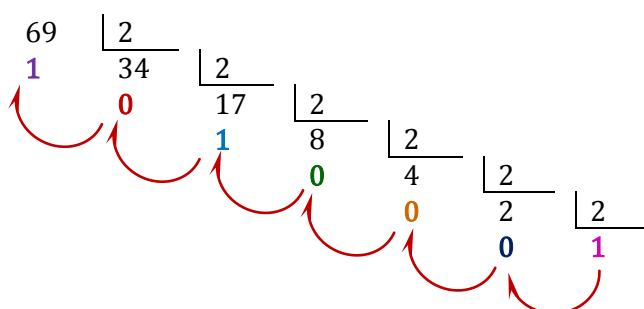
$$11010|_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 16 + 8 + 0 + 2 + 0 = 26|_{10}$$

Este método sirve para realizar la conversión de un número binario a un número decimal.

Para realizar la conversión inversa (del sistema decimal al binario) se divide el número del sistema decimal entre 2, cuyo resultado se vuelve a dividir entre 2, y así sucesivamente hasta que el número a dividir sea 1.

Después se ordenan los restos de las divisiones desde el último al primero (en orden inverso a como aparecen). Éste será el número binario que buscamos.

$$69|_{10} = 1000101|_2$$



Potencias de base 2			
2 <sup>0</sup>	1		
2 <sup>1</sup>	2	2 <sup>-1</sup>	$\frac{1}{2} = 0,5$
2 <sup>2</sup>	4	2 <sup>-2</sup>	$\frac{1}{4} = 0,25$
2 <sup>3</sup>	8	2 <sup>-3</sup>	$\frac{1}{8} = 0,125$
2 <sup>4</sup>	16	2 <sup>-4</sup>	$\frac{1}{16} = 0,0625$
2 <sup>5</sup>	32	2 <sup>-5</sup>	$\frac{1}{32} = 0,03125$
2 <sup>6</sup>	64	2 <sup>-6</sup>	$\frac{1}{64} = 0,015625$
2 <sup>7</sup>	128	2 <sup>-7</sup>	$\frac{1}{128} = 0,0078125$
2 <sup>8</sup>	256	2 <sup>-8</sup>	$\frac{1}{256} = 0,00390625$
2 <sup>9</sup>	512	2 <sup>-9</sup>	$\frac{1}{512} = 0,001953125$
2 <sup>10</sup>	1024	2 <sup>-10</sup>	$\frac{1}{1024} = 0,0009765625$

Para números con cifras decimales, en el caso de querer pasar de una cantidad binaria a una cantidad decimal, se extiende la suma de la conversión con potencias de exponente negativo.

$$1011,01 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 8 + 0 + 2 + 1 + 0 + 0,25 = 11,25$$

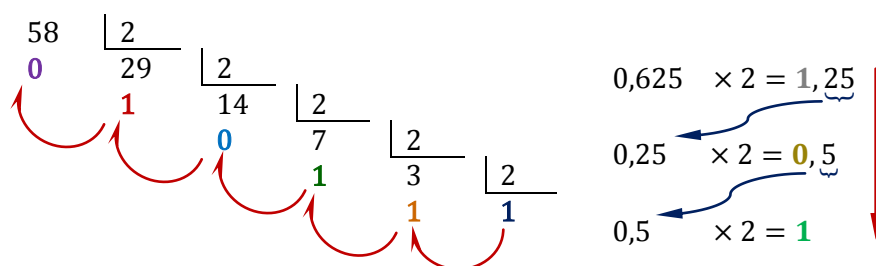
Para realizar la conversión de un número decimal de base 10 al sistema binario, primero se transforma la parte entera a binario. Se sigue con las cifras decimales, multiplicando cada número por 2. Si el resultado obtenido es mayor o igual a 1 se anota como un 1 binario. Si es menor que 1 se anota como un 0 binario.

$$0,7 \times 2 = 1,4 \rightarrow \text{Se anota un 1}$$

$$0,4 \times 2 = 0,8 \rightarrow \text{Se anota un 0}$$

Después de realizar cada multiplicación, se colocan los números obtenidos en el orden en el que se obtienen. Se repite esta operación hasta que el resultado es 1 exacto. (Algunos números por mucho que los multipliques por dos, nunca dan 1 exacto porque tienen en su representación binaria infinitas cifras decimales).

$$58,625|_{10} = 111010,101|_2$$



## b. El sistema hexadecimal

El sistema de numeración hexadecimal utiliza 16 dígitos para representar las cantidades. Estos son: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F. Es un sistema de numeración **base 16**. Sigue unas reglas parecidas al sistema binario.

Decimal	Binario	Hexadecimal	Decimal	Binario	Hexadecimal
0	0	0	11	1011	B
1	1	1	12	1100	C
2	10	2	13	1101	D
3	11	3	14	1110	E
4	100	4	15	1111	F
5	101	5	16	10000	10
6	110	6	17	10001	11
7	111	7	20	10100	14
8	1000	8	100	1100100	64
9	1001	9	255	11111111	FF
10	1010	A	256	100000000	100

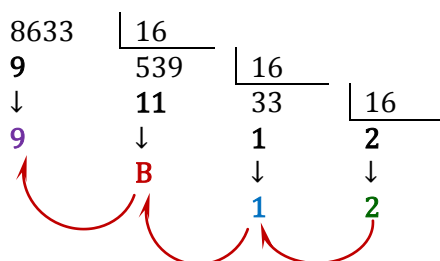
Análogamente a los otros sistemas numéricos, el sistema binario se puede expresar un número hexadecimal como una suma de potencias de **base 16**. Es

$$\text{B2}_{16} = \text{B} \cdot 16^1 + \text{2} \cdot 16^0 = 11 \cdot 16 + 2 \cdot 1 = 176 + 2 = 178_{10}$$

$$\text{A1D}_{16} = \text{A} \cdot 16^2 + \text{1} \cdot 16^1 + \text{D} \cdot 16^0 = 10 \cdot 256 + 1 \cdot 16 + 13 \cdot 1 = 2560 + 16 + 13 = 2589_{10}$$

Para convertir un número en el sistema decimal al sistema hexadecimal se utiliza el método que ya hemos visto para el binario, ahora se utilizara la base hexadecimal como divisor.

$$8633_{10} = \text{21B9}_{16}$$



En informática se utiliza el sistema hexadecimal porque es muy fácil transformar números en base 16 al sistema binario. Por otro lado para los informáticos es más fácil utilizar en el sistema hexadecimal que el sistema binario y a veces incluso más fácil que el propio sistema decimal.

Para pasar un número binario a base hexadecimal se hacen grupos de cuatro dígitos y se hace la conversión directamente.

Binario	Hexadecimal	Binario	Hexadecimal
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

$$\begin{array}{c} \text{1110101101}_2 \\ \text{0011} \quad \text{1010} \quad \text{1101} \\ \downarrow \quad \downarrow \quad \downarrow \\ 3 \quad \quad A \quad \quad D \\ \text{3AD}_{16} \end{array}$$

Para realizar la conversión de números hexadecimales a la base binaria sólo hay que operar al revés.

$$\text{CA32}_{16} \rightarrow \text{1100} \quad \text{1010} \quad \text{0011} \quad \text{0010} \rightarrow 1100101000110010$$

¡Ojo! Estas conversiones directas sólo funcionan entre el binario y el hexadecimal. No sirve para el sistema decimal.

## 2. Representación interna de la información

Los procesadores almacenan la información en unos elementos de memoria fáciles de manejar. Estas agrupaciones de bits suelen ser múltiplos de la base binaria: 8, 32, 64, 128...

Las principales agrupaciones de bits son:

- **Byte**: agrupación de 8 **bytes**, se utiliza habitualmente para cuantificar el tamaño de una unidad de memoria.
- **Palabra**: es la unidad mínima manejable por el sistema, y depende de su arquitectura ya que corresponde con el tamaño de los **registros** internos de la CPU. Cuanto mayor es el ancho de palabra, mayor es el número que puede almacenar y mayor es su precisión, o lo que es lo mismo, mayor es la potencia del cálculo del procesador. Actualmente las palabras más comunes que podemos encontrar en los sistemas son de 32 bits y 64 bits.

### a. Representación de los números enteros

El conjunto de los números enteros comprende a todas las cantidades enteras, tanto los positivos como los negativos, incluyendo el cero.

Para representar los números enteros con el sistema binario en un sistema informático siempre hay que tener en cuenta que existe un límite de dígitos para cada dato. Ese límite es el ancho de palabra. Si en una arquitectura tenemos palabras de  $n$  bits, según las leyes de la combinatoria, en el sistema binario sólo podemos representar  $2^n$  números.

$$\boxed{\text{Nº de representaciones posibles} = 2^n}$$

#### Representación de enteros positivos

Para representar números binarios positivos se almacenan tal cual en el espacio disponible. Si disponemos de  $n$  bits, el número más pequeño será **0** y el más grande  $2^n - 1$ .

$$\boxed{\text{Rango } 0 \leq x \leq (2^n - 1)}$$

Si  $n = 8$  bits:

Nº repr= 256

Rango  $0 \leq x \leq 255$

42<sub>10</sub>

0	0	0	1	0	1	1	0
---	---	---	---	---	---	---	---

203<sub>10</sub>

1	1	0	0	1	0	1	1
---	---	---	---	---	---	---	---

Si  $n = 16$  bits:

Nº repr= 65536

Rango  $0 \leq x \leq 65535$

38<sub>10</sub>

0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

473<sub>10</sub>

0	0	0	0	0	0	1	1	1	0	1	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

## Representación signo-magnitud

Para poder representar números negativos se debe reservar un bit para indicar el signo (normalmente el que está más a la izquierda) y su valor será **0 si el número es positivo** y **1 si el número es negativo**. El resto de bits ( $n - 1$ ) se utilizarán para representar la magnitud. Ahora el rango representable será el siguiente:

$$\text{Rango } -(2^{n-1} - 1) \leq x \leq (2^{n-1} - 1)$$

Si  $n = 8$  bits:

Nº repr= 256

Rango  $-127 \leq x \leq 127$

$102_{10}$  0 1 1 0 0 1 1 0

$-102_{10}$  1 1 1 0 0 1 1 0

$47_{10}$  0 0 1 0 1 1 1 1

$-47_{10}$  1 0 1 0 1 1 1 1

$0_{10}$  0 0 0 0 0 0 0 0

$-0_{10}$  1 0 0 0 0 0 0 0

Este sistema tiene un problema, se desperdicia una representación ya que el número cero tiene dos representaciones (positiva y negativa).

## Representación complemento a 1 (Ca1)

Con el complemento a 1 también se reserva el bit de la izquierda para indicar el signo y el valor 1 indica que el número es negativo, pero las magnitudes se representan de forma diferente. Los números negativos se obtienen **cambiando todos los números de ceros a unos y de unos a ceros**:

$$\text{Rango } -(2^{n-1} - 1) \leq x \leq (2^{n-1} - 1)$$

Si  $n = 8$  bits:

Nº repr= 256

Rango  $-127 \leq x \leq 127$

$37_{10}$  0 0 1 0 0 1 0 1

$-37_{10}$  1 1 0 1 1 0 1 0

$-127_{10}$  1 0 0 0 0 0 0 0

$-1_{10}$  1 1 1 1 1 1 1 0

$0_{10}$  0 0 0 0 0 0 0 0

$-0_{10}$  1 1 1 1 1 1 1 1

Este sistema tiene el mismo problema que la anterior, se desperdicia una representación en el caso del cero.

## Representación complemento a 2 (Ca2)

Con el complemento a 2 es muy parecido al complemento a 1, también se reserva el bit de la izquierda para indicar el signo y el valor 1 indica que el número es negativo. Las magnitudes positivas se obtienen normalmente, las negativas, sin embargo, se representan obteniendo el **complemento a 1 y sumando una unidad a la magnitud**:

Si  $n = 8$  bits:

10      

0	0	1	0	0	1	0	1
---	---	---	---	---	---	---	---

Ca1      

1	1	0	1	1	0	1	0
							1

-10 (Ca2)      

1	1	0	1	1	0	1	1
---	---	---	---	---	---	---	---

68      

0	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---

Ca1      

1	0	1	1	1	0	1	1
							1

-68 (Ca2)      

1	0	1	1	1	1	0	0
---	---	---	---	---	---	---	---

La ventaja de este sistema es que el cero tiene una única representación.

0      

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Ca1      

1	1	1	1	1	1	1	1
							1

-0 (Ca2)      

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

← Desbordamiento

Así pues ganamos una representación, se puede representar un número más.

$$\text{Rango } -(2^{n-1}) \leq x \leq (2^{n-1} - 1)$$

Si  $n = 8$  bits:

Nº repr= 256

Rango  $-128 \leq x \leq 127$

$22_{10}$ 

0	0	0	1	0	1	1	0
---	---	---	---	---	---	---	---

$-22_{10}$ 

1	1	1	0	1	0	1	0
---	---	---	---	---	---	---	---

$-1_{10}$ 

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

$-127_{10}$ 

1	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

$-128_{10}$ 

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

El complemento a 2 se utiliza muy a menudo porque es más fácil realizar algunas operaciones aritméticas que con otras representaciones.

## Representación sesgada

Esta representación no utiliza ningún bit para el signo, todos los bits se utilizan para representar la magnitud. Esta magnitud se determina de forma muy parecida a la representación de números positivos salvo que en esta representación hay que sumar primero una cantidad, a la que llamaremos **exceso**. Con esta representación tampoco desperdiciamos una representación por culpa del cero.

$$\text{Exceso } 2^{n-1}$$

$$\text{Rango } -(2^{n-1}) \leq x \leq (2^{n-1} - 1)$$

Si  $n = 8$  bits:

Nº repr= 256

Exceso= **128**

Rango  $-128 \leq x \leq 127$

$$10_{10} \rightarrow \mathbf{128} + 10 = 138 \quad \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ \hline \end{array}$$

$$-10_{10} \rightarrow \mathbf{128} - 10 = 118 \quad \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline \end{array}$$

$$109_{10} \rightarrow \mathbf{128} + 109 = 237 \quad \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ \hline \end{array}$$

$$-109_{10} \rightarrow \mathbf{128} - 109 = 19 \quad \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ \hline \end{array}$$

$$0_{10} \rightarrow \mathbf{128} + 0 = 128 \quad \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

$$-128_{10} \rightarrow \mathbf{128} - 128 = 0 \quad \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

Representaciones para una palabra de 4 bits (n=4)					
Cantidad	Positivo	Signo-magnitud	Ca1	Ca2	Sesgada (Exceso=8)
-8	-	-	-	1000	0000
-7	-	1111	1000	1001	0001
-6	-	1110	1001	1010	0010
-5	-	1101	1010	1011	0011
-4	-	1100	1011	1100	0100
-3	-	1011	1100	1101	0101
-2	-	1010	1101	1110	0110
-1	-	1001	1110	1111	0111
0	0000	1000 0000	1111 0000	0000	1000
1	0001	0001	0001	0001	1001
2	0010	0010	0010	0010	1010
3	0011	0011	0011	0011	1011
4	0100	0100	0100	0100	1100
5	0101	0101	0101	0101	1101
6	0110	0110	0110	0110	1110
7	0111	0111	0111	0111	1111
8	1000	-	-	-	-
9	1001	-	-	-	-
10	1010	-	-	-	-
11	1011	-	-	-	-
12	1100	-	-	-	-
13	1101	-	-	-	-
14	1110	-	-	-	-
15	1111	-	-	-	-



## Suma aritmética de enteros

Al igual que en el sistema decimal, en el sistema binario también se puede realizar operaciones aritméticas como la suma, resta, multiplicación, división, etc. En el caso de la suma, es muy parecida a la suma decimal, aunque tan sólo se utilizan dos dígitos. Si la suma excede de 1, se agrega una unidad a la suma siguiente, es lo que llamamos acarreo (c).

Suma Aritmética			
0	+	0	0
0	+	1	1
1	+	0	1
1	+	1	0 (c=1)

$$\begin{array}{r}
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ \hline \end{array} & 37 \\
 + & \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ \hline \end{array} & 18 \\
 \hline
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ \hline \end{array} & 55
 \end{array}$$

$$\begin{array}{r}
 \begin{array}{ccccccc} & & & 1 & & 1 & & 1 \\ & & & \cap & & \cap & & \cap \\ \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline \end{array} & 87 \\
 + & \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ \hline \end{array} & 33 \\
 \hline
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline \end{array} & 120
 \end{array}$$

$$\begin{array}{r}
 \begin{array}{ccccccc} & & & 1 & & 1 & & 1 \\ & & & \cap & & \cap & & \cap \\ \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ \hline \end{array} & 230 \\
 + & \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ \hline \end{array} & 50 \\
 \hline
 \text{Desbordamiento} \rightarrow \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline \end{array} & 24
 \end{array}$$

En el caso de las representaciones con bit de signo, la suma de números positivos se realiza igualmente, teniendo en cuenta que ahora tenemos un bit menos. (para los números negativos se opera como una resta, lo que veremos en el punto siguiente):

$$\begin{array}{r}
 \begin{array}{ccccccc} & & & 1 & & 1 & & \\ & & & \cap & & \cap & & \\ \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ \hline \end{array} & 53 \\
 + & \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ \hline \end{array} & 19 \\
 \hline
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline \end{array} & 72
 \end{array}$$

$$\begin{array}{r}
 \begin{array}{ccccccc} & & & 1 & & 1 & & 1 \\ & & & \cap & & \cap & & \cap \\ \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ \hline \end{array} & 115 \\
 + & \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ \hline \end{array} & 25 \\
 \hline
 \text{Desbordamiento} \rightarrow \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline \end{array} & -12
 \end{array}$$

## Resta aritmética de enteros

Las restas no existen como tales en informática, lo que se hace es sumar en números negativos en complemento a 1 o en complemento a 2. Cuando se trata del primer caso, el número a restar se pasa a negativo (Ca1). Si hubiese desbordamiento, se suma una unidad al resultado.

$$\begin{array}{r} \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline \end{array} & 12 \\ + & \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ \hline \end{array} & -5 \\ \hline \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ \hline \end{array} & \\ + & \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 1 & & & & & & & & 1 \\ \hline \end{array} & \\ \hline \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ \hline \end{array} & 7 \end{array}$$

En el otro caso, el número a restar se pasa a negativo Ca2. Si hubiese desbordamiento, se ignora.

$$\begin{array}{r} \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline \end{array} & 12 \\ + & \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ \hline \end{array} & -5 \\ \hline \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline \text{X} & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ \hline \end{array} & 7 \end{array}$$

## Multiplicación de enteros por la base

La multiplicación por la base es muy sencilla en cualquier sistema de numeración. Como en la base decimal, se trata de añadir ceros por la derecha, la representación se irá desplazando por el registro hacia la izquierda. Si un 1 llega al extremo generará un desbordamiento.

$$\begin{array}{r} \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ \hline \end{array} & 29 \\ \times & & & & & & & 2^1 & 2 \\ \hline \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ \hline \end{array} & 58 \\ \hline \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ \hline \end{array} & 29 \\ \times & & & & & & & 2^4 & 16 \\ \hline \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline \end{array} & 208 \end{array}$$

## División de enteros por la base

La división, de forma parecida a la multiplicación se efectúa añadiendo ceros, esta vez por la izquierda. La representación se irá desplazando hacia la derecha por el registro y se irá truncando.

$$\begin{array}{r}
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ \hline \end{array} & 29 \\
 \div & 2^1 & 2 \\
 \hline
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ \hline \end{array} & 14 \\
 \hline
 \end{array}$$
  

$$\begin{array}{r}
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ \hline \end{array} & 29 \\
 \div & 2^3 & 8 \\
 \hline
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline \end{array} & 3 \\
 \hline
 \end{array}$$

### Operaciones lógicas binarias

En el sistema binario existen, además de las tradicionales operaciones aritméticas, una serie de operaciones que tratan de representar silogismos lógicos haciendo coincidir los valores 0 y 1 con falso y verdadero respectivamente. Estas operaciones no generan acarreo ni desbordamiento.

Suma Lógica (OR)			
0	∨	0	0
0	∨	1	1
1	∨	0	1
1	∨	1	1

$$\begin{array}{r}
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ \hline \end{array} \\
 \vee \\
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline \end{array} \\
 \hline
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ \hline \end{array}
 \end{array}$$

Producto Lógico (AND)			
0	∧	0	0
0	∧	1	0
1	∧	0	0
1	∧	1	1

$$\begin{array}{r}
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ \hline \end{array} \\
 \wedge \\
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline \end{array} \\
 \hline
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ \hline \end{array}
 \end{array}$$

Exclusión (XOR)			
0	⊕	0	0
0	⊕	1	1
1	⊕	0	1
1	⊕	1	0

$$\begin{array}{r}
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ \hline \end{array} \\
 \oplus \\
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline \end{array} \\
 \hline
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ \hline \end{array}
 \end{array}$$

Negación Lógica (NOT)		
~	0	1
~	1	0

$$\begin{array}{r}
 \sim \\
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline \end{array} \\
 \hline
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ \hline \end{array}
 \end{array}$$

En informática es habitual utilizar sentencias lógicas para programar operaciones condicionales. Cualquier expresión condicional se puede expresar como una función lógica que dependiendo de las condiciones de partida devuelva los valores de verdadero (1) o falso (0):

**“Si llueve hoy y no llevo paraguas, me mojaré”**

Llover =  $L$

Llevar paraguas =  $P$

Mojarse =  $M$

$$L \wedge \sim P \rightarrow M$$

$L$	$P$	$M = L \wedge \sim P$
0	0	0
0	1	0
1	0	1
1	1	0

No llover ( $L = 0$ ) y salir sin paraguas ( $P = 0$ ) implica no mojarme ( $M = 0$ )

No llover ( $L = 0$ ) y salir con paraguas ( $P = 1$ ) implica no mojarme ( $M = 0$ )

Llover hoy ( $L = 1$ ) y salir sin paraguas ( $P = 0$ ) implica mojarme ( $M = 1$ )

Llover hoy ( $L = 1$ ) y salir con paraguas ( $P = 1$ ) implica no mojarme ( $M = 0$ )

**“Si el nombre de usuario o la contraseña es incorrecta, se notificará un error”**

Usuario correcto =  $U$

Contraseña correcta =  $C$

Error =  $E$

$$\sim U \vee \sim C \rightarrow E$$

$U$	$C$	$E = \sim U \vee \sim C$
0	0	1
0	1	1
1	0	1
1	1	0

**“Para que funcione un motor de coche debes repostar gasolina o bien gasoil”**

Gasolina =  $G1$

Gasoil =  $G2$

Funcionar =  $F$

$$G1 \oplus G2 \rightarrow F$$

$G1$	$G2$	$F = G1 \oplus G2$
0	0	0
0	1	1
1	0	1
1	1	0

## b. Representación de los números reales

El conjunto de los números reales comprende a todos las cantidades numéricas, eso incluye negativos, fracciones, raíces, y otros números irracionales. En la mayoría de las ocasiones no se podrán representar estos números de forma exacta (como ocurría en representaciones enteras) sino que tendremos que recurrir a representaciones aproximadas. Esto hará que las cantidades a veces se vean truncadas. La precisión de estas representaciones dependerá del número de bits, a mayor ancho de palabra mayor precisión.

La representación en **coma o punto flotante** usa la notación exponencial, que nos permitirá representar un amplio rango de números:

$$N = m \cdot 2^{exp}$$

$$49|_{10} = 110001|_2 = 1,10001 \cdot 2^5$$

$$-5,1875|_{10} = -101,0011|_2 = -1,010011 \cdot 2^2$$

$$0,1328125|_{10} = 0,0010001|_2 = 1,0001 \cdot 2^{-3}$$

De un número en coma flotante se necesita almacenar en los registros el valor de la parte significativa o **mantisa** ( $m$ ), su respectivo signo ( $s$ ) y el valor del **exponente** ( $exp$ ) en representación sesgada. El valor del primer uno (el de la parte entera) ni la coma se almacenan, porque se suponen implícitos. En algunos casos se utilizan dos palabras para representar una cantidad con extrema precisión.

### Precisión simple (*single o float*)

Los números en representación se almacena en una palabra de **32 bits**, de estos se utilizan 8 bits para el exponente en representación sesgada y 24 bits para la mantisa en Ca1 (un bit se reserva para su signo).



$$20,75|_{10} = 10100,11|_2 = +1,010011 \cdot 2^4$$

$$s = 0$$

$$exp = 4 + 2^7 = 4 + 128 = 132|_{10} = 10000100 \text{ (Representación sesgada)}$$

$$m = 010011000000000000000000 \text{ (Ca1)}$$

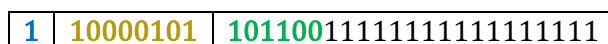


$$-20,75|_{10} = -10100,11|_2 = -1,010011 \cdot 2^4$$

$$s = 1$$

$$exp = 4 + 2^7 = 4 + 128 = 132|_{10} = 10000100 \text{ (Representación sesgada)}$$

$$m = 101100111111111111111111 \text{ (Ca1)}$$

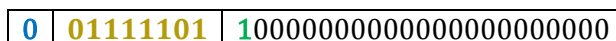


$$+0,1875|_{10} = 0,0011|_2 = 1,1 \cdot 2^{-3}$$

$$s = 0$$

$$exp = -2 + 2^7 = -2 + 128 = 126|_{10} = 01111101 \text{ (Representación sesgada)}$$

$$m = 100000000000000000000000 \text{ (Ca1)}$$



### Doble precisión (*double*)

Los números en representación se almacena en una palabra de **64 bits** o en dos palabras de 32 bits, de estos se utilizan 11 bits para el exponente en representación sesgada y 53 bits para la mantisa en Ca1 (un bit se reserva para su signo).



### c. Representación de valores alfanuméricos

Además de cantidades numéricas, en un sistema informático se puede almacenar otros tipos de datos no matemáticos: texto, sonidos, imágenes, vídeo, etc. Cuando se trata de almacenar caracteres alfanuméricos englobamos a todas las letras del abecedario en mayúscula y en minúscula, los signos de puntuación, los caracteres que simbolizan los números del 0 al 9 (sin significado matemático), y otros símbolos de texto.

#### Código ASCII

Utiliza **7 bits** para almacenar cada carácter permitiendo en total un repertorio de  $2^7=128$  caracteres. Lo suficiente para el alfabeto en mayúscula, minúscula, los caracteres del 0 al 9 y los signos de puntuación (lo que podríamos encontrar en una antigua máquina de escribir). Se reservan algunas combinaciones para códigos de control de uso interno. Para numerar cada carácter se utiliza el sistema hexadecimal.

0	1	2	3	4	5	6	7		
000	001	010	011	100	101	110	111		
00 NUL	10 DLE	20	30 0	40 @	50 P	60 `	70 p	0000	0
01 SOH	11 DC1	21 !	31 1	41 A	51 Q	61 a	71 q	0001	1
02 STX	12 DC2	22 "	32 2	42 B	52 R	62 B	72 r	0010	2
03 ETX	13 DC3	23 #	33 3	43 C	53 S	63 C	73 s	0011	3
04 EOT	14 DC4	24 \$	34 4	44 D	54 T	64 D	74 t	0100	4
05 ENQ	15 NAK	25 %	35 5	45 E	55 U	65 e	75 u	0101	5
06 ACK	16 SYN	26 &	36 6	46 F	56 V	66 F	76 v	0110	6
07 BEL	17 ETB	27 ' ,	37 7	47 G	57 W	67 g	77 w	0111	7
08 BS	18 CAN	28 (	38 8	48 H	58 X	68 h	78 x	1000	8
09 HT	19 EM	29 )	39 9	49 I	59 Y	69 i	79 y	1001	9
0A LF	1A SUB	2A *	3A :	4A J	5A Z	6A j	7A z	1010	A
0B VT	1B ESC	2B +	3B ;	4B K	5B [	6B k	7B {	1011	B
0C FF	1C FS	2C ,	3C <	4C L	5C \	6C l	7C	1100	C
0D CR	1D GS	2D -	3D =	4D M	5D ]	6D m	7D }	1101	D
0E SO	1E RS	2E .	3E >	4E N	5E ^	6E n	7E ~	1110	E
0F SI	1F US	2F /	3F ?	4F O	5F _	6F o	7F DEL	1111	F

## Código ASCII extendido

Esta ampliación de **8 bits** añade 128 nuevos (hasta un total de 256) caracteres para añadir el código de caracteres extranjeros y símbolos gráficos. Cada carácter tiene ahora el tamaño de **1 byte**. La primera parte de la tabla es igual a la del ASCII de 7 bits, por lo que es totalmente compatible con esta codificación. Este fue uno de los códigos más utilizados y aunque ahora está superado por el *Unicode*, aun se utiliza en ocasiones.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111		
00 NUL	10 ►	20	30 0	40 @	50 P	60 `	70 p	80 Ç	90 É	A0 á	B0 ☒	C0 Ł	D0 ŏ	E0 Ó	F0 -	0000	0
01 ☺	11 ◄	21 !	31 1	41 A	51 Q	61 a	71 q	81 ü	91 æ	A2 í	B1 ☒	C1 ⊥	D1 Ð	E1 ß	F1 ±	0001	1
02 ☹	12 ⇕	22 "	32 2	42 B	52 R	62 b	72 r	82 é	92 Æ	A2 ó	B2 ☒	C2 ⊥	D2 Ê	E2 Ô	F2 =	0010	2
03 ♥	13 !!	23 #	33 3	43 C	53 S	63 c	73 s	83 â	93 ô	A3 ú	B3	C3	D3 Ë	E3 Ò	F3 ¾	0011	3
04 ♦	14 ¶	24 \$	34 4	44 D	54 T	64 d	74 t	84 ä	94 ö	A4 ñ	B4	C4 —	D4 È	E4 õ	F4 ¶	0100	4
05 ♣	15 §	25 %	35 5	45 E	55 U	65 e	75 u	85 à	95 ò	A5 Ñ	B5 Á	C5 †	D5 ı	E5 Õ	F5 §	0101	5
06 ♠	16 —	26 &	36 6	46 F	56 V	66 f	76 v	86 â	96 û	A6 a	B6 Â	C6 ã	D6 Í	E6 μ	F6 ÷	0110	6
07 •	17 ⇕	27 ´	37 7	47 G	57 W	67 g	77 w	87 ç	97 ù	A7 °	B7 À	C7 Ã	D7 Î	E7 Þ	F7 ´	0111	7
08 ◼	18 ↑	28 (	38 8	48 H	58 X	68 h	78 x	88 ê	98 ÿ	A8 ÿ	B8 ©	C8 ℒ	D8 Ï	E8 Ñ	F8 °	1000	8
09 ○	19 ↓	29 )	39 9	49 I	59 Y	69 i	79 y	89 ë	99 Ö	A9 ®	B9 ¶	C9 ¶	D9 Ĵ	E9 Ú	F9 ..	1001	9
0A ◼	1A →	2A *	3A :	4A J	5A Z	6A j	7A z	8A è	9A Ü	AA ¬	BA ¶	CA ¶	DA ¶	EA Û	FA ·	1010	A
0B ♂	1B ←	2B +	3B ;	4B K	5B [	6B k	7B {	8B ï	9B ø	AB ½	BB ¶	CB ¶	DB █	EB Û	FB 1	1011	B
0C ♀	1C ⊥	2C ,	3C <	4C L	5C \	6C l	7C	8C î	9C £	AC ¼	BC ¶	CC ¶	DC █	EC ý	FC 3	1100	C
0D 🎵	1D ⇕	2D -	3D =	4D M	5D ]	6D m	7D }	8D ì	9D Ø	AD ÿ	BD ¢	CD =	DD	ED Ý	FD 2	1101	D
0E 🎵	1E ▲	2E .	3E >	4E N	5E ^	6E n	7E ~	8E Ä	9E ×	AE «	BE ¥	CE ¶	DE Ì	EE -	FE █	1110	E
0F ☀	1F ▼	2F /	3F ?	4F O	5F _	6F o	7F ◻	8F Å	9F f	AF »	BF ¶	CF ¶	DF █	EF ´	FF	1111	F

## Código Unicode

Es la codificación que se utiliza en los sistemas modernos y por los navegadores de internet, utiliza **16 bits** para representar cada carácter, aunque a la hora de codificar dicho carácter se pueden utilizar hasta 32 bits por lo que contiene el conjunto de caracteres de los lenguajes más importantes del mundo, el objetivo de las versiones sucesivas es representar cada uno de los símbolos de cualquier idioma del planeta incluyendo puntuación, símbolos especiales, símbolos matemáticos, etc. Antes de la aparición del Unicode, cada idioma debía poseer su propio sistema de codificación o bien adaptarse al de otro idioma. A veces esto daba problemas de compatibilidad.

El Unicode es compatible con el ASCII de tal forma que cualquier cadena de texto en estos sistemas se representa sin cambios ya que los primeros símbolos del Unicode corresponden con los del ASCII.

Cada carácter Unicode se denomina con la notación **U+XXXX** siendo X un numero en hexadecimal.

**M** (ASCII) → **4D**

**M** (Unicode) → U+**004D**

**€** (Unicode) → U+**20AC**

A veces utilizamos dos códigos de 16 bits, o sea **32 bits** en total, para representar un carácter compuesto por dos caracteres puros, como el caso de la puntuación:

**A** (U+0041) + **º** (U+030A) → **Å** (U+0041 + U+030A)

Actualmente existen diversos tipos de variantes de la codificación Unicode, pero la que está teniendo más éxito es la codificación **UTF-8** que usa símbolos de longitud variable:

Se utiliza **1 byte** para almacenar los caracteres compatibles con el código ASCII.

Se utiliza **2 bytes** para almacenar un carácter Unicode puro.

Se utiliza **4 bytes** para almacenar un carácter compuesto por dos caracteres puros.