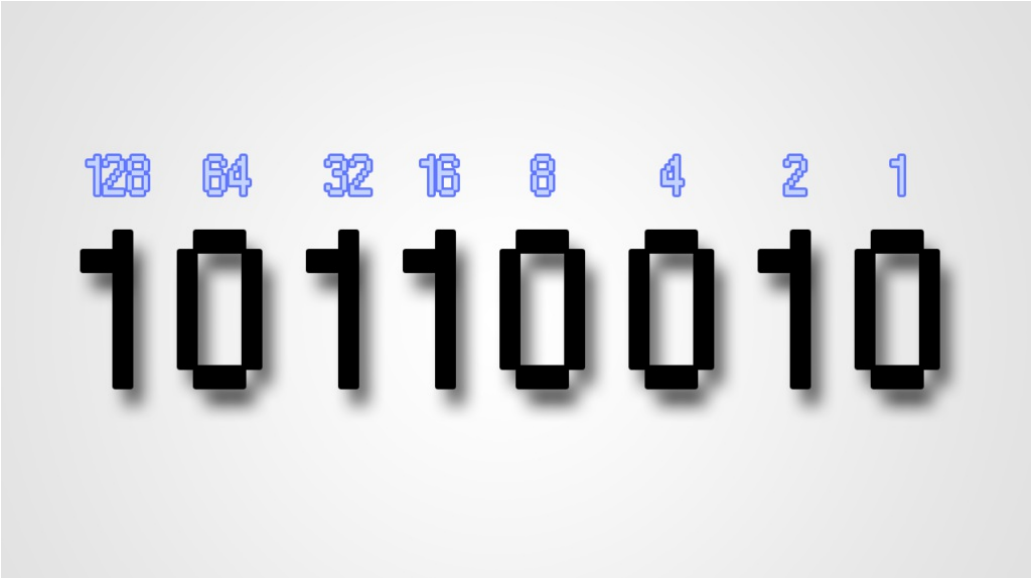
Sistemas de numeración usados en la tecnología digital

Publicado el El Informatico - 12 de diciembre de 2021 -



Artículo anterior: Bits y bytes. Cómo funciona la información digital.

Aprende los sistemas de numeración binario, octal y hexadecimal de forma sencilla con éste artículo.

Sistema binario

Aunque ya expliqué en otro artículo en qué consiste el sistema de numeración binario y por qué se usa en la tecnología digital, lo vuelvo a explicar aquí por redundancia.

Para explicar lo que es el sistema binario, tomemos por ejemplo un sistema de numeración que nosotros conocemos muy bien y que usamos a diario, que es el sistema **decimal**. Cuando nosotros contamos en decimal, contamos con un único dígito desde el número 0 hasta el número 9. Pero **como es un sistema decimal, sólo tenemos diez números para contar por cifra**. Así que cuando queremos pasar del 9 al siguiente valor, tenemos que añadir otro dígito a la izquierda. Que en éste caso sería el 10, y que representa cuántas vueltas hemos dado al primer dígito.

Así volveríamos a contar desde el 10 al 19, y tendríamos que sumar 1 al segundo dígito para ir al siguiente número que es el 20. Repite ésto hasta el 99, y tendrás que añadir otro dígito a la izquierda para pasar al número 100. Y así constantemente.

El sistema de numeración binario es igual, **pero con la diferencia de que contamos desde 0 hasta 1**. Y como es un sistema binario, sólo tenemos dos dígitos para contar. Así que **la siguiente cifra despues del 1, sería el 10**. Despues contaríamos 10, 11, 100, 101, 110, 111, 1000, y así constantemente.

DECIMAL	BINARIO
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000
17	10001
18	10010
19	10011
20	10100

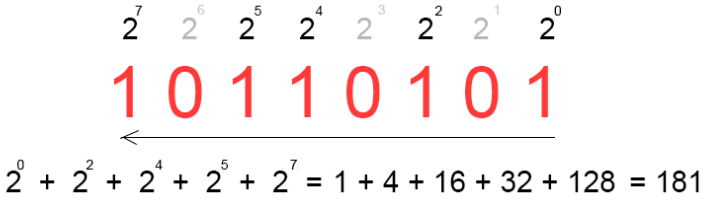
Los primeros 20 números, representados en binario

Los números binarios se suelen representar con la letra 'b' a la izquierda. Por ejemplo, b1011 significa que es un 11 en binario.

Conversión de binario a decimal

Convertir un valor binario en un valor decimal es sencillo. Dado que cada bit puede tener 2 valores, el valor de cada bit se representa en base 2 con exponente según la posición del bit (de derecha a izquierda), menos 1. Es decir, empezando por 2 elevado a 0 (Porque 2 = 1). El valor de cada bit será 0 o 1, multiplicado por 2 elevado a la posición del bit (Valor del bit = { 0 1 } 2 [1]). **El valor del número binario en decimal será la suma de los valores de todos los bits cuyo valor sea 1** ([1 , 2 ... n]).

Una imágen para dejarlo todo mucho más claro:



En el caso del número binario 10110101, tomamos el valor de todos los bits que se encuentren en 1, que son los bits 0,2,4,5,7. El valor de éstos bits son 1,4,16,32 y 128 respectivamente. **Si hacemos el sumatorio de todos éstos valores (sumamos todos los valores), el resultado es 181 en decimal**.

Conversión de decimal a binario

Supongamos que queremos representar el valor 213 decimal en binario.

Para representar un valor en binario, primero debemos encontrar el número de bits necesarios para representarlo. Como el valor de cada bit es una potencia de 2, debemos encontrar la potencia de dos cuyo valor sea igual o inmediatamente superior al número que queramos representar. En éste caso, al ser 213, **la potencia inmediatamente superior será 2 elevado a 8 (256)**. Es decir, **9 bits** (recuerda, el primer exponente es 0). Pero, **¡Ojo!** Porque **hay que tener en cuenta si el valor decimal es igual o inferior a la potencia**.

Si el valor de la potencia fuese igual al número que queremos representar, serían 9 bits. El último bit será un 1 y el resto seran ceros (100000000). Esto sería si quisieramos representar el valor 256.

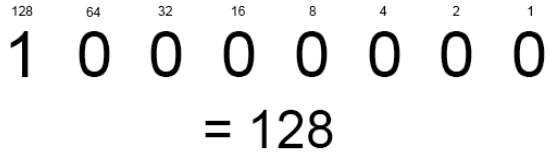
Pero como en éste caso el número es inferior (213 < 256), **el último bit sería 0. En este caso, el número de bits será el exponente, menos 1. Ya que el último bit no cuenta**. Es decir, que **necesitamos 8 bits**.

Recuerda que el orden de los bits va siempre de derecha a izquierda, siendo el primer bit (de menor valor) el bit más a la derecha, y el último (De mayor valor) el más a la izquierda.

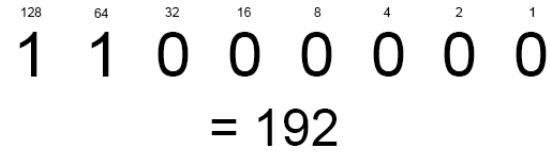
En el siguiente paso, vamos recorriendo todos los bits de izquierda a derecha. Es decir, de mayor valor (último), a menor valor (primero). **El objetivo es colocar**

todos los bits a 1, siempre y cuando al hacerlo la suma del valor de los bits no exceda el valor decimal que queremos representar. Es decir, que la suma de los valores de cada bit no exceda, en éste caso, 213.

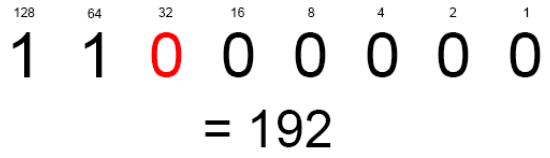
El último bit en éste caso tendría un valor de 2 = 128. Como 128 es menor que 213, **pondremos éste bit a 1**.



El siguiente bit tiene un valor de 2 = 64. 128 + 64 = 192. Como 192 es menor que 213, ponemos éste bit también a 1.

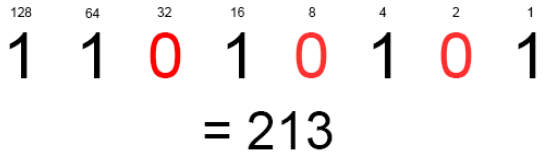


El siguiente bit tendría un valor de 32. 192 + 32 = 224. 224 es **mayor** que 213, con lo cuál éste bit se queda en 0.



Seguímos repitiendo éste proceso con el resto de bits. El siguiente serían 16. 192+16 = 208. 208 < 213, con lo que se pondría a 1. El siguiente serían 8. 208+8 = 216, 216 > 213 (se queda a 0). El siguiente serían 4. 208 + 4 = 212, 212 < 213 (1). El siguiente serían 2. 212 + 2 = 214, 214 > 213 (0). Y por último 212 + 1 = 213. 213 = 213 (1).

La representación binaria de la cifra 213 sería 11010101.

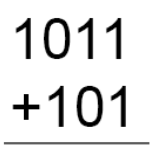


Aritmética binaria

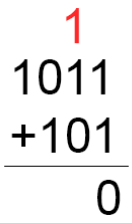
La aritmética en binario funciona exáctamente de la mísma manera que en cualquier otro sistema de numeración, pero teniendo en cuenta de que cada dígito sólo tiene un rango entre 0 y 1, y que cada vuelta que demos añadirá o sustraerá 1 al acarreo del siguiente dígito.

Por ejemplo, si sumamos 1 a la cifra binaria 110, el resultado sería 111. Pero si sumamos 1 a la cifra 1011, el resultado sería 1100.

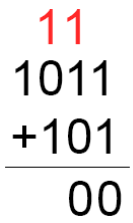
Por poner otro ejemplo, si quisieramos sumar 101 (5) a la cifra 1011 (11):



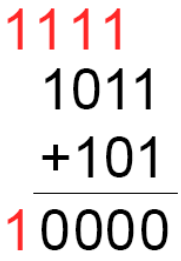
Al igual que en decimal, sumamos cada dígito de menor a mayor (de derecha a izquierda), teniendo en cuenta el acarreo (lo que nos «llevamos» de la suma del anterior dígito para añadirselo al siguiente). En éste caso, empezamos por los bits más a la derecha, que sería 1+1=10. Como el valor es 10, nos llevamos 1 al acarreo.



El siguiente bit sería 1+0, pero tenemos un 1 en el acarreo, así que el resultado sería 1+0+1 =10. Igualmente nos llevamos 1 al acarreo.



Repetimos ésta acción con el resto de bits. 0+1+1=10, 1+1=10, y el último acarreo se lo añadimos al final al resultado. **El resultado serían 10000 (16)**.



De igual manera, es posible realizar operaciones de **resta**, **multiplicación** y **división** de manera similar a como lo hacemos en decimal.

Números negativos en binario y Complemento a dos (C2)

Representar un valor negativo en binario es muy sencillo. Normalmente cuando representamos un valor negativo en decimal, usamos el símbolo «-«. Por ejemplo, -20 sería un valor de 20 en negativo (por debajo de 0).

En binario, añadimos lo que se denomina un **bit de signo**. En éste caso, el valor

10100 (20), pasaría a ser 110100, siendo el último bit el bit de signo. De éste modo, **todos los valores negativos empiezan con un uno**.

Ésto tiene más sentido cuando trabajamos con un número de bits fijo. Y por eso, por norma general, **solemos trabajar con un número de bits fijo**. Siendo el último el bit de signo. Por ejemplo, cuando queremos representar un valor negativo con un único byte (8 bits), el bit de mayor valor sería el de signo. **Pudiendo así representar valores de entre -127 (10000000) hasta 127 (01111111), siendo el primer valor negativo (-1) 11111111**.

Cuando trabajamos con **valores de bits fijos**, podemos invertir esos valores usando una técnica conocida como el **complemento a dos**.

El complemento a dos consiste en dos pasos: primero invertimos el valor de todos y cada uno de los bits (los unos pasan a ser ceros y los ceros pasan a ser unos). Y despues, al resultado de ésta operación, le sumamos un 1. El resultado será el valor invertido.

Por ejemplo, el número 27 en binario usando 8 bits con signo se representaría como 00011011.

Para representar su valor en negativo, primero invertimos todos los bits. Los ceros pasan a ser unos, y vice versa.

```
0 0 0 1 1 0 1 1
1 1 1 0 0 1 0 0
```

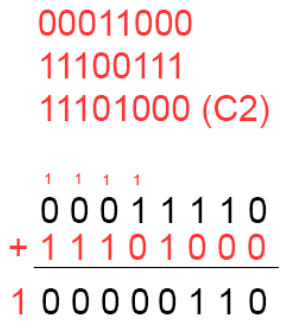
Y a éste valor, le sumamos 1.

```
0 0 0 1 1 0 1 1
1 1 1 0 0 1 0 0
+             1
1 1 1 0 0 1 0 1
```

El resultado (11100101) sería el equivalente en binario de -27. Puedes volver a realizar estos dos pasos sobre éste número (11100101) para comprobar que, en efecto, el resultado será otra vez 00011011 (+27).

Restas en binario usando C2

Cuando ambas cifras son positivas, podemos realizar una resta invirtiendo el valor de la segunda cifra y sumando ambos valores. Por ejemplo, para restar 30-24, primero convertimos 24 (00001100) en su forma negativa usando C2 (11110100), y después sumamos (00011110+11110100) Sería el equivalente de realizar la operación 30 + -24.



Pero, ¿Qué pasa con el acarreo final? Como tenemos un número fijo de bits (8 bits), en éste caso el acarreo **se ignora**, ya que queda fuera de los límites de bits disponibles. Es decir, que el resultado final de la operación sería 00000110 (6).

Cuando hable sobre procesadores, vereis que éste acarreo adiccional va a otra parte. Pero en éste caso, símplemente se ignora.

¿Por qué es ésto importante? **Porque éste es el método que se usa en los componentes como los procesadores a la hora de restar**.

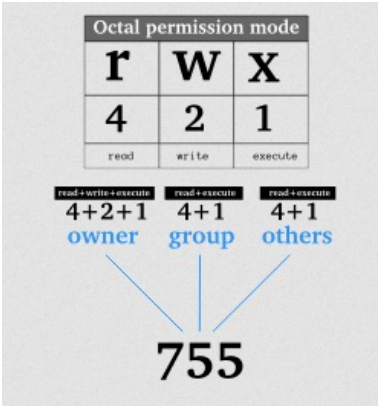
Numeración en Octal

De igual forma que en binario tenemos un sistema de numeración con dos estados por dígito, **en octal tenemos hasta 8 valores por dígito**, del 0 al 7. Es decir, contamos del 0 al 7 para pasar al 10. Los valores de cada dígito son en éste caso 8 elevado a la posición del dígito (empezando siempre por 0). Es decir, $8^{} = 1$, $8^{1} = 8$, $8^{2} = 16$, etcétera.

La razón por la que se usa la numeración octal es porque a veces queremos usar grupos de por ejemplo 3 bits, con valores del 0 al 7. El ejemplo más claro de ésto es el sistema de permisos en UNIX (Y los sistemas basados en Linux), donde el valor de los permisos se representa en octal y cada cifra representa los permisos para un usuario o grupo específico.

En el sistema de permisos UNIX, hay 3 bits donde cada bit representa un parámetro (lectura, escritura y ejecución). Los bits que se encuentren a 1 permiten al usuario o grupo realizar dicha acción. Hay 3 grupos con 3 conjuntos de 3 bits cada uno (usuario, grupo y «otros»).

Como son 3 bits, **el rango máximo que podemos representar numéricamente por grupo es del 0 al 7. Por eso se usa el sistema octal en éste caso**.



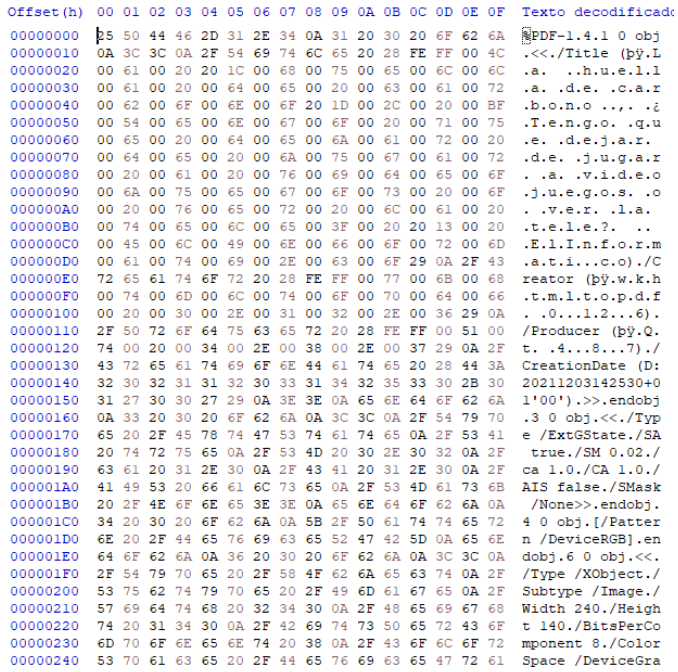
Sistema octal para los permisos de un archivo o directorio en UNIX. Fuente: freesvg.org

Sistema de numeración hexadecimal

El sistema hexadecimal es el más importante, después del binario. Al igual que en los anteriores casos, cada dígito nos permite contar desde 0 hasta, en éste caso, 15. Para pasar al 10.

Pero, ¿Cómo representamos cifras superiores al 9 en éste sistema? Como nuestra numeración sólo dispone de 10 dígitos del al 9, **usamos caracteres alfanuméricos para representar las cifras del 10 al 15**. Es decir, A representaría el 10, B el 11, C el 12, D el 13, E el 14, y F el 15. De éste modo, decimos que **podemos contar desde 0, hasta F, para pasar a 10**.

En informática se usa el hexadecimal para representar el valor de cada byte con sólo dos cifras. Del 0 al FF (255). A la hora de representar multitud de valores en una pantalla, es mucho más fácil de organizar y representar de ésta manera, ayudando a que su lectura sea más símple.



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Texto decodificad
00000000	25	50	44	46	2D	31	2E	34	0A	31	20	30	20	6F	62	6A	%PDF-1.4.1 0 obj
00000010	0A	3C	3C	0A	2F	54	69	74	6C	65	20	28	FE	FF	00	4C	.<<./Title (þÿ.L
00000020	00	61	00	20	20	1C	00	68	00	75	00	65	00	6C	00	6C	.a. ..h.u.e.l.l
00000030	00	61	00	20	00	64	00	65	00	20	00	63	00	61	00	72	.a. .d.e. .c.a.r
00000040	00	62	00	6F	00	6E	00	6F	20	1D	00	2C	00	20	00	BF	.b.o.n.o ..,. .¿
00000050	00	54	00	65	00	6E	00	67	00	6F	00	20	00	71	00	75	.T.e.n.g.o. .q.u
00000060	00	65	00	20	00	64	00	65	00	6A	00	61	00	72	00	20	.e. .d.e.j.a.r.
00000070	00	64	00	65	00	20	00	6A	00	75	00	67	00	61	00	72	.d.e. .j.u.g.a.r
00000080	00	20	00	61	00	20	00	76	00	69	00	64	00	65	00	6F	. .a. .v.i.d.e.o
00000090	00	6A	00	75	00	65	00	67	00	6F	00	73	00	20	00	6F	.j.u.e.g.o.s. .o
000000A0	00	20	00	76	00	65	00	72	00	20	00	6C	00	61	00	20	. .v.e.r. .l.a.
000000B0	00	74	00	65	00	6C	00	65	00	3F	00	20	20	13	00	20	.t.e.l.e.?. ..
000000C0	00	45	00	6C	00	49	00	6E	00	66	00	6F	00	72	00	6D	.E.l.I.n.f.o.r.m
000000D0	00	61	00	74	00	69	00	2E	00	63	00	6F	29	0A	2F	43	.a.t.i...c.o)./C
000000E0	72	65	61	74	6F	72	20	28	FE	FF	00	77	00	6B	00	68	reator (þÿ.w.k.h
000000F0	00	74	00	6D	00	6C	00	74	00	6F	00	70	00	64	00	66	.t.m.l.t.o.p.d.f
00000100	00	20	00	30	00	2E	00	31	00	32	00	2E	00	36	29	0A	. .0...1.2...6).
00000110	2F	50	72	6F	64	75	63	65	72	20	28	FE	FF	00	51	00	/Producer (þÿ.Q.
00000120	74	00	20	00	34	00	2E	00	38	00	2E	00	37	29	0A	2F	t. .4...8...7)./
00000130	43	72	65	61	74	69	6F	6E	44	61	74	65	20	28	44	3A	CreationDate (D:
00000140	32	30	32	31	31	32	30	33	31	34	32	35	33	30	2B	30	20211203142530+0
00000150	31	27	30	30	27	29	0A	3E	3E	0A	65	6E	64	6F	62	6A	1'00').>>.endobj
00000160	0A	33	20	30	20	6F	62	6A	0A	3C	3C	0A	2F	54	79	70	.3 0 obj.<<./Typ
00000170	65	20	2F	45	78	74	47	53	74	61	74	65	0A	2F	53	41	e /ExtGState./SA
00000180	20	74	72	75	65	0A	2F	53	4D	20	30	2E	30	32	0A	2F	true./SM 0.02./
00000190	63	61	20	31	2E	30	0A	2F	43	41	20	31	2E	30	0A	2F	ca 1.0./CA 1.0./
000001A0	41	49	53	20	66	61	6C	73	65	0A	2F	53	4D	61	73	6B	AIS false./SMask
000001B0	20	2F	4E	6F	6E	65	3E	3E	0A	65	6E	64	6F	62	6A	0A	/None>>.endobj.
000001C0	34	20	30	20	6F	62	6A	0A	5B	2F	50	61	74	74	65	72	4 0 obj.[/Patter
000001D0	6E	20	2F	44	65	76	69	63	65	52	47	42	5D	0A	65	6E	n /DeviceRGB].en
000001E0	64	6F	62	6A	0A	36	20	30	20	6F	62	6A	0A	3C	3C	0A	dobj.6 0 obj.<<.
000001F0	2F	54	79	70	65	20	2F	58	4F	62	6A	65	63	74	0A	2F	/Type /XObject./
00000200	53	75	62	74	79	70	65	20	2F	49	6D	61	67	65	0A	2F	Subtype /Image./
00000210	57	69	64	74	68	20	32	34	30	0A	2F	48	65	69	67	68	Width 240./Heigh
00000220	74	20	31	34	30	0A	2F	42	69	74	73	50	65	72	43	6F	t 140./BitsPerCo
00000230	6D	70	6F	6E	65	6E	74	20	38	0A	2F	43	6F	6C	6F	72	mponent 8./Color
00000240	53	70	61	63	65	20	2F	44	65	76	69	63	65	47	72	61	Space /DeviceGra

Representación de valores hexadecimales en un editor hexadecimal.
Cada cifra se corresponse con el valor de un único byte.

Conversión de decimal a hexadecimal y hexadecimal a decimal

Para convertir un valor decimal a hexadecimal, primero convertimos el valor decimal en un valor binario, y despues dividimos dicho valor en **grupos de 4 bits**, de derecha a izquierda. Si alguno de los grupos no tiene 4 bits, le añadimos tantos ceros como sean necesarios a la izquierda.

Por ejemplo, para representar el número 137 en hexadecimal, primero lo convertimos a binario siguiendo los pasos ya mencionados.

$$137 = 10001001$$

Dividimos la cifra en grupos de 4 bits, **de derecha a izquierda**. Recuerda, si alguno de los grupos a la izquierda no tuviese 4 bits, añadele ceros a la izquierda hasta que tenga al menos 4 bits.

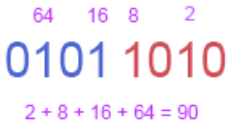
Como trabajamos ahora con grupos de 4 bits, el valor de cada grupo de 4 bits (nibble) tiene un rango de valores del 0 al 15. En hexadecimal, ésto es del 0 al F. **Símplemente representa el valor de cada nibble en hexadecimal**.

El número 137 en hexadecimal sería **89**.

Para representar los valores en hexadecimal, normalmente añadimos el prefijo 0x a la izquierda, o una 'H' a la derecha (89H, 0x89).

Para convertir cualquier valor hexadecimal a decimal, seguimos los pasos inversos. Por ejemplo, para pasar el número 0x5A a decimal, primero pasamos los dígitos 5 y A a binario.

Unimos ambas cifras binarias y pasamos el valor binario a decimal



El valor en decimal de 0x5A es **90**.

ANTERIOR

Bits y bytes. Cómo funciona la información digital.

SIGUIENTE

Historia de los medios de almacenamiento digitales

Buscar ...

Entradas Recientes

- La memoria del ordenador, a fondo
- Operaciones lógicas (Lógica booleana) y Bit Shifting
- Historia de los medios de almacenamiento digitales
- Sistemas de numeración usados en la tecnología digital
- Bits y bytes. Cómo funciona la información digital.
- Desinformación, Bulos, Fake News y manipulación de la información

Categorías

Actualidad | Android | Básicos | Ciberseguridad | Clima | Criptografía | Emulación / Virtualización | FOSS | Hacking | Hardware | Informática | Internet | Juegos | Opinion | Otros | Personal | Privacidad | Programación | Tecnología | Time Machine | Tutoriales

RSS

Subscribirse al feed RSS