SISTEMA BINARIO

Todo sistema de comunicación necesita un mensaje, un emisor del mismo, un receptor, un canal para transmitir la información y un código que tanto el emisor como el receptor sean capaces de entender. Así, por ejemplo, cuando dos seres humanos hablan entre sí (emisor y receptor) utilizan un idioma que ambos entiendan (código) y transmiten el mensaje mediante el sonido de la voz y su propagación por el aire (canal).

El avance de las comunicaciones ha ido dando lugar a modificaciones de los canales usados para transmitir la información, así como los códigos empleados. En la era digital, el desarrollo de sistemas que usan la electricidad y las ondas electromagnéticas como canales para emitir información ha supuesto una vía de desarrollo de códigos que se adapten a los mismos.

La transmisión de información mediante estos sistemas puede responder a formas alámbricas (por un cable e impulsos eléctricos) o inalámbricas (mediante la propagación de ondas electromagnéticas).

Los códigos empleados para ambos canales responden a dos formas:

- Transmisión analógica: los valores que se transmiten son continuos, es decir, se mide la intensidad de la señal que llega, y esta toma una infinidad de valores entre unos límites.
- Transmisión digital: Los valores que se transmiten son discretos y adoptan un número concreto de valores. El caso mas habitual es utilizar sólo dos posibilidades, que responden a recibir señal (primera posibilidad) o no recibirla (segunda posibilidad).

En ambos casos, de la señal recibida se toman valores regularmente (es decir, se divide la misma en "trocitos" entre los cuales pasa un tiempo determinado y se mide el valor de la misma en cada "trocito"). En el caso de una señal analógica cada vez que tomemos un dato la señal tendrá una intensidad y ésta tendrá un significado. En el caso de una señal digital al tomar un dato obtendremos una señal o la ausencia de la misma.

Pero, ¿cómo se envía la información en un ordenador?. Partiendo de la base de que los ordenadores funcionan con electricidad y envían la información que procesan mediante impulsos eléctricos, el primer paso es entender porqué es mejor una señal digital. Una señal analógica necesita codificar la información (hablando de impulsos eléctricos) con diferentes voltajes, estableciendo un código que transforme la información en diferentes voltajes. El problema sería que si la señal eléctrica sufre interferencias o se atenúa[1] el mensaje cambiaría completamente y, con toda seguridad, pasaría a no tener sentido. Estableciendo una transmisión digital, la intensidad de la señal carece de sentido, pues sólo es importante establecer cuándo se recibe señal y cuando no se recibe señal.

Por ello, se desarrolló un sistema, llamado código binario, que responde a los dos estados anteriormente mencionados (recibir señal o no recibirla) y establece para ellos dos valores:

- Recibo señal.....1
- No recibo señal.....0

El ordenador va tomando valores de forma periódica, en pequeñísimos tramos de tiempo, y

1 Atenuar es el término usado para definir el proceso por el cual una señal eléctrica o electromagnética se debilita.

va marcando si recibe señal (1) o si no la recibe (0). De ello surge una sucesión de 1 y 0. Ya tenemos la base para desarrollar un código, que tiene que utilizar esos dos números y combinarlos para que respondan a nuestras necesidades. De esta forma, podríamos establecer que la "A" es un 1010, que la "B" es un 1110, que la "C" sea un 0110, etc (o cualquier combinación de 1 y 0).

A la unidad mas pequeña que se puede representar en binario (es decir, a cada 1 o 0) se le denomina **bit**.

Cada uno podríamos establecer nuestro código binario y marcar nuestras combinaciones de 1 y 0, pero no podríamos entendernos entre nosotros al estar los mensajes en diferentes "idiomas" binarios. Por ello, es importante que todos los ordenadores trabajen bajo un mismo código.

El código binario que emplean de forma general los ordenadores se denomina código ASCII (en inglés siglas de "American Standard Code for Information Interchange") y establece un número concreto de **bits** para cada uno de los símbolos que utilizamos.

Pero, ¿qué número de bits es necesario?. Si utilizamos un sólo bit las posibilidades son dos, el 1 y el 0. Si utilizamos dos bits las combinaciones aumentan, como se muestra en la siguiente tabla:

Número de bits	Combinaciones posibles	
1	**2**	*0,1*
2	**4**	*00, 01, 10, 11*
3	**8**	*000, 001, 010, 011, 100, 101, 110, 111*
4	**16**	*0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1111*

La cantidad necesaria, para poder abarcar todos nuestros caracteres, sería de **8 bits**, que nos da 256 combinaciones posibles, las cuales nos permiten dar cabida a los caracteres de nuestro alfabeto.

El código ASCII, para no incurrir en errores, establece una combinación concreta de **8 bits** a cada caracter. A cada grupo de **8 bits** que representan un caracter se le denomina **byte u octeto**.

A continuación se establecen algunos ejemplos de caracteres y su equivalencia en código ASCII:

Caracter	Equivalencia en ASCII
A	01000001
L	01001100
W	01010111
1	00110001

Para codificar un texto en código ASCII se escribirá el **byte** correspondiente a cada caracter. A modo de ejemplo se representa a continuación una codificación de un texto a código ASCII:

Texto:	Traducción a código ASCII:
¿Cómo está usted?	11000010 10111111 01000011 11000011 10110011 01101101 01101111 00100000 01100101 01110011 01110100 01100001 00100000 01110101 01110011 01110100 01100101 01100100 00111111

Como verás, para una frase de tres palabras son necesarios 19 **bytes**, es decir, 152 **bits** (1 y 0). La cantidad de **bytes** para cualquier texto o información es grande, lo cual hace incómodo trabajar con números tan altos y hace falta buscar múltiplos para unidades muy grandes (de la misma forma que hacemos con otras unidades de medida, por ejemplo, podemos expresar una longitud en metros, decímetros, kilómetros, etc).

A partir del **byte** las unidades de capacidad de información, en el sistema internacional, se miden en múltiplos de 1000, siendo un **kilobyte (Kb)** 1000 bytes, tal y como se refleja en la tabla siguiente:

Unidad	Equivalencia	Cantidad de bits (1 y 0)
Kilobyte (KB)	1000 bytes	8000
Megabyte (MB)	1000 Kilobytes	$8x10^{6}$
Gigabyte (GB)	1000 Megabytes	$8x10^{9}$
Terabyte (TB)	1000 Gigabytes	$8x10^{12}$
Petabyte (PB)	1000 Gigabytes	$8x10^{15}$
Exabyte (EB)	1000 Petabytes	$8x10^{18}$
Zettabyte (ZB)	1000 Exabytes	$8x10^{21}$
Yottabyte (YB)	1000 Zettabytes	$8x10^{24}$

Como veréis en la tabla, las siglas de **Kilobyte** son **KB**, ya que la palabra **byte** se representa con una **B** y el bit con una **b**. Frecuentemente esta nomenclatura es usada de forma errónea y se representa **Kilobyte** como **Kb**, pero no es correcto.

Es usual, en el mundo informático, usar múltiplos de 1024 en lugar de múltiplos de 1000 a la hora de representar la cantidad de **bits**. Esto se debe a que el sistema binario se basa en dos estados, por lo que al principio las capacidades iban creciendo según potencias de base 2, y se estandarizó el uso de 1024 como múltiplo para ir estableciendo las equivalencias, ya que es la potencia de base 2 más cercana a 1000 (2^{10} = 1024). En muchas ocasiones se suele leer y escuchar que **1 Kilobyte** son **1024 bytes**, pero no es correcto.

Para las equivalencias basadas en potencias de base 2 (múltiplos de 1024) se usa la palabra **Kibibyte**, tal como se muestra en la siguiente tabla:

Unidad	Equivalencia	Cantidad de bits (1 y 0)
Kibibyte (KB)	1024 bytes	2^{10}
Mebibyte (MB)	1024 Kibibytes	2^{20}
Gibibyte (GB)	1024 Mebibytes	2^{30}
Tebibyte (TB)	1024 Gibibytes	2^{40}
Pebibyte (PB)	1024 Gibibytes	2^{50}
Exbibyte (EB)	1024 Pebibytes	2^{60}
Zebibyte (ZB)	1024 Exbibytes	2^{70}
Yobibyte (YB)	1024 Zebibytes	2^{80}

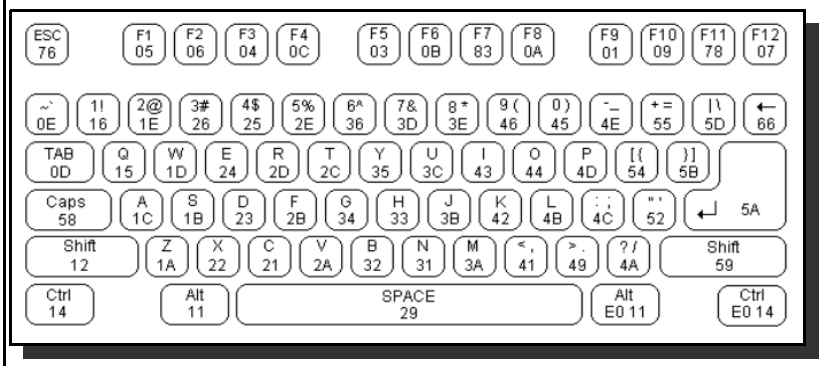
Como el código ASCII representa cada *byte* con grupos de 8 *bits*, existe un método algo mas simplificado para representar la información, denominado código *HEXADECIMAL*. Este código sólo sirve para representar información a nivel usuario. El ordenador no trabaja con el mismo, pero el programador si ve facilitado su trabajo.

El código *hexadecimal* convierte cada grupo de 4 bits en un solo caracter. De esta forma, un *byte* (ocho 1 y 0) se representará por 2 caracteres. A cada grupo de 4 bits se le denomina **nibble**, de forma que 2 nibbles forman un byte. Si contamos todas las combinaciones con 4 bits veremos que hay 16 posibilidades. El código *hexadecimal* establece un solo caracter a cada uno de ellos, empezando por la combinación mas baja posible (0000) a la cual le otorga el 0, siguiendo con los números en orden hasta el 9 y finalizando con letras en orden alfabético, como muestra la siguiente tabla:

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

Es importante entender que al convertir cada cuatro 1 y 0 en un solo carácter la cantidad de texto se divide entre cuatro y al programador le es mas sencillo escribir y leer la información, aunque el ordenador sigue trabajando con la codificación binaria.

De esta forma, por ejemplo, el 10010010 en código hexadecimal se corresponde con el 92, y el 11000110 con el C6.



Esquema que muestra un teclado y la equivalencia de cada símbolo en código hexadecimal, ya que cada caracter del teclado corresponde a un byte binario, que se expresa como dos nibbles hexadecimales.

APRENDE MÁS:

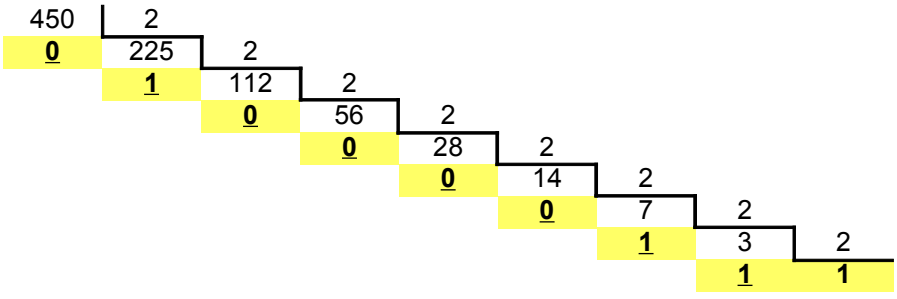
¿Cómo contar en código binario?

Igual que nosotros contamos usando los números del 0 al 9 y combinándolos, es posible contar en binario. De esta forma, empezaríamos con el número mas pequeño que podemos hacer con 1 y 0, que sería el 0. Después seguiría el número mas próximo a 0 que se pueda hacer con código binario, que corresponde con el 1, a continuación iría el 10, luego el 11, el 100, 101, 110... Así podemos hacer una equivalencia entre números decimales y números binarios.

Decimal	Binario
0	0
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

Pero... ¿me tengo que aprender todas las combinaciones posibles?, ¿cómo sería el número 450 en código binario?...

Existe una forma de transformar cualquier **número decimal** en un **número binario**, con la cual obtenemos el mismo resultado que conseguiríamos si siguiésemos la secuencia de la tabla anterior. Para ello, hay que ir dividiendo el número decimal entre 2, sin sacar decimales (por lo que el resto de cada división, al ser el divisor 2, siempre será 0 ó 1). Seguiremos dividiendo los resultados hasta que obtengamos un 1, como muestra el siguiente ejemplo en el cual vamos a transformar el número decimal **450** en número binario:



Una vez realizadas las sucesivas divisiones entre 2 tenemos una serie de **restos** (marcados en **negrita** y subrayados), así como el **1** final que queríamos. Todos los restos y dicho 1 están resaltados en amarillo. Nuestro número binario, ya obtenido, se lee de derecha a izquierda, encadenando el 1 final y todos los restos obtenidos. De esta forma sabemos que...:

450 en decimal se corresponde con **111000010** en binario.

Y... ¿Si quiero saber qué número decimal corresponde a un número binario?.

El proceso a seguir es sencillo, basta con realizar una serie de pasos y acabaremos obteniendo el **número decimal** que se corresponde con el **número binario**. Para ello vamos a hacer el proceso en seis partes:

1) Separamos los bits (1 y 0) del número binario.
2) Situamos un 2 multiplicando a cada bit.
3) Vamos a poner potencias a los 2 que hemos situado en el paso anterior, empezando por el 2 de la derecha, al que le pondremos como potencia el 0 (2^0) y seguiremos poniendo potencias cada una un grado mas alto que la anterior (0, 1, 2, 3, etc).
4) Resolvemos las potencias de base 2.
5) Realizamos la multiplicación del bit por el resultado de la potencia de base 2.
6) Sumamos todos los resultados anteriores.

En el siguiente cuadro se realizan los seis pasos para transformar el **110010110** en número decimal:

			Binario:	**110010110**					
1)	1	1	0	0	1	0	1	1	0
2)	1·2	1·2	0·2	0·2	1·2	0·2	1·2	1·2	0·2
3)	$1 \cdot 2^8$	$1 \cdot 2^7$	$0 \cdot 2^6$	$0 \cdot 2^5$	$1 \cdot 2^4$	$0 \cdot 2^3$	$1 \cdot 2^2$	$1 \cdot 2^1$	$0 \cdot 2^0$
4)	1·256	1·128	0·64	0·32	1·16	0·8	1·4	1·2	0·1
5)	256	128	0	0	16	0	4	2	0
6)	256+128+16+4+2								
			Decimal:	**456**					

¿Cómo practico estos procesos?

Es sencillo, no tienes mas que elegir un número decimal, por ejemplo el 650, y convertirlo en número binario. Una vez que has obtenido el número binario, vuelve a operarlo para pasarlo a número decimal. Si tras ambos procesos vuelves a obtener el 650 es que has operado de forma correcta.