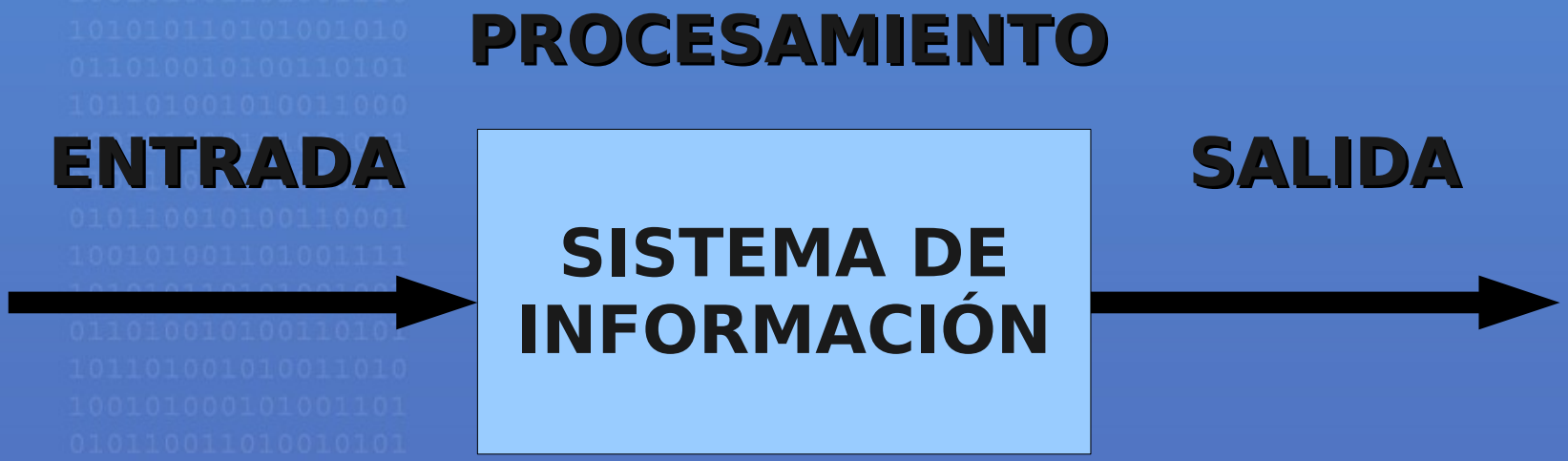
FUNDAMENTOS DE DATOS

Componentes lógicos de un ordenador

Para que un ordenador funcione, además de hardware necesita información...

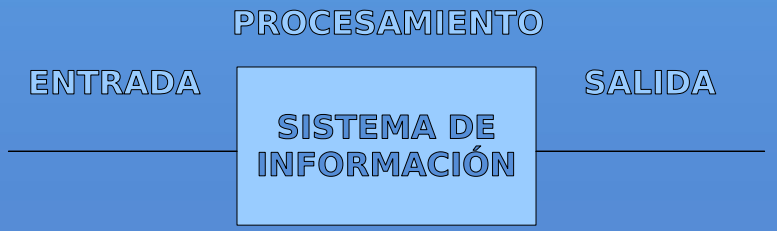
- PROGRAMAS (SOFTWARE): Información sobre qué hay que hacer con los datos de entrada
- DATOS: La información que se ha de procesar

Con ella se hace el tratamiento automático de la información:



Tipos de datos

- Datos según su localización:
  - De entrada
  - Intermedios
  - Salida (resultados)
- Según su estado durante el procesamiento:
  - Fijos (constantes)
  - Variables (variables)



Tipos de datos

- Datos según como los usa el ordenador:
  - Numéricos
  - Alfabéticos
  - Alfanuméricos
  - Imagen
  - Sonido
  - Vídeo
  - ...

Tipos de datos

- Datos según el código que emplean:
  - Sistemas de codificación numérica:
    - Conversión directa a binario
    - Signo y magnitud
    - BCD Exceso-8
    - Complemento a dos
    - IEE 754 en coma flotante ...
  - Sistemas de codificación alfanumérica
  - Sistemas de codificación de imagen
  - Sistemas de codificación de sonido
  - Sistemas de codificación de vídeo
  - ...

Sistemas de codificación

- Lenguaje ordenador ǂ Lenguajes humanos
- Para conseguir que el ordenador entienda los datos del lenguaje humano se ha de usar un **sistema de codificación**.
- Cualquier sistema de codificación de compone de:
  - Símbolos, que componen cada lenguaje
  - Reglas, que relacionan los símbolos de un lenguaje con otro
- Vamos a estudiar estos sistemas.

Sistemas de codificación

Estudiaremos los sistemas usados para codificar estos tipos de datos...

- Numéros
  - Naturales
  - Enteros
  - Reales
- Caracteres
- Imágenes
- Sonido
- Vídeo

Sistemas de codificación numérica: Números naturales

- Cualquier sistema de numeración se compone de...
  - Un conjunto de símbolos, llamados BASE (por ejemplo 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9 en el sistema decimal)
  - Una regla, llamada SISTEMA POSICIONAL, para combinar esos símbolos, que indica que el valor de cada cifra depende de su posición (la que da en el sistema decimal los valores a unidades, decenas, centenas...)

Sistemas de codificación numérica: Números naturales

- Todos los sistemas posicionales se basan en el **Teorema Fundamental de la Numeración** (TFN), que relaciona una cantidad expresada en cualquier sistema de numeración con su valor decimal:

$$\text{Valor}_{\text{número}} = \Sigma(X_i \cdot B^i)$$

i=Posición de la cifra (0 la de la derecha, incrementándose a medida que se va hacia la izquierda)

X_i= Valor absoluto de la cifra en la posición i

B=número de símbolos de la base

Sistemas de codificación numérica: Números naturales

- Ejemplo del TFN para el número decimal 743:

POSICIÓN	2	1	0
NÚMERO	7	4	3

$$Valor_{número} = \Sigma (X_i \cdot B^i) = X_2 \cdot B^2 + X_1 \cdot B^1 + X_0 \cdot B^0 = 7 \cdot 10^2 + 4 \cdot 10^1 + 3 \cdot 10^0 = 7 \cdot 100 + 4 \cdot 10 + 3 \cdot 1 = 700 + 40 + 3 = \mathbf{743}$$

- Ejemplo del TFN para el número en 312 en una base de 5 símbolos (0,1,2,3,4):

POSICIÓN	2	1	0
NÚMERO	3	1	4

$$Valor_{número} = \Sigma (X_i \cdot B^i) = X_2 \cdot B^2 + X_1 \cdot B^1 + X_0 \cdot B^0 = 3 \cdot 5^2 + 1 \cdot 5^1 + 4 \cdot 5^0 = 3 \cdot 25 + 1 \cdot 5 + 4 \cdot 1 = 75 + 5 + 4 = \mathbf{84}$$

Sistemas de codificación numérica: Números naturales

- Como vamos a empezar a trabajar con números en diversas bases, para distinguirlos a partir de ahora pondremos a la derecha del número como subíndice la base en la que está representado, así...
  - 4563_{10} : número 4563 en base decimal
  - 3012_{5}: número 3012 en base 5
  - 11101_{2}: número 11101 en base binaria
  - 6130_{8}: número 4563 en base octal

Sistemas de codificación numérica: Números naturales: Sistema binario

- El ordenador sólo utiliza internamente carga o no carga (0 o 1), por lo cual su sistema de numeración es el binario.
- Por lo tanto para pasar de binario a decimal sólo hemos de aplicar el TFN:

POSIC	7	6	5	4	3	2	1	0
Nº BINARIO	1	0	1	0	0	1	1	0

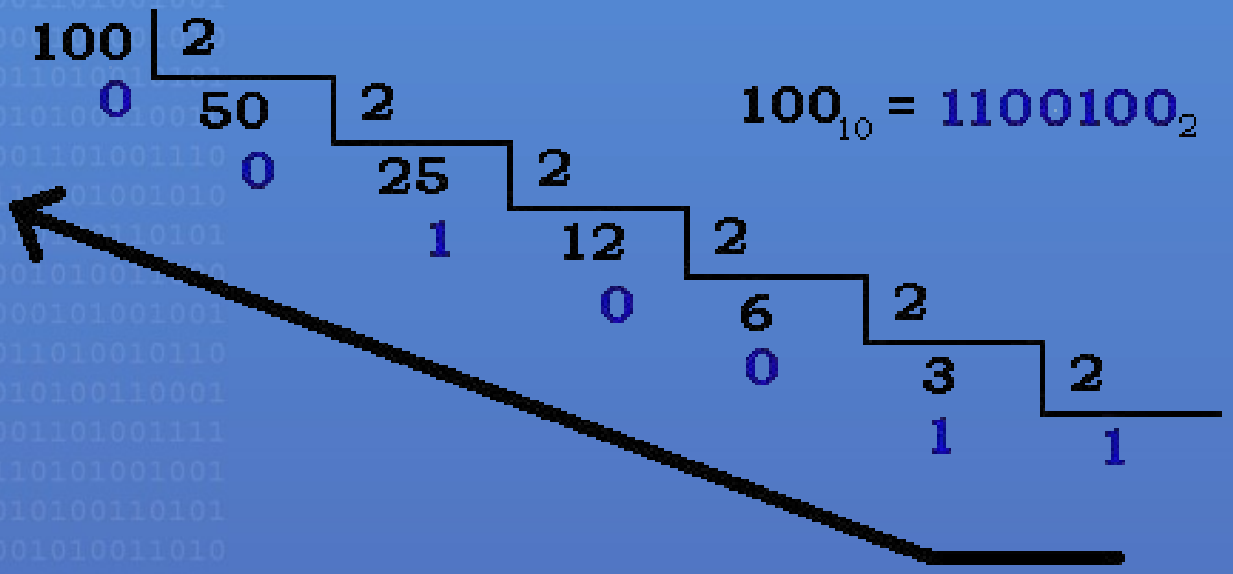
→ nº binario

$$Valor_{número} = \Sigma (X_i \cdot B^i) = X_7 \cdot B^7 + X_6 \cdot B^6 + X_5 \cdot B^5 + X_4 \cdot B^4 + X_3 \cdot B^3 + X_2 \cdot B^2 + X_1 \cdot B^1 + X_0 \cdot B^0 = 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 1 \cdot 128 + 0 \cdot 64 + 1 \cdot 32 + 0 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 128 + 32 + 4 + 2 = \mathbf{166}_{10}$$

→ nº decimal

Sistemas de codificación numérica: Números naturales: Sistema binario

- Para pasar de decimal a binario, hemos de realizar varias divisiones sucesivas y quedarnos con los restos en orden inverso. Por ejemplo, así pasamos el número decimal 100 a binario:



Sistemas de codificación numérica: Números naturales: Sistema binario

- El número de combinaciones que se pueden hacer con X bits es 2^X. Así...
  - Con 8 bits se pueden realizar 256 (2^8)
  - Con 16 bits se pueden realizar 65536 (2^{16})
- Como tenemos que reservar una combinación para el 0, el número natural más alto que podemos representar con X bits es 2^X-1. Así...
  - Con 8 bits alcanzamos el 255 (2^8-1)
  - Con 16 bits alcanzamos 65535 (2^{16}-1)

Sistemas de codificación numérica: Números naturales: Sistema binario

- Si colocamos secuencialmente los números naturales en binario tendríamos...

• 0	• 100	• 1000	• 1100
• 1	• 101	• 1001	• 1101
• 10	• 110	• 1010	• 1110
• 11	• 111	• 1011	• 1111

Sistemas de codificación numérica: Números naturales: Sistema binario

- Trabajar con números binarios es poco práctico y puede llevar a errores de transcripción muy fácilmente, pues por ejemplo número como el 834245_{10} se convierte en 11001011101011000101_{2}.
- Para evitarlo, en informática se usan sistemas de base 8 (octal) y base 16 (hexadecimal). Por ejemplo el mismo número anterior sería 3135305_{8} o $CBAC5_{16}$.

Sistemas de codificación numérica: Números naturales: Sistema octal

- Base: 0, 1, 2, 3, 4, 5, 6, 7 (8 símbolos)
- Regla: Sistema posicional
- Para pasar de octal a decimal...
  - Aplicamos el TFN
- Para pasar de decimal a octal...
  - Divisiones sucesivas entre 8, tomando cociente final y restos en orden inverso.

Sistemas de codificación numérica: Nºs naturales: Sistema hexadecimal

- Base: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (16 símbolos)
- Regla: Sistema posicional
- Para pasar de octal a decimal...
  - Aplicamos el TFN
- Para pasar de decimal a octal...
  - Divisiones sucesivas entre 16, tomando cociente final y restos en orden inverso.

Si alguna cifra es una letra se sustituye por su valor (A=10, B=11, C=12, D=13, E=14, F=15).

Sistemas de codificación numérica: Nº naturales: Correspondencias

SISTEMA DECIMAL (Base 10)	SISTEMA BINARIO (Base 2)	SISTEMA OCTAL (Base 8)	SISTEMA HEXADECIMAL (Base 16)
0	00000	00	00
1	00001	01	01
2	00010	02	02
3	00011	03	03
4	00100	04	04
5	00101	05	05
6	00110	06	06
7	00111	07	07
8	01000	10	08
9	01001	11	09
10	01010	12	0A
11	01011	13	0B
12	01100	14	0C
13	01101	15	0D
14	01110	16	0E
15	01111	17	0F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13

Sistemas de codificación numérica: Números naturales: Sistema octal

Para pasar de octal a binario...

- Sustituimos cada símbolo octal por 3 símbolos binarios de acuerdo con la tabla de correspondencias (rellenando con 0s hasta los 3 digitos si hace falta).

$307_8 = ?_2$

3 0 7

$011000111 \rightarrow 11000111_2$

Sistemas de codificación numérica: Números naturales: Sistema octal

Para pasar de binario a octal...

- Agrupamos los bits en bloques de 3 empezando por la derecha y sustituimos cada bloque por su símbolo octal de acuerdo con la tabla de correspondencias.

$1000100_2 = ?_8$

$$\underbrace{1}_{1}\ \underbrace{000}_{0}\ \underbrace{100}_{4} \rightarrow 104_8$$

Sistemas de codificación numérica: Nºs naturales: Sistema hexadecimal

Para pasar de hexadecimal a binario...

- Sustituimos cada símbolo octal por 4 símbolos binarios de acuerdo con la tabla de correspondencias (rellenando con 0s hasta los 3 digitos si hace falta).

$5B7_{16} = ?_{2}$

$\overbrace{5}\ \overbrace{B}\ \overbrace{7}$

$0101101110111 \rightarrow 10110110111_{2}$

Sistemas de codificación numérica: Nºs naturales: Sistema hexadecimal

Para pasar de binario a hexadecimal...

- Agrupamos los bits en bloques de 4 empezando por la derecha y sustituimos cada bloque por su símbolo octal de acuerdo con la tabla de correspondencias.

$$1111000101_2 = ?_{16}$$

$$\underbrace{11}_{3}\ \underbrace{1100}_{A}\ \underbrace{0101}_{5} \rightarrow 3A5_{16}$$

Sistemas de codificación numérica: Nºs naturales: Hexadecimal-Octal

Para pasar de octal a hexadecimal o de hexadecimal a octal...

- Pasamos de la base inicial a binario y luego de binario a la base final

$3072_8 = ?_{16}$

3 0 7 2

1100011 1010

6 3 A → $63A_{16}$

Sistemas de codificación

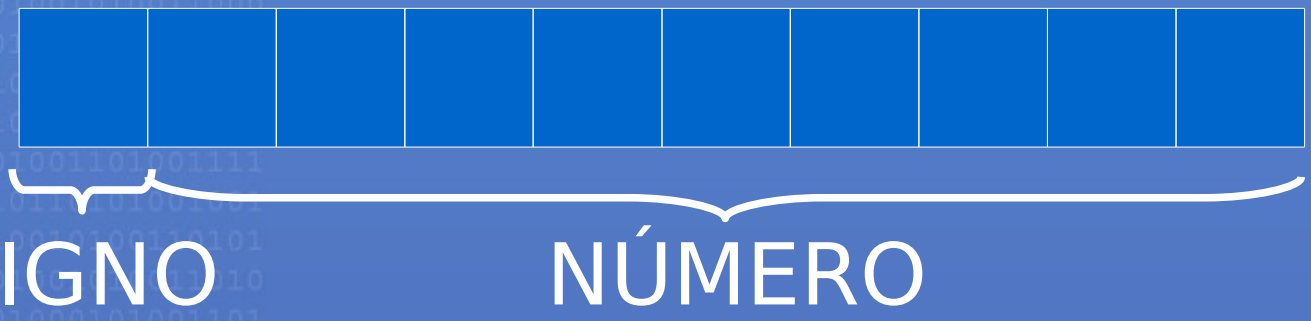
A partir de ahora, siempre que representemos los bits de u dato de cualquier tipo tal y como estará almacenado en la memoria del ordenador, si no especificamos lo contrario vamos a usar HEXADECIMAL.

Sistemas de codificación numérica: Números enteros

- La forma anterior de representar números no tiene manera de representar signos, por lo que para guardar números enteros en binario deberemos de usar otra.

Sistemas de codificación numérica: Nºs enteros: Signo y magnitud

- Signo y magnitud es la forma más sencilla (aunque no la más usada) de representar números enteros.
- Se reserva el bit de más a la izquierda para el signo (0=positivo, 1=negativo) y el resto para almacenar el valor absoluto del número en binario:



Sistemas de codificación numérica: Nºs enteros: Signo y magnitud

- Ejemplos:
  - El byte 10000100 (84_{SM}) sería -4
  - El byte 00000011 (03_{SM}) sería 3
  - El byte 10001111 ($8F_{SM}$) sería -15
  - El byte 00000000 (00_{SM}) sería 0
  - El byte 10000000 (80_{SM}) sería -0

  } Hay dos maneras de representar el número 0

- Los números que se pueden representar si tenemos n bits van desde el $-(2^{n-1} - 1)$ hasta el $+(2^{n-1} - 1)$.

Sistemas de codificación numérica: Nºs enteros: Complemento a 2

- El complemento a 2 es la forma más habitual de representar números enteros, pues es muy útil operar con circuitos electrónicos digitales.
- En el complemento a 2, el bit de más a la izquierda también representa el signo, pero el número almacenado a contiuación está codificado de forma especial.

Sistemas de codificación numérica: Nºs enteros: Complemento a 2

- Para codificar el signo (bit izquierdo) directamente ponemos 0 (+) o 1 (-).
- Para codificar el nº en complemento a 2 pasamos su valor absoluto a binario y...
  - Si es positivo, quedarnos con el valor binario obtenido. Para +5 con 4 bits:

$$5_{10} = 0101_{2} \rightarrow \mathbf{0101} \rightarrow \mathbf{5_{c2}}$$

  - Si es negativo, invertir el valor de todas las cifras y luego sumar 1 al valor resultante. Para -5 con 4 bits:

INVERTIMOS LOS BITS SUMAMOS 1

$$5_{10} = 0101_{2} \rightarrow 1010 \rightarrow \mathbf{1011} \rightarrow \mathbf{B_{c2}}$$

Sistemas de codificación numérica: Nºs enteros: Complemento a 2

- Para decodificar un número en complemento a 2 y pasarlo a decimal...
  - El signo lo da su primer bit: 0 → **+** 1 → **-**
  - Su valor absoluto lo dan el resto de los bits...
    - Si el número es positivo, simplemente transcribimos los bits
    - Si el número es negativo, para hallar el valor absoluto hemos de complementar a 2 invirtiendo y sumando 1 como vimos antes. Por ejemplo si tenemos 1111100:

	INVERTIMOS LOS BITS		SUMAMOS 1	
1111100	→	0000011	→	**3**$_{10}$

Sistemas de codificación numérica: Nºs enteros: Complemento a 2

- Ejemplos:
  - El byte 00001100 ($0C_{C2}$) sería 12
  - El byte 11111101 (FD_{C2}) sería -3
  - El byte 11110001 ($F1_{C2}$) sería -15
  - El byte 00000000 (00_{C2}) sería 0
  - El byte 11111111 (FF_{C2}) sería -1
- Los números que se pueden representar si tenemos n bits van desde el -2^{n-1} hasta el $+(2^{n-1} - 1)$. Al tener una sola forma de poner 0, representamos un número más.

Sistemas de codificación numérica: Números reales

- Con la notación anterior, no podemos representar números que tengan parte fraccionaria, por lo que tenemos que utilizar otros sistemas de codificación para ello.

Sistemas de codificación numérica: Nºs reales: Coma fija

- La notación de coma fija es la más sencilla para representar números enteros.
- En coma fija se predetermina que cierto número de dígitos binarios (los situados más a la derecha) corresponden a la parte fraccionaria, y el resto a la entera del número a representar.

Sistemas de codificación numérica: Nºs reales: Coma fija

- Para pasar de <u>binario en coma fija a decimal</u> aplicamos el TFN, sabiendo que las partes fraccionarias tienen posición indicada en negativo. Así, si tenemos $E6_{CFI3}$ (el binario 11100110 en coma fija con 3 fraccionarios):

POSIC	4	3	2	1	0	-1	-2	-3
Nº BINARIO	1	1	1	0	0 ,	1	1	0

→ nº binario en coma fija con 3 fraccionarios

$$Valor_{número} = \Sigma (X_i \cdot B^i) = X_4 \cdot B^4 + X_3 \cdot B^3 + X_2 \cdot B^2 + X_1 \cdot B^1 + X_0 \cdot B^0 + X_{-1} \cdot B^{-1} + X_{-2} \cdot B^{-2} + X_{-3} \cdot B^{-3} = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} = 1 \cdot 16 + 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 0 \cdot 1 + 1 \cdot 0{,}5 + 1 \cdot 0{,}25 + 0 \cdot 0{,}125 = 16 + 8 + 4 + 0{,}5 + 0{,}25 = \mathbf{28{,}75}_{10}$$

→ nº decimal

Sistemas de codificación numérica: Nºs reales: Coma fija

- Para pasar de <u>decimal a binario en coma fija</u> convertimos la parte entera como sabemos y la fraccionaria con multiplicaciones por dos sucesivas de la parte fraccionaria de las que nos vamos quedando con los dígitos enteros en orden. Por ejemplo, así pasamos el nº $19{,}427_{10}$ a binario coma fjia con 3 decimales:

0,**427** x 2 = 0,854
0,854 x 2 = 1,708
0,708 x 2 = 1,416

Parte fraccionaria: **0,011**$_2$

Número Binario: **10011,011**$_2$

En coma fija 3 decimales:

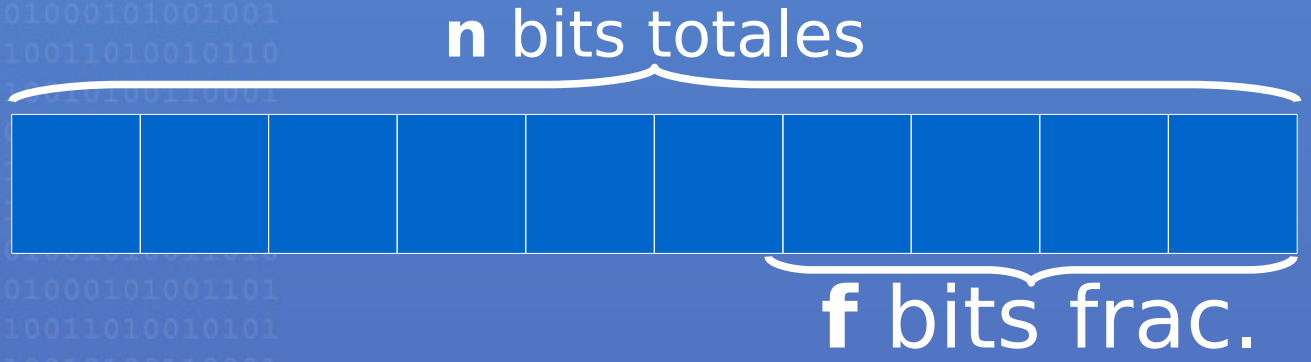
10011011 → **9B**$_{CFI3}$

Sistemas de codificación numérica: Nºs reales: Coma fija

- La notación estándar de coma fija no tiene signo, así que para poder trabajar con números reales positivos y negativos hemos de usar alguna de las notaciones de signo.
- Así, sabiendo que $19{,}427_{10} \simeq 10011{,}011_{2}$...
  - Si utilizáramos además 12 bits con signo y magnitud
    - $+19{,}427_{10} \rightarrow 000010011011 \rightarrow 09B_{CFI3SM}$
    - $-19{,}427_{10} \rightarrow 100010011011 \rightarrow 89B_{CFI3SM}$
  - Si utilizáramos además 12 bits con complemento a 2
    - $+19{,}427_{10} \rightarrow 000010011011 \rightarrow 09B_{CFI3C2}$
    - $-19{,}427_{10} \rightarrow 111101100101 \rightarrow F65_{CFI3C2}$

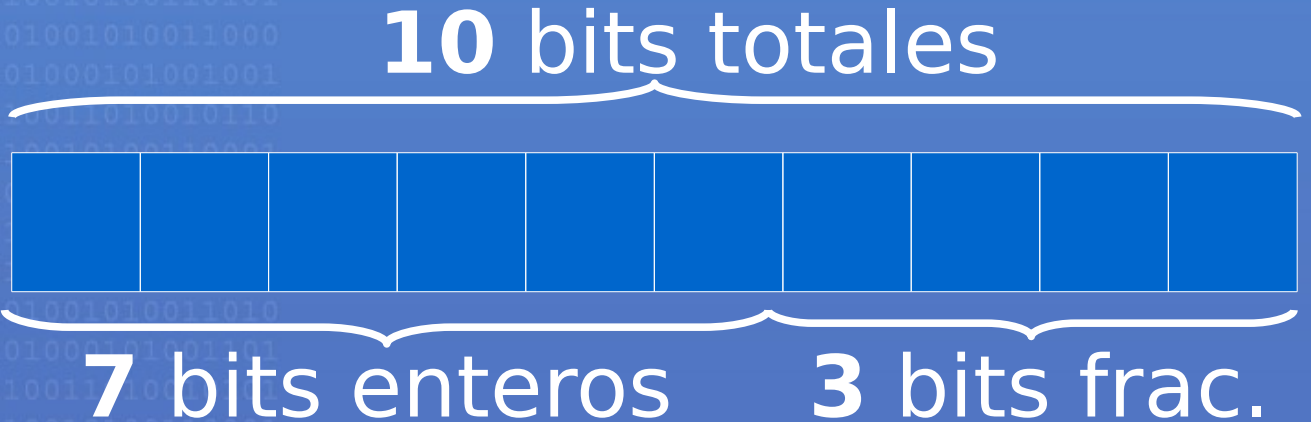
Sistemas de codificación numérica: Nº reales: Coma fija

- En coma fija se puede representar...
  - Directamente, desde el 0 al $+(2^{(n-f)} - 2^{-f})$.
  - Con signo y magnitud desde el $-(2^{(n-f-1)} - 2^{-f})$ hasta el $+(2^{(n-f-1)} - 2^{-f})$.
  - En complemento a dos desde el el $-2^{(n-f-1)}$ hasta el $+(2^{(n-f-1)} - 2^{-f})$.
- Es decir, están muy limitados, por eso la notación de coma fija apenas se usa.



Sistemas de codificación numérica: Nºs reales: Coma fija

- En coma fija con 10 bits y 3 fraccionarios...
  - Directamente, desde el 0 al $127{,}875_{10}$
  - Con signo y magnitud desde el $-63{,}875_{10}$ ($-111111{,}111_2$) al $63{,}875_{10}$ ($1111111{,}111_2$).
  - En complemento a dos desde el el -64_{10} (-10000000_2) hasta el $63{,}875_{10}$ ($1111111{,}111_2$).

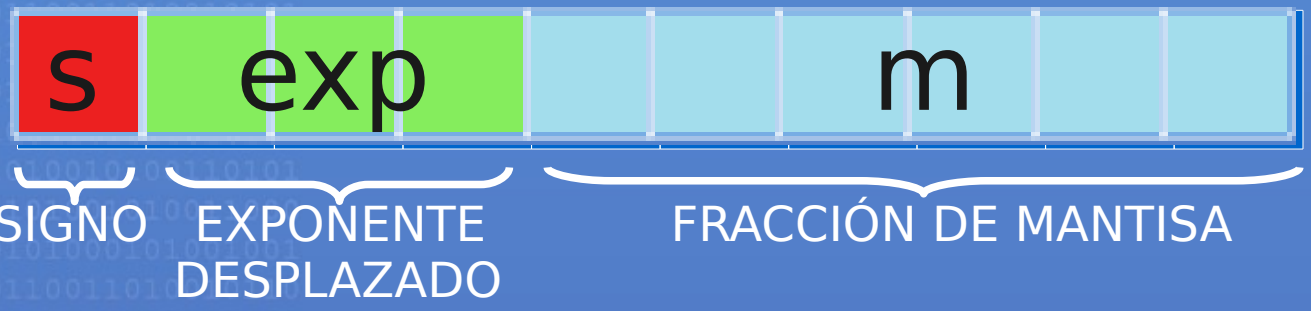


Sistemas de codificación numérica: Nº reales: Coma flotante

- La notación de coma fija nos limita mucho las cantidades que pueden ser representadas con un número limitado de bits. Para solucionarlo, existen los sistemas de representación de coma flotante en los cuales la posición de la coma fraccionaria puede variar (*flotar*) dependiendo de lo que se determine en el campo exponente.
- Vamos a explicar la notación de coma flotante **IEEE 754**, que es la estándar y más ampliamente usada.

Sistemas de codificación numérica: Nºs reales: Coma flotante

- Un número en coma flotante se compone de
  - Un campo de signo **s**, de 1 bit
  - Un campo de exponente desplazado **exp**, que indica la posición de la coma
  - Un campo fracción de mantisa **m**, que indica los primeros bits <u>significativos</u> del valor del nº



$$\text{Valor del número} = \mathbf{s} \times \mathbf{M} \times 2^{e}$$

Siendo d el desplazamiento y **e=exp-d**

Sistemas de codificación numérica: Nºs reales: Coma flotante

La mantisa **M** que utilizamos para calcular el valor de un número de coma flotante se obtiene muy fácilmente a partir del valor fraccionario de la mantisa **m**. Simplemente hemos de tratar el valor como un binario que tuviera como parte entera 1 y cuyo valor fraccionario fuese el indicado en m.

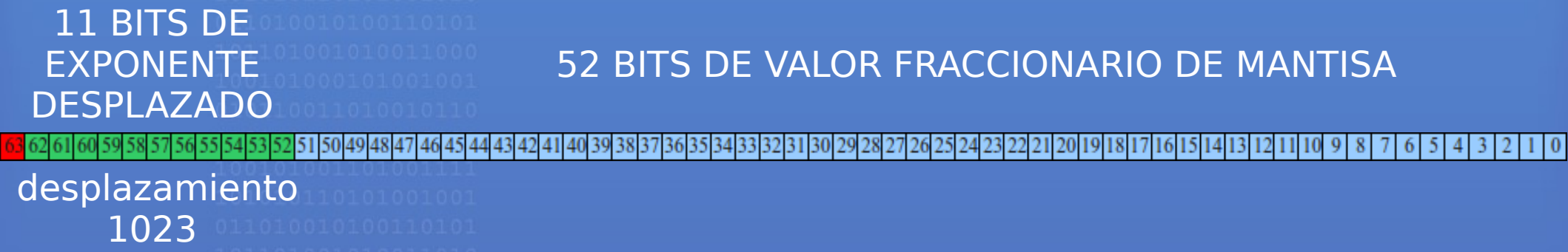
Así, si el valor fraccionario **m** fuese 1001_2 esto indicaría que el valor de la mantisa **M** sería $1{,}1001_2$.

Sistemas de codificación numérica: Nºs reales: Coma flotante

El estándar IEE 754 define dos variantes:

- Precisión simple. 32 bits totales (1+8+23)

- Precisión doble. 64 bits totales (1+11+52)



Sistemas de codificación numérica: Nºs reales: Coma flotante

La mantisa **M** que utilizamos para calcular el valor de un número de coma flotante se obtiene muy fácilmente a partir del valor fraccionario de la mantisa **m**. Simplemente hemos de tratar el valor como un binario que tuviera como parte entera 1 y cuyo valor fraccionario fuese el indicado en m.

Así, si el valor fraccionario **m** fuese 1001 esto indicaría que el valor de la mantisa **M** sería $1{,}1001_2$.

Sistemas de codificación numérica: Nºs reales: Coma flotante

- Convertir $X_2 \rightarrow ?_{CFL}$ en precisión simple

$101110{,}01010111010000111110000111110010011_2 \rightarrow ?_{CFL}$

Pasamos al formato

$valor_{número} = \mathbf{s} \times \mathbf{M} \times 2^{\mathbf{e}}$

$+1 \times 1{,}01110010101110100001111100001111100010011_2 \times 2^5$

$M=1,\mathbf{01110010101110100001111}10000111110010011_2$

m (los 23 primeros fraccionarios)

$e=5_{10}$ $exp= e + 127_{10}$ **exp**$= e+127 = 5_{10} + 127_{10} = 132_{10} =$ **10000100**$_2$

Signo = +1 → **s=0** (tal y como utilizábamos en *signo y magnitud*)

s **exp** **m**

0 10000100 01110010101110100001111 → **42395D0F**$_{CFL}$

Sistemas de codificación numérica: Nºs reales: Coma flotante

- Convertir $X_{CFL} \rightarrow ?_2$ o $?_{10}$ en precisión simple

C 0 D 0 0 0 0 0 $C0D00000_{CFL} \rightarrow ?_2 \rightarrow ?_{10}$

1 10000001 10100000000000000000000

s **exp** **m**

S=1 → Número negativo

$exp=10000001_2=1 \times 2^7 + 1 \times 2^0 = 129_{10}$ $e=exp-127=129-127=2_{10}$

$M=1,m_2=1,1010000_2$

$valor_{número} = \mathbf{s} \times \mathbf{M} \times 2^{\mathbf{e}} = -1 \times 1,101_2 \times 2^2 =$ **$-110,1_2$**

Y como por el TFN $110,1= 1x2^2 + 1x2^1 + 0x2^0 + 1x2^{-1} = 4 + 2 + 0,5 = 6,5_{10}$

Entonces obtenemos que el valor en decimal es **$-6,5_{10}$**

Sistemas de codificación numérica: Nºs reales: Coma flotante

- Convertir $X_{10} \rightarrow ?_{CFL}$ en precisión doble (1)

$140{,}5625_{10} \rightarrow ?_{CFL}$

140 | 2
0 70 | 2
0 35 | 2
1 17 | 2
1 8 | 2
0 4 | 2
0 2 | 2
0 1

Parte entera
10001100_2

$0{,}\mathbf{5625} \times 2 = 1{,}125$
$0{,}125 \times 2 = 0{,}25$
$0{,}25 \times 2 = 0{,}5$
$0{,}5 \times 2 = 1{,}0$

Parte fraccionaria: **$0{,}1001_2$**

Número Binario:
$10001100{,}1001_2$

Pasamos al formato
$valor_{número} = \mathbf{s} \times \mathbf{M} \times 2^{\mathbf{e}}$

Sistemas de codificación numérica: Nºs reales: Coma flotante

- Convertir $X_{10} \rightarrow ?_{CFL}$ en precisión doble (2)

$10001100{,}1001_2 \longrightarrow +1 \times 1{,}00011001001_2 \times 2^7$

Pasamos al formato

$valor_{número} = \mathbf{s} \times \mathbf{M} \times 2^{\mathbf{e}}$

$M=1,\underbrace{\mathbf{00011001001}0000000000000000000000000000000000000000000}_{\mathbf{m}\text{ (los 52 primeros bits fraccionarios)}}{}_2$

$exp=7_{10}$ $exp= e-1023_{10}$ $\mathbf{e}=exp+1023=7_{10}+1023_{10}=1030_{10}=\mathbf{10000000110}_2$

Signo = +1 → 5 **s=0** (tal y como utilizábamos en *signo y magnitud*)

s	**exp**	**m**
0	10000000110	0001100100100000000000000000000000000000000000000000
4	0 6	1 9 2 0 0 0 0 0 0 0 0 0 0

4061920000000000_{CFL}

Sistemas de codificación numérica: Nºs reales: Coma flotante

- En coma flotante se puede representar...
  - En precisión simple:
    Desde $-((2 - 2^{-23}) \times 2^{127}) \simeq 3{,}4 \times 10^{28}$
    Hasta $+((2 - 2^{-23}) \times 2^{127}) \simeq -3{,}4 \times 10^{28}$
  - En precisión doble:
    Desde $-((2 - 2^{-52}) \times 2^{1023}) \simeq 1{,}8 \times 10^{308}$
    Hasta $+((2 - 2^{-52}) \times 2^{1023}) \simeq -1{,}8 \times 10^{308}$

Sistemas de codificación alfanumérica

- Ahora que ya sabemos como representar números a partir de ceros y unos, vamos a estudiar la forma en la que podemos representar caracteres alfanuméricos:
  - Letras (a, b, c, A, B, C, ...)
  - Símbolos (!, @, ,, ., *, /, ...)
  - Caracteres numéricos (1, 2, 3, 4, +, -, ...)
  - Caracteres especiales (cambio de línea, borrado, ...)

Sistemas de codificación alfanumérica: Código ASCII

- ASCII= **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange (código americano estándar para el intercambio de información)
- Es el estándar en el que se basan casi todos los sistemas informáticos para representar textos, aunque normalmente usen una extensión compatible.
- Fue creado por la Asociación Americana de Estándares.
- Define el alfabeto latino inglés

Sistemas de codificación alfanumérica: Código ASCII

- 1 byte → 1 carácter alfanumérico
- Sólo usa 7 bits del byte
- Sólo indica la manera de representar 2^7=128 caracteres, de los cuales:
  - 33 caracteres son no-imprimibles, la mayoría de ellos caracteres de control
  - 95 son caracteres imprimibles (32-126)
- Tiene una tabla con las correspondencias

Sistemas de codificación alfanumérica: Código ASCII

Dec	Char		Dec	Chr	Dec	Chr	Dec	Chr
0	NUL	(null)	32	Space	64	@	96	`
1	SOH	(start of heading)	33	!	65	A	97	a
2	STX	(start of text)	34	"	66	B	98	b
3	ETX	(end of text)	35	#	67	C	99	c
4	EOT	(end of transmission)	36	$	68	D	100	d
5	ENQ	(enquiry)	37	%	69	E	101	e
6	ACK	(acknowledge)	38	&	70	F	102	f
7	BEL	(bell)	39	'	71	G	103	g
8	BS	(backspace)	40	(	72	H	104	h
9	TAB	(horizontal tab)	41	)	73	I	105	i
10	LF	(NL line feed, new line)	42	*	74	J	106	j
11	VT	(vertical tab)	43	+	75	K	107	k
12	FF	(NP form feed, new page)	44	,	76	L	108	l
13	CR	(carriage return)	45	-	77	M	109	m
14	SO	(shift out)	46	.	78	N	110	n
15	SI	(shift in)	47	/	79	O	111	o
16	DLE	(data link escape)	48	0	80	P	112	p
17	DC1	(device control 1)	49	1	81	Q	113	q
18	DC2	(device control 2)	50	2	82	R	114	r
19	DC3	(device control 3)	51	3	83	S	115	s
20	DC4	(device control 4)	52	4	84	T	116	t
21	NAK	(negative acknowledge)	53	5	85	U	117	u
22	SYN	(synchronous idle)	54	6	86	V	118	v
23	ETB	(end of trans. block)	55	7	87	W	119	w
24	CAN	(cancel)	56	8	88	X	120	x
25	EM	(end of medium)	57	9	89	Y	121	y
26	SUB	(substitute)	58	:	90	Z	122	z
27	ESC	(escape)	59	;	91	[	123	{
28	FS	(file separator)	60	<	92	\	124	|
29	GS	(group separator)	61	=	93	]	125	}
30	RS	(record separator)	62	>	94	^	126	~
31	US	(unit separator)	63	?	95	_	127	DEL

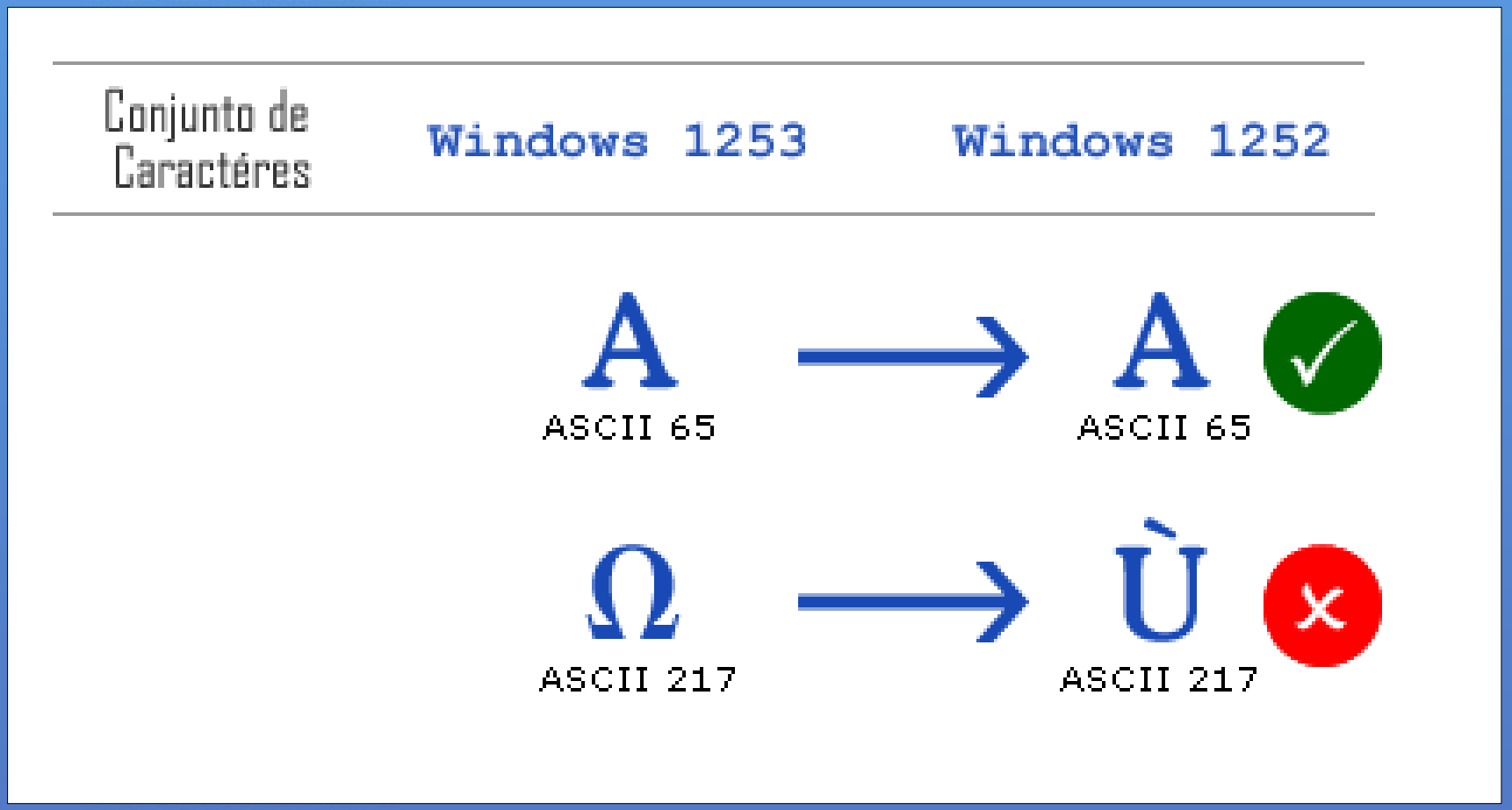
Sistemas de codificación alfanumérica: Código ASCII

- Ejemplos:
  - 61_{ASCII} → a
  - 41_{ASCII} → A
  - arbol → $6172626F6C_{ASCII}$
  - Hola! → $486F6C6121_{ASCII}$
  - $432E492E462E502E_{ASCII}$ → C.I.F.P.

Sistemas de codificación alfanumérica: Código ASCII

- El estándar ASCII tiene muy limitados los caracteres posibles, y era insuficiente para muchas funciones
- Al estar definidas solamente caracteres para 7 bits, aún se podía aprovechar el siguiente.
- Por ello, surgieron diversos sistemas de codificación que usaban estas 128 combinaciones extra, pero de forma incompatible entre si.

Sistemas de codificación alfanumérica: Código ASCII



Sistemas de codificación alfanumérica: Códigos ISO 8859

- Para solventar los problemas debidos a las diferentes implementaciones de los caracteres ASCII 128-255, ISO (International Standards Organization) ha definido estándares empleando el byte completo:
  - ISO 8859-1 (Latin-1): alemán, español...
  - ISO 8859-2 (Latin-2): polaco, croata...
    - ...
  - ISO 8859-6 (Latin-Arabic): árabe (básico)
  - ISO 8859-7 (Latin-Greek): griego
    - ...
  - ISO 8859-15 (Latin-9): ISO8859-1 + € + ...
    - ...

Sistemas de codificación alfanumérica: Código ISO 8859-1

- Este juego de caracteres también se llama
  - Latin-1
  - Western European (Europa occidental)
  - ISO/IEC 8859-1:1998
- Define el alfabeto latino (incluyendo letras especiales) necesario para representar textos en numerosas lenguas de los países situados en el oeste de Europa.
- Ha sido durante bastante tiempo el juego de caracteres habitual en España y otros países cercanos.

Sistemas de codificación alfanumérica: Código ISO 8859-1

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL 0000	STX 0001	SOT 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
10	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
20	SP 0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	DEL 007F
80																
90																
A0	NBSP 00A0	¡ 00A1	¢ 00A2	£ 00A3	¤ 00A4	¥ 00A5	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	ª 00AA	« 00AB	¬ 00AC	- 00AD	® 00AE	¯ 00AF
B0	° 00B0	± 00B1	² 00B2	³ 00B3	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	¹ 00B9	º 00BA	» 00BB	¼ 00BC	½ 00BD	¾ 00BE	¿ 00BF
C0	À 00C0	Á 00C1	Â 00C2	Ã 00C3	Ä 00C4	Å 00C5	Æ 00C6	Ç 00C7	È 00C8	É 00C9	Ê 00CA	Ë 00CB	Ì 00CC	Í 00CD	Î 00CE	Ï 00CF
D0	Ð 00D0	Ñ 00D1	Ò 00D2	Ó 00D3	Ô 00D4	Õ 00D5	Ö 00D6	× 00D7	Ø 00D8	Ù 00D9	Ú 00DA	Û 00DB	Ü 00DC	Ý 00DD	Þ 00DE	ß 00DF
E0	à 00E0	á 00E1	â 00E2	ã 00E3	ä 00E4	å 00E5	æ 00E6	ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ì 00EC	í 00ED	î 00EE	ï 00EF
F0	ð 00F0	ñ 00F1	ò 00F2	ó 00F3	ô 00F4	õ 00F5	ö 00F6	÷ 00F7	ø 00F8	ù 00F9	ú 00FA	û 00FB	ü 00FC	ý 00FD	þ 00FE	ÿ 00FF

Sistemas de codificación alfanumérica: ISO 8859-1

- Ejemplos:
  - $D4_{ISO8859-1}$ → Ô
  - $35A4B4_{ISO8859-1}$ → 5¤´
  - árbol → $E172626F6C_{ISO8859-1}$
  - ¡Hola! → $A1486F6C6121_{ISO8859-1}$
  - $432E492E462E502E_{ISO8859-1}$ → C.I.F.P.

Sistemas de codificación alfanumérica: Código ISO 8859-15

- Este juego de caracteres también se llama
  - Latin-9
  - ISO/IEC 8859-15:1999
- Modifica el ISO-8859-1 eliminando 8 de los caracteres que menos se usaban y añadiendo a cambio el símbolo del Euro (€) y algún carácter más.
- Desde el establecimiento del Euro como moneda ha sustituido al ISO-8859-1 en España, pues incluye todos los caracteres del castellano + el símbolo del Euro.

Sistemas de codificación alfanumérica: Código ISO 8859-15

ISO-8859-15																
	x0	**x1**	**x2**	**x3**	**x4**	**x5**	**x6**	**x7**	**x8**	**x9**	**xA**	**xB**	**xC**	**xD**	**xE**	**xF**
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8x	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
9x	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
Ax	NBSP	¡	¢	£	€	¥	Š	§	š	©	ª	«	¬	SHY	®	¯
Bx	°	±	²	³	Ž	µ	¶	·	ž	¹	º	»	Œ	œ	Ÿ	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Đ	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Sistemas de codificación alfanumérica: ISO 8859-15

- Ejemplos:
  - $D4_{ISO8859\text{-}15}$ → Ô
  - $35A4B4_{ISO8859\text{-}15}$ → 5€Ž
  - árbol → $E172626F6C_{ISO8859\text{-}15}$
  - ¡Hola! → $A1486F6C6121_{ISO8859\text{-}15}$
  - $432E492E462E502E_{ISO8859\text{-}15}$ → C.I.F.P.

Sistemas de codificación alfanumérica: Unicode

- Los juegos de caracteres ISO-8859 tienen la desventaja de ser diferentes para cada idioma, con lo cual para leer cualquier documento codificado con ellos hay que conocer el sistema que usa.
- Para solucionar este problema, se ha trabajado en desarrollar un único código de caracteres universal más amplio en el que estén contenidos todos los símbolos existentes en el mundo. El más respetado actualmente con este fin es el Unicode.

Sistemas de codificación alfanumérica: Unicode

- La última versión de Unicode es la 5.2, de Octubre de 2009, y define 107.361 caracteres distintos.
- Unicode es simplemente una lista que asigna un número llamado punto de código (code point) a cada uno de los símbolos que define.
- Nomenclatura: U+00B4 significa el carácter con code point B4.

Sistemas de codificación alfanumérica: Unicode

- Para facilitar compatibilidad se hizo que los 256 primeros puntos de código de Unicode fuesen iguales a los 256 primeros códigos de ISO-8859-1.
- Sin embargo presenta dos problemas:
  - Necesita varios bytes (2 bytes para la versión 3.0 o 3 para las posteriores) para cada carácter en vez de uno solo
  - Si abrimos un documento con caracteres, aunque sean simples, nos aparecen símbolos “extraños” de por medio.

Sistemas de codificación alfanumérica: UTF-8

- Para solventar los problemas de la codificación “estándar” de Unicode se creó UTF-8.
- UTF = **U**nicode **T**ransformation **F**ormat
- Es el sistema de codificación que se está imponiendo actualmente para representar juegos de caracteres.

Sistemas de codificación alfanumérica: UTF-8

- Utiliza longitud de carácter variable:
  - Los caracteres más comunes (ASCII) los codifica con 1 byte.
  - Los caracteres menos frecuentes utilizan 2, 3, 4... bytes (cuanto menos frecuentes, más bytes utilizan).
- Para ello, usa una estructura variable en la cual el número de 1s al principio del primer byte indica el número total de bytes del carácter

Sistemas de codificación alfanumérica: UTF-8

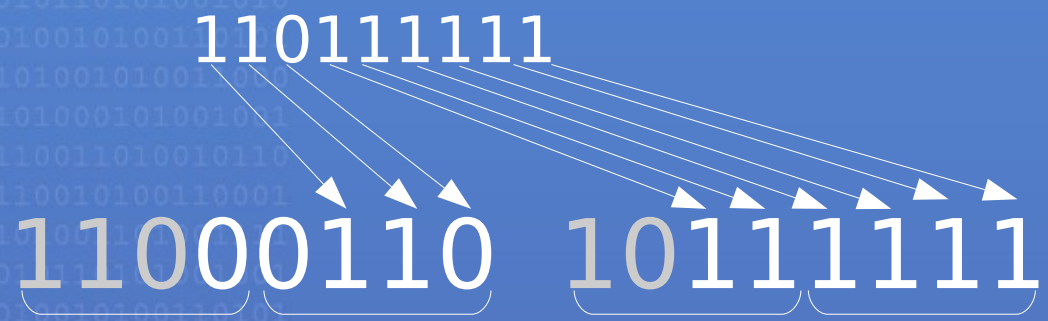
- Esta tabla indica el número de bytes y estructura a usar en función del rango en el que está el Code Point a representar:

Rango Unicode (hexadecimal)	UTF-8 secuencia de octetos (binario)
0000 0000-0000 007F	0xxxxxxx
0000 0080-0000 07FF	110xxxxx 10xxxxxx
0000 0800-0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
0001 0000-0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
0020 0000-03FF FFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
0400 0000-7FFF FFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

El espacio marcado con xxx... debe de ser sustituido por el punto de código del carácter a representar en binario, rellenando con 0s a la izquierda si hiciera falta.

Sistemas de codificación alfanumérica: UTF-8

- Para pasar el carácter ß → ?$_{\text{UTF-8}}$
  - Buscando en tablas de Unicode vemos que ß tiene el punto de código DF (U+00DF).
  - Buscando en la tabla, vemos que DF está en el rango 80-7F, por lo que le corresponde un formato 110xxxxx 10xxxxxx.
  - DF en binario es 110111111
  - Colocamos los bits de DF en su lugar y...



C 6 B F → C6BF$_{\text{UTF-8}}$

Sistemas de codificación alfanumérica: UTF-8

- Ejemplos:
  - $C394_{UTF-8}$ → Ô
  - $35E282ACC5BD_{UTF-8}$ → 5€Ž
  - árbol → $C3A172626F6C_{UTF-8}$
  - ¡Hola! → $C2A1486F6C6121_{UTF-8}$
  - $432E492E462E502E_{UTF-8}$ → C.I.F.P.

Sistemas de codificación alfanumérica

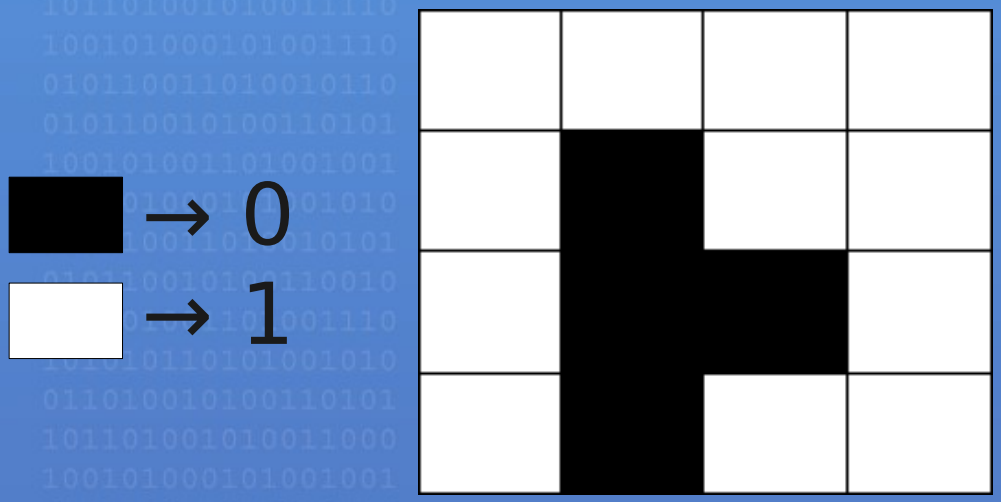
- Estos sistema de codificación que hemos visto son utilizados por programas EDITORES de texto, que guardan únicamente texto, sin formato (.TXT)
- Los programas PROCESADORES de texto han de guardar también información extra como : márgenes, fuentes, tamaños, subrayados, colores, alineados... Para ello tienen que añadir códigos adicionales al texto en formato ISO-8859-15 o UTF-8 (.DOC, .ODT)

Sistemas de codificación gráfica

- Ya sabemos representar números y letras, pero sin embargo, gran parte de la información que manejamos hoy en un ordenador es gráfica: imágenes y dibujos.
- Este tipo de información exige una gran capacidad de memoria.
- Para codificar una imagen cualquier ordenador la divide en una trama de numerosos puntos de colores llamados píxeles y se representa el color de cada uno con uno o varios bits.

Sistemas de codificación gráfica

- Por ejemplo, vamos a representar esta pieza del tetris como un gráfico de 1 bpp (bit pot pixel).

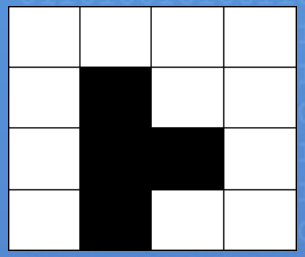


1 1 1 1
1 0 1 1
1 0 0 1
1 0 1 1

Esta imagen sería 1111101110011011, o en hexadecimal F B 9 B

Sistemas de codificación gráfica

Por lo tanto podríamos decir que:



FB9B

GR1B4P

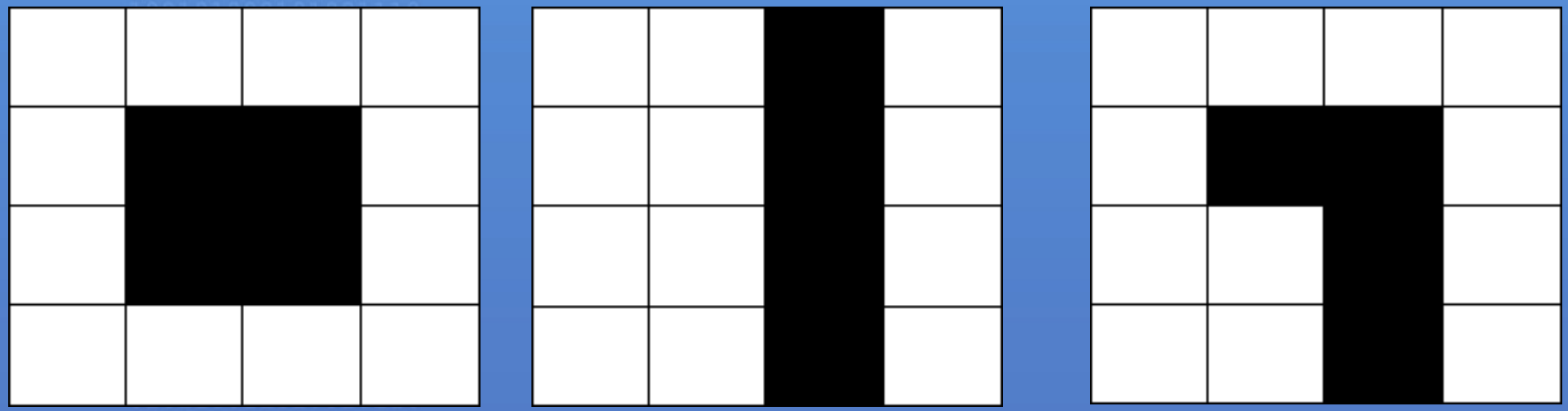
GR=Codificación gráfica

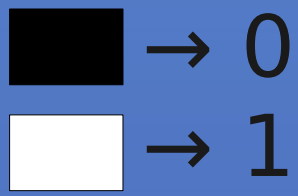
1B=Usando 1 bit por pixel

4P=Gráfico de 4 píxeles de ancho

Sistemas de codificación gráfica

- Usando también 1 bit por píxel y 4 bits de ancho, codifica el resto de las piezas del Tetris:





Sistemas de codificación gráfica

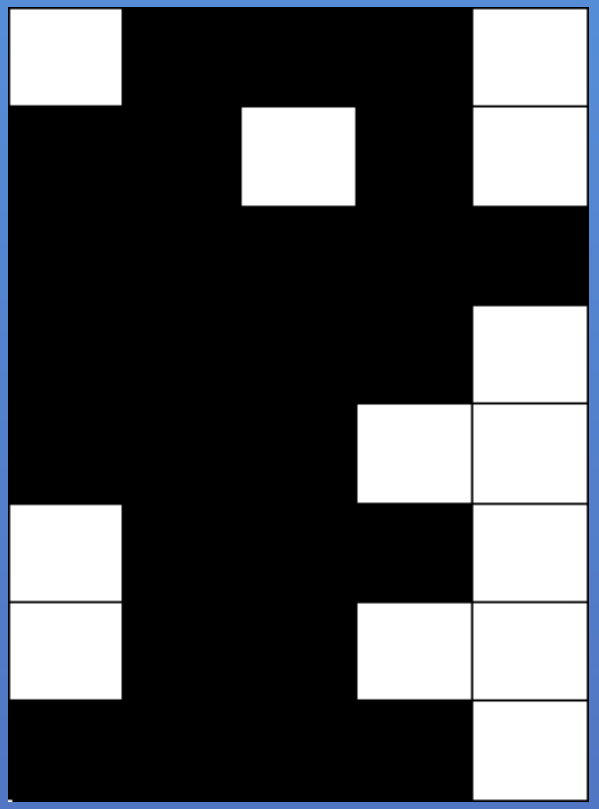
- También podemos hacer lo contrario...

$894011C661_{GR1B5P}$ → ?

1000100101000000001000111000110011000 01

1 0 0 0 1
0 0 1 0 1
0 0 0 0 0
0 0 0 0 1
0 0 0 1 1
1 0 0 0 1
1 0 0 1 1
0 0 0 0 1

→ 0
→ 1



Sistemas de codificación gráfica

- Ahora haz lo contrario, decodifica como imagen de 1 bit por píxel y 11 píxeles de ancho lo siguiente:

208220FE376FFF7F6828D8$_{\text{GR1B11P}}$

?

Sistemas de codificación gráfica

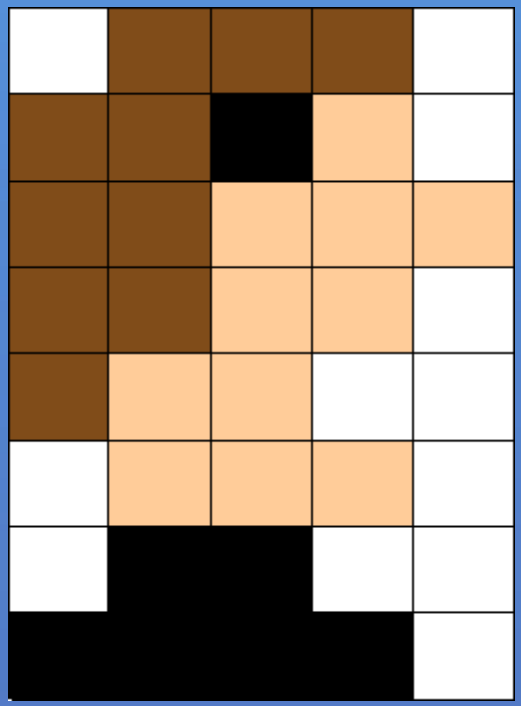
- Si empleamos más bits por píxel, la imagen tendrá tantos colores como combinaciones se puedan hacer con esos bits. Por ejemplo, con 2:

→ 11

→ 10

→ 01

→ 00



11 01 01 01 11
01 01 00 10 11
01 01 10 10 10
01 01 10 10 11
01 10 10 11 11
11 10 10 10 11
11 00 00 11 11
00 00 00 00 11

Es decir,

1101010111010100101101011010100101101011011010111111010101111000011110000000011

D 5 D 4 B 5 A 9 6 B 6 B F A B C 3 C 0 3

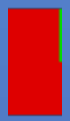
Sistemas de codificación gráfica

- Ahora haz lo contrario, decodifica como imagen de 2 bits por píxel y 10 píxeles...

  3000ABA815AB477655555DD5DD5515574
  1DD5455D001D400050000000 GR2B10P


→ 11


→ 10


→ 01


→ 00

?

Sistemas de codificación gráfica

- Es decir, la imagen anterior (cara) de 5x8 píxeles con 2 bpp (bits por pixel) es equivalente a:

  $D5D4B5A96B6BFABC3C03_{GR2B5P}$

- Si hacemos la cuenta, ocupa 10 bytes:

  2 bits/pixel x 5 píxeles de ancho x 8 píxeles de alto =

  80 bits = 80 / 8 bytes = **10 bytes**

- De esta manera podemos calcular muy fácilmente los bits que ocupa una imagen

Sistemas de codificación gráfica

- Por ejemplo si queremos calcular cuanto ocuparía una imagen de 320x200 píxeles con 8 bits por píxel...

  8 x 320 x 200 = 512000 bits = 512000/8 bytes = 64000 bytes = 64000/1024 Kbytes = 62,5 KB

- ¿Cuanto ocuparía una imagen de 1024 bits de alto, 768 de ancho y 12 bpp?
- ¿Y una imagen de 800x600 y 18 bpp?
- ¿Y una foto de 3664 x 2748 y 36 bpp (cámara Réflex a tope de calidad)?

Sistemas de codificación gráfica

- También conviene saber cuantos colores tiene una imagen con X bits. Por ejemplo...¿Cuántos colores tiene una imagen de 8 bits por píxel?

  Nº colores=nº combinaciones posibles=2^8=256 colores

- También puede ser interesante lo contrario... Por ejemplo, ¿cuantos bits por píxel hacen falta para representar 5000 colores en una imagen?

  2^1=2, 2^2=4, 2^3=8, 2^4=16, 2^5=32, 2^6=64, , 2^7=128, 2^8=256, 2^9=512, 2^{10}=1024, 2^{11}=2048, 2^{12}=4096, 2^{13}=8192 → Hacen falta por lo menos 13 bits

Sistemas de codificación gráfica

- ¿Cuántos colores tiene una imagen de 16 bits por píxel?
- ¿Cuántos colores tiene una imagen de 24 bits por píxel?
- ¿Cuántos bits por píxel hacen falta para representar dos milones de colores?
- ¿Cuántos bits por píxel hacen falta para representar 12 colores?

Sistemas de codificación gráfica

- ¿Cuanto ocupa una imagen normal de fondo de escritorio que tiene 1280 x 1024 píxeles y 16 millones de colores?

  Como 2^{24} = 16777216 → 24 bits
  1280 x 1024 x 24 = 31457280 bits = 3932160 bytes = **3,75 MB**

- ¿ Y una foto de alta calidad de 3600 x 2400 con 1000 millones de colores?

  2^{29} = 536870912 2^{30} = 1073741824 → 30 bits
  3600 x 2400 x 30 = 259200000 bits = 32400000 bytes = **30,9 MB**

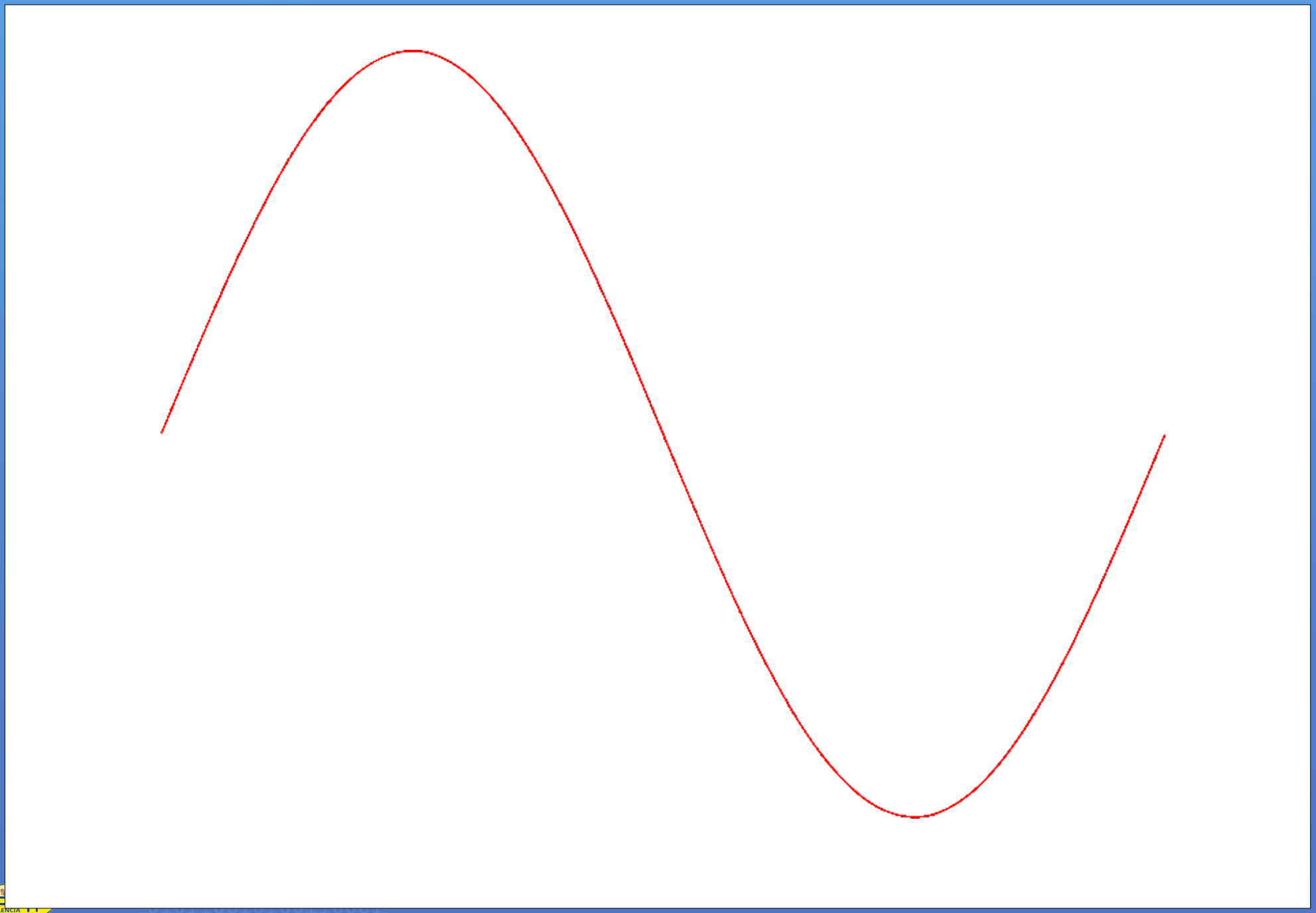
Sistemas de codificación gráfica

- Una imagen puede almacenarse:
  - En bruto o raw: “Tal cual”, sin ningún tipo de compresión ni pérdida de calidad. Todos los tamaños que hemos calculado se refieren a este tipo (.RAW)
  - Sin pérdida (de calidad) o lossless: Se comprime la información, normalmente aprovechando patrones repetidos, sin perder ningún dato después de la descompresión (.PNG)
  - Con pérdida (de calidad) o lossy: Se comprime la información en base a generar una imagen no idéntica, en la que se pierden algunos detalles (.JPG)

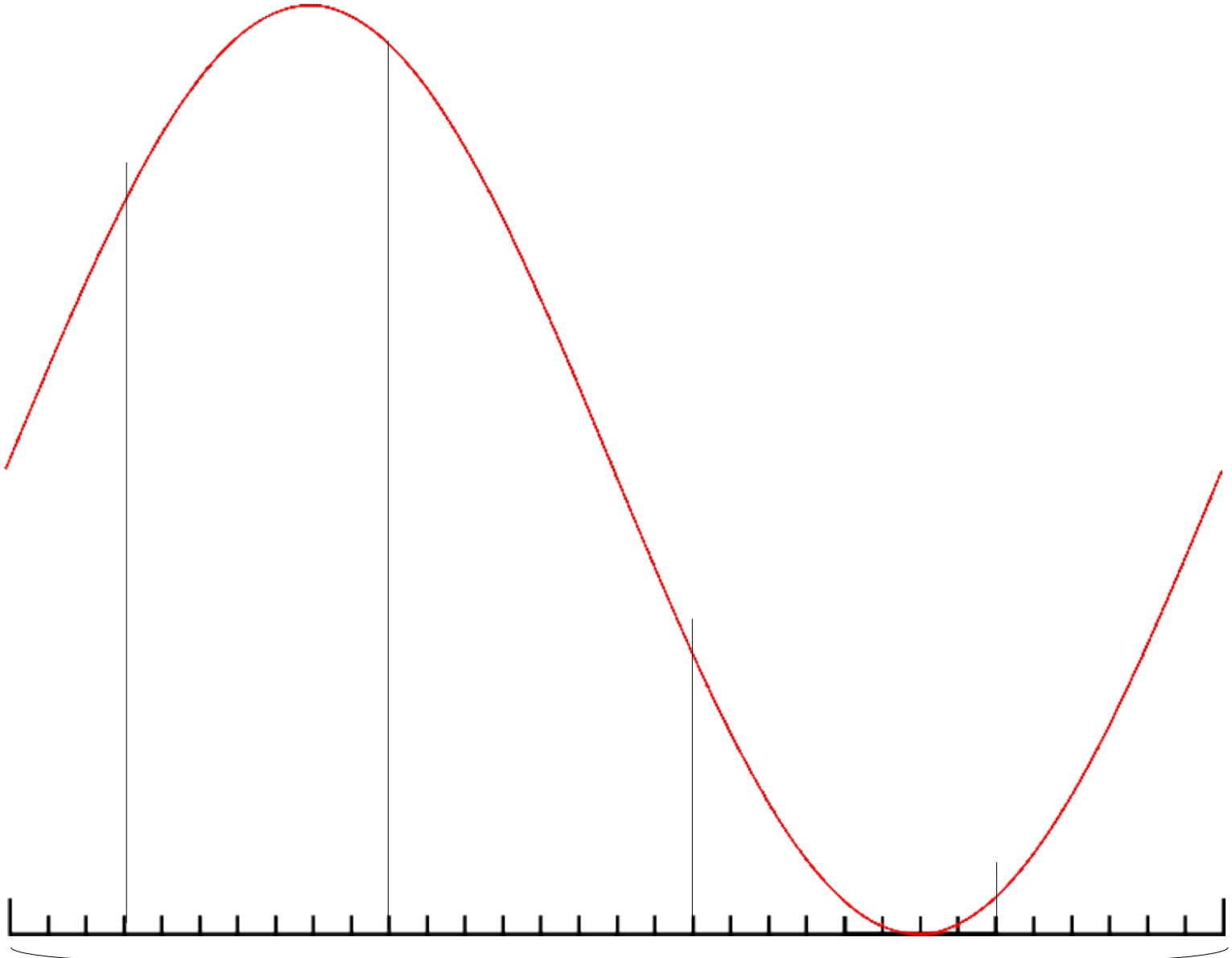
Sistemas de codificación de audio

- Otro tipo de información con la que cualquier ordenador trabaja habitualmente hoy en día es el sonido.
- Para codificar audio hemos de tener en cuenta que cualquier cosa que escuchamos no es más que una serie de ondas que se propagan por el aire y que están en el rango de frecuencias que nuestro oído puede percibir.

Sistemas de codificación de audio

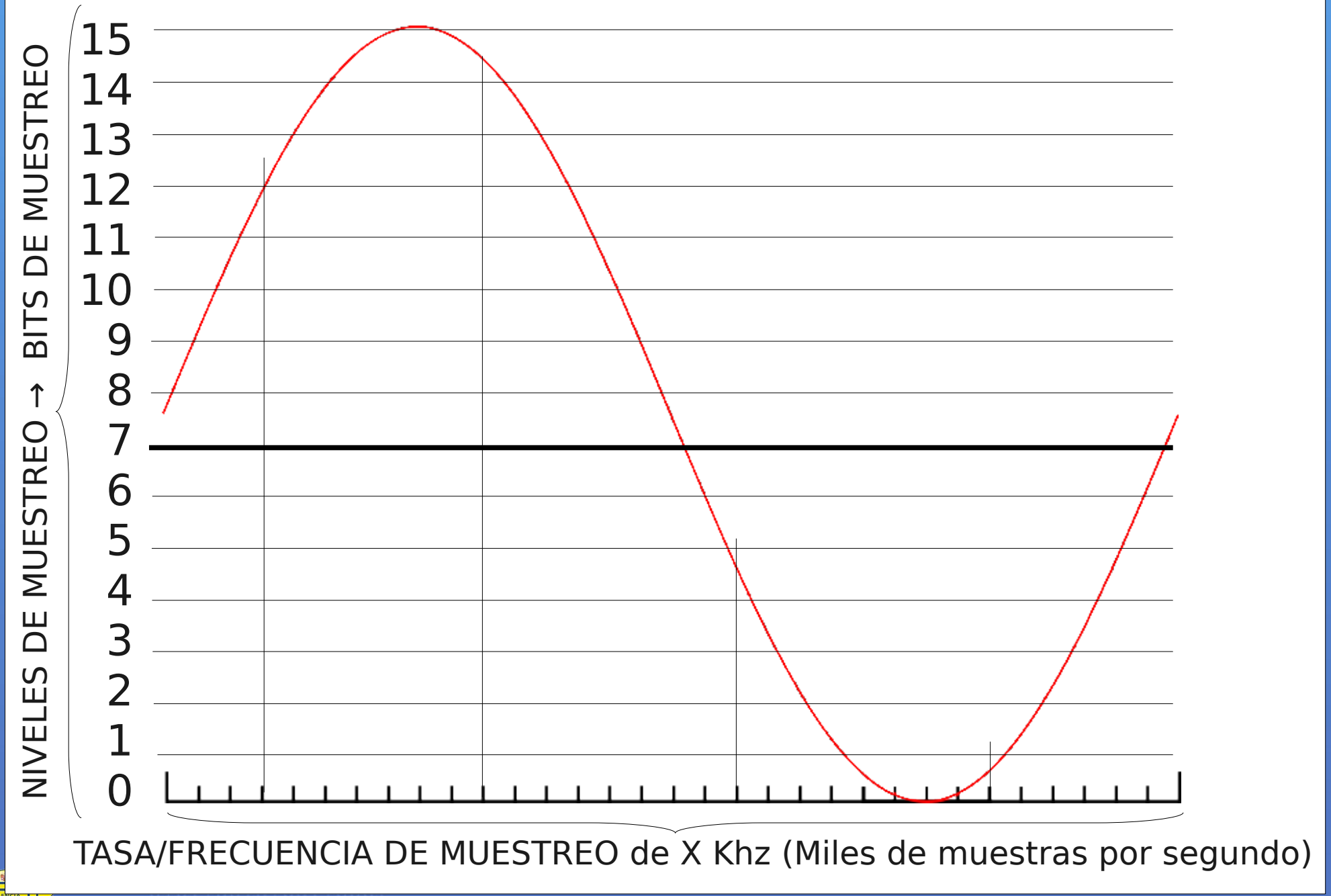


Sistemas de codificación de audio

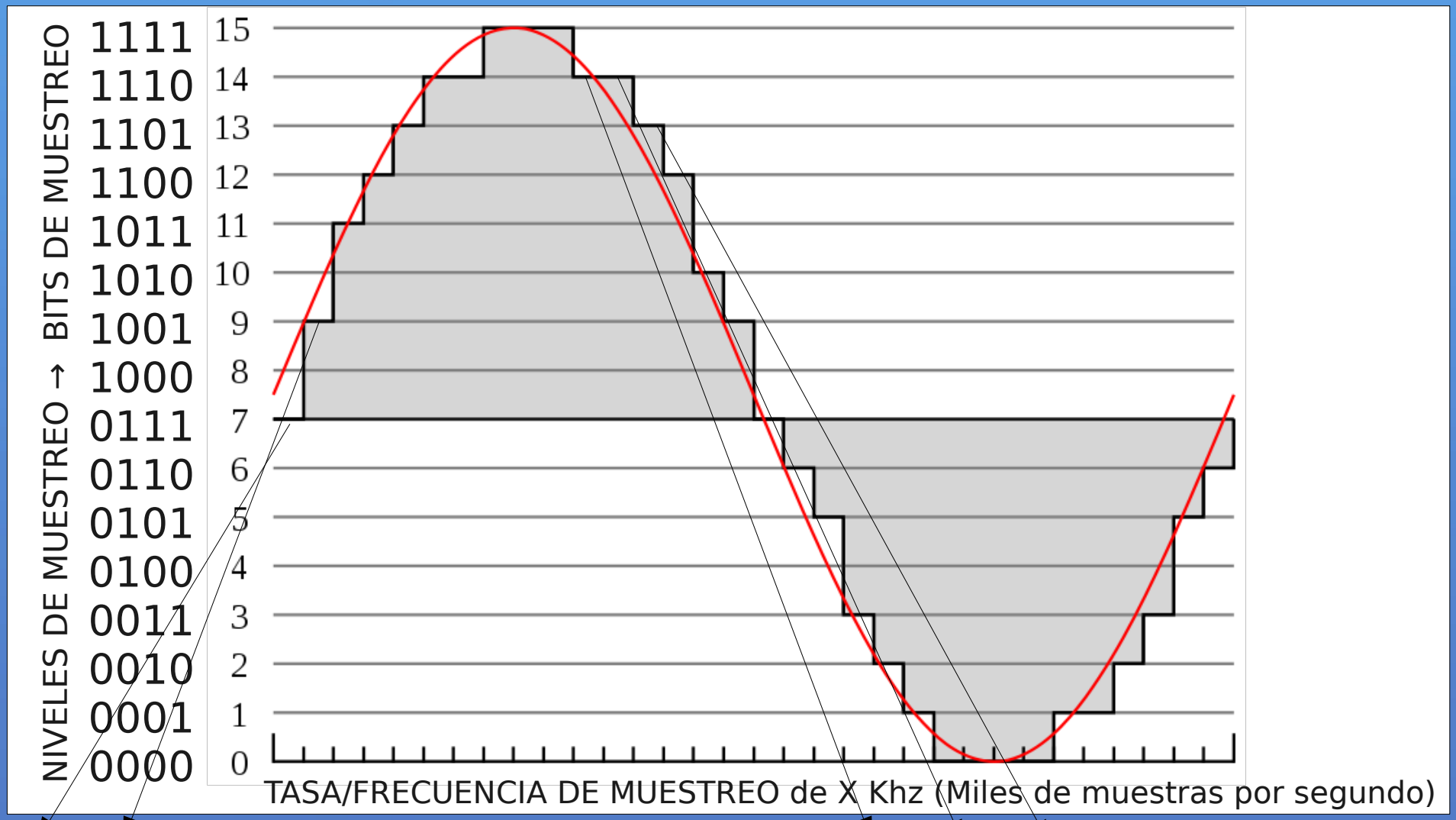


TASA/FRECUENCIA DE MUESTREO de X Khz (Miles de muestras por segundo)

Sistemas de codificación de audio



Sistemas de codificación de audio

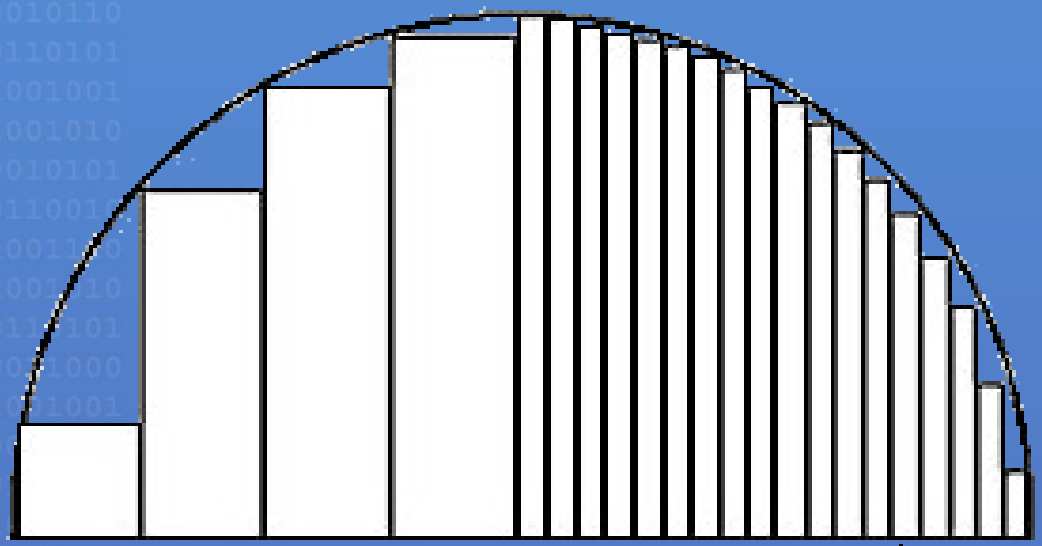


0111100110111100110111101110111111111110111011011100101010010111
0110010100110010000100000000000000000001000100100011010101100111

Tipo de codificación: Audio, a 32 Hz, con 4 bits de muestreo

Sistemas de codificación de audio

- Cuanto más alta sea la frecuencia de muestreo y los bits de muestreo, más se parecerá el sonido digitalizado al original que queremos reproducir.



Sistemas de codificación de audio

- El sonido anterior tiene 32 muestras, y utiliza 4 bits de muestreo... ¿cuánto ocupa?

  32 muestras x 4 bits/muestra = 128 bits = 16 bytes

- Si codificásemos un sonido de 18 segundos a 32 Herzios (muestras/segundo), utilizando 5 bits de muestreo... ¿cuánto ocupa el resultado?

  32 Hz x 18 segundos x 5 bits/muestra = 2880 bits = 360 bytes

- ¿Y 50 segundos a 500 Herzios con 6 bits de muestreo?

  500 Hz x 50 segundos x 6 bits/muestra = 150000 bits = 18,31 KB

Sistemas de codificación de audio

- Las frecuencias de muestreo y bits de muestreo anteriores son, sin embargo totalmente insuficientes. A día de hoy, se considera:
  - Audio de muy mala calidad: frecuencia de muestreo de 8 KHz y 8 bits de muestreo.
  - Audio estándar (calidad CD): Frecuencia de muestreo de 44,1 KHz y 16 bits de muestreo
  - Audio de alta calidad (grabaciones de estudio semiprofesional): 96 KHz y 24 bits de muestreo

Ojo, ¡no confundir Khz con Kbps en los MP3!

Sistemas de codificación de audio

- ¿Cuanto ocuparía una grabación de voz de 1 minuto que me he hecho a 16KHz, con 8 bits de muestreo?

  16000 Hz x 60 segundos x 8 bits = 7680000 bits = 960000 bytes = 937,5 KB

- ¿Cuantos bytes ocuparía una canción de 3 minutos y 20 segundos en calidad CD (esto es 44,1 KHz y 16 bits de muestreo)?

  44100 Hz x 16 bits x 200 segundos x 2 canales = 282240000 bits = 35280000 bytes = 33,65 MB

Ojo, ¡que es estéreo!

Sistemas de codificación de audio

- ¿Cuánto ocupa un efecto de sonido de un disparo a calidad CD que dura 3,3 segundos?

  44100 Hz x 16 bits x 3,3 segundos = 2328480 bits = 291060 bytes = 284,24 KB
- ¿Cuánto ocupa un CD de audio de 74 minutos?

  44100 Hz x 16 bits x 74 minutos x 60 segs x 2 canales = 6265728000 bits = 783216000 bytes = 746,93 MB
- ¿Cuanto ocuparía una grabación de altísima calidad de 2 minutos a 192 Khz y 30 bits de muestreo en estéreo?

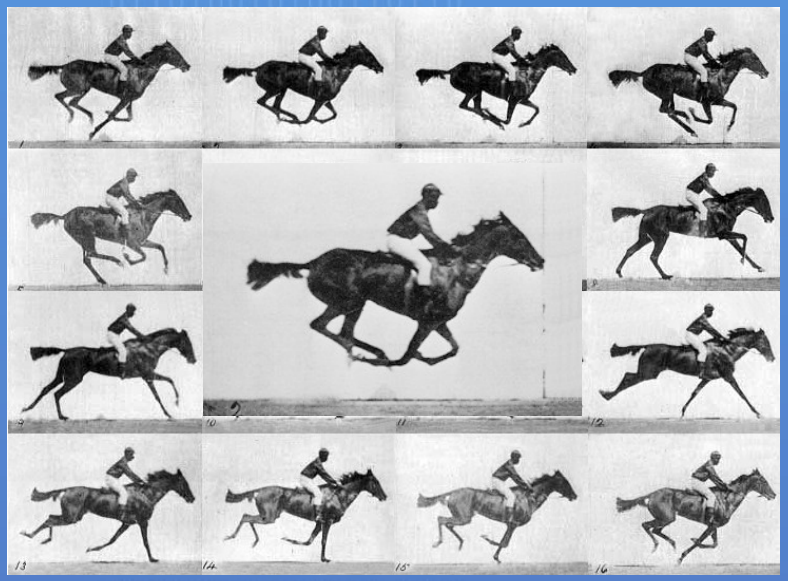
  192000 Hz x 30 bits x 120 segundos x 2 canales = 1382400000 bits = 164,79 MB

Sistemas de codificación de audio

- Al igual que una imagen, el audio puede almacenarse:
  - En bruto o raw: “Tal cual”, sin ningún tipo de compresión ni pérdida de calidad. Todos los tamaños que hemos calculado se refieren a este tipo (.RAW, .PCM, .AIFF, .WAV, CD Audio)
  - Sin pérdida (de calidad) o lossless: Se comprime la información, normalmente aprovechando comportamientos predecibles de la onda, sin perder ningún dato (.FLAC, .APE)
  - Con pérdida (de calidad) o lossy: Se comprime la información normalmente en base a eliminar frecuencias de sonido que el oído humano apenas percibe (.MP3, .OGG, .WMA)

Sistemas de codificación de vídeo

- Cualquier vídeo al fin y al cabo se compone de...



Un montón de imágenes que se muestran consecutivamente de forma rápida

+

Una o varias pistas de audio que suenan de forma simultánea y sincronizada

Sistemas de codificación de vídeo: Modos de escaneo

- Las imágenes que se muestran en pantalla consecutivamente pueden guardarse:
  - En modo de escaneo entrelazado (interlaced scan), si sólo se guarda la mitad de la información en cada imagen (frame): las líneas pares o impares.
  - En modo de escaneo progresivo (progressive scan), si se guarda la información completa en cada imagen (frame).

Sistemas de codificación de vídeo

- Algunos de los formatos de vídeo más populares hoy pueden utilizan comúnmente estas calidades:

FORMATO	IMÁGENES			SONIDO		
	Resolución	Bits de color	Imágenes por segundo	Canales	Frecuencia de muestreo	Bits de muestreo
Video CD	355x288	24	25p	2	44,1 KHz	16
Youtube	320x240 400x226	24	25p	1	22,5 KHz	16
DVD PAL (España)	720x576	24	50i	1-6 (5.1)	48/96 KHz	16/20/24
DVD NTSC (EEUU)	720x480	24	60i	1-6 (5.1)	48/96 KHz	16/20/24
Blu-Ray HD-Ready	1280x720	24	24p/50p	1-8 (7.1)	48/96/192 KHz	16/20/24
Blu-Ray Full-HD	1920x1080	24	24p/50i	1-8 (7.1)	48/96/192 KHz	16/20/24

Sistemas de codificación de vídeo

- ¿Cuanto ocuparía un vídeo de un minuto con resolución de 200x150 píxeles a 10 imágenes por segundo (escaneo progresivo), 24 bits de color y sonido mono con 16 bits de muestreo y 12,25 Khz?
- ¿Cuanto ocuparía un vídeo musical sacado de un DVD en Español (ver características técnicas) que dura 4:41 cuyo sonido es estéreo de 48 KHz y 16 bits de muestreo?
- ¿Cuanto ocuparía una película de DVD en formato PAL cuya duración es de 1 hora y 29 minutos y tiene sonido 5.1 en inglés, español y alemán codificado a 48 KHz y 20 bits de muestreo?

Sistemas de codificación de vídeo

- ¿Cuanto ocuparía un vídeo de un minuto con resolución de 200x150 píxeles a 10 imágenes por segundo (escaneo progresivo), 24 bits de color y sonido mono con 16 bits de muestreo y 12,25 Khz?

Imágenes:
$Tamaño_{imagen}$ = 200 píxeles x 150 píxeles x 24 bits = 720000 bits = 90000 bytes
$Tamaño_{vídeo}$ = 90000 bytes x 10 imagenes/segundo x 60 segundos = 54000000 bytes = 51,5 MB de imágenes de vídeo

Audio:
$Tamaño_{audio}$ = 12250Hz x 16 bits de muestreo x 60 segundos = 11760000 bits de audio = 1470000 bytes de audio = 1,4 MB de audio

$Tamaño_{total} = Tamaño_{vídeo} + Tamaño_{audio}$
$Tamaño_{total}$ = 51,5 MB + 1,4 MB= **<u>52,9 MB</u>**

Sistemas de codificación de vídeo

- ¿Cuanto ocuparía un vídeo musical sacado de un DVD en Español (ver características técnicas) que dura 4:41 cuyo sonido es estéreo de 48 KHz y 16 bits de muestreo?

Se divide ya que usa modo entrelazado y cada imagen sólo guarda la mitad de las líneas

Imágenes:

$Tamaño_{imagen}$ = (720 píxeles x 576 píxeles x 24 bits) / 2 = 4976640 bits = 622080 bytes

$Tamaño_{vídeo}$ = 622080 bytes x 50 imagenes/segundo x 281 segundos = 8740224000 bytes = 8,14 GB de imágenes de vídeo

Audio:

$Tamaño_{audio}$ = 48000Hz x 16 bits de muestreo x 281 segundos x 2 = 431616000 bits de audio = 53952000 bytes de audio = 51,45 MB de audio = 0,05 GB de audio

$Tamaño_{total}$ = $Tamaño_{vídeo}$ + $Tamaño_{audio}$

$Tamaño_{total}$ = 8,14 GB + 0,05 GB= **8,19 GB**

Sistemas de codificación de vídeo

- ¿Cuanto ocuparía una película de DVD en formato PAL cuya duración es de 1 hora y 29 minutos y tiene sonido 5.1 en inglés, español y alemán codificado a 48 KHz y 20 bits de muestreo?

Se divide ya que usa modo entrelazado y cada imagen sólo guarda la mitad de las líneas

Imágenes:

$Tamaño_{imagen}$ = (720 píxeles x 576 píxeles x 24 bits) / 2 = 4976640 bits = 622080 bytes

$Tamaño_{vídeo}$ = 622080 bytes x 50 imagenes/segundo x (89x60) segundos = 166095360000 bytes = 154,688 GB de imágenes de vídeo

Cada canal de audio tiene su propia pista de sonido, y Además son diferentes para cada idioma.

Audio:

$Tamaño_{audio}$ = 48000 Hz x 20 bits de muestreo x (89x60) segundos x 6 canales x 3 idiomas = 92275200000 bits de audio = 11534400000 bytes de audio= 10,742 GB de audio

$Tamaño_{total}$ = $Tamaño_{vídeo}$ + $Tamaño_{audio}$

$Tamaño_{total}$ = 154,688 GB + 10,742 GB= **<u>165,43 GB</u>**

Sistemas de codificación de vídeo

- ¿Cuanto ocuparía un capítulo de "Que vida más triste" del Yotube, que dura 4 minutos 47 segundos, tiene una resolución de a 320 x 240 píxeles con 25 imágenes por segundo (escaneado progresivo) y sonido estéreo a 22,5 KHz a 16 bits de muestreo?
- ¿Cuanto ocuparía cada hora de película Blu-Ray en Full-HD a 1920x1080 a 24 imágenes por segundo progresivas, con sonido 7.1 a 96 Khz con 20 bits de muestreo en 2 idiomas?
- ¿Cuanto ocuparía una película DivX de 1 hora 55 minutos en formato cinemascope que está capturada a 704 x 288 con 25p y tiene sonido estéreo con calidad CD (44,1 KHz y 16 bits de muestreo)?

Sistemas de codificación de vídeo

- Al igual que el audio e imágenes, un vídeo puede almacenarse:
  - En bruto o raw: “Tal cual”, sin ningún tipo de compresión ni pérdida de calidad. Todos los tamaños que hemos calculado se refieren a este tipo (.RAW)
  - Sin pérdida (de calidad) o lossless: Se comprime la información, normalmente aprovechando patrones repetidos dentro de una misma imagen o imágenes contiguas, sin perder ningún dato. Apenas se usa (.FFV1, .MSU)
  - Con pérdida (de calidad) o lossy: Se comprime la información normalmente en base a lo mismo que la codificación sin pérdida, pero modificando audio y vídeo (.MPG, .AVI, .MKV)