Tema 3: Codificación de la información

509: Informática básica

2006/2007

Índice

Multimedia: sonido y vídeo

Compresión

Run-length encoding :: Compresión con y sin pérdidas

Formatos de imagen

Formatos de sonido

Formatos de vídeo

Formatos de documento

Formatos abiertos y cerrados

Conceptos básicos

La Informática trata de la **adquisición**, **representación**, **tratamiento** y **transmisión** de la información. Para llevar a cabo automáticamente estas operaciones se utilizan unas máquinas llamadas ordenadores.

Definición R.A.E. para el término **Informática**: Conjunto de conocimientos científicos y técnicas que hacen posible el tratamiento automático de la **información** por medio de ordenadores.

La figura siguiente muestra un esquema de lo que se debe entender por tratamiento automático de la información. La información consistente en un conjunto de datos de entrada, se somete a unas operaciones o transformaciones automáticas para producir unos resultados o datos de salida.

Información

Tratamiento de la Información

Resultados

Datos de Entrada $\Rightarrow$ Operaciones o Transformaciones $\Rightarrow$ Datos de Salida

Algunos ejemplos de tratamiento de la información pueden ser:

- ordenar un conjunto de datos,
- dar formato a un texto,
- calcular estadísticas (media, desviación típica, moda, etc.) de un conjunto de valores numéricos,
- analizar el código de un programa escrito en C y producir el código máquina correspondiente, listo para ser ejecutado en un ordenador determinado.

Un **ordenador**, **computador** o **computadora**, es una máquina capaz de aceptar unos datos de entrada, efectuar con ellos operaciones aritméticas y lógicas, y proporcionar la información resultante a través de un medio de salida.

Todo ello se realiza de forma automática **bajo el control de un programa** de instrucciones previamente almacenado en la memoria del propio ordenador.

Los **datos** son conjuntos de símbolos utilizados para expresar o representar un valor numérico, un hecho, un objeto o una idea, y que se presentan en la forma adecuada para ser manipulados.

Pueden ser cosas tan diferentes como una temperatura, una altura, una matrícula de coche, una frase de un libro, una imagen, etc.

¿Cómo representamos estos datos en el ordenador de forma adecuada?

Una **codificación** es una transformación de los elementos de un conjunto en los de otro, de tal manera que a cada elemento del primer conjunto le corresponde un elemento distinto del segundo. Por tanto, **asocia representaciones** (signos o símbolos) **a los elementos de un conjunto** (datos o "significados").

Ejemplo: En occidente codificamos los números con sucesiones de símbolos de un "alfabeto" con 10 elementos: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Ponemos en correspondencia una sucesión de símbolos con un conjunto (infinito) de significados concretos: los números naturales.

$$
\begin{aligned}
1 &\longrightarrow \circ \\
3 &\longrightarrow \circ\,\circ\,\circ \\
10 &\longrightarrow \circ\,\circ\,\circ\,\circ\,\circ\,\circ\,\circ\,\circ\,\circ\,\circ \\
15 &\longrightarrow \circ\,\circ\,\circ\,\circ\,\circ\,\circ\,\circ\,\circ\,\circ\,\circ\,\circ\,\circ\,\circ\,\circ\,\circ
\end{aligned}
$$

Ejemplo: Ahora tenemos un alfabeto compuesto por dos símbolos: $\{A, B\}$. Codificamos el conjunto: $\{\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \theta, \kappa, \lambda, \omega\}$. Asociamos una secuencia distinta formada por A's y B's a cada letra griega:

$$\begin{array}{cccccc} \alpha & \rightarrow & A & \zeta & \rightarrow & BBB \\ \beta & \rightarrow & B & \theta & \rightarrow & ABAB \\ \gamma & \rightarrow & AA & \kappa & \rightarrow & BABA \\ \delta & \rightarrow & BB & \lambda & \rightarrow & AAAB \\ \epsilon & \rightarrow & AAA & \omega & \rightarrow & BBBA \end{array}$$

Unos y ceros

Los sistemas **digitales** son aquellos en los que los datos están representados mediante una magnitud física que varía de forma **discreta**. Un sistema digital en el que los valores posibles son sólo dos se llama **binario**. En la actualidad, **los ordenadores son digitales binarios**.

A la **unidad de información** empleada en los ordenadores se le conoce con el nombre de **bit** ("**bi**nary digi**t**"): sólo puede tener uno de dos posibles valores (0 ó 1). ¿Cómo interpretamos el significado de un bit individual?

- todo/nada,
- cierto/falso,
- sí/no,
- uno/cero.

La tecnología actual de los ordenadores hace que resulte fácil almacenar y manejar bits ya que en un ordenador digital binario, los componentes más básicos son **dispositivos electrónicos** con **dos estados estables** (representados lógicamente mediante 0 y 1). De esta forma, todas las informaciones se representan mediante **códigos binarios**.

- El valor 1 puede consistir en que hay un cierto **voltaje** en un circuito y el valor 0 en la **ausencia de dicho voltaje** (o la presencia de un voltaje diferente).
- También pudiera ser que el valor 1 consista en que un condensador está **cargado** y el valor 0, en que está **descargado**.
- O bien, el valor 1 puede consistir en que la superficie de cierto material presenta un **agujero**, y 0 en que es **lisa**.
- etc.

Además, existe un **valor añadido**: representar sólo símbolos de un alfabeto tan reducido:

- **simplifica en gran medida el diseño** de ordenadores,
- **facilita su construcción** a gran escala (se abaratan mucho los costes ya que los circuitos a utilizar son bastante simples) y
- **los hace más fiables** (es más fácil construir circuitos sencillos que no fallen).

Memoria

La **memoria principal** (memoria **RAM**) de un ordenador sirve para almacenar **temporalmente** programas y datos. El procesador sólo puede ejecutar un programa cuando éste está almacenado en la memoria principal. Sin embargo, la memoria RAM tiene la desventaja de ser **volátil** y de tener una **capacidad relativamente pequeña** (en relación con dispositivos de almacenamiento como los discos duros).

La memoria principal de un ordenador se compone de una gran cantidad de celdillas, y cada celdilla contiene sólo una unidad de información. En este sentido es un vasto **almacén de bits**. Un ordenador actual de prestaciones modestas almacena y maneja varios miles de millones de bits. Por ejemplo, un CD-ROM, que es un tipo de memoria de ordenador, puede almacenar cerca de 6.000 millones de bits.

Es frecuente utilizar ciertos prefijos para expresar las enormes capacidades de almacenamiento con las que trabajamos:

Prefijo	Núm. bits		Aprox.
Kilobit (Kb.)	1024	2^{10}	$\approx 10^3$ (mil)
Megabit (Mb.)	$1024 \times 1024 = 1\,048\,576$	2^{20}	$\approx 10^6$ (millón)
Gigabit (Gb.)	$1024 \times 1024 \times 1024 = 1\,073\,741\,824$	2^{30}	$\approx 10^9$ (mil mill.)
Terabit (Tb.)	$1024 \times 1024 \times 1024 \times 1024 = 1\,099\,511\,627\,776$	2^{40}	$\approx 10^{12}$ (billón)

Direcciones y agrupamientos

En principio, cada bit de la memoria es accesible mediante un número: su **posición** o **dirección**.

El ordenador puede hacer preguntas como "*¿qué valor tiene ahora el bit que ocupa la posición 653 373 106?*" o ejecutar acciones como "*haz que el bit con dirección 1 015 valga 1*".

Raramente tiene interés manejar un bit aislado. Es habitual que la información se codifique utilizando un **grupo de bits**. Los ordenadores organizan la memoria como una serie de grupos de bits **de tamaño fijo**.

Así, ya se dispone de **gran capacidad expresiva** para representar instrucciones y datos en la memoria principal, ya que cada grupo puede contener múltiples combinaciones de ceros y unos.

La "potencia expresiva" de un bit es muy pequeña: sólo puede representar dos valores o estados diferentes. Con **un bit** solamente podría **codificar** los elementos de un **conjunto de cardinal 2**.

¿Y con **2 bits**? Posibles combinaciones: 00, 01, 10 y 11. Podría llegar a representar hasta **4 valores distintos**. Observa que, simplemente **añadiendo un bit**, he **duplicado** la potencia expresiva.

¿Y con **4 bits**? 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110 y 1111. En total, **16**.

En general, con n **bits** puedo representar hasta 2^n valores distintos. Cuantos más bits agrupe, entonces mi "potencia expresiva" será mayor: **multiplicaré por 2** la que ya tenía antes **si añado un nuevo bit** a la agrupación. Luego, en la práctica tenemos la "potencia expresiva" suficiente como para representar casi cualquier tipo de información: sonido, imágenes, vídeo. . .

La agrupación de bits más frecuente es la de 8 bits. **Cada 8 bits forman un byte** (u octeto).

Ejemplo: Un byte arbitrario:

01101110

O sea que, en realidad, cada bit individual no tiene una dirección, sino cada grupo de 8 bits o byte.

Preguntas como las de antes se formulan más bien así "*¿qué valor tiene ahora el byte (los 8 bits) que ocupa la posición 653 373 106?*" o "*haz que el byte con dirección 1 015 valga 00110101*".

En realidad, cuando se quiere indicar la capacidad de memoria RAM (principal) de un ordenador o bien la de un medio de almacenamiento secundario (por ejemplo un disco duro), se expresa en múltiplos de bytes:

Término	Núm. bits	Aprox.
Kilobyte (KB)	$8 \times 1\,024 = 8\,192$	$\approx 10^3$ (mil) bytes
Megabyte (MB)	$8 \times 1024^2 = 1\,048\,576$	$\approx 10^6$ (millón) bytes
Gigabyte (GB)	$8 \times 1024^3 = 1\,073\,741\,824$	$\approx 10^9$ (mil millones) bytes
Terabyte (TB)	$8 \times 1024^4 = 1\,099\,511\,627\,776$	$\approx 10^{12}$ (billón) bytes

Fíjate en las siguientes proporciones:

- Unos 1000 bytes (en realidad, 1024) son 1 KB.
- Unos 1000 KB (en realidad, 1024) son 1 MB.
- Unos 1000 MB (en realidad, 1024) son 1 GB.

¿Qué clase de información podemos representar con bits y grupos de bits?

La respuesta es: prácticamente cualquier tipo de información.

En este tema nos encargamos de ver cómo se puede representar y manipular información con bits:

- números,
- textos,
- imágenes,
- sonido,
- etc.

Sistemas de representación posicional de los números

Sólo utilizamos 10 dígitos para expresar los números naturales y, sin embargo, podemos representar números arbitrariamente grandes: la posición de los dígitos en un número cambia su "significado".

Ejemplo:

$$2105 = 2 \times 10^3 + 1 \times 10^2 + 0 \times 10^1 + 5 \times 10^0$$

Cada posición numérica presenta un **peso** asociado. La posición p (contando desde la derecha y empezando en 0) tiene peso 10^p.

El valor 10 es la **base** del sistema de numeración **decimal**.

No todos los sistemas de numeración son posicionales. El sistema numérico romano, por ejemplo, no lo es.

Ejemplo: El número 2105 se escribe como MMCV en el sistema numérico romano.

Puesto que el sistema romano de numeración **no** es posicional, se requieren más símbolos para representar números cada vez más grandes.

Ejemplo: Para representar números hasta 100 se utilizan los símbolos {I,V,X,L,C}; para representar números hasta 1000 se requieren dos símbolos más: D y M.

Es posible definir sistemas de numeración con otras bases.

Uno especialmente interesante en informática es el **sistema binario**, es decir, el sistema de **base 2**.

En base 2, cada dígito puede ser un 0 o un 1, es decir, cada dígito es un bit.

El sistema binario

La posición p (contando desde la derecha y empezando en 0) tiene peso 2^p.

Ejemplo: El número binario 1001 representa el valor decimal 9

$$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 9$$

El bit de más a la izquierda recibe el nombre de **bit más significativo** (es el de mayor peso) y el de más a la derecha, **bit menos significativo**.

Ejemplo:

bit más significativo ↓ bit menos significativo ↓

1 0 1 1 1

Aquí tienes una tabla con los primeros treinta números binarios y otros más junto a su equivalente decimal.

binario	decimal	binario	decimal	binario	decimal	binario	decimal
0	0	1010	10	10100	20	11110	30
1	1	1011	11	10101	21	101000	40
10	2	1100	12	10110	22	110010	50
11	3	1101	13	10111	23	111100	60
100	4	1110	14	11000	24	1000110	70
101	5	1111	15	11001	25	1010000	80
110	6	10000	16	11010	26	1011010	90
111	7	10001	17	11011	27	1100100	100
1000	8	10010	18	11100	28		
1001	9	10011	19	11101	29		

Ciertos números son "**ambiguos**": si sólo constan de unos y ceros, no está clara la base:

- ¿10 es diez (interpretación en base 10) o dos (interpretación en base 2)?
- ¿111 es ciento once (interpretación en base 10) o siete (interpretación en base 2)?

Utilizaremos una notación en la que, cuando convenga, emplearemos un **subíndice** para indicar la base:

- 10_2 es dos, pues indicamos que está en base 2.
- 10_{10} es diez, pues indicamos que está en base 10.

Ejercicio 1: ¿Qué valores decimales corresponden a los siguientes números binarios?

a) 1111_2

b) 1_2

c) 0_2

d) 0111_2

e) 10_2

f) 1000_2

Ejercicio 2: ¿Cuál es el valor más grande que podemos expresar con cuatro bits?, ¿y con ocho?, ¿y con n, siendo n un entero positivo?

Conversión de números decimales a otra base

Debes saber que, en general, podemos transformar un número decimal **entero** a otra base **dividiendo reiteradamente por la base en cuestión**:

1.- Realizar una división **entera** entre 2 del número decimal entero. El resto siempre será 0 ó 1, ya que el divisor es 2.

2.- Considerar el cociente obtenido como nuevo número en decimal a dividir y aplicar el paso 1.

3.- Repetir los pasos 1 y 2 hasta que el cociente obtenido sea 1. Siempre que se sigan estos pasos se llegará a obtener un cociente 1, excepto si el número entero en decimal a dividir es 0 ó 1 (en cuyo caso su representación en binario es la misma que en decimal).

4.- La representación del número entero decimal en binario se obtiene **tomando el resto de cada división y el último cociente en orden inverso**.

Ejemplo: ¿Cómo convertir el valor decimal 108 a base 2?

$$\begin{array}{r|l} 108 & 2 \\ \hline \end{array}$$

Ejemplo: ¿Cómo convertir el valor decimal 108 a base 2?

108	2
08	54
0	

Ejemplo: ¿Cómo convertir el valor decimal 108 a base 2?

108	2	
08	54	2
0		

Ejemplo: ¿Cómo convertir el valor decimal 108 a base 2?

108	2		
08	54	2	
0	14	27	
	0		

Ejemplo: ¿Cómo convertir el valor decimal 108 a base 2?

108	2		
08	54	2	
0	14	27	2
	0		

Ejemplo: ¿Cómo convertir el valor decimal 108 a base 2?

108	2		
08	54	2	
0	14	27	2
	0	07	13
		1	

Ejemplo: ¿Cómo convertir el valor decimal 108 a base 2?

108	2			
08	54	2		
0	14	27	2	
	0	07	13	2
		1		

Ejemplo: ¿Cómo convertir el valor decimal 108 a base 2?

108	2			
08	54	2		
0	14	27	2	
	0	07	13	2
		1	**1**	6

Ejemplo: ¿Cómo convertir el valor decimal 108 a base 2?

108	2				
08	54	2			
0	14	27	2		
	0	07	13	2	
		1	**1**	6	2

Ejemplo: ¿Cómo convertir el valor decimal 108 a base 2?

108	| 2						
08	54	| 2					
0	14	27	| 2				
	0	07	13	| 2			
		1	**1**	6	| 2		
				0	3		

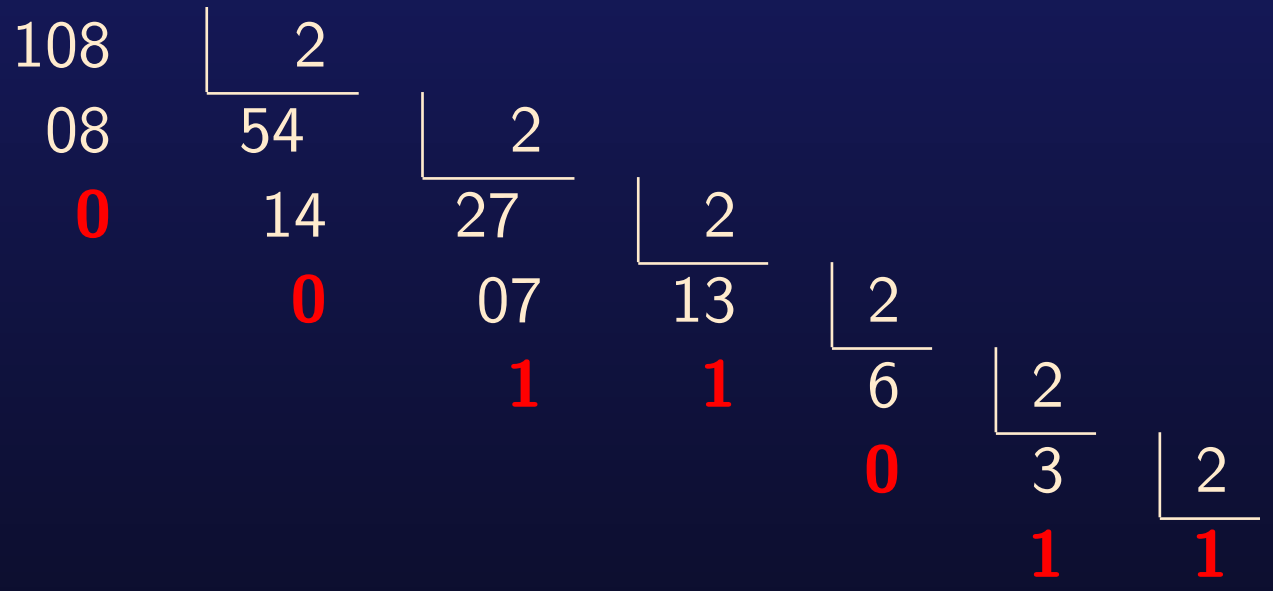
Ejemplo: ¿Cómo convertir el valor decimal 108 a base 2?

108	2					
08	54	2				
0	14	27	2			
	0	07	13	2		
		1	**1**	6	2	
				0	3	2

Ejemplo: ¿Cómo convertir el valor decimal 108 a base 2?

108	| 2						
08	54	| 2					
0	14	27	| 2				
	0	07	13	| 2			
		1	**1**	6	| 2		
				0	3	| 2	
					1	**1**	

Ejemplo: ¿Cómo convertir el valor decimal 108 a base 2?



Solución: 108_{10} es 1101100_2.

Ejercicio 3: Convierte los siguientes números decimales a base 2:

a) 0

b) 1

c) 18

d) 128

e) 127

f) 863

g) 2^{10}

Suma binaria

Antes de aprender a sumar en base 2, hemos de conocer la "tabla de sumas":

sumando	sumando	suma	acarreo
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Ahora podemos sumar números en base dos de modo similar a como sumamos números en base 10: llevando el exceso de una columna a la siguiente. Ya sabes: aquello de ". . . ocho y seis, catorce y llevo una. . . " ahora se convierte en ". . . uno y uno, cero y llevo uno. . . " ;-)

	Acarreo			
	1	0	1	1
+			1	1

Ahora podemos sumar números en base dos de modo similar a como sumamos números en base 10: llevando el exceso de una columna a la siguiente. Ya sabes: aquello de ". . . ocho y seis, catorce y llevo una. . . " ahora se convierte en ". . . uno y uno, cero y llevo uno. . . " ;-)

			1		Acarreo
	1	0	1	1	
+			1	1	
				0	

Ahora podemos sumar números en base dos de modo similar a como sumamos números en base 10: llevando el exceso de una columna a la siguiente. Ya sabes: aquello de ". . . ocho y seis, catorce y llevo una. . . " ahora se convierte en ". . . uno y uno, cero y llevo uno. . . " ;-)

Ejemplo:

		1	1		Acarreo
	1	0	1	1	
+			1	1	
			1	0	

Ahora podemos sumar números en base dos de modo similar a como sumamos números en base 10: llevando el exceso de una columna a la siguiente. Ya sabes: aquello de ". . . ocho y seis, catorce y llevo una. . . " ahora se convierte en ". . . uno y uno, cero y llevo uno. . . " ;-)

Ejemplo:

		1	1		Acarreo
	1	0	1	1	
+			1	1	
		1	1	0	

Ahora podemos sumar números en base dos de modo similar a como sumamos números en base 10: llevando el exceso de una columna a la siguiente. Ya sabes: aquello de ". . . ocho y seis, catorce y llevo una. . . " ahora se convierte en ". . . uno y uno, cero y llevo uno. . . " ;-)

Ejemplo:

		1	1		Acarreo
	1	0	1	1	
+			1	1	
	1	1	1	0	

Ejercicio 4: Calcula las siguientes sumas de números binarios:

a) $101011_2 + 100101_2$

b) $1_2 + 1001_2$

c) $10000_2 + 10000_2$

d) $101010_2 + 10101_2$

e) $101010_2 + 101010_2$

Circuitería para las sumas

Es posible implementar circuitos capaces de llevar a cabo **operaciones lógicas** con bits: **1 es verdadero**, **0 es falso**.

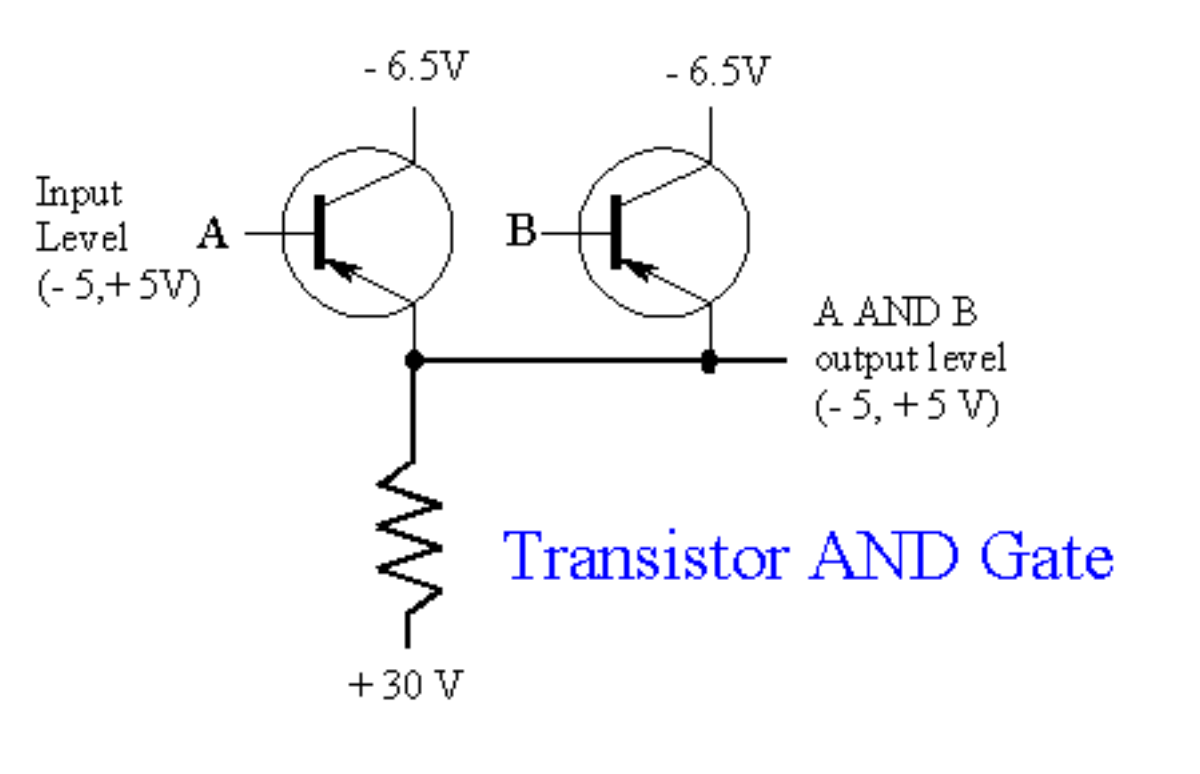
Hay tres operaciones lógicas básicas:

X	Y	X and Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X or Y
0	0	0
0	1	1
1	0	1
1	1	1

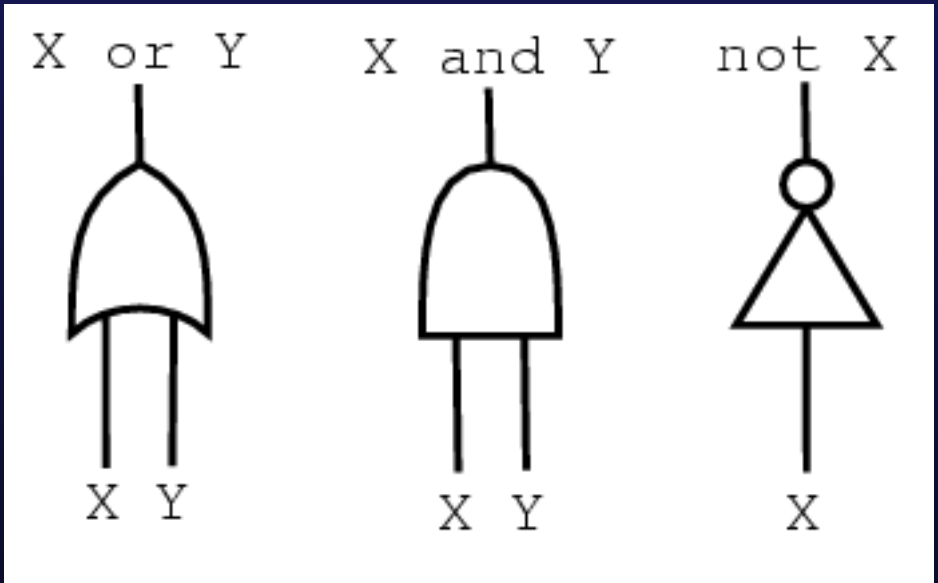
X	not X
0	1
1	0

Cada operación lógica puede implementarse con circuitería electrónica sencilla (un par de transistores y una resistencia):



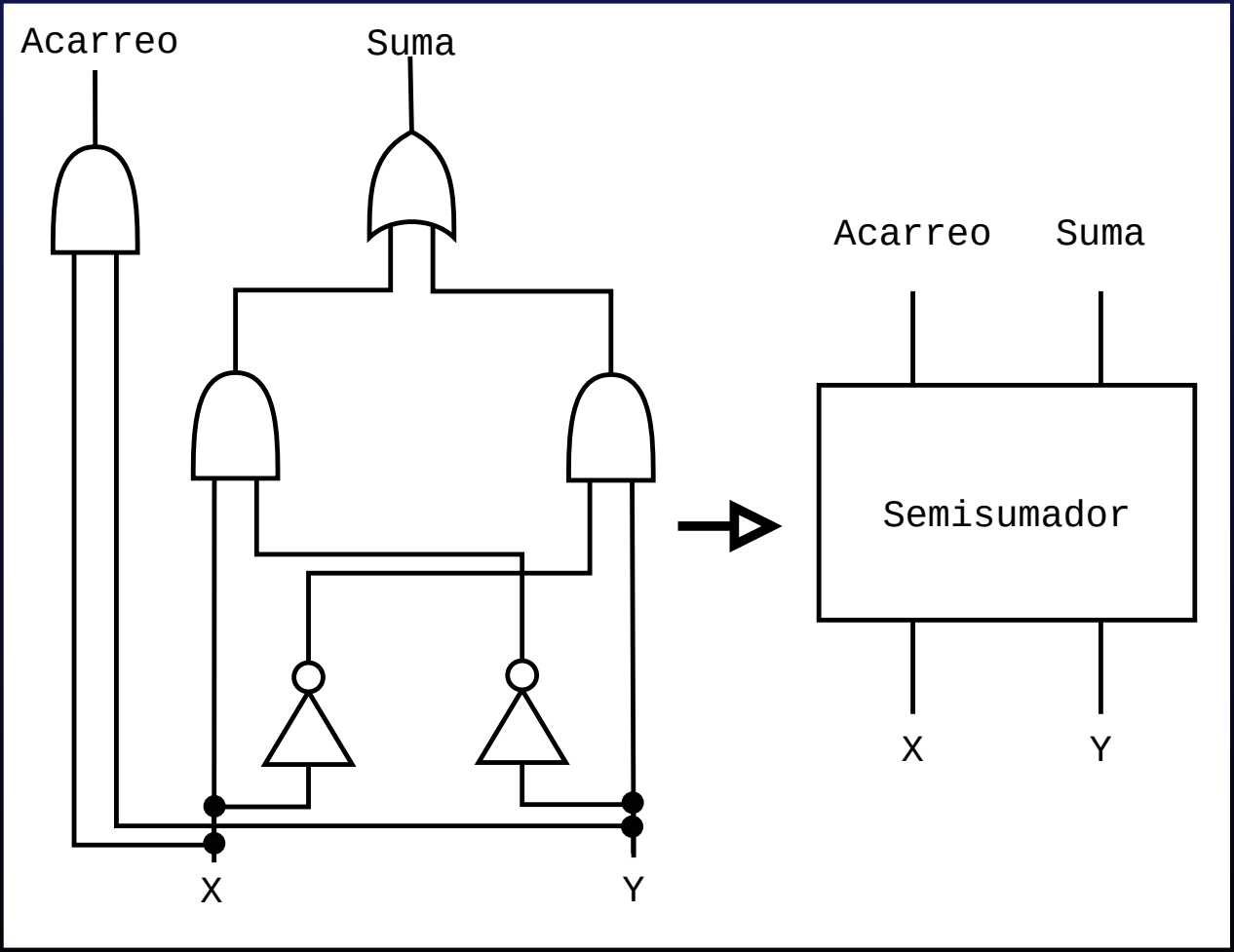
En un chip caben millones de transistores.

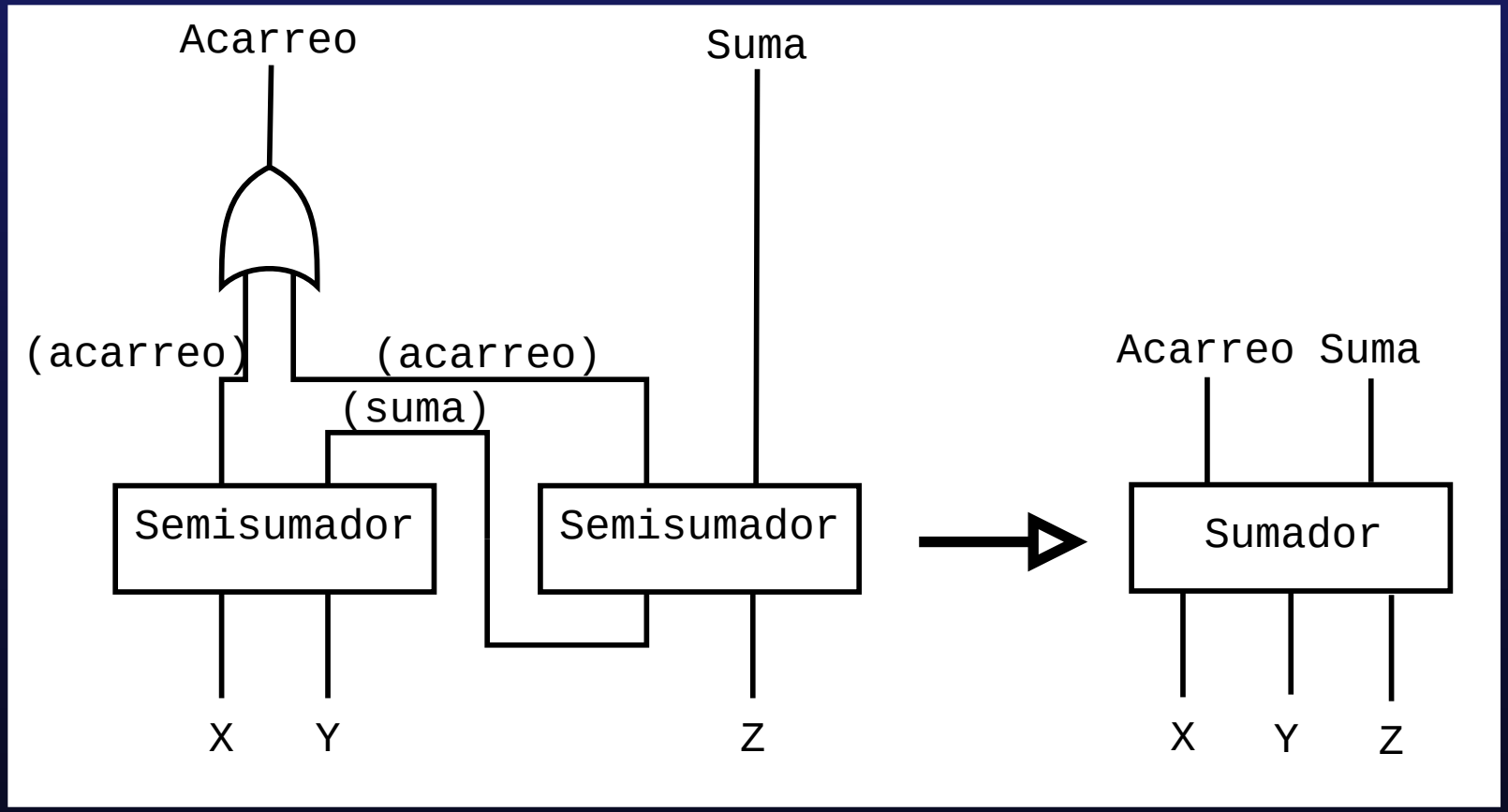
En los circuitos electrónicos lógicos (es decir, que manejan bits) las operaciones lógicas se representan gráficamente con unos símbolos:



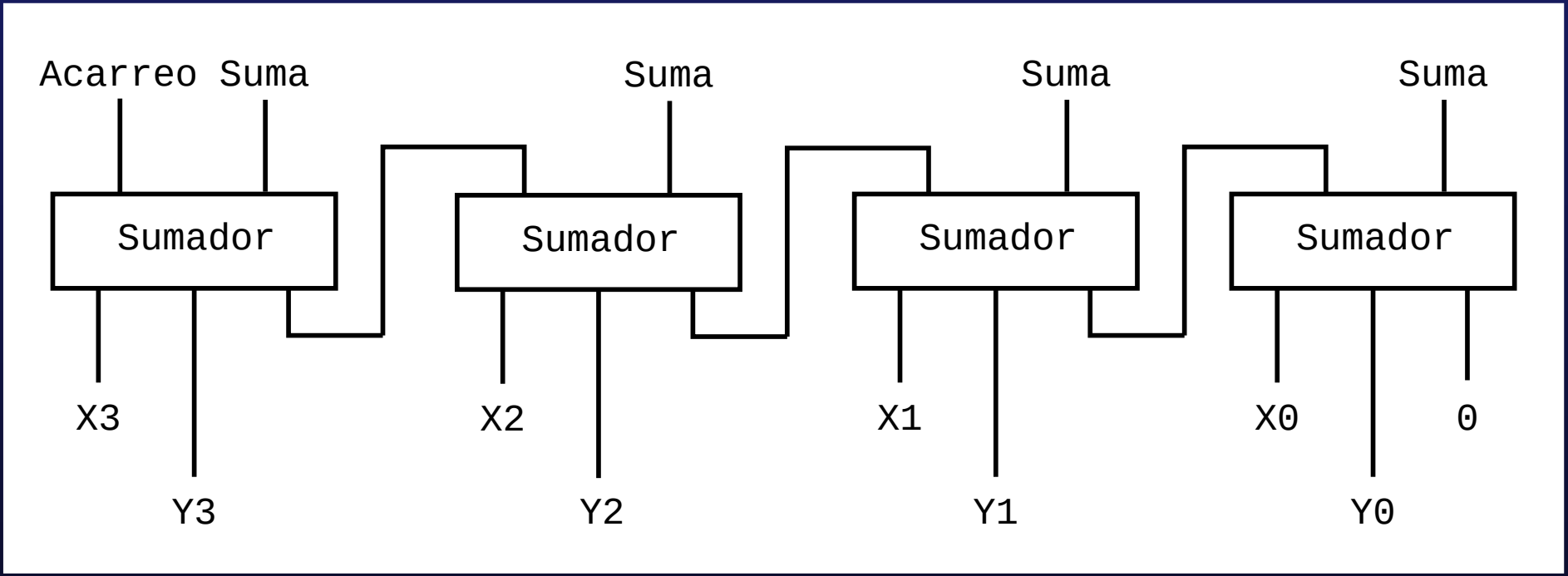
Combinando circuitos lógicos, podemos implementar operaciones más complicadas que las tres operaciones básicas.

La circuitería con la que se implementa la tabla de la suma es relativamente sencilla (este circuito suma 2 bits y obtiene correctamente su suma y el valor del acarreo):





Dos semisumadores y una puerta lógica OR bastan para sumar correctamente 3 bits y producir su acarreo.



Con cuatro sumadores consigo implementar un circuito capaz de sumar correctamente dos operandos compuestos por secuencias de 4 bits.

Podemos efectuar otras operaciones aritméticas con números binarios extendiendo de forma natural los métodos que aplicamos al operar con números en base 10: resta, multiplicación y división.

No lo vamos a ver. Nuestro objetivo didáctico se cumple sobradamente con la ilustración de la suma (esperamos que sepas extraer los importantes conceptos básicos explicados en el proceso).

Ahora bien, para simplificar la circuitería de los ordenadores, la resta se plantea como una suma en la que el segundo sumando cambia de signo. Y esto nos conduce a otro importante concepto básico.

¿Cómo podemos representar números con signo?

Números con signo

Los números que hemos aprendido a representar en base 2 son positivos.

Podríamos representar **números negativos** con un procedimiento similar al que adoptamos al trabajar con el sistema decimal: introduciendo el símbolo "–" (signo negativo).

Surge un problema: en un ordenador sólo podemos almacenar los símbolos 1 y 0 *y el signo es un símbolo adicional*.

Solución: se trabaja con **números de ancho fijo** (típicamente 8, 16, 32 ó 64 bits) y **se codifica el signo en el bit más significativo**.

Fijar la anchura no es un artificio forzado. La tecnología de los ordenadores actuales hace natural el agrupamiento de bits en bloques de 8, 16, 32 ó 64 unidades.

Ejemplo: Con anchura 8, por ejemplo, el valor 2 es

0	0	0	0	0	0	1	0

Ten en cuenta que fijar una anchura determinada para los números presenta algunos problemas:

- La suma de dos números de p bits puede ocupar $p+1$ bits. Se dice entonces que se ha producido un **desbordamiento** (si se produce un acarreo al final).

  La unidad aritmética del ordenador da un aviso cuando se produce un desbordamiento, de modo que el programador pueda tomar la decisión más acertada.

- El producto de dos números de p bits puede ocupar $2 \times p$ bits.

  Las operaciones de multiplicación devuelven un número codificado en el doble de bits.

Hay varias codificaciones distintas para números con signo y anchura fija.

Nosotros estudiaremos brevemente:

- **complemento a 1**,
- **complemento a 2**.

Complemento a 1

En la notación en complemento a 1:

- Se escribe el **valor absoluto** del número en binario.
- Y, si es negativo, a continuación **se invierten sus bits** (los unos pasan a ser ceros y los ceros unos).

Por interés didáctico, asumiremos en adelante que trabajamos con una anchura fija de 4 bits.

Ejemplo: El valor 2 se codifica como 0010, y el valor -2 como 1101.

Ejercicio 5: Convierte a base 10 estos números binarios en complemento a uno de cuatro bits:

a) 1101

b) 0101

c) 0000

d) 1111

Ejercicio 6: ¿Cuáles son los números mayor y menor que podemos expresar en complemento a uno de cuatro bits?

Veamos un ejemplo de suma empleando la representación en complemento a 1.

Ejemplo: 1_{10} más -1_{10} da 0:

$$\begin{array}{rcccc} & 0 & 0 & 0 & 1 \\ + & 1 & 1 & 1 & 0 \\ \hline & 1 & 1 & 1 & 1 \end{array}$$

observa que la representación de signo y magnitud **no funciona**:
$0001 + 1001 = 1010$

Pero la suma no siempre funciona bien.

Ejemplo: Sumemos 3 y -2. El resultado debe ser 1.

$$\begin{array}{rcccc} & 0 & 0 & 1 & 1 \\ + & 1 & 1 & 0 & 1 \\ \hline (1) & 0 & 0 & 0 & 0 \end{array}$$

El bit adicional aparece porque hay un **desbordamiento** en la suma.

Cuando hay un desbordamiento, el resultado es incorrecto pero se soluciona fácilmente: **basta con sumar un uno al resultado**.

Ejemplo: El resultado correcto del ejemplo es 0001.

$$\begin{array}{rcccc} & 0 & 0 & 1 & 1 \\ + & 1 & 1 & 0 & 1 \\ \hline (1) & 0 & 0 & 0 & 0 \end{array}$$

$$\downarrow$$

$$\begin{array}{rcccc} & 0 & 0 & 0 & 0 \\ + & 0 & 0 & 0 & 1 \\ \hline & 0 & 0 & 0 & 1 \end{array}$$

Ejercicio 7: Efectúa las siguientes operaciones codificando los números en base 10 a sus equivalentes en complemento a uno con cuatro bits:

a) $3 + 2$

b) $3 - 2$

c) $1 + 1$

d) $1 - 1$

e) $7 - 7$

f) $-2 - 2$

g) $-2 + 3$

Problemas con el complemento a 1:

- Hay dos formas diferentes de representar el cero: 0000 y 1111.
- La **suma** es relativamente **lenta**: una vez realizada debe comprobarse el valor del acarreo y, si es 1, sumar uno al resultado.

Complemento a 2

La notación en complemento a 2 permite que la sumas sean más sencillas (y rápidas) y presenta una forma única para el cero.

Es la que se usa realmente en los computadores.

Para **cambiar el signo** de un número:

1. Se **complementan todos los bits** del número (los 1 pasan a ser 0 y viceversa).
2. Y **se suma 1** al resultado.

Ejemplo: Para representar el valor -4 (con 4 bits) partimos de la representación binaria del número 4: 0100; se complementan sus bits: 1011; y se suma 1 al resultado: 1100.

La suma en complemento a 2 es una suma convencional en la que *descartamos* los bits de exceso (si los hay).

Ejemplo: Sumemos 3 y -2. El resultado debe ser 1.

$$\begin{array}{rcccc} & 0 & 0 & 1 & 1 \\ + & 1 & 1 & 1 & 0 \\ \hline (1) & 0 & 0 & 0 & 1 \end{array}$$

Ejercicio 8: Efectúa las siguientes operaciones en complemento a 2 con 4 bits:

a) $3 + 2$

b) $3 - 2$

c) $1 + 1$

d) $1 - 1$

e) $7 - 7$

f) $-2 - 2$

g) $-2 + 3$

Números con parte fraccionaria

Los números fraccionales en los sistemas numéricos posicionales se representan usando pesos negativos para la parte decimal.

Ejemplo: En base 10, el número $13,14$ es

$$1 \times 10^1 + 3 \times 10^0 + 1 \times 10^{-1} + 4 \times 10^{-2}$$

Ejemplo: En base 2, el número $10,01$ es

$$1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$$

es decir, es el número $2,25$ en base 10.

Conversión de números fracciones en base 10 a base 2

- Se hace la conversión de la parte entera mediante divisiones sucesivas por 2, tal como vimos. . .
- . . . y la parte fraccionaria se procesa así: Se toma la parte decimal del número y se multiplica reiteradamente por dos, quedándose con la parte entera como cifra a añadir al número binario, hasta llegar a un número entero (es decir, **con parte decimal nula**).

Ejemplo: Conversión del número decimal $4{,}125$ a binario:

- la parte entera es 100 (se puede calcular por divisiones sucesivas),
- y la parte decimal se calcula así:

1. $0{,}125 \times 2 = 0{,}25$, así que 0 es el primer decimal.
2. $0{,}25 \times 2 = 0{,}5$, así que 0 es el segundo decimal.
3. $0{,}5 \times 2 = 1{,}0$, así que 1 es el tercer decimal,

- y como la parte decimal es nula $(1, 0)$, hemos acabado y el número resultante es $100{,}001$.

Ejercicio 9: Codifica en binario los siguientes números en base 10:

- $10,25$
- $128,125$
- $1,75$
- $2,875$
- $3,3$

Notación exponencial o de coma flotante

Un número en coma flotante consta de:

- signo
- mantisa
- base
- exponente

Estas partes se combinan así:

$$\text{signo} \quad \text{mantisa} \times \text{base}^{\text{exponente}}$$

Hay varias representaciones posibles para un mismo número en coma flotante.

Ejemplo: Supongamos que trabajamos en el sistema decimal. El número -50 puede representarse así (y de más formas) si usamos el valor 10 como base:

signo y mantisa	base$^{\text{exponente}}$
-50000	10^{-3}
-5000	10^{-2}
-500	10^{-1}
-50	10^{0}
$-5{,}0$	10^{1}
$-0{,}5$	10^{2}
$-0{,}05$	10^{3}

Hay una representación que denominamos **normalizada**:

- su mantisa es un número menor que 10,
- y su parte entera es no nula.

Ejemplo: La forma normalizada de -50 es $-5{,}0 \times 10^1$.

La IEEE (Institute of Electrical and Electronics Engineers) ha producido un estándar para **coma flotante en binario**. Es el formato **IEEE 754** que define dos tipos de números en coma flotante:

- números en coma flotante de **32 bits**,
- números en coma flotante de **64 bits**.

Como ejemplo de interés didáctico, detallaremos el formato de 32 bits:

- es un formato normalizado,
- 1 bit para el signo (el bit más significativo),
- 8 bits para el exponente, el cual se representa como número entero con desplazamiento 127; es decir, el 0 se escribe como 01111111 (o sea $127 + 0 = 127$); el 1 como 10000000 $(127 + 1 = 128)$,
- 23 bits para la mantisa (en realidad, 24, pero como el bit más significativo siempre es 1 –en el formato normalizado la parte entera no puede ser 0, por tanto en el sistema binario sólo puede ser 1– se omite),
- la base tiene valor 2.

Ejemplo: El número binario $-1{,}11_2$ se representa así en coma flotante normalizada:

$$-1{,}11_2 \times 2^0$$

en el formato IEEE 754:

1	01111111	11000000000000000000000

El número $0{,}0000001101011_2$ se representa así en coma flotante normalizada:

$$1{,}101011_2 \times 2^{-7}$$

en el formato IEEE 754:

0	01111000	10101100000000000000000

¿Qué **ventajas** tiene el formato de coma flotante de la IEEE?

- Cubre un **gran rango** de valores.

  Por ejemplo, la norma IEEE 754 para coma flotante de 64 bits puede representar números entre -10^{308} y 10^{308}.

- Su **precisión se "adapta"** a la magnitud del número.

  Por ejemplo, en el mismo formato, podemos representar números tan pequeños como 10^{-323}.

- Es una **representación estándar**, con lo que podemos intercambiar informaciones de este tipo entre diferentes ordenadores.

Has de ser consciente de que:

- No todo número real puede representarse como un número en coma flotante ($3{,}3$, por ejemplo):
  - Hay un rango acotado,
  - y dentro de él, errores de redondeo.
- Las operaciones matemáticas afectan a la precisión de los resultados.
- Las operaciones aritméticas con flotantes son mucho más costosas.

  (Hasta la aparición del 486 era necesario instalar un *coprocesador* matemático en los ordenadores para poder efectuar operaciones en coma flotante a una velocidad razonable.)

Caracteres y texto

Ya hemos visto que es posible representar:

- **números enteros**, siempre en un **rango acotado** (p.e., 32 bits, 1 bit para el signo: $2^{31} - 1 = 2147483647$) pero **fielmente** y
- **números reales**, no sólo en un **rango acotado**, sino también con una **precisión** (número de decimales) **limitada** (p.e., hay infinitos números reales en $]0, 1[$), de ahí que la representación **no sea fiel** y hablemos de números de coma flotante (y no reales)

usando sólo bits.

¿Cómo representar letras y caracteres especiales?

ASCII

La codificación **ASCII** (*American Standard for Coded Information Interchange*) es un formato que usa 7 bits para representar cada uno de 128 símbolos.

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
010		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
101	P	Q	R	S	T	U	V	W	X	Y	Z	[	/	]	^	_
110	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
111	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

- Ciertos símbolos son **caracteres de control** (en la tabla se muestran con códigos de tres letras): sirven para controlar los dispositivos en los que se muestra el texto (impresoras, pantallas).
  Por ejemplo, DEL significa "borrar" (*delete*), CR es "retorno de carro" (*carriage return*), etc.
- Se respeta el **orden alfabético** en las minúsculas, en las mayúsculas y en los dígitos: si un carácter va alfabéticamente delante de otro, su código ASCII es menor que el de ese otro.
- Las **mayúsculas** tienen códigos que van del 1000001 al 1011010 y las **minúsculas** del 1100001 al 1111010: **se diferencian en un bit** (es fácil pasar de mayúsculas a minúsculas sumando una cantidad fija al código binario que representa cada letra).
- **No hay representación para las letras** acentuadas, la letra "ñ" o signos de puntuación del **español**, **catalán**, **francés**. . .

Cada carácter ocupa 7 bits, así que:

- es posible representarlo como un valor entre 0 y 127,
- ocupa un byte (aunque sobra un bit).

Una página de texto mecanografiado ocupa unos 2 KB (contiene unos 2000 caracteres).

Ejercicio 10: ¿Cuántas páginas de texto mecanografiado caben en un CD-ROM? (un CD-ROM permite albergar unos 700 MB)

Como muchos lenguajes usan símbolos que no pertenecen a la tabla ASCII, hay muchas **extensiones estándar**.

Las que más nos interesan son las extensiones **ISO-8859-1** (también conocida como **Latin1**) e **ISO-8859-15**:

- usan **un bit** más (con lo que cada carácter ocupa 8 bits y sigue "cabiendo" en un byte),
- extienden la tabla para incorporar los símbolos comunes a **lenguas románicas occidentales**,
- la **ISO-8859-15** incluye además el símbolo del **euro**.

Texto

Un texto es una sucesión de caracteres. El único problema estriba en especificar su longitud. Hay dos soluciones comúnmente adoptadas:

- indicar al principio la longitud del texto,
- utilizar un símbolo terminador (por ejemplo, el valor 0).

Ejemplo: La palabra “Pepe” se puede representar

- indicando la longitud en primer lugar (el primer byte es el número 4):

00000100	01010000	01100101	01110000	01100101
Número 4	Letra P	Letra e	Letra p	Letra e

- o usando un carácter nulo (NUL), por ejemplo, como terminador:

01010000	01100101	01110000	01100101	00000000
Letra P	Letra e	Letra p	Letra e	Terminador (0)

Unicode

- **8 bits por carácter son insuficientes** para representar los símbolos de uso común en las lenguas naturales,
- la **gran variedad de formatos** alternativos al ASCII dificulta el intercambio de información,

Por ello, se ha diseñado una representación de **16 bits** por letra que permite representar 65536 símbolos diferentes: **Unicode**.

Nos encontramos en un momento de **transición** de las codificaciones de 8 bits a Unicode.

℀	2100	ℐ	2110	SM	2120	ℰ	2130		2150	←	2190	↠	21A0	↰	21B0	—	21C0	⇐	21D0	⇠	21E0
℁	2101	ℑ	2111	TEL	2121	ℱ	2131		2151	↑	2191	↡	21A1	↱	21B1	→	21C1	⇑	21D1	⇡	21E1
ℂ	2102	ℒ	2112	TM	2122	∃	2132		2152	→	2192	↜	21A2	↲	21B2	↓	21C2	⇒	21D2	⇢	21E2
℃	2103	ℓ	2113	℣	2123	ℳ	2133		2153	↓	2193	↝	21A3	↳	21B3	↕	21C3	⇓	21D3	⇣	21E3
℄	2104	℔	2114	ℤ	2124	ℴ	2134		2154	↔	2194	↤	21A4	↴	21B4	⇄	21C4	⇔	21D4	⇠	21E4
℅	2105	ℕ	2115	℥	2125	ℵ	2135		2155	↕	2195	↥	21A5	CR	21B5	⇅	21C5	⇕	21D5	⇢	21E5
℆	2106	№	2116	Ω	2126	ℶ	2136		2156	↖	2196	↦	21A6	↶	21B6	⇆	21C6	⇖	21D6	⇦	21E6
ℇ	2107	℗	2117	℧	2127	ℷ	2137		2157	↗	2197	↧	21A7	↷	21B7	⇇	21C7	⇗	21D7	⇧	21E7
℈	2108	℘	2118	ℨ	2128	ℸ	2138		2158	↘	2198	↨	21A8	↸	21B8	⇈	21C8	⇘	21D8	⇨	21E8
℉	2109	ℙ	2119	℩	2129		2139		2159	↙	2199	↩	21A9	↹	21B9	⇉	21C9	⇙	21D9	⇩	21E9
ℊ	210A	ℚ	211A	K	212A		213A		215A	↚	219A	↪	21AA	↺	21BA	⇊	21CA	⇚	21DA	⇪	21EA
ℋ	210B	ℛ	211B	Å	212B		213B	⅛	215B	↛	219B	↫	21AB	↻	21BB	⇋	21CB	⇛	21DB		21EB
ℌ	210C	ℜ	211C	ℬ	212C		213C	⅜	215C	↮	219C	↬	21AC	↼	21BC	⇌	21CC	⇜	21DC		21EC
ℍ	210D	ℝ	211D	ℭ	212D		213D	⅝	215D	↯	219D	↭	21AD	↽	21BD	⇍	21CD	⇝	21DD	⇳	21ED
ℎ	210E	℞	211E	℮	212E		213E	⅞	215E	↚	219E	↮	21AE	↾	21BE	⇎	21CE	⇞	21DE		21EE
ℏ	210F	℟	211F	ℯ	212F		213F		215F	↛	219F	↯	21AF	↿	21BF	⇏	21CF	⇟	21DF		21EF

Representación gráfica

Aunque hemos visto que podemos representar con 8 bits cada carácter y el texto como una secuencia de caracteres, en pantalla no aparecen como esas secuencias de bits, sino como imágenes.

- ¿Cómo es que vemos las secuencias de bits como imágenes?
- ¿Cómo se representan esas imágenes?

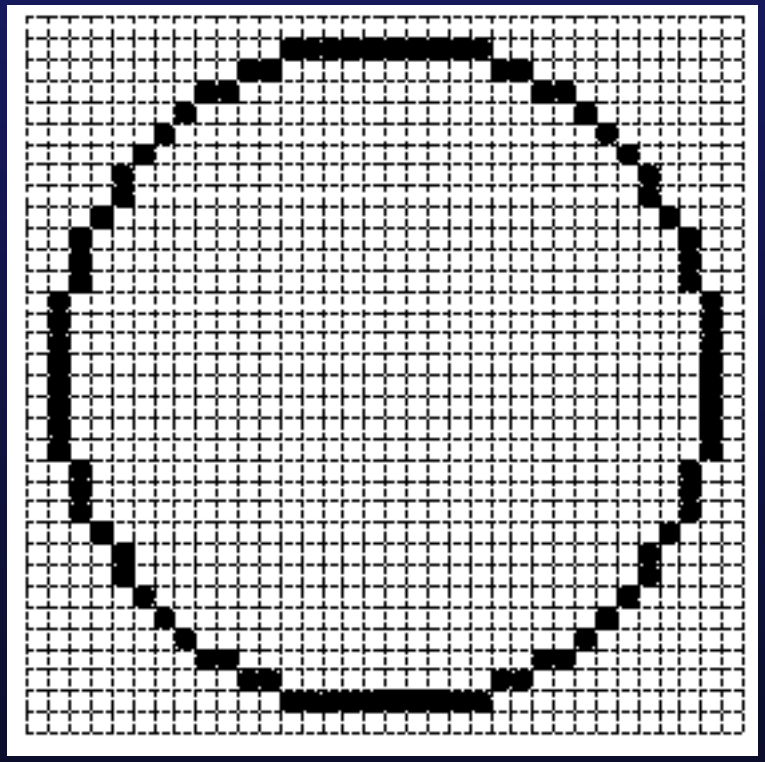
Imágenes

Las imágenes también se representan con bits (en un ordenador *todo* se representa con bits).

Una imagen es una **matriz de píxels**.

Un **píxel** es un punto de la imagen.

En el caso más sencillo, dicho valor es un 0 o un 1: imagen en blanco y negro (puro).

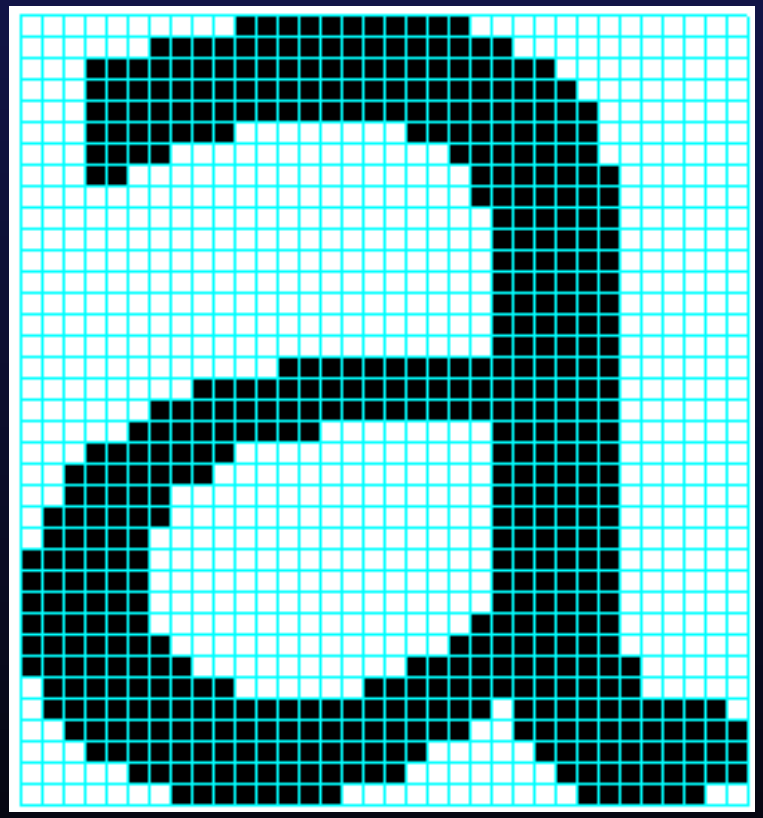


```
00000000000000000000000000000000
00000000000111111111000000000000
00000000011000000000110000000000
00000001100000000000001100000000
00000010000000000000000010000000
...
```

Ejercicio 11: ¿Cuántos bytes ocupa una imagen en blanco y negro de 800×600 píxels?

Recuperemos ahora la cuestión de cómo se muestran los caracteres en pantalla.

Cada carácter es una imagen en blanco y negro:



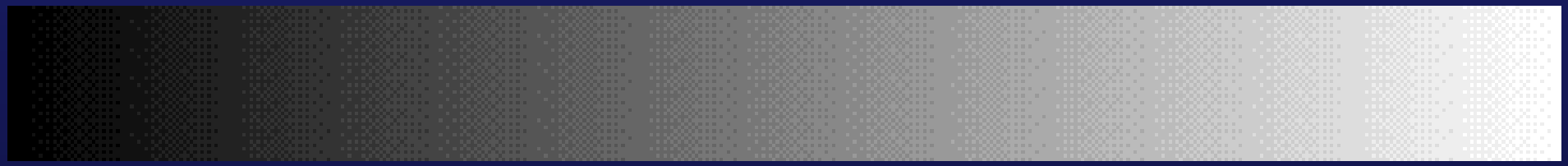
Las imágenes de los caracteres se almacenan en memoria y una **tabla** indica en qué zona de memoria está guardada la imagen de cada letra.

Cuando queremos mostrar el carácter de código ASCII `01100001`, se muestra en pantalla la imagen correspondiente.

Los sistemas con tipos de letra más avanzados (los que se usan en la actualidad) no guardan en memoria una imagen, sino **un programa** capaz de generar esa imagen a diferentes escalas partiendo de una serie de tipos de letra que almacenan imágenes a unos tamaños preestablecidos: es lo que se conoce como tipografías o **fuentes** de letra.

Una imagen en blanco y negro con tonalidades de gris necesita varios bits por píxel:

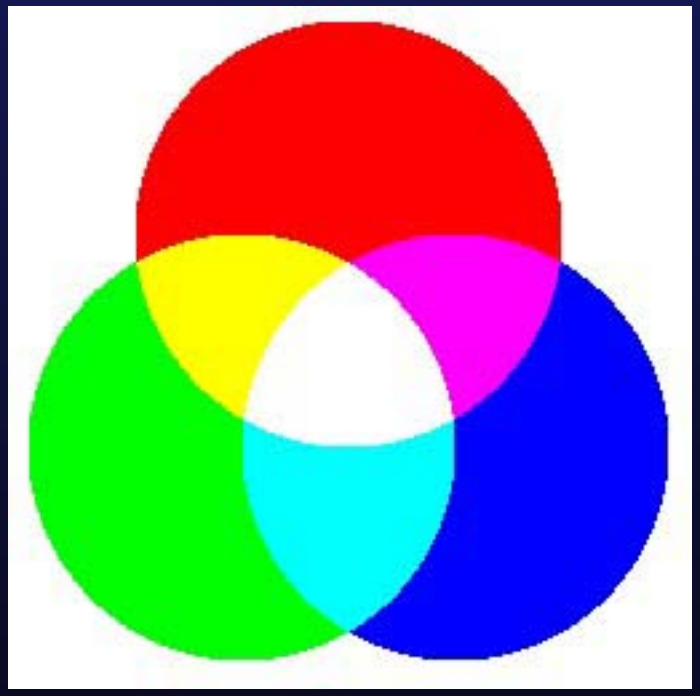
- 1 bits: 2 tonos (blanco y negro).
- 2 bits: 4 tonos.
- 3 bits: 8 tonos.
- . . .
- 8 bits (1 byte): 256 tonos.



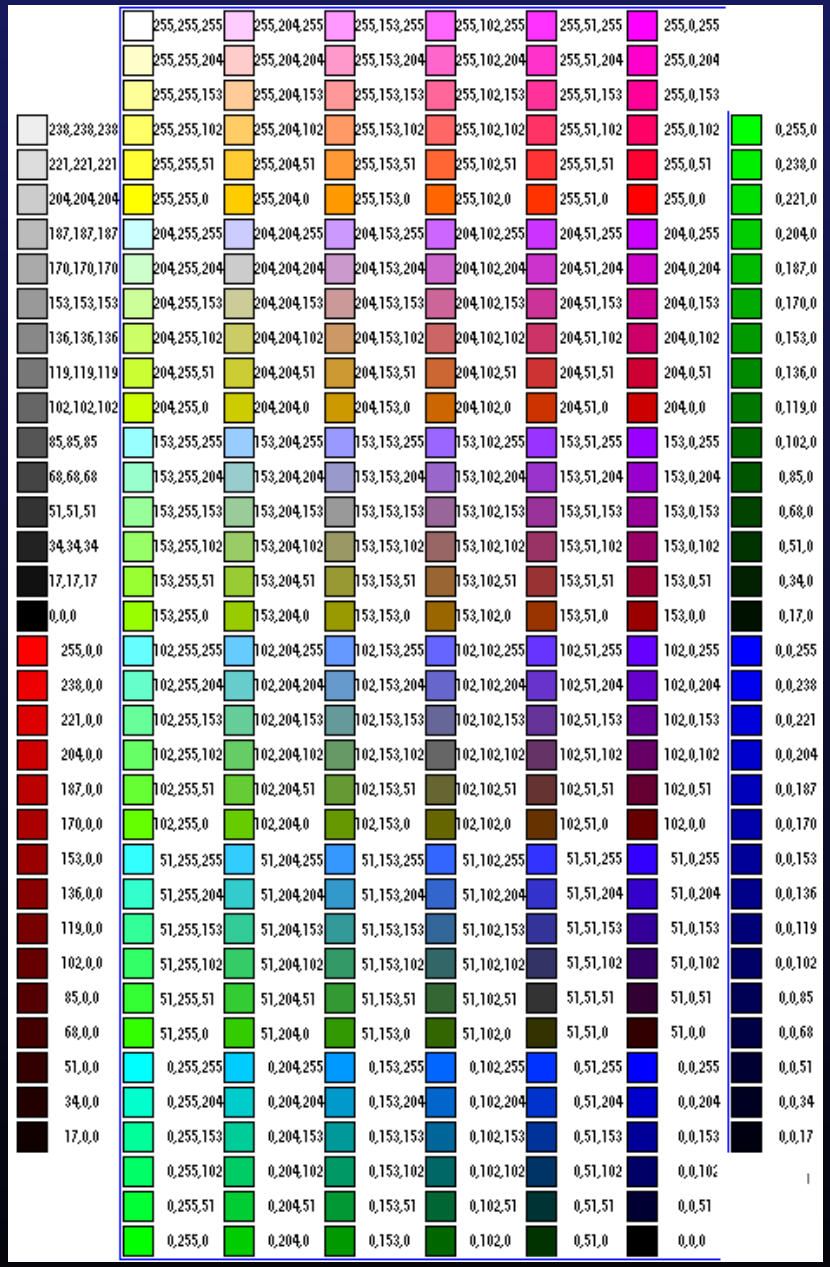
```
255 254 253 252 ... 3 2 1 0
255 254 253 252 ... 3 2 1 0
255 254 253 252 ... 3 2 1 0
...
```

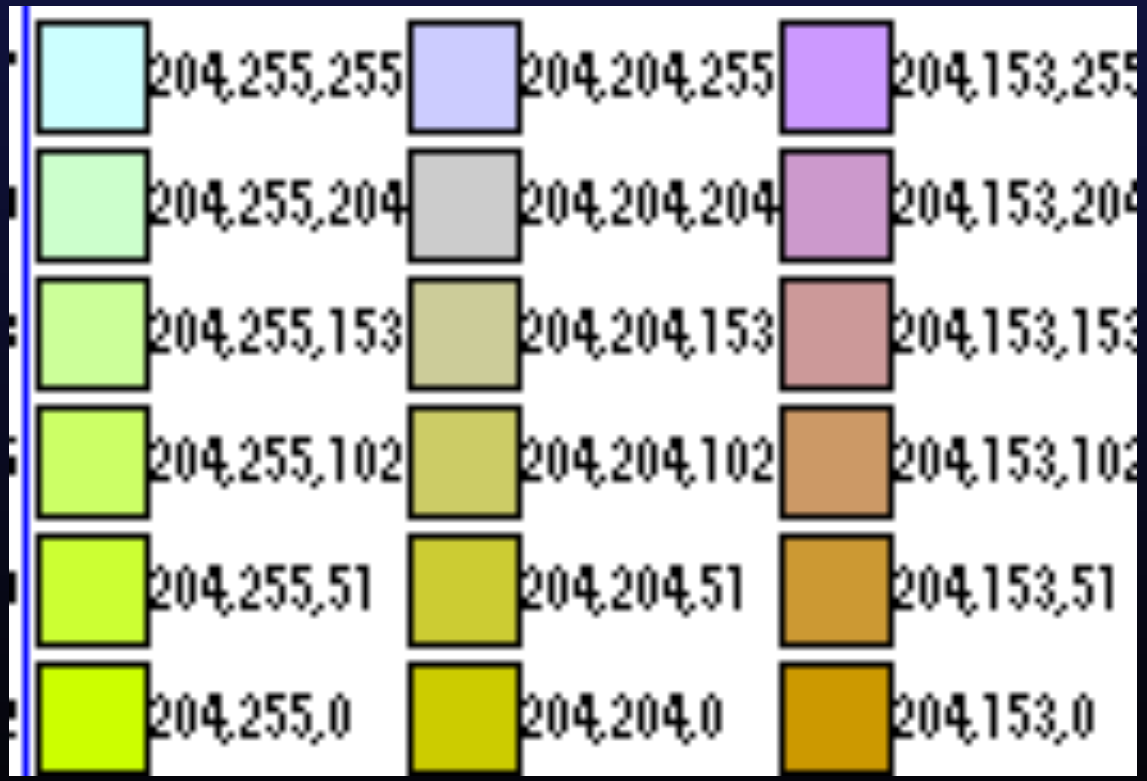
Ejercicio 12: ¿Cuántos bytes ocupa una imagen con 256 tonos (niveles) de gris de 800×600 píxels?

Las imágenes en color se forman combinando tonos de tres colores básicos: rojo, verde y azul (RGB, del inglés “red, green, blue”).



Cada color se describe con tres números que indican el tono de cada color básico. Cada número está entre 0 y 255. Describir cada color requiere 3 bytes (existen otros “modelos” de color, pero éste es el más extendido).




204,255,255
204,204,255
204,153,255
204,255,204
204,204,204
204,153,204
204,255,153
204,204,153
204,153,153
204,255,102
204,204,102
204,153,102
204,255,51
204,204,51
204,153,51
204,255,0
204,204,0
204,153,0

Ejercicio 13: ¿Cuántos colores pueden representarse con la codificación RGB de 3 bytes?

Ejercicio 14: ¿Cuántos bytes ocupa una imagen de 800×600 píxels en este caso?

Representar con 3 bytes cada píxel puede ser un desperdicio de memoria si una imagen usa pocos colores. Un truco consiste en usar una **paleta de colores**.

La paleta de colores registra, por ejemplo, 256 colores que podemos usar en la imagen:

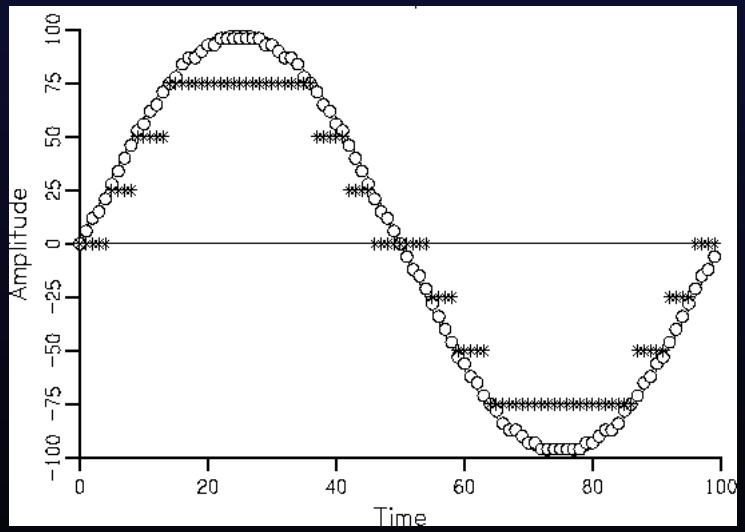
índice	R	G	B	color
0	255	242	12	
1	242	65	22	
2	200	4	120	
3	1	0	42	
⋮	⋮	⋮	⋮	⋮
255	42	0	255	

La imagen necesita ahora un byte por píxel: el que indica el índice.

Multimedia: sonido y vídeo

El sonido es una onda de presión. Podemos "capturar" sonido con un micrófono que registra la variación de una señal eléctrica a lo largo del tiempo, a través de una conexión analógica de la tarjeta de sonido (AUX, p.e. vinilo, casette, radio,. . .) o bien audio digital directamente (conversión de formatos, p.e. CDA-MP3).

La amplitud es una magnitud continua. Para almacenarla en el ordenador se discretiza o digitaliza, es decir, se convierte en un valor entero.



Hay dos factores que determinan la calidad del sonido digitalizado:

- la cantidad de bits usada para representar el sonido (la **cuantización** o número de niveles para interpolar valores),
- la cantidad de valores (muestras) medidos por segundo (la **frecuencia**), que se mide en hercios (Hz).

Una grabación con calidad de CD requiere:

- dos formas de onda (sonido estéreo) con 16 bits para la cuantización ($2^{16} = 65536$ niveles),
- tomando 44100 muestras por segundo (44 KHz).

Esto implica que una canción entre 3 ó 4 minutos ocupa entre 30 y 40 MB.

¿Qué entendemos por **vídeo**?

Una secuencia de **frames** (imágenes estáticas o fotogramas) en **movimiento** tal como ocurre en el mundo real.

La diferencia con una **animación** es el concepto de **movimiento real**: la secuencia de imágenes tiene que mostrarse lo suficientemente rápido como para constituir una representación convincente del movimiento.

Esto se consigue a partir de la **frecuencia de fusión**: más de 40 imágenes por segundo (en las películas se usan 24 frames/s., pero ten en cuenta que los proyectores muestran dos veces cada frame produciendo efectivamente una tasa de 48 frames/s.).

Hay dos estándares:

(A) NTSC (EE.UU. y Japón); tamaño frames: 640×480, 24 bits de color por píxel y $29{,}97$ frames/s. ($59{,}94$ en la práctica).

(B) PAL (Europa occidental y Australia); tamaño frames: 768×576, 24 bits de color por píxel y 25 frames/s. (50 en la práctica).

Estos son los estándares que se aplican en televisión, transmisión por cable y vídeo digital (DVD, p.e.)

Para capturarlo: entrada analógica para conexión de VHS, tarjeta sintonizadora de televisión o **mejor**: utilizar una cámara de vídeo digital.

Compresión

Ya hemos visto que es posible representar cualquier tipo de información con bits.

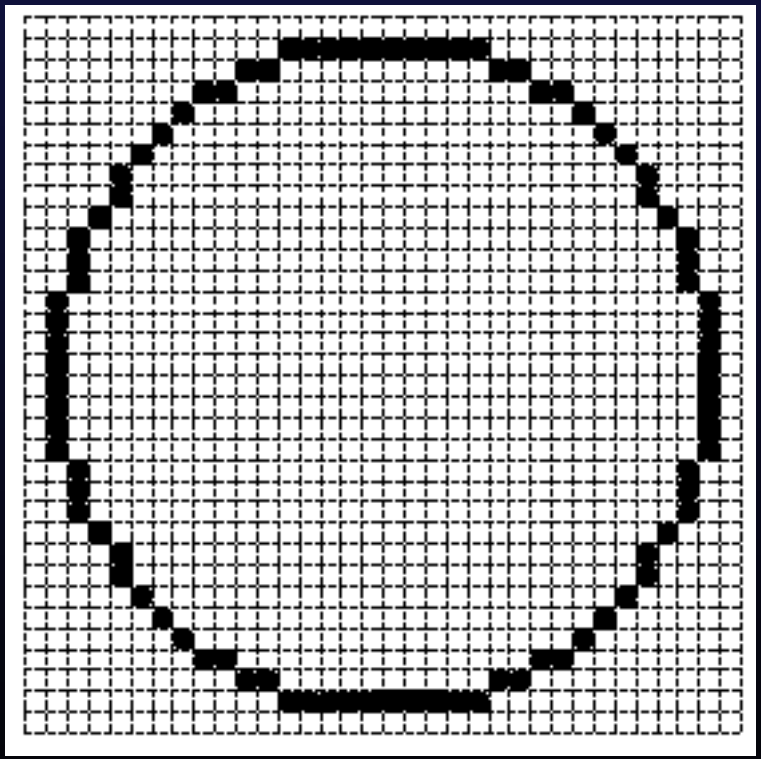
Sin embargo, la representación directa de la información **puede ocupar mucha memoria**.

Se han diseñado muchas técnicas de **compresión de la información** para reducir la ocupación de memoria. El objetivo es aprovechar la redundancia de la información.

Veamos un ejemplo: el *run-length encoding*.

Run-length encoding

Es una técnica que permite comprimir imágenes en blanco y negro. En lugar de almacenar los bits de la imagen, almacenamos, para cada fila, el número de bits de cada color:



```
34
12 10 12
10 2 10 2 10
9 1 14 1 9
...
```

Compresión con y sin pérdidas

La técnica del "run-length encoding" **permite "recuperar" la información original** tal cual: es una técnica de **compresión sin pérdidas**.

Las **técnicas de compresión con pérdidas** son **útiles** en la compresión de **imágenes, sonido y vídeo**: no permiten recuperar la información original, pero:

- La compresión JPEG permite almacenar imágenes ocupando mucho menos espacio gracias a que el ojo humano no afina mucho en la distinción de variaciones de color.
- La compresión MP3 comprime el sonido haciendo uso de información psicoacústica: no oímos sonidos de frecuencias próximas, ante un sonido fuerte los más débiles quedan ocultos, etc. . .

La compresión en vídeo es **fundamental**: 1 seg. de NTSC son 26MB, 1 min. son $1{,}6$GB, 1 seg. de PAL son 31MB, 1 min. son $1{,}85$GB. Para vídeo se utilizan los llamados **codecs** (acrónimo de **compresor/descompresor**). La mayoría se basan en el almacenamiento único de las **partes invariables** comunes a todos los frames en la secuencia, junto con **índices** que permiten **reconstruir** los (pequeñísimos) cambios de un frame a otro.

Se pierde información al comprimir, por lo que se puede apreciar una disminución en la calidad de la imagen. Pero si se usan los adecuados **niveles de muestreo** (frecuencia y cuantización –número de bits) la pérdida es **inapreciable** para el ser humano.

Niveles más altos implican **más calidad**, pero también **menos compresión**. Ejemplos de codecs: MPEG1, MPEG2, DivX, Xvid, ffmpeg4, etc.

Dejamos la cuestión abriendo un par de reflexiones:

1.- ¿Qué crees que pasaría si usásemos una técnica de compresión con pérdidas al texto de El Quijote?

Que ocuparía bastante menos espacio de almacenamiento, pero **no podríamos recuperar la novela original** a partir de la copia comprimida.

¿Obtendríamos una versión resumida del libro?

No. Las técnicas de compresión **no analizan la semántica** para comprimir. Eso sería propio de la Inteligencia Artificial (y no es un problema resuelto). Simplemente buscan **patrones repetidos** y **eliminan información redundante** de tipo "estructural".

2.- ¿Qué pasaría si usásemos repetidamente una técnica de compresión sin pérdidas al texto de El Quijote?

Llegaría un momento que no podríamos comprimir más: **existe un límite para la capacidad de compresión**.

¿Sería posible llegar a almacenarlo en un solo bit?

Como hemos dicho, hay un límite. El mínimo número de bits necesario para codificar una información **sin pérdidas** guarda relación con el concepto de **entropía**: cuanto más orden, más capacidad de predicción y menos bits; a más desorden, menos predecible será la secuencia y necesitaremos más bits para codificar.

Formatos de imagen

Los factores que afectan al espacio en disco que ocupará el archivo que contiene una imagen son los dos vistos con anterioridad: **número de píxels** (tamaño imagen) y **modelo de color** (blanco y negro, grises, color RGB). La mayoría de los formatos de imagen aplican **compresión** (con o sin pérdidas):

- BMP (bitmap): almacena la imagen tal cual, **sin comprimir**. Es, básicamente, el formato descrito en transparencias anteriores.

- GIF (Graphics Interchange Format):
  - usa un esquema de compresión **sin pérdidas** LZW (Lempel-Ziv-Welch),
  - **apropiado** para imágenes con bordes muy marcados y grandes áreas de colores planos (p.e. **imágenes con textos y dibujos basados en líneas de forma predominante**),
  - modelo de color: **paleta**; sólo 8 bits por píxel, es decir, puede indexar **256 colores** distintos, por lo que debe tenerse en cuenta la **pérdida de calidad** si la imagen original representaba más de 256 colores,
  - permite **visualización progresiva** (entrelazado), adecuado para imágenes en web,
  - permite **animaciones**, ya que se puede almacenar más de una imagen en un único fichero.

- JPEG (Joint Photographic Experts Group):
  - **apropiado** para imágenes con transiciones de color sutiles y suavizadas (**fotografías** e imágenes en escala de grises o con **millones de colores**).
  - realiza una **compresión con pérdidas**; se debe buscar un **equilibrio** entre la ocupación y la calidad de la imagen,
  - el **método de compresión trocea** la imagen en pequeños bloques y **promedia** el color en cada bloque; se basa en que el ojo humano no afina mucho en la distinción de variaciones de color (esto hace que el texto y los dibujos basados en líneas no sean apropiados para JPEG),
  - permite hasta **24 bits** de color por píxel,
  - el formato JPEG progresivo permite **entrelazado** (el estándar, no).

- PNG (Portable Network Graphics):
  - **apropiado** para **cualquier** tipo de imagen,
  - utiliza el **método de compresión** "Deflate", similar al LZW; pero aumentando el grado de compresión,
  - permite hasta **48 bits** de color por píxel,
  - realiza una **compresión sin pérdidas** (al contrario que JPEG), lo que junto al **ilimitado número de colores** (al contrario que GIF), proporciona **gran calidad** con **cualquier** tipo de imagen,
  - permite entrelazado.
  - todo ello le convierte en un formato muy adecuado para representar imágenes en páginas web, aunque **comprime menos que JPEG**.

Formatos de sonido

Los factores que afectan al espacio en disco que ocupará el archivo que contiene sonido (y a la calidad del mismo) son los dos vistos con anterioridad: **frecuencia de muestreo** y **niveles de cuantización** (número de bits). La mayoría de los formatos de sonido aplican **compresión** (con pérdidas):

- CDA (Compact Disc Audio): es el formato empleado en los CD-audio (música). Gran calidad: 44KHZ, 16 bits, estéreo. Inconveniente: ocupa mucho espacio.
- WAV: desarrollado por Microsoft para la reproducción y edición de audio en PC; misma calidad que CDA; mismo inconveniente (espacio ocupado).

- MP3 (Capa III de MPEG-2, Moving Picture Experts Group): **compresión con pérdidas**. Se basa en que no percibimos sonidos de frecuencias próximas y que ante un sonido fuerte los más débiles quedan ocultos (características psico-acústicas) para descartar información que no es valiosa para el ser humano.

  Frecuencia estándar: 44KHZ (mejor no tocar) y con la **tasa de bits** o **bitrate** controlamos la relación entre calidad del sonido y espacio que ocupa el fichero comprimido.

  Los expertos dicen que a un bitrate de **160 kbs** el ser humano no distingue entre original y MP3. Se necesita un reproductor con soporte MP3 para poder escuchar estos ficheros. Hoy en día prácticamente cualquier aparato con capacidad de reproducir audio lo soporta.
  Está patentado (se paga licencia). Se usa como estándar "de facto" para el intercambio de ficheros en Internet.

- OGG Vorbis: formato de **compresión con pérdidas** de audio, **más evolucionado** que MP3 y **libre** (público). Sus especificaciones son de **uso gratuito**, incluso si se reproduce con software de pago.

  Usa principios matemáticos distintos a MP3, pero la idea básica es la misma: usar modelos humanos de audición para eliminar la información que el oído humano es incapaz de captar.

  Como en MP3 el **bitrate** controla el compromiso calidad audio/espacio ocupado. Obtiene una calidad de sonido similar a MP3 (en algunos casos mejor) y ocupa poco más o menos el mismo espacio. También se usa para intercambio de ficheros en Internet.

Reproducción:

- Windows Media Player (Windows) y XMMS (multiplataforma) reproducen todos los mencionados: CD-Audio, WAV, MP3, OGG y otros. . .
- Existen reproductores de bolsillo que soportan MP3 (y muchos también OGG). Por ejemplo, el **IPod** de Apple.
- La gran mayoría de DVDs electrodomésticos soportan MP3 (y algunos empiezan a incluir OGG).

Conversión:

- CDA-WAV: Programas de "ripeo", Easy CD Ripper (Windows), Grip (Linux). . .
- WAV-MP3: la mayoría de programas de "ripeo" incluyen una herramienta para efectuar esta conversión. También hay herramientas específicas como LAME (Linux).
- WAV-OGG: la mayoría de programas de "ripeo" incluyen una herramienta para efectuar esta conversión, p.e. Grip.
- WAV-CDA: se necesita un programa de grabación de CD como Nero Burning (Windows) o K3b (Linux). Seleccionar los ficheros y escoger "grabar CD de audio".
- MP3,OGG-CDA: los buenos programas de grabación de CD hacen la conversión de formato automática a CD-Audio, p.e. Nero (Windows) y K3b (Linux).

Para editar audio: (ver tema 1), Adobe Premiere. SW. libre: Jahshaka (multiplataforma).

Formatos de vídeo

Para almacenar un vídeo o una película en un fichero se ha de utilizar un **contenedor**. Un **contenedor** es como un "sobre" (envoltorio) que comprende las secuencias de audio, vídeo y (posiblemente) subtítulos que conforman un vídeo en un **único fichero**. ¿Por qué es necesario?

- para sincronizar perfectamente audio, vídeo (y subt.),
- para poder realizar rebobinado y marcha adelante/atrás,
- para hacer búsquedas rápidas en el fichero,
- para reproducir a partir de un minuto dado,
- para seleccionar (posibles) pistas alternativas de audio,
- para seleccionar subtítulos alternativos,
- para recuperar errores (frames "en mal estado"),
- . . .

Los **contenedores** más usados hoy en día:

- AVI (Audio Video Interleaved). Primer formato de vídeo de Microsoft. Muy extendido. La información de índice (cabecera) se almacena al final del fichero. Tiene un soporte limitado para codecs (compresor/descompresor). Ha sufrido tantas implementaciones no propias de Microsoft que hoy en día hay bastantes AVIs no conformes al estándar.
- MOV. El de Apple para su famoso QuickTime. Soporta múltiples secuencias de audio y vídeo, índice (cabecera) al principio del fichero, pistas de texto, anotaciones, soporte para cientos de codecs, etc.
- MPEG (Motion Picture Experts Group). Contenedor para vídeo MPEG-1 (VCD) y MPEG-2 (DVD). Normalmente se mezcla con una pista de audio.
- OGG. Similar al AVI, soporta los mismos codecs y, además, OGG Vorbis (formato de compresión de audio con características similares a MP3 y libre de patentes).

Algunos codecs (compatibles con AVI):

- DivX, basado en el estándar MPEG-4, es capaz de comprimir una película en DVD hasta 1/10 parte de su tamaño sin apenas merma de calidad. Calidad DVD (90 min. aprox. en 1 CD, con sonido Dolby 75 min. 1 CD). Es el formato "de facto" para el intercambio de películas en Internet. Es al vídeo digital (DVD) lo que el MP3 a la música.
- Xvid (DivX al revés ;-) es la implementación **sw. libre** de MPEG-4. Prestaciones iguales o superiores a DivX. Cada vez se usa más en Internet (para películas).
- ffmpeg4. . .

MPEG, como contenedor, también usa codecs pero deben ser los propios de MPEG (MPEG1, MPEG2).

Conversión DVD-DivX,Xvid,ffmpeg4,VCD,SVCD,. . . Hay programas como DVD Rip N' Burn (Windows, www.dvdripnburn.com) y dvd::rip (Linux, www.exit1.org/dvdrip/).

Ya hay bastantes reproductores DVD (electrodomésticos) que soportan DivX/Xvid y son actualizables por Internet ;-)

Formatos de documento

- Documentos Word (**doc**): se necesita Microsoft Word para su perfecta visualización/impresión (requiere licencia). Writer de OpenOffice.org es capaz de editar este tipo de ficheros, pero **no se garantiza compatibilidad 100 %** (aunque la mayoría de veces va bien).
- Documentos Writer (**odt**): se necesita Writer de OpenOffice.org para visualización/impresión. OpenOffice.org es **sw. libre**, **gratuito** y **multiplataforma**.
- Portable Document Format (**pdf**): es un formato **públicamente conocido** propiedad de Adobe. Existen programas gratuitos capaces de visualizar/imprimir correctamente en cualquier plataforma.
- HTML: el "lenguaje" en el que están hechas las páginas web también se puede emplear para formatear documentos. Más limitado, pero es estándar y cualquier navegador permite visualizar/imprimir.

Formatos abiertos y cerrados

Los ficheros en un formato determinado se atienen a unas reglas precisas que indican **qué** se puede almacenar en ellos y **cómo**.

- Los **formatos abiertos** siguen normativas públicas y que puede seguir cualquiera. Usar formatos abiertos hace posible que se comparta información entre diferentes usuarios sin coste alguno.

- Los **formatos cerrados** siguen normativas que fijan empresas y mantienen en secreto. Mantener un formato cerrado puede suponer una ventaja competitiva, sobre todo cuando el formato se convierte en un estándar de facto en el intercambio de información: los usuarios se ven obligados a adquirir productos comerciales para poder leer o modificar los ficheros.

Utilizar un **formato cerrado** para el **intercambio de información** puede resultar **contraproducente**, ya que **estás obligando** al receptor a:

(a) adquirir una **licencia** del programa, con el consiguiente dispendio u

(b) obtener una **copia ilegal** del programa, lo cual puede conllevar su **ingreso en prisión** con el nuevo código penal en la mano (1 de octubre de 2004 en vigor).