SSII UT.01
Explotación de Sistemas Informáticos

Representación de la Información

Licencia

¿Qué vamos a ver?

Introducción.

Sistemas de numeración usuales en Informática.

Representación de textos.

Representación de sonidos.

Representación de imágenes.

Representación de datos numéricos.

Detección de errores.

Compresión de datos.

Introducción

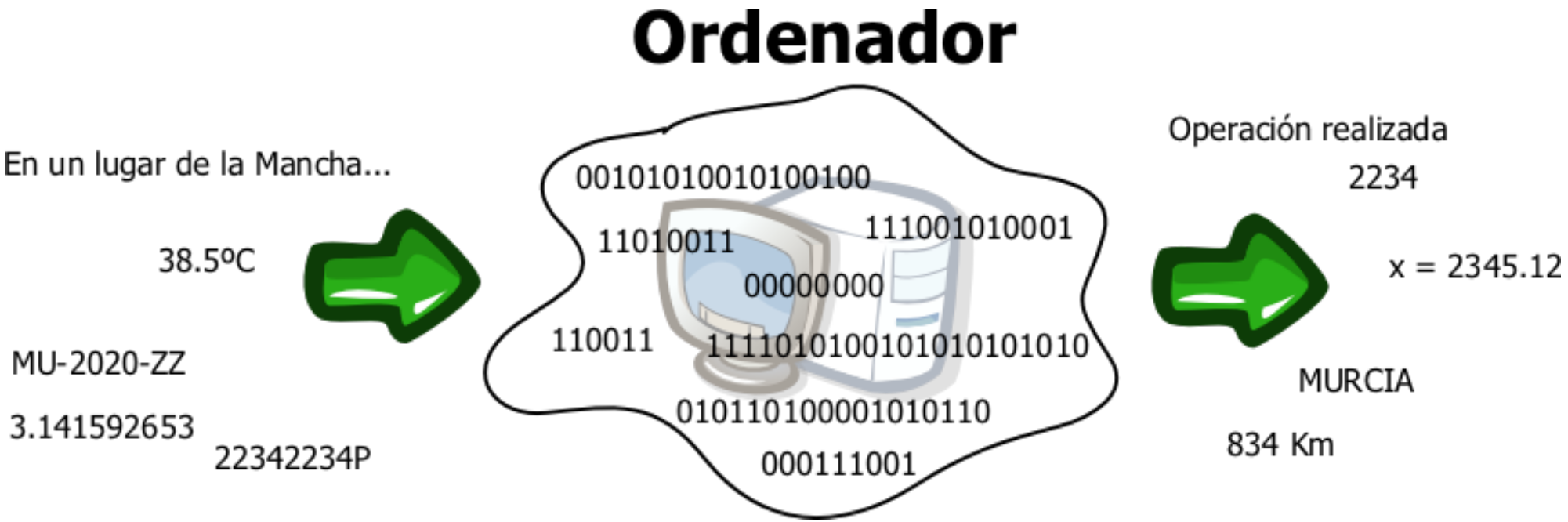
Un ordenador es, básicamente, una máquina que procesa una información de entrada y devuelve una información de salida.

Toda la información se maneja internamente utilizando un sistema binario.

La información utilizada la forman, básicamente:

- Números.
- Texto.
- Imágenes.
- Sonido.

Procesamiento de la información



Representaciones

Veremos cuatro tipos de representaciones:

- Representaciones numéricas.
- Representación de textos.
- Representación de imágenes.
- Representación de sonidos.
- Representación de instrucciones.

Y dos temas relacionados:

- Detección de errores.
- Compresión de datos.

Pero todo es binario...

Para un ordenador TODO son números.

Pero estos números se representan en un sistema de numeración en base dos.

A este sistema lo llamaremos binario natural o binario.

Pero...

¿Por qué nosotros usamos un sistema en base diez?

¿Por qué los ordenadores utilizan un sistema en base dos?

Representación posicional de los números

Un sistema de numeración en base b utiliza un alfabeto A compuesto por b símbolos o cifras.

Todo número se expresa por un conjunto de cifras, contribuyendo cada una de ellas a un valor que depende de:

- La cifra en sí.
- La posición que ocupa dentro del número.

En un sistema base 10 (o decimal) el alfabeto lo componen diez símbolos o cifras:

$$\mathbf{S_{10} = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}}$$

Ejemplos de sistemas de numeración

Sistema de numeración decimal (base 10):

b = 10, A = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Sistema de numeración binario (base 2):

b = 2, A = { 0, 1}

Sistema de numeración octal (base 8):

b = 8, A = { 0, 1, 2, 3, 4, 5, 6, 7}

Sistema de numeración hexadecimal (base 16):

b = 16, A = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

Representación posicional

El número 23.456'34 se puede obtener así:

$$
\begin{aligned}
2 \times 10^{4} &= 20000 \\
3 \times 10^{3} &= 3000 \\
4 \times 10^{2} &= 400 \\
5 \times 10^{1} &= 50 \\
6 \times 10^{0} &= 6 \\
3 \times 10^{-1} &= 0.3 \\
4 \times 10^{-2} &= 0.04 \\
\hline
&= 23456.34
\end{aligned}
$$

Generalizando

Si tenemos cualquier número en cierta base b:

$$N \equiv \ldots n_4\ n_3\ n_2\ n_1\ n_0\ n_{-1}\ n_{-2}\ n_{-3}\ n_{-4} \ldots$$

... podemos expresar su valor así:

$$\mathbf{N \equiv \ldots n_3 \cdot b^3 + n_2 \cdot b^2 + n_1 \cdot b^1 + n_0 \cdot b^0 + n_{-1} \cdot b^{-1} + n_{-2} \cdot b^{-2} + n_{-3} \cdot b^{-3} \ldots}$$

Podemos utilizar esto para convertir números entre bases:

$$\mathbf{74,32)_8} = 7 \cdot 8^1 + 4 \cdot 8^0 + 3 \cdot 8^{-1} + 2 \cdot 8^{-2} =$$

$$56 + 4 + 0'375 + 0'03125 = \mathbf{60'40625)_{10}}$$

Sistema de numeración en base dos

En el sistema de numeración binario b = 2 y se necesitan tan sólo dos elementos para representar cualquier número:

$$S = \{ 0, 1 \}$$

Los elementos de S se denominan cifras binarias o bits.

Conversión: binario -> decimal

Para transformar números a decimal basta con utilizar la expresión anterior:

$$N \equiv \ldots n_3 \cdot b^3 + n_2 \cdot b^2 + n_1 \cdot b^1 + n_0 \cdot b^0 + n_{-1} \cdot b^{-1} + n_{-2} \cdot b^{-2} + n_{-3} \cdot b^{-3} \ldots$$

Ejemplos:

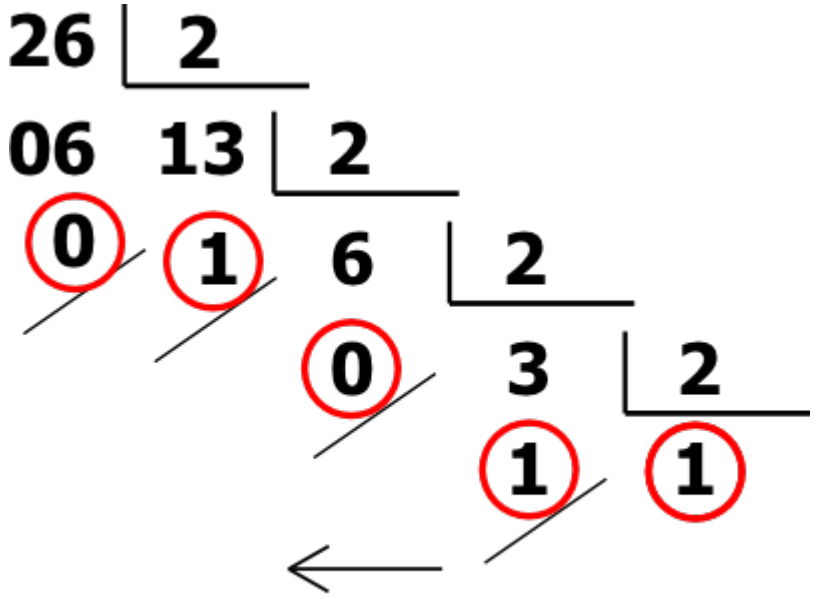
$10101)_2 =$

$100)_2 =$

$100001)_2 =$

Conversión: decimal -> binario (I)

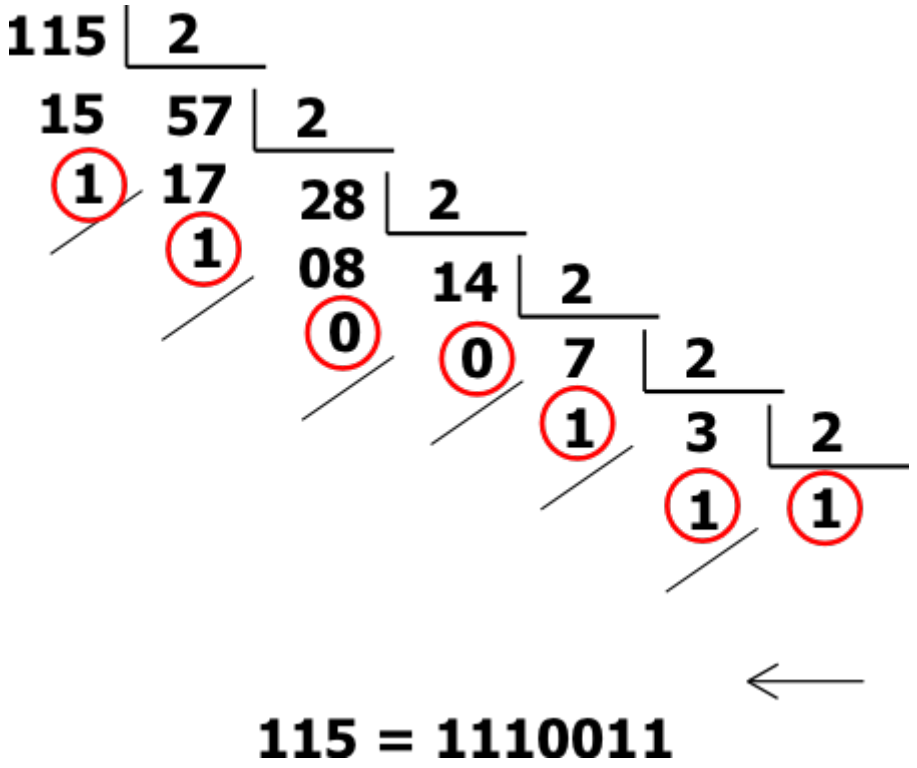
Pasar un número de decimal a binario es un poquillo más complicado. Ejemplo:



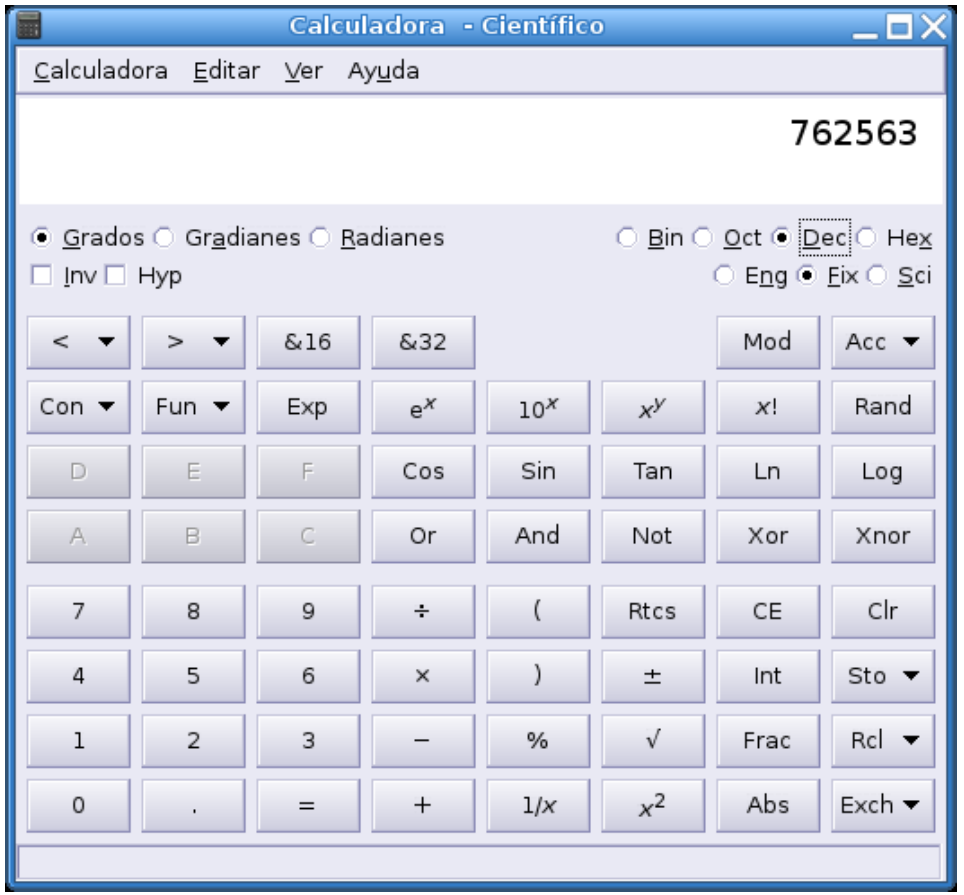
26 = 1 1 0 1 0

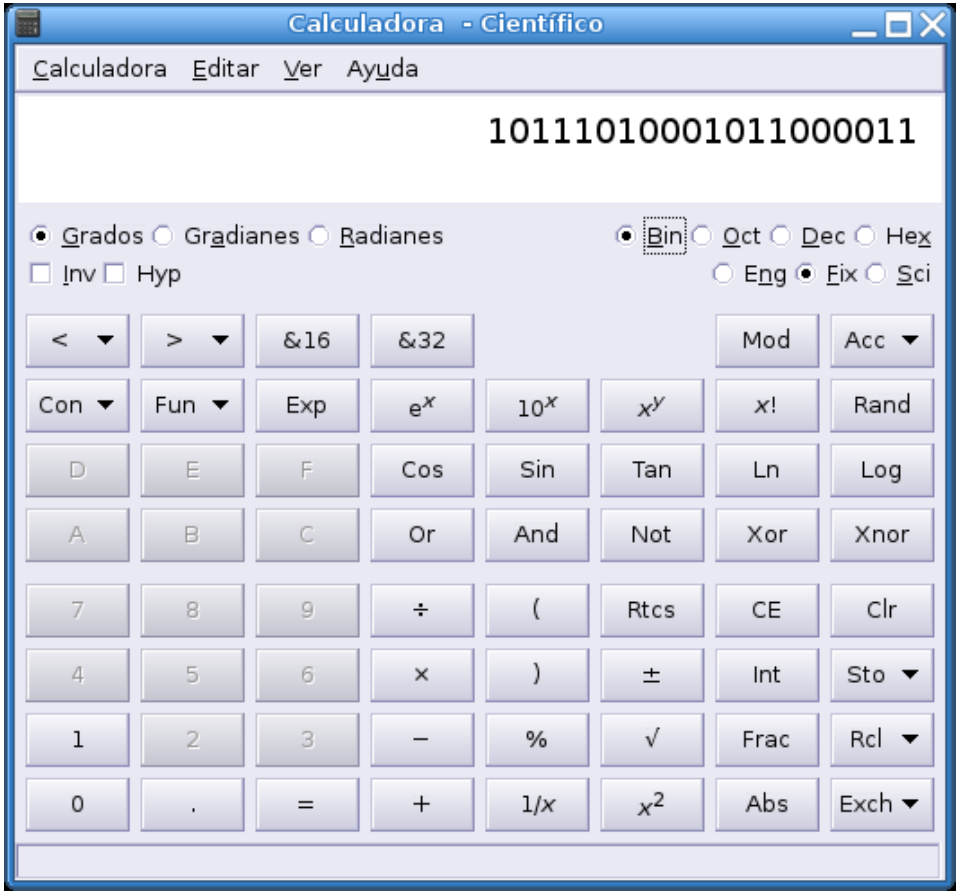
Conv.: decimal -> binario (y III)

¿A qué número corresponde el 115?



Siempre nos quedará la calculadora...





Operaciones aritméticas en binario

Suma

A	B	**A + B**
0	0	0
0	1	1
1	0	1
1	1	0 y me llevo 1

Ejemplos (I)

1011010	90
+ 100100	+ 36
1111110	126

1110101	117
+ 1110110	+ 118
11101011	235

Como ejercicio ...

10011001 + 11011011

11100110 + 11011101

Códigos intermedios

Los códigos intermedios se fundamentan en la facilidad que existe para transformar un número en base 2 a otra base que sea potencia de 2, y viceversa.

- Base 8 (2^3, sistema octal)
- Base 16 (2^4, sistema hexadecimal).

En Informática es muy usual utilizar los siguientes sistemas de numeración:

- Sistema de numeración base 8 (**octal**).
- Sistema de numeración base 16 (**hexadecimal**).

Decimal/binario/octal/hexadecimal

Lo único que hay que saber:

Hexadecimal	Decimal	Binario
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Octal	Decimal	Binario
0	0	000
1	1	001
2	2	010
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111

Octal

Decimal	Octal	Decimal	Octal	Decimal	Octal	Decimal	Octal
0	0	10	12	20	24	30	36
1	1	11	13	21	25	31	37
2	2	12	14	22	26	32	40
3	3	13	15	23	27	33	41
4	4	14	16	24	30	34	42
5	5	15	17	25	31	35	43
6	6	16	20	26	32	36	44
7	7	17	21	27	33	37	45
8	10	18	22	28	34	38	46
9	11	19	23	29	35	39	47

Octal: conversiones

Para transformar un número binario a octal se forman **grupos de tres cifras binarias** a partir del punto decimal hacia la izquierda y hacia la derecha.

Se rellenan con ceros si hace falta para tener grupos de tres.

Posteriormente se efectúa directamente la conversión a octal de cada grupo individual.

Proceso análogo para pasar de octal a binario.

Ejemplos

10011010101

En binario -> 010 011 010 101

En octal -> 2 3 2 5

2325

537

En octal -> 5 3 7

En binario -> 101 011 111

101011111

Hexadecimal: conversiones

Para transformar un número binario a hexadecimal se forman **grupos de cuatro cifras binarias** a partir del punto decimal hacia la izquierda y hacia la derecha.

Se rellenan con ceros si hace falta para tener grupos de cuatro.

Posteriormente se efectúa directamente la conversión a hexadecimal de cada grupo individual.

Proceso análogo para pasar de hexadecimal a binario.

Ejemplos

10110011110

En binario -> 0101 1001 1110

En hexadecimal -> 5 9 E

59E

FA345

En hexadecimal -> F A 3 4 5

En binario -> 1111 1010 0011 0100 0101

11111010001101000101

Representación de textos (I)

Podemos representar cualquier información escrita (texto) a través de caracteres.

Los caracteres que más se utilizan en Informática se podrían clasificar en cinco categorías:

- Alfabéticos.
- Numéricos.
- Especiales.
- Geométricos y gráficos.
- De control.

Representación de textos (II)

Alfabéticos.

A, B, C, ... X, Y, Z, a, b, c, ..., x, y, z

Numéricos.

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Especiales.

. : , ; _ ! " · $ % & ^ { } [ ] + - * / Ç **SP**

Geométricos y gráficos.

┌ └ ├ ¬ £ © ☎ ☯ ☠ ♥ ♠ ♣ ♦

De control.

Ordenes de control: retorno de carro, nueva línea, pitido, tabulador, ...

Representación de textos (III)

Al introducir un texto en un ordenador desde cualquier periférico de E/S, los caracteres se codifican según un código de E/S.

A cada carácter se le asocia una determinada combinación de n bits.

Un código de E/S (tabla de códigos) no es más que una correspondencia entre los siguientes conjuntos:

$$\{ A, B, \ldots 0, 1, 2, \ldots +, -, ., ; \ldots \} \rightarrow \{ 0, 1 \}^n$$

¿Qué tamaño debe tener n?

Representación de textos (y IV)

Con 2 bits (n = 2) tenemos 2^2 combinaciones.

Podemos representar 4 caracteres.

Con 3 bits (n = 3) tenemos 2^3 combinaciones.

Podemos representar 8 caracteres.

Con 7 bits (n = 7) tenemos 2^7 combinaciones.

Podemos representar 128 caracteres.

Si queremos representar m caracteres necesitamos n bits de tal forma que:

$$2^{n-1} < m \leq 2^n \quad \rightarrow \quad \mathbf{n \geq \log_2 m}$$

Ejemplo

Si quisiéramos representar los diez números podríamos hacerlo según estos dos códigos:

Alfabeto	Código1	Código2
0	0000	0000
1	0001	0100
2	0010	1000
3	0011	1100
4	0100	0001
5	0101	0101
6	0110	1001
7	0111	1101
8	1000	0010
9	1001	0110

ASCII

(American Standard Code for Information Interchange)

El código ASCII es actualmente uno de los más usados para representar texto.

Tambien llamado **ANSI**-X3.4 ó **ISO** 646.

¡¡ Es de 7 bits !!

Da a cada carácter una combinación de 7 bits.

¿Es suficiente, al menos en castellano?

Código ASCII

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00	0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	16	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	32	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	
60	96	`	a	b	c	d	e	F	g	h	i	j	k	L	m	n	o
70	112	p	q	r	s	t	u	V	w	x	y	z	{	|	}	~	DEL

Códigos ISO-8859-n

Existen varias ampliaciones de ASCII que utilizan 8 bits y respetan el cósigo ASCII básico.

Se duplica el número de caracteres ($2^8 = 256$).

Entre ellas destacan los códigos ISO-8859-n.

- n representa el número que identifica el juego de caracteres añadido.
- Cada valor de n se ajusta a un área geográfica.

Las usadas en España son:

- ISO-8859-1 (Latín-1 ó Western).
- ISO-8859-15 (Latín 15).

Códigos ISO-8859-1

La tabla de códigos ISO-8859-1 deja varios caracteres sin definir.

Dichos caracteres se pueden definir en otra tabla no normalizada pero compatible con ISO-8859-1.

Ese es el ejemplo de la tabla de códigos 850 o de la tabla 1252 utilizada mucho en PC's (con Windows).

ISO-8859-1

Tabla 3.12 Conjunto de caracteres ISO 8859-1 (Latín 1).

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00	0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	16	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	32	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	96	`	a	b	c	d	e	f	g	h	i	j	k	L	m	n	o
70	112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
80	128																
90	144																
A0	160		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B0	176	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C0	192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D0	208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E0	224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	Ì	í	î	ï
F0	240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	Ü	ý	þ	ÿ

Unicode (I)

Tanto ASCII como ISO-8859 presentan ciertos inconvenientes:

- Símbolos insuficientes para representar ciertos caracteres especiales que requieren ciertas aplicaciones.
- Los caracteres de 8 bits no están normalizados.
- Basados en un subconjunto muy limitado de caracteres latinos (inglés). Otras culturas utilizan símbolos distintos para letras y cifras numéricas.
- Los lenguajes de culturas orientales (China, Japón ...) se basan en la utilización de ideogramas o símbolos que representan palabras, frases o ideas completas. Estos códigos no son apropiados.

Unicode (II)

http://unicode.org

Es un estándar industrial cuyo objetivo es proporcionar el medio por el cual un texto en cualquier forma e idioma puede ser codificado para su uso informático.

- Forman parte del consorcio Unicode empresas como: Adobe, Apple, HP, IBM, Microsoft, Xerox, etc.

Proyecto muy ambicioso.

- Origen en 1991: Unicode 1.0. Actualmente versión 10

Hace posible escribir aplicaciones que sean capaces de procesar texto de muy diversos sistemas de escritura.

Propiedades de Unicode

Universalidad.

Se persigue cubrir la mayoría de lenguajes escritos existentes en la actualidad.

Unicidad.

A cada carácter se le asigna exactamente un único código.

Uniformidad.

Todos los símbolos se representan con un número fijo de bits, concretamente 16.

Estandarización.

Es un estándar ISO/IEC 10646.

Escrituras soportadas

La mayor parte de escrituras usadas actualmente:

Árabe, armenio, bengalí, braille, cheroqui, copto, cirílico, devanagari, esperanto, ge'ez, georgiano, griego, guyaratí, gurmukhi, hangul, han, japonés, hebreo, jémer, kannadam lao, latino, malayalam, mongol, burmese, oriya, syriac, tailandés, tamil, tibetano, yi, zhuyin.

También se han ido añadiendo escrituras históricas menos utilizadas:

Cuneiforme, griego antiguo, linear b, fenicio, rúnico, sumerio, ugarítico.

En la versión 10.0 de 2017 se agrega el símbolo de Bitcoin y 56 caracteres emoji

Zonas de Unicode

- Las 65536 (código de 16 bits, 2^{16}) símbolos posibles combinaciones se han dividido en cuatro zonas (0x0000 – 0xFFFF):
  - Zona A (0x0000 – 0x3FFF). Aquí se incluyen:
    - ISO-8859-1 (0x0000 – 0x00FF).
    - Caracteres latinos, griegos, cirílicos, armenios, hebreos ...
    - Símbolos generales (diacríticos, subíndices, superíndices, monedas, matemáticos, formas geométricas ...).
  - Zona I (0x4000 – 0x9FFF).
    - Más de 20000 caracteres Han (ideogramas chinos, japoneses y coreanos).
  - Zona O (0xA000 – 0xDFFF). Pendiente de reservar.
  - Zona R (0xE000 – 0xFFFF). Reservado para usuarios.

Soporte de Unicode

Todos los sistemas operativos modernos.

Plataformas de programación como Java y .NET.

Clientes web (navegadores).

MIME (mecanismo de codificación del correo electrónico).

XML.

Tecnologías web (XML, HTML, CSS ...).

Almacenamiento, transferencia y procesamiento

Por el momento, Unicode solo es una forma de asignar un código único a cada carácter utilizado en los lenguajes escritos del mundo.

El almacenamiento de esos números en el procesamiento de texto es otro tema. Problemas:

- Gran cantidad de software que solo puede manejar codificaciones de 8 bits.
- Para ciertas escrituras incluso 65536 caracteres son insuficientes.

Unicode puede ser implementado de diferentes formas.

Implementaciones de Unicode

Unicode define dos métodos de mapeo de caracteres:

- **Codificación UTF** (Unicode Transformation Format).
  - Aquí incluimos la popular y muy utilizada UTF-8.
  - Además tenemos el UTF-16 y el UTF-32
- **Codificación UCS** (Universal Character Set).
  - Aquí incluimos UCS-2 que es una codificación fija de 16 bits que solamente permite el mapeo de la zona A.

Codificación UTF-8

Se utiliza entre 1 y 4 bytes por carácter.

Relativamente compacto para escritura con caracteres latinos.

Compatible con ASCII.

Codificación estándar para el intercambio de texto en Unicode.

Soportado por distintos SO (Windows, Linux ...).

Representación de sonidos

Muchas son las aplicaciones que necesitan procesar tanto texto como sonidos e imágenes:

- Aplicaciones multimedia.
- Juegos.
- Etc.

¿Qué es el sonido?

Sonido

Es una sensación que se origina en el órgano del oído, producida por el movimiento ondulatorio en un medio elástico (normalmente el aire), debido a rapidísimos cambios de presión generados por el movimiento vibratorio de un cuerpo sonoro.

El oído humano es capaz de oir:

- Sonidos con una intensidad entre:
  - 0 dB (umbral de audición) y
  - ... 140 dB (umbral de dolor).
- Sonidos con frecuencias entre 20 y 20000 Hz.

Muestreo (I)

Una señal de sonido se capta por medio de un micrófono.

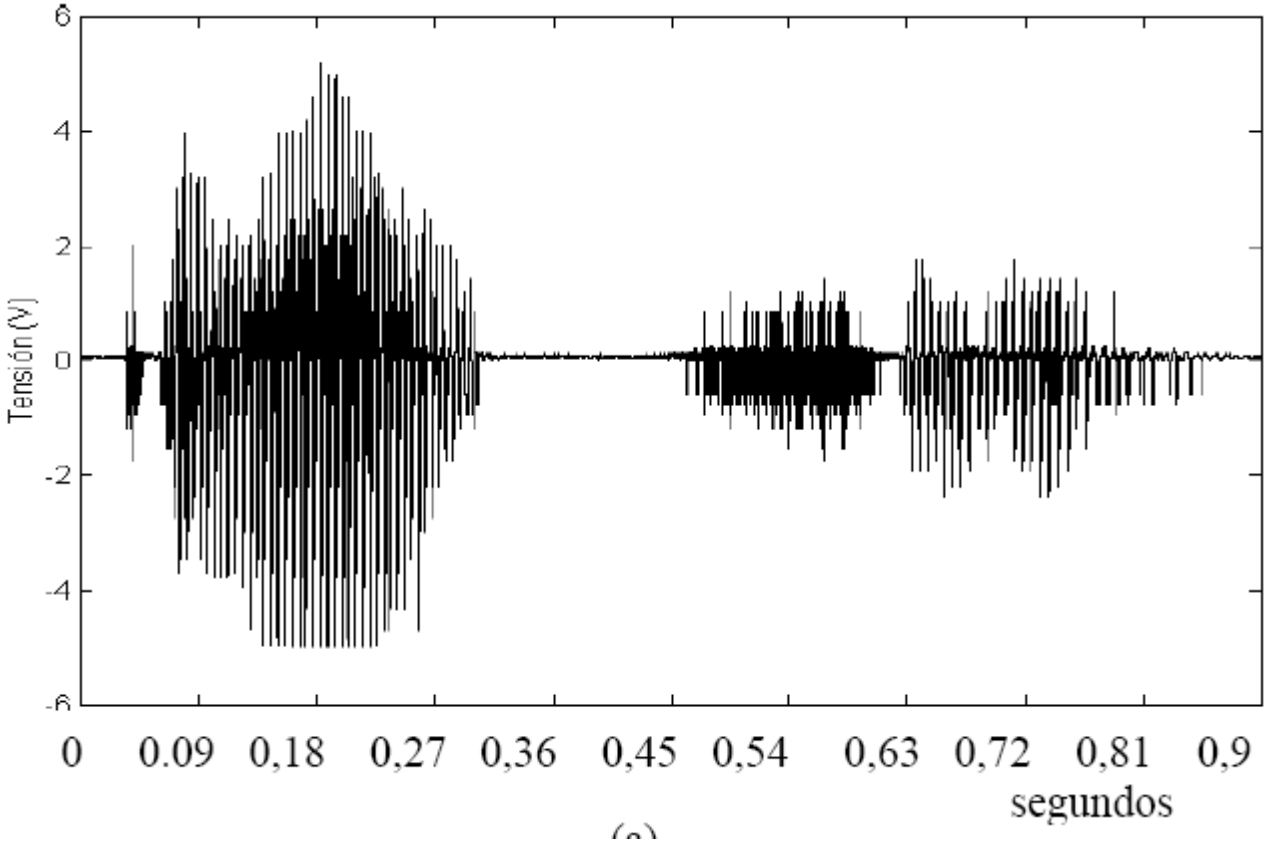
El micrófono produce una señal analógica, es decir, una señal que puede tomar cualquier valor dentro de un determinado intervalo continuo.

Posteriormente la señal es amplificada para encajarla dentro de dos valores límite.

Como por ejemplo -5 y 5 voltios.

Muestreo (II)

Esta figura muestra la señal producida al pronunciar la palabra "casa":



Muestreo (III)

En un intervalo de tiempo continuo (en la figura entre 0 y 0'9 sg) se obtienen **infinitos valores** de la señal analógica ...

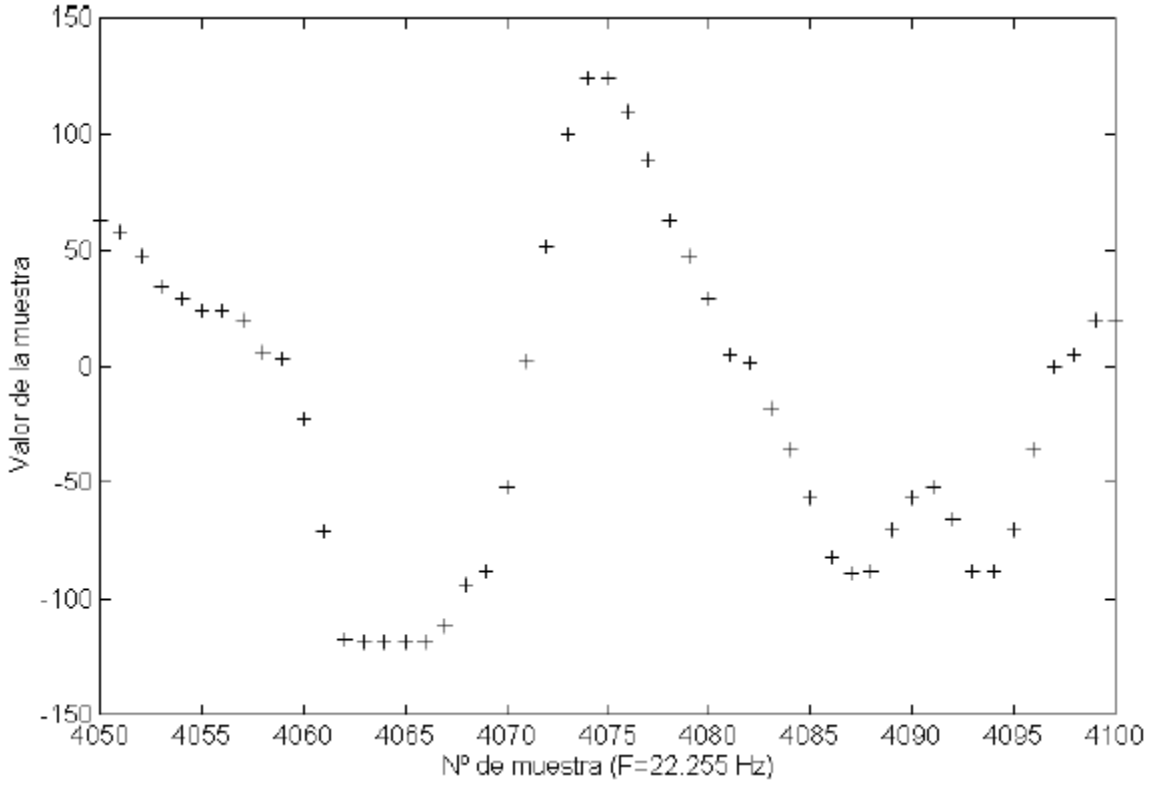
... pero para poder almacenarla y procesarla por medio de técnicas digitales hay que realizar previamente un proceso de **muestreo**.

El proceso de muestreo consiste en tomar el valor de la señal cada cierto intervalo de tiempo.

El número de muestras en un determinado tiempo se denomina **frecuencia**.

Muestreo (IV)

Si muestreamos la señal a una frecuencia de 22'05 KHz (F=22'05 KHz), en un intervalo de 0 a 0'9 obtendremos aprox. 20000 muestras.



Posición	contenido
1	2
2	2
3	2
4	2
.	.
.	.
4050	63
4051	58
4052	48
4053	35
4054	29
4055	24
4056	24

posición	contenido
4057	20
4058	6
4059	3
4060	-23
4061	-71
4062	-118
.	.
.	.
19996	1
19997	1
19998	1
19999	1
20000	1

Muestreo (V)

Simultáneamente al muestreo la muestran se digitalizan (se transforman a binario) por medio de un conversor analógico/digital.

Resumiendo, la señal de sonido queda representada por una secuencia de valores, por ejemplo de 8 bits, correspondiendo cada uno de ellos a una muestra analógica.

- La palabra "casa" del ejemplo quedaría guardada en una tabla de unos 20000 elementos, siendo cada uno de los elementos el valor en binario de cada una de las muestras.

Muestreo (y VI)

En el muestreo intervienen principalmente dos parámetros:

- **Frecuencia** del muestreo.
  - Veces que tomamos la muestra por unidad de tiempo.
  - Debe ser lo suficientemente grande como para no perder la forma de la señal original.
- **Número de bits** de la muestra.
  - Tamaño de cada muestra.

A mayor frecuencia y mayor número de bits:

- Más calidad del sonido.
- Mayor tamaño del fichero que almacena el sonido.

Características de distintos sonidos

	Bits/muestra	**Frecuencia (Hz)**	**Periodo (µsg)**
Teléfono	8	8000	125
Radio	8	22050	45'4
CD	16	44100	22'7
TV digital, DVD		48000	20'8
SACD		2'8 Mhz	0,357

Ejemplo de cálculo de espacio

Ejemplo

Espacio requerido para almacenar sonido digital

Seguidamente, determinaremos el espacio que vamos a necesitar para almacenar una canción de 3 minutos de duración, con calidad de CD, en el disco duro del ordenador.

Como hemos visto antes, el estándar definido para el sonido digital en un CD establece:

- una frecuencia de muestreo de 44 100 Hz,
- una resolución de 16 bits,
- sonido estéreo (2 canales).

Así pues, tenemos:

Tiempo: 3 ~~minutos~~ · 60 segundos / ~~minuto~~ = 180 segundos

Número total de muestras: 180 ~~segundos~~ · 44 100 muestras / ~~segundo~~ = 7 938 00

Número de bits por canal: 7 938 000 · 16 bits / canal = 127 008 000 bits / canal

Número total de bits: 127 008 000 bits / ~~canal~~ · 2 ~~canales~~ = 254 016 000 bits

Si expresamos el resultado anterior en bytes, será: 254 016 000 ~~bits~~ · 1 / 8 ~~bits~~ = 31 752 000 bytes, que equivalen, aproximadamente, a 30 megabytes (MB).

Este resultado nos indica la gran cantidad de espacio necesario para almacenar sonido digital. Un archivo estándar, en calidad CD, suele ocupar unos 10 MB por cada minuto de audio. En las páginas siguientes analizaremos diferentes mane-

Formatos de sonido digital

Una primera clasificación sería:

- Sonido digitalizado
- Sonido MIDI (Musical Instrument Digital Interface). No almacena sonido propiamente, ya que utiliza patrones (instrucciones, códigos y mensajes) que ya están activos en la placa de sonido.

También podríamos clasificarlos:

- Sin compresión
- Con compresión.
  - Sin Pérdida. Una vez realizado el proceso de comprimir-descomprimir, se obtiene el fichero original sin cambios.
  - Con Pérdida. Al comprimir-descomprimir el fichero obtenido no es idéntico al original (ej. MP3)

Representación de imágenes

Las imágenes se pueden adquirir de muy diversas formas:

- Escáneres.
- Cámaras de vídeo.
- Cámaras fotográficas (analógicas y/o digitales).
- Simplemente dibujando.

Hay varios sistemas de codificación de imágenes, los podemos clasificar en:

- Mapas de bits.
- Mapas de vectores (imágenes vectoriales).

Mapas de bits (I)

Una imagen está compuesta por infinitos puntos.

A cada uno de dichos puntos se le asocia:

- Un nivel de gris, si se trata de una imagen en blanco y negro.
- Un color, si se trata de imágenes a color.

Para codificar y almacenar una imagen hay que tener en cuenta dos factores:

- Número de puntos a considerar.
- Código de atributo asociado a cada uno de ellos.

Mapas de bits (II)

No podemos almacenar y/o procesar:

- Ni infinitos puntos.
- Ni infinitos colores.

Sobre los puntos:

- La imagen se divide en una retícula de celdas o elementos de imagen (**pixels**).
- La calidad de la imagen vendrá definida por su resolución.
- La **resolución** es el número de pixels horizontales que tiene la imagen multiplicados por el número de pixels verticales.
- ¿Cuántos puntos forman una imagen 800x600?

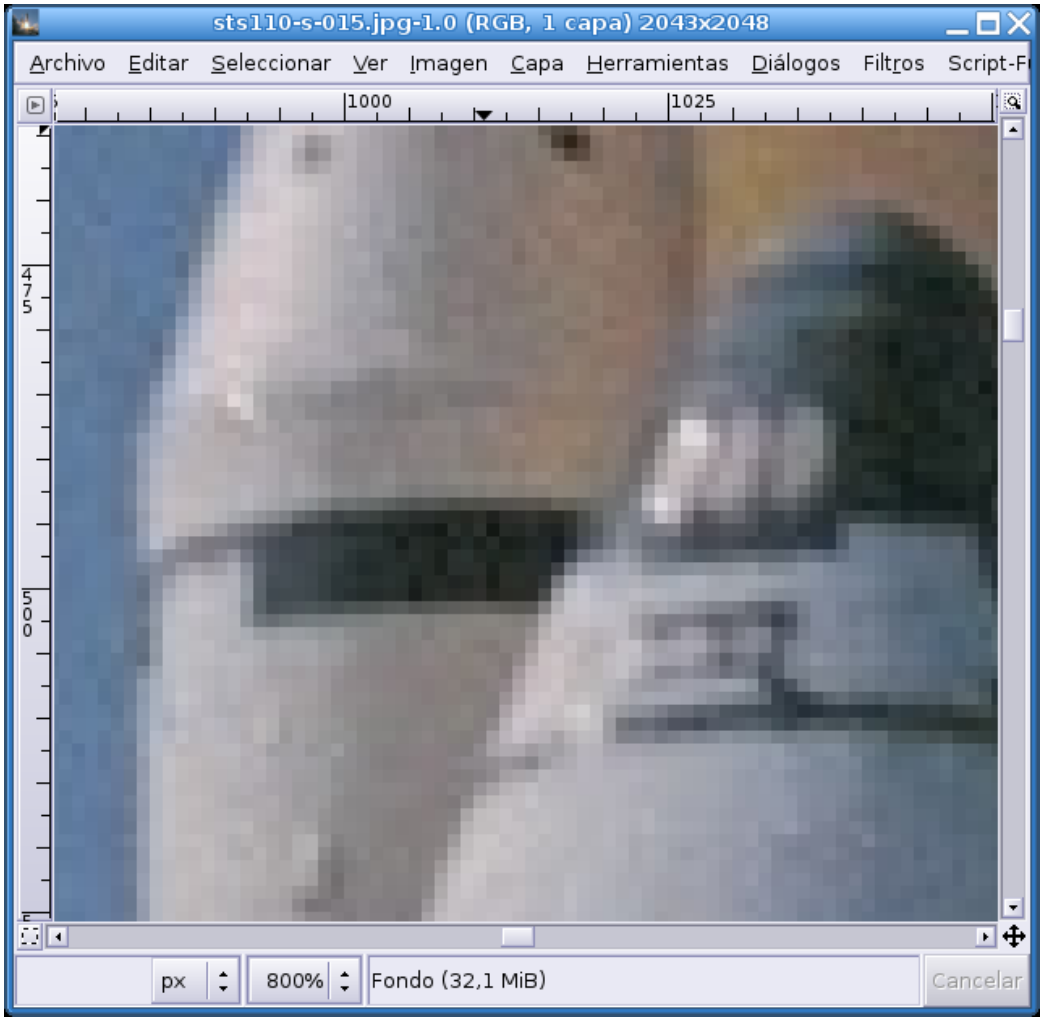
Mapas de bits (III)

La imagen de una fotografía también se forma por puntos, el ojo humano la considera continua (sin puntos) a resoluciones típicas de 640x480, 800x600, 1024x768 etc.

El tamaño influye.

Para una misma resolución, a mayor tamaño peor calidad.

Mapa de bits (y IV)



¿Y qué pasa con el color? (y II)

Pero el ojo humano presenta ciertas limitaciones:

No puede distinguir más de 16 millones de colores.

Solución:

Asignamos a cada color un número de 24 bits, comenzando por el negro y acabando por el blanco.

Negro: 0x000000

Blanco: 0xFFFFFF

Configurar resolución (I)

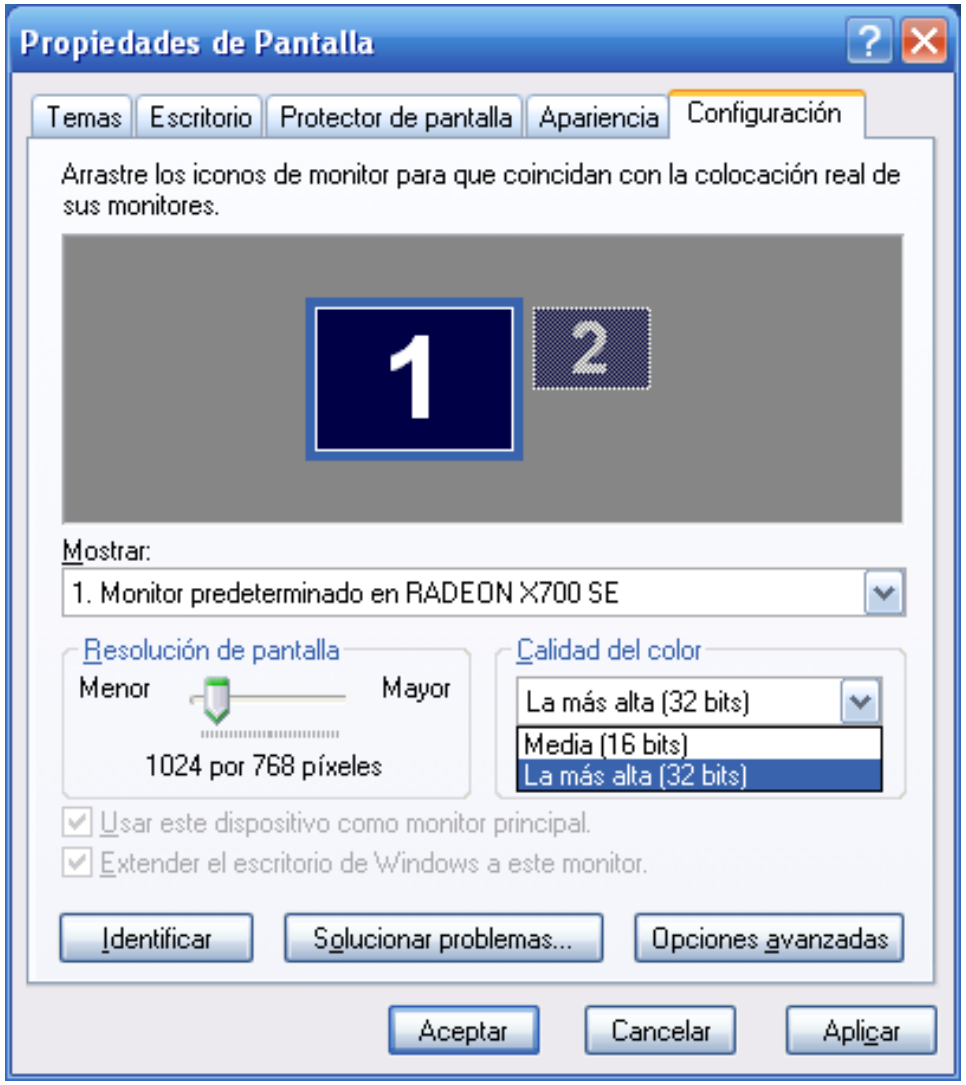
Cuando fijamos la resolución en nuestro ordenador siempre fijamos:

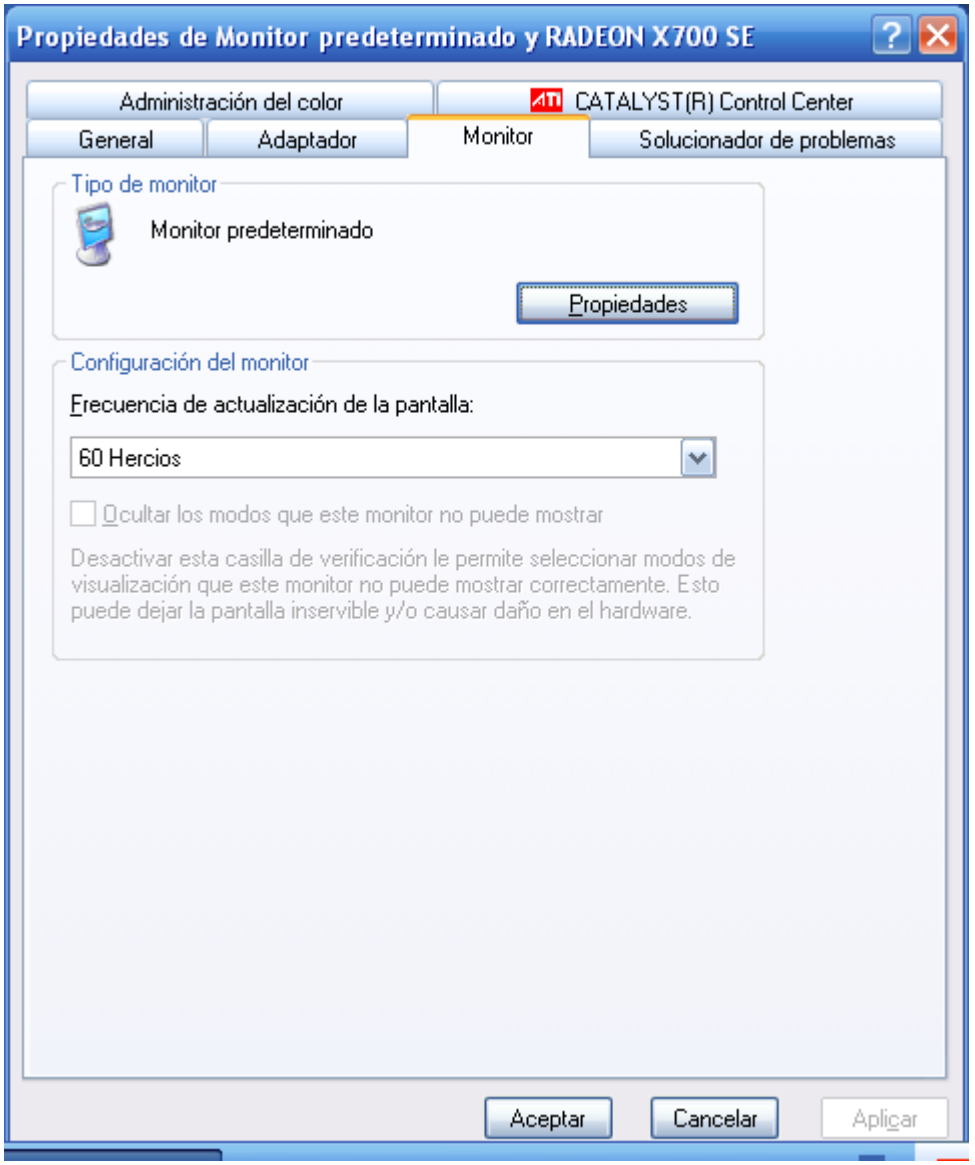
La resolución de puntos.

La profundidad de color:

2 bits.	Monocromo.
4 bits.	16 colores.
8 bits.	256 colores.
16 bits.	65536 colores (High Color).
24 bits.	16 millones de colores (True Color).
32 bits.	16 millones de colores (True Color 32 bits).

Configurar resolución (y II)





Mapa de vectores

Hay otra forma de codificar imágenes sin recurrir a los mapas de bits.

Una imagen se puede descomponer en una colección de objetos como:

- Líneas, polígonos, círculos, ...

Una imagen vectorial almacena toda la información necesaria para poder visualizarla posteriormente.

El software que la visualiza debe poder "dibujar" la imagen siguiendo las instrucciones.

Características

Adecuadas para gráficos de tipo geométrico, no son adecuadas para representar imágenes reales.

Mucho menor tamaño que los mapas de bits.

Adecuadas también para almacenar dibujos.

Usada en aplicaciones gráficas CAD/CAM.

Escalan muy fácilmente sin perder calidad.

Formatos vectoriales conocidos

DXF (Document eXchange Format).

- Utilizado en programas como AutoCAD, CorelDRAW ...).

PS (Postscript), PDF (Portable Document Format).

- Utilizados para la generación de documentos de impresión.

TrueType.

- Utilizados para tipos de letra.

SVG (Scalable Vector Graphics).

- Basado en XML.

Representación de datos numéricos

En la E/S los números son tratados y codificados como caracteres de texto.

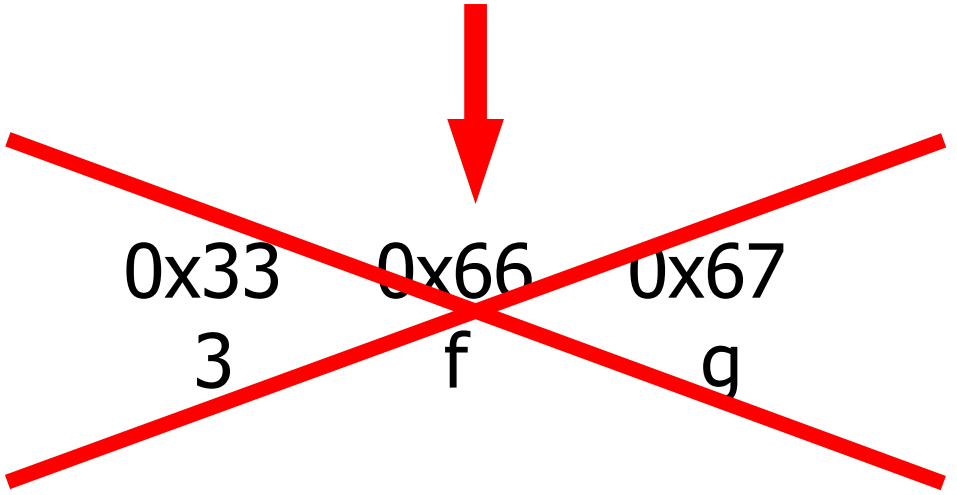
Esta codificación no sirve a la hora de hacer operaciones.

Si un número se va a utilizar en un programa como dato numérico, es necesario convertir el número desde la representación ASCII a un número en base 2 normal con el que poder operar.

Ejemplo

Queremos sumar 345 + 22 (resultado 367).

345	$(0x333435)_{ASCII}$	00110011 00110100 00110101
+ 22	$(0x3232)_{ASCII}$	+ 00110010 00110010
367		00110011 01100110 01100111



Representaciones numéricas

- Es mejor no utilizar la representación ASCII para hacer operaciones.
  - Las cantidades ocupan menos.
  - Hay algoritmos muy eficientes que trabajan directamente con cantidades.
  - La representación ASCII simplemente no sirve.
- Tenemos que representar (y además con bits):
  - Números enteros.
  - Números reales.

Tipos de datos

Si se trata directamente los números decimales tal como se representan en binario, sí se puede operar con ellos.

Cuando programemos, guardaremos nuestros datos en variables, cada variable será de un tipo de datos en concreto.

Depende del tipo de datos de la variable la trataremos de una forma u otra.

Tipo de datos en C

Tipo	Tamaño en bits (aproximado)	Intervalo mínimo
- char	8	-127 a 127
- unsigned char	8	0 a 255
- signed char	8	-127 a 127
- int	16 ó 32	-32.768 a 32.768
- unsigned int	16 ó 32	0 a 65.535
- signed int	16 ó 32	Igual que **int**
- short int	16	-32.768 a 32.768
- unsigned short int	16	0 a 65.535
- signed short int	16	Igual que **short int**
- long int	32	-2.147.483.647 a 2.147.483.647
- long long int	64	$-(2^{63} - 1)$ a $2^{63} - 1$
- signed long int	32	Igual que **long int**
- unsigned long int	32	0 a 4.294.967.295
- unsigned long long int	64	0 a 2^{64} -1
- float	32	1e-37 a 1e+37 (6 dígitos de precisión)
- double	64	1e-37 a 1e+37 (10 dígitos de precisión)
- long double	80	1e-37 a 1e+37 (10 dígitos de precisión)

Datos de tipo entero

Hay varias formas de representar datos de tipo entero:

- Enteros sin signo (naturales):
  - **Binario natural.**
- Enteros con signo:
  - Representación signo-magnitud.
  - Representación en complemento a 1.
  - **Representación en complemento a 2.**
  - Representación sesgada.

Enteros sin signo

Elegido un tamaño en bits, los enteros sin signo se representan tal cual en binario, desde el cero hasta el mayor número que se pueda representar.

Si tenemos 16 bits:

- 2^{16} números.
- 65536 números.
- Desde el 0 al 2^{16}-1.
- De 0 a 65535.

Decimal	Binario
0	0000000000000000
1	0000000000000001
2	0000000000000010
3	0000000000000011
...	...
65532	1111111111111100
65533	1111111111111101
65534	1111111111111110
65535	1111111111111111

Representación signo/magnitud

El signo se representa con el bit más significativo:

- 0 -> números positivos.
- 1 -> números negativos.

Si suponemos 8 bits podemos representar:

- Desde el $-(2^{n-1}-1)$ hasta el $2^{n-1}-1$.
- Desde el -127 hasta el 127.
- Vamos a tener 2^8 numeros: 2^7 positivos y 2^7 negativos.

Pequeño problema:

- Vamos a tener dos ceros: -0 y +0.

Representación signo/magnitud

Decimal	Binario (8 bits)
-127	11111111
-126	11111110
-125	11111101
...	...
-2	10000010
-1	10000001
-0	10000000
+0	00000000
1	00000001
2	00000010
...	...
125	01111101
126	01111110
127	01111111

Representación en complemento a la base (complemento a 2)

Utilizado en los procesadores actuales.

Se reserva el bit más significativo para representar el signo (0 positivos, 1 negativos).

Los números positivos se representan con el bit más significativo a cero, y su valor en binario natural.

Los números negativos se representan con el bit más significativo puesto a 1, y su valor en complemento a 2.

Complemento a 2

Pero... ¿qué ventajas tiene esto?

Principalmente dos:

1.- Pasar un número a su complemento es muy sencillo.

2.- Las restas se hacen como sumas.

Pero también:

3.- No vamos a tener dos ceros.

4.- Podemos representar números desde -2^{n-1} a $2^{n-1}-1$.

Todos los procesadores actuales utilizan esta representación para los números enteros.

Ventaja 1

Para convertir un número a su complemento basta con:

- Invertir todos los bits.
- Sumar 1.

Pasar 6 a su complemento, -6, es tan sencillo como:

6 = 0110 -> 1001 -> 1001 + 1 = **1010**

Supongamos un ejemplo en 16 bits, 23456

0101101110100000 -> 1010010001011111

1010010001011111 + 1 = **1010010001100000**

-23456 = **1010010001100000**

Ventaja 2

Supongamos 4 bits.

Para restar basta sumar al minuendo el complemento a 2 del sustraendo, tras la operación, despreciamos el último bit.

Queremos realizar la operación 6-3=**3**

La realizamos como una suma:

$$\begin{array}{r} 6 \\ +\ -3 \\ \hline 3 \end{array} \qquad 3 = 0011 \rightarrow 1100 + 1 = 1101 \qquad \begin{array}{r} 0110 \\ +\ 1101 \\ \hline (1)0011 \end{array}$$

En el resultado se descarta el último bit que es 1 (tan sólo tenemos 4 bits).

Otro ejemplo

Supongamos 8 bits.

Queremos realizar la operación 122 – 87 = 35.

122
+ -87 87 = 01010111 -> 10101000 + 1 = **10101001**

122	01111010	
+ -87	+10101001	
35	1 00100011	**Resultado: 00100011 (35)**

Represent. enteros con 4 bits

Nº decimal	Signo magnitud	Complemento a 1	Complemento a 2	Sesgado
7	0111	0111	0111	1111
6	0110	0110	0110	1110
5	0101	0101	0101	1101
4	0100	0100	0100	1100
3	0011	0011	0011	1011
2	0010	0010	0010	1010
1	0001	0001	0001	1001
+0	0000	0000	0000	1000
-0	1000	1111	-	-
-1	1001	1110	1111	0111
-2	1010	1101	1110	0110
-3	1011	1100	1101	0101
-4	1100	1011	1100	0100
-5	1101	1010	1011	0011
-6	1110	1001	1010	0010
-7	1111	1000	1001	0001
-8	-	-	1000	0000

Datos de tipo real

También necesitaremos una notación para poder expresar números reales.

Dentro de los números reales tendremos que manejar tanto números muy grandes como números muy pequeños:

Número grande: 8.000.000.000.000.000.000.000.000

Número pequeño: 0'000000000000123

Para estos números será muy útil la notación exponencial.

Notación exponencial

También llamada notación científica o notación en coma flotante.

Consiste en expresar un número así:

$$\mathbf{N = M * B^E}$$

... siendo: M = mantisa, B = base y E = exponente.

Ejemplos:

$8.000.000.000 = 8 * 10^9$

$0'00000000001 = 1 * 10^{-11}$

Notación exponencial

Se la llama coma flotante porque un mismo número puede expresarse de diferentes formas según dónde pongamos la coma decimal:

13257'3285 = 13257'3285 * 10^{0} = 1'32573285 * 10^{4} = 0'132573285 * 10^{5} = 132573'285 * 10^{-1} = 13257328500 * 10^{-6}

Coma flotante

La responsabilidad y manejo de datos en coma flotante puede ser responsabilidad del:

Hardware.

Para ello el procesador debe contar con una unidad en coma flotante.

Algunos procesadores no tienen esta unidad pero podían contar con un coprocesador matemático.

Software.

Siempre se pueden realizar por software dichas operaciones.

Son mucho más lentas.

Coma flotante: IEEE 754

Hasta los años 80 no había una representación única para los números reales.

Cada fabricante utilizaba la suya.

El IEE lo normalizó creando un estándar de representación llamado: IEEE 754.

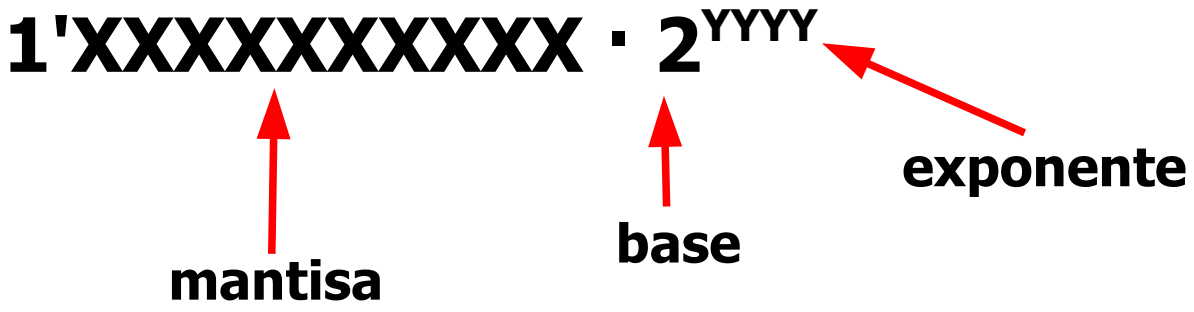
IEEE 754 es capaz de representar números reales:

- En simple precisión (32 bits).
  - Rango: $1'401*10^{-45}$ a $3'403 * 10^{38}$
- En doble precisión (64 bits).
  - Rango $4'941*10^{-324}$ a $1'798 * 10^{308}$

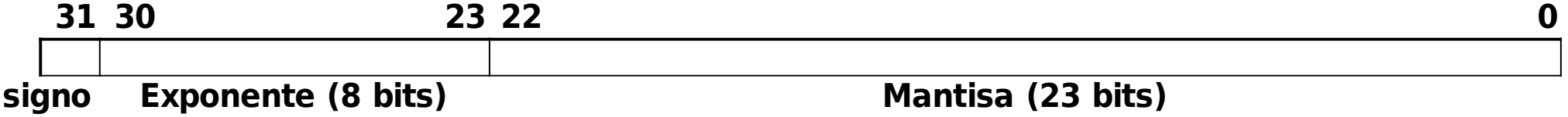
Características IEEE 754 (I)

(simple precisión)

Los números se representan con un sólo dígito a la izquierda del punto decimal:



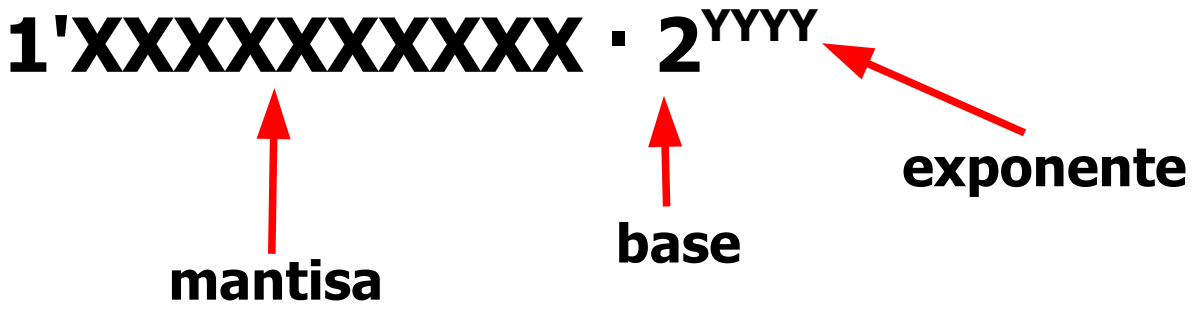
El orden de almacenamiento (en **32 bits**) es:



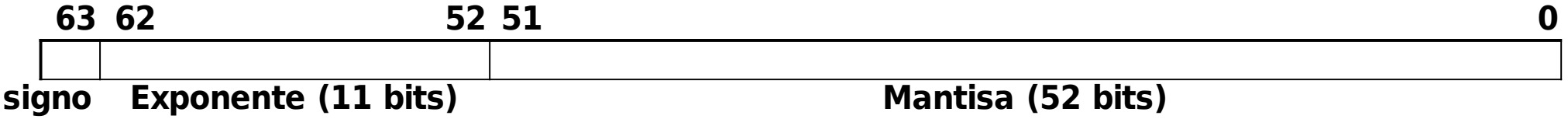
Características IEEE 754 (II)

(doble precisión)

Los números se representan con un sólo dígito a la izquierda del punto decimal:



El orden de almacenamiento (en 64 **bits**) es:



Códigos de detección de errores

Un código de detección de errores introduce información redundante para poder detectar si se ha producido algún tipo de error en la comunicación.

Hablamos de comunicación en un sentido amplio entre un emisor y un receptor:

- CPU y memoria.
- Al leer un CD.
- Dos máquinas que se comunican a través de la red.
- etc.

Bit de paridad

- Consiste en añadir un bit para determinar posibles errores.
  - Bit de paridad par.
    - Se añade un bit (0 ó 1) de tal forma que el número total de 1's sea par.
  - Bir de paridad impar.
    - Se añade un bit (0 ó 1) de tal forma que el número total de 1's sea impar.
- Muy utilizado por ejemplo en memorias RAM.

Ejemplos de bit de paridad

Tenemos los siguientes bytes:

11100101

10011010

11100011

10001101

10000000

00000000

¿Cuál sería su bit de paridad?

Paridad par

Bytes con sus bits de paridad par:

11100101 **1**

10011010 **0**

11100011 **1**

10001101 **0**

10000000 **1**

00000000 **0**

Paridad impar

Bytes con sus bits de paridad impar:

11100101 **0**

10011010 **1**

11100011 **0**

10001101 **1**

10000000 **0**

00000000 **1**

Código de redundancia cíclica (CRC)

El bit de paridad solo puede detectar un error, o a lo sumo un número impar de errores.

No es adecuado para bloques grandes (1KB, 64 KB, etc.).

El CRC es un código polinómico muy utilizado actualmente, tanto en software como en hardware.

Es capaz de detectar cualquier error en un bloque de bytes.

Muy utilizado en redes, medios de almacenamiento, etc.

Más información en:

PRIETO A., LLORIS A. y TORRES J.C. **Introducción a la Informática (3ªed.)**. Ed. McGraw-Hill. 2002.

Wikipedia, la enciclopedia libre. http://es.wikipedia.org/wiki/Portada