Tema 2
Representación de la información

Fundamentos de Computadores

Curso 2018/19

Índice

Índice

Introducción

- Un ordenador es máquina que procesa información
  - INSTRUCCIONES que se ejecutan con DATOS.
- El humano introduce la información mediante caracteres:
  - Alfabéticos: {a,b, ..., z, A, B, ..., Z}.
  - Numéricos: {0, 1, ..., 9}.
  - Especiales: {(,), *, +, -, ?, ...}.
  - Control: {fin de línea, alimentación de línea, ... }.
- La máquina sólo entiende secuencias de 0 y 1
  - Por tanto, la codificación es el proceso de asignar a cada uno de los caracteres una secuencia de ceros y unos (Código E/S)
  - La correspondencia anterior es arbitraria, aunque existen códigos normalizados.
- Para la realización de operaciones aritméticas, se establece una segunda correspondencia entre el código de E/S y un código basado en el sistema de numeración posicional en base 2.

Índice

Representación posicional de los números

- Sistema de numeración posicional en base b:
  - Utilización de un alfabeto de b símbolos diferentes (o cifras).
  - Representación de cualquier número como una secuencia cifras, contribuyendo cada una de ellas con un valor que depende de la cifra en sí y de su posición en la secuencia.
  - Número N como una secuencia de cifras

$$N = \ldots\ n_2\ n_1\ n_0\ ,\ n_{-1}\ n_{-2}\ \ldots$$

  - Valor numérico del número N interpretado en base b

$$N = \ldots + n_2*b^2 + n_1*b^1 + n_0*b^0 + n_{-1}*b^{-1} + n_{-2}*b^{-2} + \ldots$$

- Ejemplo:
  - b = 10 ⇒ Conjunto de símbolos = {0, 1, 2, 3, 4, ..., 9}

$$3278{,}52 = 3000 + 200 + 70 + 8 + 0{,}5 + 0{,}02 =$$
$$3*10^3 + 2*10^2 + 7*10^1 + 8*10^0 + 5*10^{-1} + 2*10^{-2}$$

Representaciones más comunes

- Puede usarse cualquier base b, aunque nos centraremos en:
  - Base 2 ($b = 2$): sistema binario natural.
    - Alfabeto = {0, 1}.
  - Base 8 ($b = 8$): sistema octal.
    - Alfabeto = {0 ... 7}.
  - Base 10 ($b = 10$): sistema decimal.
    - Alfabeto = {0 ... 9}.
  - Base 16 ($b = 16$): sistema hexadecimal.
    - Alfabeto = {0 ... 9, A ... F}.

Representación binaria

- Base 2 ($b = 2$): sistema binario natural.
  - Alfabeto = {0, 1}.
- Ejemplo: números binarios de 3 cifras existentes y sus valores decimales.

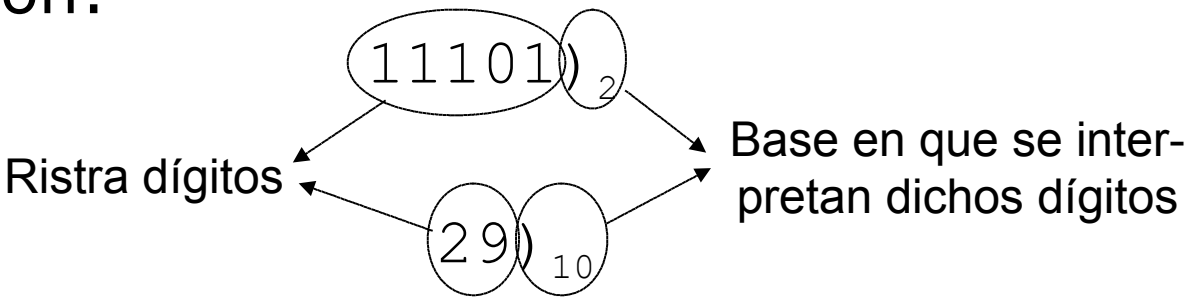
$000)_2 = 0)_{10}$ $\quad$ $100)_2 = 4)_{10}$

$001)_2 = 1)_{10}$ $\quad$ $101)_2 = 5)_{10}$

$010)_2 = 2)_{10}$ $\quad$ $110)_2 = 6)_{10}$

$011)_2 = 3)_{10}$ $\quad$ $111)_2 = 7)_{10}$

- Notación:



Conversión de binario a decimal

- Ejemplos:
  - $10100)_2 = 1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 0*2^0 = 20)_{10}$
  - $0,101)_2 = 1*2^{-1} + 1*2^{-3} = 0,625)_{10}$

- Ejercicio propuesto:
  - $11110,101)_2 = ?$

Conversión de decimal a binario (I)

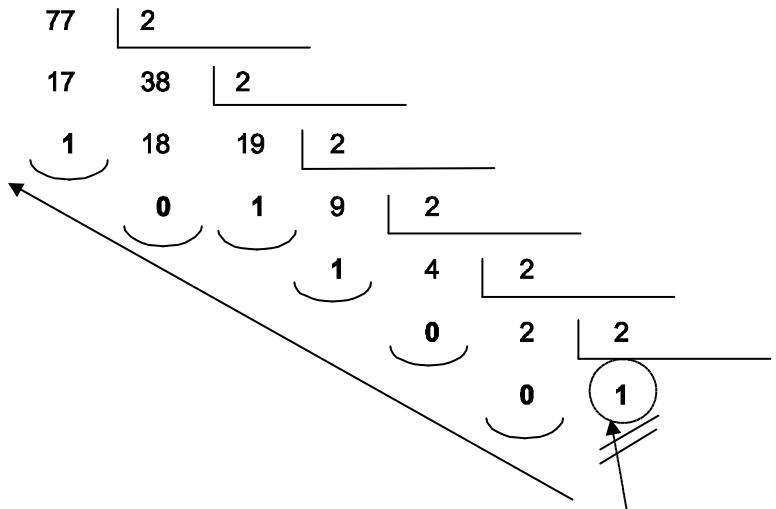
- Procedimiento:
  - Parte entera: ir dividiendo entre 2 la parte entera del número decimal original y, sucesivamente, los cocientes que se van obteniendo. Los restos de estas divisiones y el último cociente son las cifras binarias. El último cociente es el bit más significativo y el primer resto el menos significativo.

  - Parte fraccionaria: ir multiplicando por 2 la parte fraccionaria del número decimal original y, sucesivamente, las partes fraccionarias de los números obtenidos. El número binario se forma con las partes enteras que se van obteniendo.

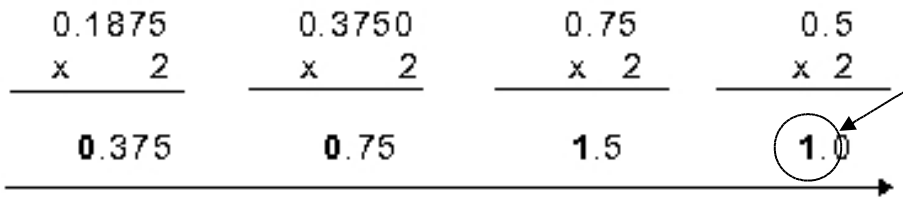
Conversión de decimal a binario (I)

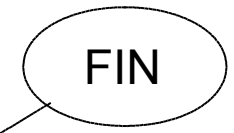
- Ejemplos:

$77)_{10} = 1001101)_2 = 64+8+4+1$



$0,1875)_{10} = 0,0011)_2 = 1/8+1/16$





Funciona exactamente igual para otra base cualquiera (p.e. 8, 16, etc.)

Operaciones aritméticas

- Se usan las siguientes tablas:

SUMA
$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$ y llevo 1

RESTA
$0 - 0 = 0$
$0 - 1 = 1$ y debo 1
$1 - 0 = 1$
$1 - 1 = 0$

PRODUCTO
$0 \cdot 0 = 0$
$0 \cdot 1 = 0$
$1 \cdot 0 = 0$
$1 \cdot 1 = 1$

DIVISIÓN
$0 / 0 =$ Indetermin.
$0 / 1 = 0$
$1 / 0 = \infty$
$1 / 1 = 1$

- Ejemplos (intentar hacer como ejercicios):

$1011101)_2$	$93)_{10}$	$1011101)_2$	$93)_{10}$
$+\ 1000101)_2$	$+\ 69)_{10}$	$-\ 1000111)_2$	$-\ 71)_{10}$
$1101010)_2$	$106)_{10}$	$13{,}25)_{10}$	$5)_{10}$
$\times\quad 101)_2$	$\times\ 5)_{10}$	$1101{,}01)_2$	$101)_2$

Desplazamientos

- Desplazamiento a la izquierda:
  - Multiplicar por $10)_2$ (e.d. la base) es equivalente a añadir un cero al final o, en su caso, desplazar el punto decimal a la derecha.
- Desplazamiento a la derecha:
  - Dividir entre $10)_2$ (e.d. la base) es igual a eliminar ceros a la derecha o, en su caso, desplazar el punto decimal a la izquierda
- Ejemplos:
  - $1101010)_2 * 10)_2 = 11010100)_2$

    $106)_{10} * 2)_{10} = 212)_{10}$
  - $1101010)_2 / 10)_2 = 110101)_2$

    $106)_{10} / 2)_{10} = 53)_{10}$

Operaciones lógicas

- Se aplican bit a bit.
- Se usan las siguientes tablas:

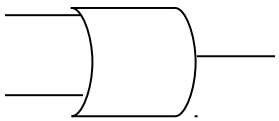
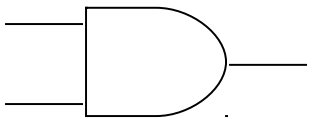
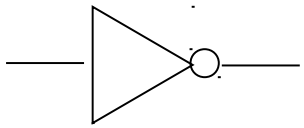
SUMA LÓGICA (OR)
$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 1$

PRODUCTO LÓGICO (AND)
$0 \cdot 0 = 0$
$0 \cdot 1 = 0$
$1 \cdot 0 = 0$
$1 \cdot 1 = 1$

COMPLEMENTO (NOT)
$\overline{0} = 1$
$\overline{1} = 0$

- Representación gráfica de las operaciones:

  Puertas lógicas, se usarán en tema 3

Representación sin signo

- Valores sin signo: valor absoluto del número entero en binario natural.

  Ejemplo: $0)_{10} = 00000000)_2$

  n=8 bits $5)_{10} = 00000101)_2$

  $247)_{10} = 11110111)_2$

  Rango = 0 ... 2^n-1

N > 0 (binario natural, n bits)

a_{n-1} . . . a_0

- Representaciones más comunes:
  - Byte: n=8
  - Entero corto: n=16
  - Entero: n=32
  - Entero largo: n=64

Representación con signo

1. Signo y magnitud:

Si N negativo ⇒ S = 1 Si N positivo ⇒ S = 0

Rango = $-(2^{n-1}-1)$ ... $2^{n-1}-1$ → Doble representación del cero

Ejemplo: $+7)_{10} = 00000111)_{SM}$ $-8)_{10} = 10001000)_{SM}$

S	N > 0 (binario natural, n-1 bits)

a_{n-1} . . . a_0

2. Sesgada:

S es una constante denominada sesgo

El resultado de la suma del entero a representar y el sesgo ya es un número natural (>=0), que se representa ya en binario natural.

Ejemplo: 8 bits -> $S=2^{8-1}=2^7=128$

$+7)_{10} = 10000111)_{sesgada}$ $-8)_{10} = 01111000)_{sesgada}$

Rango = $-(2^{n-1})$ ... $2^{n-1}-1$ → Ya no hay doble representación del 0

N + S, $S = 2^{n-1}$

a_{n-1} . . . a_0

Representación con signo (complementos)

- Se emplea para poder realizar las restas como sumas.
  - Simplificación de la circuitería de los procesadores.

3. Complemento a 2:

- Número que resulta de restar cada una de las cifras de N a la base menos 1 (2-1=1 en binario) y posteriormente sumar 1 a la diferencia obtenida.
- Es decir, para calcular el C2 de un número binario:
  - Cambiamos sus ceros por unos y viceversa
  - Le sumamos 1.
- Se cumple siempre que C2 (C2(N)) = N (e.d. -(-N) = N)
- Ejemplo:
  - $7)_{10} = 0111)_2$; $-7)_{10} = C2((0111)_2) = 1001)_2$

$4)_{10} + (-7)_{10} = 0100)_2 + 1001)_2 = 1101)_2 = 0011)_2 = -3)_{10}$

Representación C2

- Representación interna en C2

  Si N negativo ⇒ S = 1 Si N positivo ⇒ S = 0

  Rango = $-(2^{n-1})$ ... $2^{n-1}-1$ (tampoco doble representación del 0)

  Ejemplo:

  $+7)_{10} = 00000111)_{C2}$

  $-8)_{10} = 11111000)_{C2}$

S	Si S=0, N>0; Si no, C2(N>0)

a_{n-1} . . . a_0

- Por tanto, para realizar **restas**:
  - Se suma al minuendo el complemento a 2 del sustraendo despreciando, en su caso, el acarreo del resultado.
  - Ejemplo:
    - $7)_{10} - 7)_{10} = 0111)_2 +' C2(0111)_2) =$

      $0111)_2 +' 1001)_2 = 0000)_2$
    - Nota: $+'$ simboliza una suma con desprecio de acarreo.
- **Desbordamiento**:
  - Obviamente, todo esto funciona sólo si no nos salimos del rango: P.e. 62+100 = +162, ó –50-90=-140, no funcionan bien en C2 de 8 bits, ya que en ambos casos se salen del rango posible [-128,+127]

Representación C2

- Representación interna en C2 (extensión de signo)
  - Si N negativo ⇒ S = 1 Si N positivo ⇒ S = 0
  - Extensión de signo:

    Para extender un número dado en complemento a 2 a un mayor número de bits (p.e., de 8 a 16 bits o 32 bits, o de 16 a 32 bits, etc.), simplemente hay que replicar el bit de signo hacia la izquierda tantos bits como sea necesario
  - Ejemplo:

$$+7)_{10} = 00000111)_{C2(8\ bits)} =$$
$$= 00000000\ 00000111)_{C2(16\ bits)}$$
$$= 00000000\ 00000000\ 00000000\ 00000111)_{C2(32\ bits)}$$
$$-8)_{10} = 11111000)_{C2(8\ bits)}$$
$$= 11111111\ 11111000)_{C2(16\ bits)}$$
$$= 11111111\ 11111111\ 11111111\ 11111000)_{C2(32\ bits)}$$

Índice

Códigos intermedios

- El objetivo es poder expresar de forma más abreviada secuencias de 0's y 1's
- Base octal:
  - b = 8 (2^3), dígitos = {0, 1, 2, 3, 4, 5, 6, 7}
  - *Binario a octal*: se agrupan los bits en ternas empezando por la coma decimal (si la hay) y se asigna a cada una de ellas el correspondiente dígito octal.

    $010\ 001\ 101\ 100\ ,\ 110\ 100\ )_2 = 2\ 1\ 5\ 4\ ,\ 6\ 4\ )_8$

  - *Octal a binario*: al revés, cada dígito octal da lugar a 3 binarios.
- Base hexadecimal: b = 16 (2^4), dígitos={0,1,..,9,A,B,..,F}
  - *Binario a hexadecimal*: se agrupan los bits de 4 en 4 empezando por la coma decimal (si la hay) y se sustituye cada grupo por el correspondiente dígito hexadecimal.

    $0100\ 0110\ 1100\ ,\ 1101\ 0000\ )_2 = 4\ 6\ C\ ,\ D\ 0\ )_{16}$

  - *Hexadecimal a binario*: al revés, cada dígito hexadecimal da lugar a 4 binarios.

Conversión a/desde decimal

- Base octal:
  - *Octal a decimal*: aplicar la fórmula (con b=8)

    $N = n_2 * b^2 + n_1 * b^1 + n_0 * b^0 + n_{-1} * b^{-1} + n_{-2} * b^{-2}$

  - *Decimal a octal*: se utiliza el método de las divisiones para la parte entera y el de las multiplicaciones para la fraccionaria.

    $760,33)_{10} = ?)_8$

    Nota.- Ojo, una parte fraccionaria finita en base decimal puede a veces convertirse en una infinita (periódica en algún punto) en otra base (hexadecimal, octal, binaria o la que sea).
- Base hexadecimal:
  - *Hexadecimal a decimal*: aplicar la fórmula (con b=16)

    $N = n_2 * b^2 + n_1 * b^1 + n_0 * b^0 + n_{-1} * b^{-1} + n_{-2} * b^{-2}$

  - *Decimal a hexadecimal*: se utiliza el método de las divisiones para la parte entera y el de las multiplicaciones para la fraccionaria.

    $760,33)_{10} = ?)_{16}$

Índice

Notación exponencial

- Se utiliza normalmente la notación exponencial:

$N = MxB^E$ M = mantisa, B = base, E = exponente

- También llamada representación en coma flotante o punto flotante.
- Podemos cambiar la representación de N, sin cambiar su valor, simplemente aumentando (disminuyendo) en una unidad E y dividiendo (multiplicando) M por B.

$$13257,35 = \begin{cases} 13257,35 * 10^0 \\ 1,325735 * 10^4 \\ 0,1325735 * 10^5 \\ 1325735 * 10^{-2} \\ \ldots \end{cases}$$

Estructura IEEE 754

- Uso de notación exponencial.

B = 2 (al tenerla todos los números, no es necesario almacenarla, se considera implícita)

1. Campo de signo:
   - S = 0 → Número positivo.
   - S = 1 → Número negativo.
2. Campo del exponente:
   - entero sesgado, donde $S = 2^{ne-1}-1 \rightarrow e = E + S$
3. Campo de la mantisa:
   - Cuando el 1 más significativo de la mantisa se encuentra en la posición 0 (de las unidades) el número se dice que está normalizado.

     $101,111x2^{-5} = \mathbf{1},01111x2^{-3}$; $0,0011x2^{7} = \mathbf{1},1x2^{4}$
   - El campo de la mantisa sólo almacena la parte fraccionaria del número normalizado, puesto que el **1** siempre estará (se dice también que está implícito).
   - Cuando el procesador extraiga el dato de la memoria, tendrá que añadir **1**:
     - Por tanto $1 \leq M < 2$ M = 1,m
   - Ejemplo: $101,111x2^{-5} = 1,01111x2^{-3}$ m = 011110...0

Ejemplo conversión IEEE 754 → Decimal

Calcular el número real en base 10 correspondiente a la ristra de 32 bits $C0580000)_{16}$ que se encuentra en formato IEEE 754 de simple precisión ($n=32$ bits, con $ne=8$, luego $nm=23$):

- **Solución:**

Paso 1: $C0580000)_{16} = 1100\ 0000\ 0101\ 1000\ 0000\ 0000\ 0000\ 0000)_2$

$= 1\ 10000000\ 10110000000000000000000)_2$

s (1bit) e (8 bits) m (32-8-1 = 23bits)

Paso 2: $s=1 \rightarrow$ negativo (-)

Paso 3: $e=10000000)_2=128)_{10}$; $e=E+S$; $S=2^7-1)_{10}=127)_{10} \rightarrow$

$E=e-S=128-127)_{10}=1)_{10}$

Paso 4: $m=10110000000000000000000)_2 \rightarrow M=1.m=1.10110\ldots0)_2 \rightarrow$

$M=1.1011)_2$

Paso 5: $N=-M*2^E=-1.1011)_2*2^1=-11.011)_2*2^0=-11.011)_2=-3.375)_{10}$

Situaciones especiales y redondeos

- Situaciones especiales:

  a) Cuando $e = 0$ y $m \neq 0$, número desnormalizado (**1** no está implícito).

  $S = 2^{ne-1} - 2$ y $M = 0,m$

  b) e = 0 y m = 0 → N = 0

  c) e = 111...1:

  - Si m = 0 → +∞ ó -∞ (dependiendo del signo)
  - Si m ≠ 0 → NaN (Not a Number)

- Redondeos:
  - Problema: algunos números reales no pueden ser representados de forma exacta con n_m+1 cifras binarias significativas
  - Redondeo al par: Se redondea al nº representable más cercano. Si el error es el mismo se elige el nº par.
    - Analizar el bit menos significativo (posición -n_m) y 2 bits que genera la ALU: bit de redondeo (posición -n_m-1) y bit retenedor (posición -n_m-2).
    - Ejemplo: n_m = 5

```
1,01101  y  00 →  1,01101  (truncar, siempre)
1,01101  y  01 →  1,01101  (truncar, siempre)
1,01100  y  10 →  1,01100  (truncar, redondeo al par)
1,01101  y  10 →  1,01110  (sumar 1, redondeo al par)
1,01101  y  11 →  1,01110  (sumar 1, siempre)
```

Tamaños que contempla IEEE 754

- Consideraciones respecto al tamaño:
  - Precisión: cifras de la mantisa que van a poder representarse.
  - Rango: intervalo de representación.
  - A mayor n_m, mayor nº de cifras significativas → mayor precisión.
  - A mayor n_e, mayor exponente posible → mayor rango.

- Formatos más habituales IEEE 754:
  - Simple precisión:
    - n = 32 bits (n_e = 8 bits y n_m = 23), S = 2^7-1 = 127
  - Doble precisión:
    - n = 64 bits (ne = 11 bits y nm = 52), S = 2^{10}-1 = 1023

Ejemplo conversión Decimal → IEEE 754

- Representar el número $-250{,}875)_{10}$ en IEEE 754 de 32 bits:

  $N=(-1)^s M*B^E$; (fórmula general) $B=2$; (implícita, siempre)

  Paso 1: expresar valor absoluto del número en binario, separando partes entera y fraccionaria:

  $250)_{10}$ = … (div. sucesivas) … = $11111010)_2$

  $0{,}875)_{10}$ = … (mul. Sucesivas) … = $0{,}111)_2$

  Paso 2: Normalizar, desplazando la coma decimal:

  $250{,}875)_{10}=11111010{,}111)_2*2^0=1{,}1111010111)_2*2^7$ →

  $E=7$ y $M=\mathbf{1},m=\mathbf{1},11110101110000$…(rellenar hasta 23 bits en *m*)…$)_2$

  Paso 3: Obtener exponente sesgado ($S=2^{8-1}-1=127$):

  $e = E+S = 7)_{10} + 127)_{10} = 134)_{10} = 10000110)_2$ (8 bits)

  Paso 4: Construir palabra de 32 bits a partir de s, e y m:

  Num.neg.→**s=1**; **e=10000110**; M=1,m→**m=11110101110000**…

  $N = \underbrace{1}_{s\ (1bit)}\ \underbrace{10000110}_{e\ (8\ bits)}\ \underbrace{11110101110000000000000}_{m\ (23bits)} = C37AE000)_{16}$

IEEE 754: A tener en cuenta

- Es fácil que un número decimal más o menos redondo no pueda ser representado con absoluta exactitud:
  - P.e. N=-7.1: su parte decimal no se puede expresar como suma de potencias negativas de 2, de modo que necesitaría infinitos dígitos binarios tras la coma.
- Ciertas propiedades de los números reales pueden no cumplirse:
  - P.e. la asociatividad: `(a+b)+c = a+(b+c)`. Pero en IEEE 754, si `a=1.0`, `b=-1.0` y $c=2^{-30}$, `(a+b)+c` produce 2^{-30}, mientras que `a+(b+c)` produce `0`, (puesto que en este segundo caso el resultado intermedio `b+c` se ve truncado a 1.0).
  - Por supuesto, este tipo de errores se van acumulando en cálculos más largos...

Todo está relacionado con que al fin y al cabo IEEE 754 es un intento de representar un conjunto continuo de números (reales) en una representación (p.e. 32 o 64 bits)

Índice

Códigos de Entrada/Salida

- Los códigos de E/S (ó externos): asocian a cada carácter una determinada combinación de bits
  - Correspondencia entre α = {A..Z,a..z,0..9,(,),*... } y $\beta = \{0,1\}^n$
- El $|\alpha| = m$ dependerá del dispositivo de E/S o del sistema.
- Necesitaremos un número de bits n tal que: $n \geq \log_2 m$
- Ejemplo: para codificar {0,1, ..., 9} se necesitarán 4 bits.
- La elección del código es arbitraria, pero existen códigos normalizados.
- El más popular es el código ASCII (*American Standard Code for Information Interchange*)
  - Codifica, desde los años 60, la inmensa mayoría de caracteres impresos usados en textos en idioma inglés.

Código ASCII

- *Longitud n* =7 bits ⇒ $m=2^7=128$ caracteres:

ASCII	Hex	Símbolo
0	0	NUL
1	1	SOH
2	2	STX
3	3	ETX
4	4	EOT
5	5	ENQ
6	6	ACK
7	7	BEL
8	8	BS
9	9	TAB
10	A	LF
11	B	VT
12	C	FF
13	D	CR
14	E	SO
15	F	SI

ASCII	Hex	Símbolo
16	10	DLE
17	11	DC1
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US

ASCII	Hex	Símbolo
32	20	(espacio)
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/

ASCII	Hex	Símbolo
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

ASCII	Hex	Símbolo
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O

ASCII	Hex	Símbolo
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

ASCII	Hex	Símbolo
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o

ASCII	Hex	Símbolo
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	□

Código ASCII (cont.)

- Para rellenar hasta 8 bits (1 byte, unidad típica de almacenamiento), se usa 1 bit adicional:
  - Bien para control de errores (siguiente transparencia) …
  - … bien para caracteres gráficos (p.e. caracteres para hacer cajas para enmarcar texto) …
  - O, lo más común, para extensiones de determinados idiomas. Ejemplos más usados en Europa occidental:
    - ISO 8859-1 (también denominado ISO Latin 1): incluye alfabeto latino con extensiones, incluyendo ñ, vocales acentuadas, etc.
    - ISO 8859-15 es una pequeña modificación que añade el signo del euro.

Detección de errores

- En general (no sólo para caracteres), pueden usarse bits adicionales a la información a almacenar con el único propósito de comprobar que ésta no se almacenó/transmitió de manera errónea.
- Ejemplo: **bit de paridad** en código ASCII.
  - Puesto que 1 carácter ASCII = 7 bits, pero 1 byte (unidad más estándar de almacenamiento) = 8 bits, aprovechar el bit sobrante añadiéndolo de forma que el número total de unos en el byte sea **par**:
  - Ejemplo:

(7 bits) (8 bits)

- carácter ASCII 'A'=1000001 → 0 1000001
- carácter ASCII 'C'=1000011 → 1 1000011

Bit de paridad añadido tal que el número de unos es siempre par (2 en el primer caso, 4 en el segundo)

  - Ahora, si tras una transmisión por internet, o una recuperación del dato desde un disco, p.e., se produce un error y un bit cambia su valor (0↔1) nos encontraremos una cadena con un número impar de unos, e induciremos que hubo un error.

Unicode

- **Unicode**: estándar para representación de todo texto posible en todos los sistemas de escritura del mundo.
- Actualmente 110.000 caracteres, de un total de 100 *scripts* (*script* = conjunto de caracteres: alfabetos, silabarios, pictogramas …).
- Inicialmente, era de 16 bits (2^{16}=65.536 posibilidades). Actualmente, Unicode define un espacio de of 1.114.112 *code points* en el rango $0)_{16}$ a $10FFFF)_{16}$.
- Un "*code point*" se expresa como U+número (p.e. U+0058 para la 'X', o U+6708 para el ideograma chino "yue": 月).
- Para "*code points*" que no caben en 16 bits (fuera del llamado Basic Multilingual Plane, o BMP), se usan cinco o seis dígitos, según se requiera.

Unicode (cont.)

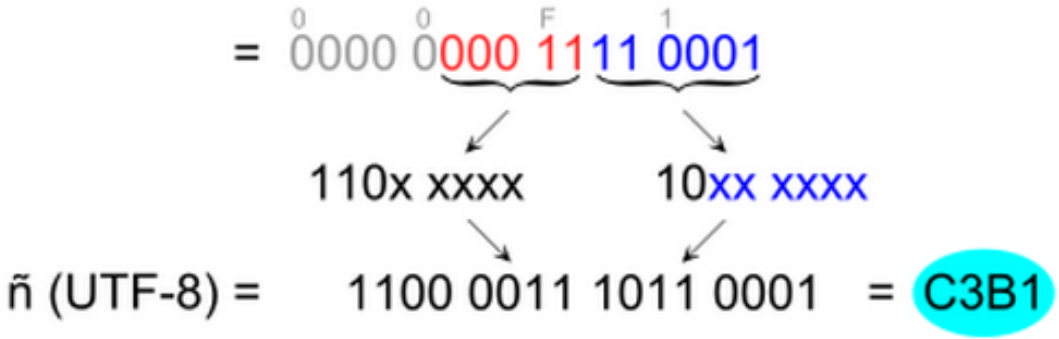
- Los primeros 256 "*code points*" son idénticos a la codificación ISO-8859-1 (Latin 1), para facilitar la conversión, y compatibilizar el subconjunto de caracteres más utilizado de largo.
- Los "*code points*" pueden concretarse en secuencias de bytes de varias formas, llamadas "*encodings*". Las más comunes son UTF-8 y UTF-16 (UTF = Unicode Transformation Format), aunque hay más (UCS-2, UCS-4, etc.).
  - UTF-8 usa un byte por cada carácter ASCII (→ compatibilidad). Sólo para caracteres no ASCII empiezan a utilizarse dos o más bytes (hasta cuatro), según un esquema basado en la comprobación de los primeros bits de cada byte, que indican si el byte siguiente forma parte aún del mismo "*code point*" (ilustración en transparencia siguiente).
  - UTF-16 similar, pero usando como base ristras de 16 bits (dos bytes), con unos pocos caracteres usando dos unidades de 16 bits (e.d. 4 bytes). Menos usado por desperdiciar más espacio.

Unicode (cont.)

- Codificación con UTF-8 (1, 2, 3 ó 4 bytes):

Rango de puntos UNICODE	Valor escalar	UTF-8
000000-00007F	00000000 0xxxxxxx	0xxxxxxx
000080-0007FF	00000yyy yyxxxxxx	110yyyyy 10xxxxxx
000800-00FFFF	zzzzyyyy yyxxxxxx	1110zzzz 10yyyyyy 10xxxxxx
010000-10FFFF	000uuuuu zzzzyyyy yyxxxxxx	11110uuu 10uuzzzz 10yyyyyy 10xxxxxx

- Ejemplo ñ (Unicode) = U+00F1



Comparación ISO-8859-15, UTF-16, UTF-8

Archivos correspondientes en okteta

Texto original

AEIOU
aeiou
ÁÉÍÓÚ
áéíóú
?!$%
¿¡ñÑ

Codificación 8859-15

Desplazam. | Bytes de contenido | Interpretación 8859-15 (por defecto en okteta)

```
0000:0000 41 45 49 4F 55 0A 61 65 69 6F 75 0A C1 C9 CD D3  AEIOU.aeiou.ÁÉÍÓ
0000:0010 DA 0A E1 E9 ED F3 FA 0A 3F 21 24 25 0A BF A1 F1  Ú.áéíóú.?!$%.¿¡ñ
0000:0020 D1 0A                                            Ñ.
```

Codificación UTF-16

BOM (Byte order mark) → Big endian / little endian

```
0000:0000 FF FE 41 00 45 00 49 00 4F 00 55 00 0A 00 61 00  ÿþA.E.I.O.U...a.
0000:0010 65 00 69 00 6F 00 75 00 0A 00 C1 00 C9 00 CD 00  e.i.o.u...Á.É.Í.
0000:0020 D3 00 DA 00 0A 00 E1 00 E9 00 ED 00 F3 00 FA 00  Ó.Ú...á.é.í.ó.ú.
0000:0030 0A 00 3F 00 21 00 24 00 25 00 0A 00 BF 00 A1 00  ..?.!.$.%...¿.¡.
0000:0040 F1 00 D1 00 0A 00                                ñ.Ñ...
```

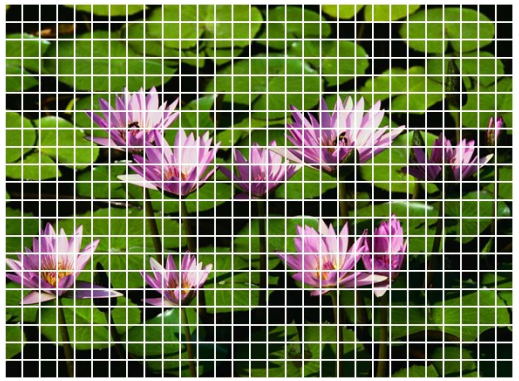
Codificación UTF-8

```
0000:0000 41 45 49 4F 55 0A 61 65 69 6F 75 0A C3 81 C3 89  AEIOU.aeiou.Ã.Ã.
0000:0010 C3 8D C3 93 C3 9A 0A C3 A1 C3 A9 C3 AD C3 B3 C3  Ã.Ã.Ã..Ã¡Ã©Ã ÃºÃ
0000:0020 BA 0A 3F 21 24 25 0A C2 BF C2 A1 C3 B1 C3 91 0A  º.?!$%.Â¿Â¡Ã±Ã..
```

Índice

Mapas de bits

- Las imágenes están compuestas por infinitos puntos
  - Cada punto tiene un nivel de gris o un color
- Para codificar una imagen:
  - Hay que determinar cuántos puntos vamos a considerar
    - No podemos almacenar y procesar infinitos puntos
  - Hay que asociarles un nivel de color o de gris
    - El nivel asociado será la media de los puntos representados
- Consideraremos que una imagen está formada por una matriz de píxeles (elementos de imagen, *picture elements*)
- Resolución de la imagen:
  - Número de elementos por línea * Número de elementos por columna
- Representación de la imagen
  - Sencilla, funciona con cualquier imagen
  - Se almacenan sucesivamente los atributos de los píxeles
    - Ej: De izquierda a derecha y de arriba a abajo



Mapas de vectores

- Representación como colección de objetos:
  - Líneas, polígonos, textos
- Los objetos se modelan mediante vectores y ecuaciones
- Al visualizar la imagen en pantalla se evalúan las ecuaciones y se escalan los vectores para generar la imagen
- Características de este tipo de imágenes:
  - Son adecuadas para gráficos geométricos, pero no para imágenes reales
  - Son ideales para aplicaciones de diseño por ordenador (CAD)
  - En comparación con los mapas de bits, los archivos de imágenes vectoriales suelen ocupar mucho menos espacio
  - Es más fácil procesar o escalar las imágenes a cualquier tamaño
  - La fidelidad o calidad de la imagen respecto a la realidad suele ser peor

Representación del color

- Las imágenes pueden representarse en escala de grises o en color
- En la representación en escala de grises, cada píxel tiene asociado un valor medio de gris
  - Normalmente cada píxel puede tomar 256 valores distintos (1 byte)
- Representación del color RGB
  - La intensidad media de los colores primarios rojo (R), verde (G) y azul (B) se codifica por separado
  - Una gran calidad de colores se consigue con 8 bits por cada color
    - RGB -> Se necesitarían 3 bytes por cada píxel: 2^{8*3} =más de 16 millones de colores distintos
  - RGB es un modelo aditivo, la suma de los colores genera el color blanco
  - El formato RGB se utiliza principalmente para la visualización en pantalla
- Representación del color CYMK
  - El color se representa mediante el cian (C), amarillo (Y), magenta (M) y negro (K)
  - CYMK es un modelo sustractivo, la suma de colores genera el negro
  - El formato CYMK se utiliza principalmente para la impresión con tinta

Ejemplos de formatos de representación de imagen

- Mapa de bits:
  - BMP (BitMap)
    - Usado en aplicaciones Windows
  - PICT (PICTure)
    - Usado en Macintosh
  - PPM (Portable PixMap)
    - Codificación sencilla. Posibilidad de ASCII.
  - JPEG (Joint Photographic Experts Group)
    - Buena calidad para imágenes naturales. Incluye compresión
    - La compresión puede ser sin pérdida o con pérdida
      - La compresión sin pérdida elimina sólo información que el ojo humano no ve
- Mapas de vectores:
  - DXF (Document eXchange Format)
    - Formato normalizado para imágenes CAD (AutoCAD, Corel Draw)
  - EPS (Encapsulated Postscript)
    - Imágenes del lenguaje Postscript (inventado por adobe para documentos con texto y gráficos)
  - ODG (Libreoffice)
  - A menudo incluyen(como subimágenes) también mapas de bits.