

Tema 1: Información digital

1.1- Introducción.

1.2- Sistemas de numeración.

1.3- Representaciones numéricas: n° Naturales.

1.4- Representaciones numéricas: n° Enteros.

1.4.1 – Signo-Magnitud.

1.4.2 – Complemento a 2.

1.4.3 – Exceso a un entero Z .

1.5- Representaciones numéricas: n° Reales.

1.5.1 – Coma fija.

1.5.2 – Coma flotante: Formato simplificado y Formato IEEE-754 simple.

1.6- Representación de caracteres.



Área de Arquitectura y Tecnología de Computadores
Departamento de Informática de la Universidad de Oviedo

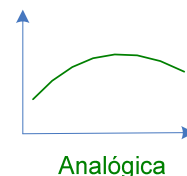
Fundamentos de Computadores

Tema 1: Información digital

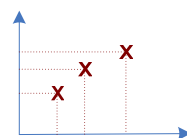
1

1.1- Introducción

- Los computadores son **sistemas electrónicos**, capaces de almacenar, mover y procesar información.
- Los dispositivos electrónicos manejan la información mediante el uso de **señales eléctricas**.
- Las señales eléctricas pueden ser de 2 tipos:
 - Analógicas: pueden tomar cualquier valor dentro de su rango de existencia. Son señales continuas, y entre dos valores dados, pueden tomar todos los valores intermedios.
 - Digitales: dentro de su rango de existencia sólo pueden tomar un conjunto discreto de valores.



Analógica



Digital



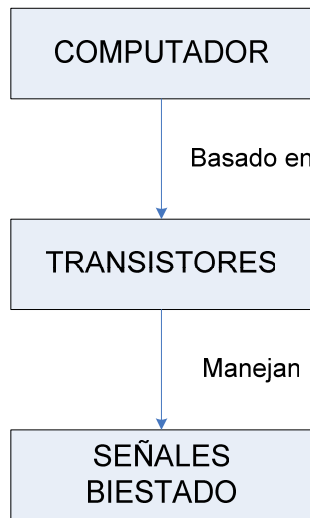
Área de Arquitectura y Tecnología de Computadores
Departamento de Informática de la Universidad de Oviedo

Fundamentos de Computadores

Tema 1: Información digital

2

- Los computadores digitales manejan señales digitales.



- A la información contenida en una **señal digital biestado** se le denomina **BIT** (Binary digiT), y representa la cantidad mínima de información que puede manejar un computador.
- Un bit puede tomar **2 valores**: “0” y “1” (correspondientes a los 2 estados posibles, 0V - 5V).
- Un bit es muy poca información.
Solución: **agrupar bits** para generar unidades de información con mayor capacidad de representación.



Nº de bits	Posibles combinaciones	Unidades de información representables
1	0 1	$2 = 2^1$
2	0 0 0 1 1 0 1 1	$4 = 2^2$
3	0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1	$8 = 2^3$
...		

Ley General: ‘n’ bits \longleftrightarrow 2^n unidades



- Unidades de información comúnmente manejadas:
 - Byte: 8 bits ($\rightarrow 2^8 = 256$ informaciones distintas).
 - Múltiplos del Byte
 - El prefijo kilo- (k) es definido por el Sistema Internacional de Unidades como igual a 1000 (*p.e. kilogramo (kg) = 1000 gramos*).
 - Por tanto kilobyte (kB) deberían ser 1000 bytes.
- Sin embargo en muchos contextos, kB = 1024 bytes. ¿Por qué?
Porque 1024 es potencia de 2 ($1024 = 2^{10}$).



CONFUSIÓN



- El Estándar Internacional crea el término “kibibyte” (kilo binario, KiB) para denotar a la potencia de 2.



Nombre	Potencias de 10 (S.I.)	Difer.	Potencias Binarias	Nombre
kilobyte (kB)	$10^3 = 1000$	2%	$2^{10} = 1024$	kibibyte (KiB)
megabyte (MB)	$10^6 = 1000000$	5%	$2^{20} = 1048576$	mebibyte (MiB)
gigabyte (GB)	$10^9 = 1000000000$	7%	$2^{30} = 1073741824$	gibibyte (GiB)
terabyte (TB)	$10^{12} = 1000000000000$	10%	$2^{40} = 1099511627776$	tebibyte (TiB)
petabyte (PB)	$10^{15} = 1000000000000000$	13%	$2^{50} = 1125899906842624$	pebibyte (PiB)
exabyte (EB)	$10^{18} = 1000000000000000000$	15%	$2^{60} = 1152921504606846976$	exbibyte (EiB)
zettabyte (ZB)	$10^{21} = 1000000000000000000000$	18%	$2^{70} = 1180591620717411303424$	zebibyte (ZiB)
yottabyte (YB)	$10^{24} = 1000000000000000000000000$	21%	$2^{80} = 1208925819614629174706176$	yobibyte (YiB)

- En la actualidad esta convención de nombres ya es empleada por algunos sistemas operativos como GNU/Linux (p.e. la distribución Ubuntu).
- Sin embargo aún son muchos los contextos en los que se sigue utilizando el término *kilobyte* como equivalente de 1024 bytes.



- Concepto de **Código Binario**: es un sistema de representación de objetos mediante secuencias de bits.
- Crear un código binario consiste en asignar una secuencia de bits a cada uno de los elementos que forman parte del conjunto de objetos que se quiere representar.

<u>Secuencia de bits</u>	<u>Objeto representado</u>
Secuencia1	Objeto1
Secuencia2	Objeto2
...	...
SecuenciaN	ObjetoN



- Ejemplo: crear un código binario para representar las letras vocales.
a, e, i, o, u → 5 elementos → ¿Cuántos bits se necesitarán?

2 bits → $2^2 = 4 < 5$ → **No vale**

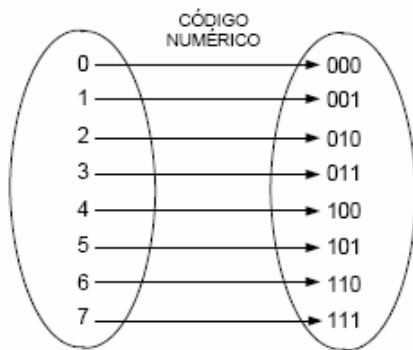
3 bits → $2^3 = 8 > 5$ → **Vale**

<u>Secuencia de bits</u>	<u>Objeto representado</u>
0 0 0	a
0 0 1	e
0 1 0	i
0 1 1	o
1 0 0	u

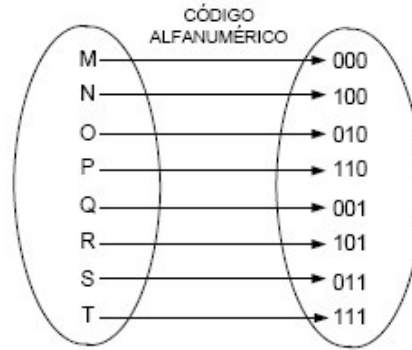


Códigos Binarios

Códigos Numéricos



Códigos Alfanuméricos



- Naturales
- Enteros
- Reales



1.2- Sistemas de numeración

• REPRESENTACIONES NUMÉRICAS

- Las representaciones numéricas se basan en el **sistema posicional**.
- En los sistemas posicionales, el valor que representa un conjunto de dígitos depende de:
 - a) Los propios dígitos.
 - b) Sus posiciones dentro del número.
 - c) La base numérica en la que están representados.

Fórmula General: $valor = \sum_{i=-\infty}^{\infty} d_i B^i = \dots + d_1 B^1 + d_0 B^0 + d_{-1} B^{-1} + \dots$

donde: d_i representa el dígito que ocupa la posición i -ésima
 B representa el valor de la base de representación

Ejemplos (en base 10):

$$1023 = 1 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 = 1000 + 0 + 20 + 3 = 1023$$

$$10'23 = 1 \times 10^1 + 0 \times 10^0 + 2 \times 10^{-1} + 3 \times 10^{-2} = 10 + 0 + 0'2 + 0'03 = 10'23$$



- Representación de números en diferentes bases:

Valor de la base = nº de dígitos disponibles para representar cualquier nº de la base.

<i>BASE</i>	<i>Dígitos disponibles</i>
2 (binaria)	0, 1
8 (octal)	0, 1, 2, 3, 4, 5, 6, 7
10 (decimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
16 (hexadecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Ejemplos:

$$1111_{(2)} = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 4 + 2 + 1 = 15$$

$$13_{(8)} = 1 \times 8^1 + 3 \times 8^0 = 8 + 3 = 11$$



1.2- Sistemas de numeración

<i>DEC</i>	<i>Binario</i>	<i>Octal</i>	<i>Hexadecimal</i>
0	0 0 0 0	0	0
1	0 0 0 1	1	1
2	0 0 1 0	2	2
3	0 0 1 1	3	3
4	0 1 0 0	4	4
5	0 1 0 1	5	5
6	0 1 1 0	6	6
7	0 1 1 1	7	7
8	1 0 0 0	10	8
9	1 0 0 1	11	9
10	1 0 1 0	12	A
11	1 0 1 1	13	B
12	1 1 0 0	14	C
13	1 1 0 1	15	D
14	1 1 1 0	16	E
15	1 1 1 1	17	F



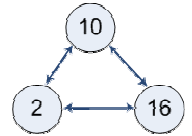
1.2- Sistemas de numeración: conversión entre bases

Bases de uso común {

- Base 10 (decimal), por ser la habitual entre las personas.
- Base 2 (binaria), adecuada para las señales binarias.
- Base 16 (hexadecimal), por ser más compacta que base 2.

Conversión entre bases {

- De base 10 a cualquier base.
- De cualquier base a base 10.
- Conversión directa entre base 2 y base 16 (sin pasar por base 10).



Se verán 3 casos {

- Conversión de números Enteros.
- Conversión de números Fraccionarios puros.
- Conversión de números mixtos.



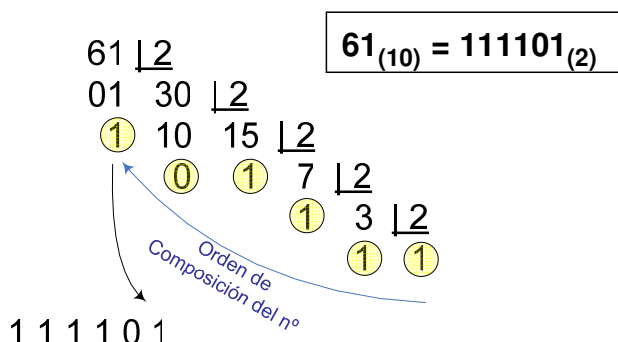
1.2- Sistemas de numeración: conversión entre bases

Caso 1: los números a convertir son enteros.

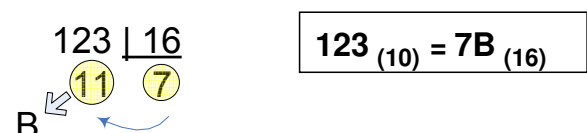
a) Paso de Base 10 a cualquier base:

Se divide el nº a convertir entre el valor de la base destino, hasta que el cociente no sea divisible.

10 → 2 **Pasar 61₍₁₀₎ a Binario**



10 → 16 **Pasar 123₍₁₀₎ a Hexadec.**



b) Paso de cualquier base a Base 10:

Aplicar el sistema posicional sobre el n° que se desea convertir.

$\textcircled{2} \rightarrow \textcircled{10}$ **Pasar $111101_{(2)}$ a Decimal**

$$\begin{aligned} 111101_{(2)} &= 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 \\ &= 32 + 16 + 8 + 4 + 1 \\ &= 61_{(10)} \end{aligned}$$

$\textcircled{16} \rightarrow \textcircled{10}$ **Pasar $7B_{(16)}$ a Decimal**

$$7B_{(16)} = 7 \times 16^1 + B \times 16^0 = 112 + 11 = 123_{(10)}$$



1.2- Sistemas de numeración: conversión entre bases

c) Cambios entre las bases 2 y 16:

B2 \rightarrow B16 Agrupar los dígitos binarios de 4 en 4, empezando por los menos significativos (si es necesario, rellenar con 0s por la izqda).

$\textcircled{2} \rightarrow \textcircled{16}$ **Pasar $111011_{(2)}$ a Hexadec.** \rightarrow Relleno 0011 1011₍₂₎ = **$3B_{(16)}$**

Pasar $11111010001_{(2)}$ a Hexadec. \rightarrow 0111 1101 0001₍₂₎ = **$7D1_{(16)}$**

B16 \rightarrow B2 Expandir cada dígito hexadecimal en 4 dígitos binarios.

$\textcircled{16} \rightarrow \textcircled{2}$ **Pasar $4CE_{(16)}$ a Binario** \rightarrow **$4CE_{(16)} = 0100\ 1100\ 1110_{(2)}$**



Caso 2: los números a convertir son fraccionarios puros.

a) Paso de Base 10 a cualquier base:

Se multiplica el n° a convertir por el valor de la base destino, hasta que la parte fraccionaria sea nula.

10 → 2 **Pasar $0'8125_{(10)}$ a Binario**

$$\begin{array}{lcl} 0'8125 \times 2 & = & 1'625 \\ 0'625 \times 2 & = & 1'25 \\ 0'25 \times 2 & = & 0'5 \\ 0,5 \times 2 & = & 1'0 \end{array} \quad \begin{array}{l} \text{Orden de} \\ \text{Composición} \\ \text{del } n^{\circ} \end{array}$$

$$0'8125_{(10)} = 0'1101_{(2)}$$

10 → 16 **Pasar $0'703125_{(10)}$ a Hex.**

$$\begin{array}{lcl} 0'703125 \times 16 & = & 11'25 \\ 0'25 \times 16 & = & 4'0 \end{array}$$

$$0'703125_{(10)} = 0'B4_{(16)}$$



1.2- Sistemas de numeración: conversión entre bases

b) Paso de cualquier base a Base 10:

Aplicar al n° que se desee convertir el sistema posicional.

2 → 10 **Pasar $0'101101_{(2)}$ a Decimal.**

$$0'101101_{(2)} = 1 \times 2^{-1} + 1 \times 2^{-3} + 1 \times 2^{-4} + 1 \times 2^{-6} = 0'703125_{(10)}$$

16 → 10 **Pasar $0'B4_{(16)}$ a Decimal.**

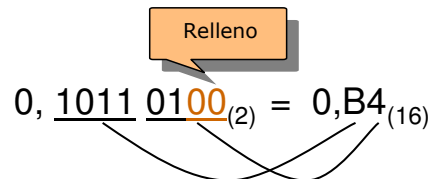
$$0'B4_{(16)} = 11 \times 16^{-1} + 4 \times 16^{-2} = 0'703125_{(10)}$$



c) Cambios entre las bases 2 y 16:

B2 → B16 Agrupar los dígitos hexadecimales de 4 en 4, empezando por el que está situado a la derecha del punto fraccionario (si es necesario, rellenar con 0s por la derecha).

$(2) \rightarrow (16)$ **Pasar $0'101101_{(2)}$ a Hexadecimal**



$$0, \underline{1011} \underline{0100}_{(2)} = 0, \text{B}4_{(16)}$$

B16 → B2 Expandir cada dígito hexadecimal en 4 dígitos binarios.

$(16) \rightarrow (2)$ **Pasar $0'1A9_{(16)}$ a Binario → $0'0001\ 1010\ 1001_{(2)}$**



1.2- Sistemas de numeración: conversión entre bases

Observación:

- Los números Enteros **SÍ** son siempre expresables en cualquier base.
- Los números Fraccionarios Puros **NO** son siempre expresables en cualquier base.

Ejemplos:

Convertir $0'2_{(10)}$ a Base 2

$$\begin{aligned}
 0'2 \times 2 &= 0'4 \\
 0'4 \times 2 &= 0'8 \\
 0'8 \times 2 &= 1'6 \\
 0'6 \times 2 &= 1'2 \\
 0'2 \times 2 &= 0'4 \\
 &\dots
 \end{aligned}$$

$$0'2_{(10)} = 0'00110011..._{(2)}$$

Convertir $0'8_{(10)}$ a Base 16

$$\begin{aligned}
 0'8 \times 16 &= 12'8 \\
 0'8 \times 16 &= 12'8 \\
 &\dots
 \end{aligned}$$

$$0'8_{(10)} = 0'CC..._{(16)}$$



Caso 3: los números a convertir son mixtos.

Operar separadamente con la parte entera y con la parte fraccionaria, y luego combinar el resultado (*aplicar Caso 1 sobre la PE y Caso 2 sobre la PF*).

Ejemplo: expresar el nº $14'375_{(10)}$ en base binaria.

a) Parte entera (*aplicar Caso 1*) → $(10) \rightarrow (2)$

$$\begin{array}{r} 14 \div 2 \\ \hline 0 \end{array} \quad \begin{array}{r} 7 \div 2 \\ \hline 1 \end{array} \quad \begin{array}{r} 3 \div 2 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \div 2 \\ \hline 1 \end{array}$$

b) Parte fraccionaria (*aplicar Caso 2*) → $(10) \rightarrow (2)$

$$\begin{array}{l} 0'375 \times 2 = 0'75 \\ 0'75 \times 2 = 1'5 \\ 0'5 \times 2 = 1'0 \end{array}$$

c) Combinar ambas partes → $14'375_{(10)} = 1110'011_{(2)}$



1.2- Sistemas de numeración: aritmética básica

• ARITMÉTICA BÁSICA (suma y resta) EN DIFERENTES BASES

Ejemplos Suma

	Sin acarreo	Con acarreo
Decimal	$\begin{array}{r} 68 \\ + 21 \\ \hline 89 \end{array}$	$\begin{array}{r} 459 \\ + 378 \\ \hline 837 \end{array}$
Binario	$\begin{array}{r} 0100000 \\ + 00010101 \\ \hline 01010101 \end{array}$	$\begin{array}{r} 111001011 \\ + 101111010 \\ \hline 1101000101 \end{array}$
Hexadecimal	$\begin{array}{r} 40 \\ + 15 \\ \hline 55 \end{array}$	$\begin{array}{r} 1CB \\ + 17A \\ \hline 345 \end{array}$

Ejemplos Resta

	Sin acarreo	Con acarreo
Decimal	$\begin{array}{r} 459 \\ - 318 \\ \hline 141 \end{array}$	$\begin{array}{r} 525 \\ - 387 \\ \hline 138 \end{array}$
Binario	$\begin{array}{r} 10100110 \\ - 00100100 \\ \hline 10000010 \end{array}$	$\begin{array}{r} 01100101 \\ - 01011010 \\ \hline 00001011 \end{array}$
Hexadecimal	$\begin{array}{r} A6 \\ - 24 \\ \hline 82 \end{array}$	$\begin{array}{r} 731C \\ - 1A46 \\ \hline 58D6 \end{array}$

Recordatorio:

A → 10
B → 11

C → 12
D → 13

E → 14
F → 15



1.2- Sistemas de numeración: ejercicios resueltos

Ejercicios:

$$\begin{array}{r} 01011101 \\ + 00101110 \\ \hline 10001011 \end{array}$$

$$\begin{array}{r} 01011101 \\ - 00101110 \\ \hline 00101111 \end{array}$$

$$\begin{array}{r} 731C \\ + 1A46 \\ \hline 8D62 \end{array}$$

$$\begin{array}{r} C3AB \\ + 9EF3 \\ \hline 1629E \end{array}$$

$$\begin{array}{r} A11C \\ - 1A48 \\ \hline 86D4 \end{array}$$



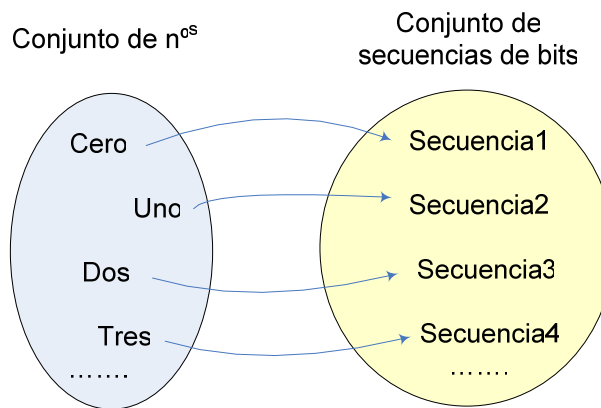
1.2- Sistemas de numeración: ejercicios propuestos

Ejercicios:

Nº a convertir	Base final	Solución
5075 ₍₁₀₎	Base 8	11723 ₍₈₎
3'375 ₍₁₀₎	Base 2	11'011 ₍₂₎
0'59375 ₍₁₀₎	Base 16	0'98 ₍₁₆₎
4'6080 ₍₁₀₎	Base 5	4'301 ₍₅₎
1100 ₍₂₎	Base 10	12 ₍₁₀₎
13 ₍₈₎	Base 10	11 ₍₁₀₎
0'98 ₍₁₆₎	Base 10	0'59375 ₍₁₀₎
1A2B'0C ₍₁₆₎	Base 10	6699'0469 ₍₁₀₎
1101011011000101011 ₍₂₎	Base 16	6B62B ₍₁₆₎
0'10011 ₍₂₎	Base 16	0'98 ₍₁₆₎



Consideración inicial: Concepto de crear un código binario para la representación de los números.



Problema: Conjuntos numéricos infinitos frente a secuencias de bits finitas (el computador tiene un límite de almacenamiento).

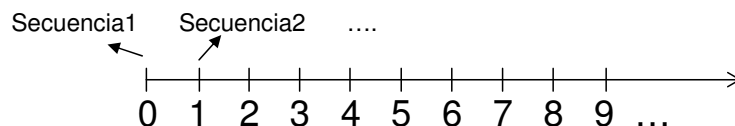


Aparición del concepto de *RANGO*.



1.3- Representaciones numéricas: N^o naturales

Planteamiento:



Pasos para crear el código:

- Determinar el n^o de objetos a representar. (p.e. [0-7])
- Determinar el n^o de bits necesarios para representar dichos objetos.
En general: $n \text{ bits} \leftrightarrow 2^n \text{ elementos}$ (p.e. para representar 8 objetos \rightarrow 3 bits)
- Asignar a cada objeto una secuencia de bits.

El criterio universalmente aceptado es asignar a cada secuencia de bits el n^o natural que representa en base 2. A este formato se le denomina **BINARIO NATURAL**.

En nuestro ejemplo:

0	\rightarrow	000
1	\rightarrow	001
...		
7	\rightarrow	111

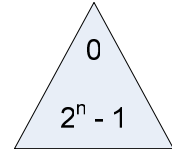


1.3- Representaciones numéricas: N^o naturales

Esquema de representación de este formato numérico:

- **Formato de representación:** **Binario natural (BN).**
- **Rango de representación:** **$[0, 2^n - 1]$** , siendo 'n' el n^o de bits.

Ejemplos: $n = 3 \rightarrow \text{Rango: } [0, 2^3 - 1] = [0, 7]$
 $n = 4 \rightarrow \text{Rango: } [0, 2^4 - 1] = [0, 15]$
 $n = 8 \rightarrow \text{Rango: } [0, 2^8 - 1] = [0, 255]$
 $n = 16 \rightarrow \text{Rango: } [0, 2^{16} - 1] = [0, 65535]$



- **Aritmética:** para sumar dos números naturales, operar directamente en B.Nat.

¡Ojo! Posibilidad de que al sumar dos n^os , cada uno representable con 'n' bits, el resultado de la suma no lo sea \rightarrow Concepto de **desbordamiento aritmético**.

✓		0 0 1 1	(3)		1 0 0 0	(8)	
	+	1 0 0 1	(9)		+	1 0 1 1	(11)
	<hr/>				<hr/>		
		1 1 0 0	(12)		1	0 0 1 1	(3)

Acarreo final \rightarrow (señalado a la celda '1' en la fila anterior)



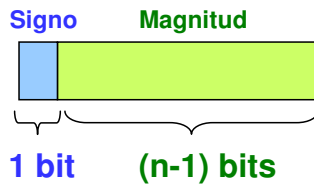
1.4- Representaciones numéricas: N^o enteros

- Los números enteros pueden ser positivos o negativos \rightarrow El signo es un elemento más a tener en cuenta.
- Hay varios tipos de soluciones:
 - Forzar la codificación del signo en un bit extra.
P.e. 0=positivo, 1=negativo (método denominado signo-magnitud).
 - Convertir los negativos en positivos sumándoles un constante Z
(método denominado Exceso a un entero Z).
 - Complemento a 2.
- Objetivo: lograr una representación de positivos y negativos que puedan ser sumados con aritmética de naturales, dando un resultado correcto.
Esto permite usar el mismo hardware para sumar naturales o enteros.



Formato 1: Signo-Magnitud

– Formato de representación (para 'n' bits):



- Número Positivo \rightarrow Bit de signo = 0
- Número Negativo \rightarrow Bit de signo = 1
- La Magnitud se representa en B. Natural

Ej: n = 4 bits

$5_{(10)} \rightarrow 0101$

$-5_{(10)} \rightarrow 1101$

NOTA: En este formato hay una doble representación del 0 (p.e. n=4 bits \rightarrow 0000 y 1000).

– Rango: $[-(2^{n-1} - 1), 2^{n-1} - 1]$, siendo 'n' el n° de bits (\rightarrow Rango Simétrico)

Ej. Si n = 4 bits \rightarrow Rango = $[-(2^3-1), 2^3-1] = [-7, 7]$

– Aritmética: no se utiliza.



1.4- Representaciones numéricas: N° enteros

Formato 2: Complemento a 2 (C-2)

N°s positivos \rightarrow MSB=0

N°s negativos \rightarrow MSB=1

– Formato de representación (para 'n' bits):

a) Números Positivos \rightarrow Se representan en Binario Natural

P.e. si n = 4 bits, representar el $3_{(10)}$ en C-2 $\rightarrow 0011$

b) Números Negativos \rightarrow 3 métodos para obtener su representación:

* Representar en BN el valor $2^n - |x|$, con 'n' = n° de bits, $-x = n^\circ$ a representar

P.e. n = 4 bits, representar el $-5 \rightarrow 2^4 - |-5| = 16 - 5 = 11_{(10)} \rightarrow 1011_{(2)}$

* Representar en BN el valor absoluto del n°.

Invertir todos los bits.

Sumarle 1 al n° obtenido en el paso anterior.

P.e. n = 4 bits, $n^\circ = -5 \rightarrow |-5| = 5_{(10)} \xrightarrow{\text{BN}} 0101_{(2)} \xrightarrow{\text{Invertir}} 1010_{(2)} \xrightarrow{+1} 1011_{(C-2)}$

* Representar en BN el valor absoluto del n° y después, empezando por el bit menos significativo, ir copiando los bits de drcha. a izqda. hasta encontrar el primer '1' (incluido), y a partir de ahí invertir bits.



– Pasar de C2 a decimal

a) Números Positivos → Como Binario Natural

b) Números Negativos → Existen 3 formas:

* Pasar el valor de BN a decimal y se obtiene un valor Y. El valor buscado es $-(2^n - Y)$

P.e. $n = 4$ bits, $1101_{(2)}$ pasado de BN a decimal es $Y=13 \rightarrow 1101_{(2)} = -(16-13) = -3$

* Restar uno en binario, invertir todos los bits. Pasar de BN a decimal para obtener el valor absoluto

P.e. $n = 4$ bits, $1101_{(2)} \rightarrow 1100 \rightarrow 0011 \rightarrow -3$

* Empezando por el bit menos significativo, ir copiando los bits de drcha. a izqda. hasta encontrar el primer '1' (incluido), y a partir de ahí invertir bits. Pasar de BN a decimal para obtener el valor absoluto

P.e. $n = 4$ bits, $1101_{(2)} \rightarrow 0011 \rightarrow -3$



1.4- Representaciones numéricas: N° enteros

– Rango: $[-2^{n-1}, 2^{n-1} - 1]$, siendo 'n' el n° de bits.

Ej. Si $n = 4$ bits \rightarrow Rango = $[-2^3, 2^3-1] = [-8, 7]$

¡Ojo! La secuencia binaria $1000_{(C-2)}$ representa el valor -8, no el 8 ($n=4$ bits).

– Aritmética en C-2:

Propiedad:

“Dadas dos cantidades de cualquier signo representadas en C-2 en un formato de 'n' bits, las operaciones aritméticas simples (como la suma) de dichas cantidades, realizada según los mecanismos de los n°s binarios naturales, genera un resultado correcto con signo siempre y cuando dicho resultado pertenezca al rango de los n°s representables con el 'n' dado”.

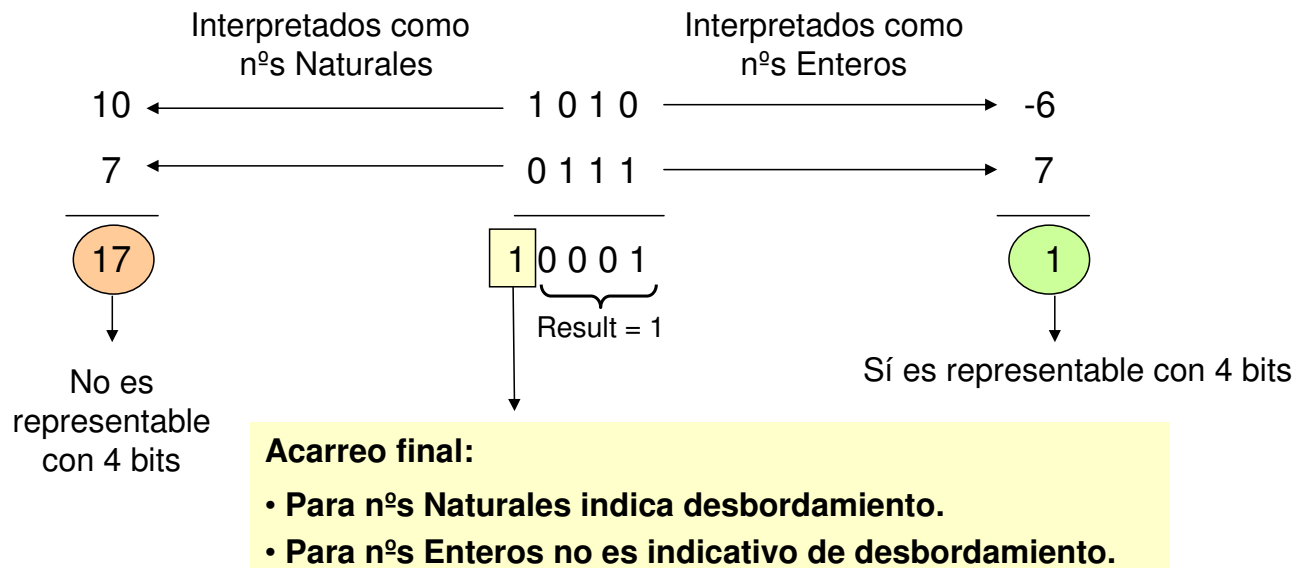
Cuando se realizan operaciones aritméticas con dos números representados con signo en un formato de 'n' bits, y el resultado no es representable con 'n' bits, entonces se produce un **desbordamiento aritmético**.

$$\begin{array}{r} -7 \rightarrow 1001_{(C-2)} \\ + \quad 5 \rightarrow 0101_{(C-2)} \\ \hline (-2) \quad 1110_{(C-2)} \rightarrow -2 \quad \checkmark \end{array}$$

$$\begin{array}{r} 7 \rightarrow 0111_{(C-2)} \\ + \quad 5 \rightarrow 0101_{(C-2)} \\ \hline (12) \quad 1100_{(C-2)} \rightarrow -4 \quad \times \end{array}$$



– Detección de desbordamiento aritmético para N° Enteros



Conclusión: el mecanismo para detectar desbordamiento en los en los n°s Naturales NO sirve para los n°s Enteros.



1.4- Representaciones numéricas: N° enteros

Para detectar el desbordamiento aritmético de n°s Enteros se plantea un mecanismo basado en el contrastar el signo de los operandos con el signo del resultado.

Operación	Signo operando A	Signo Operando B	Signo Resultado	Desbordamiento
SUMA	0	0	0	NO
SUMA	0	0	1	SÍ
SUMA	0	1	0	NO
SUMA	0	1	1	NO
SUMA	1	0	0	NO
SUMA	1	0	1	NO
SUMA	1	1	0	SÍ
SUMA	1	1	1	NO

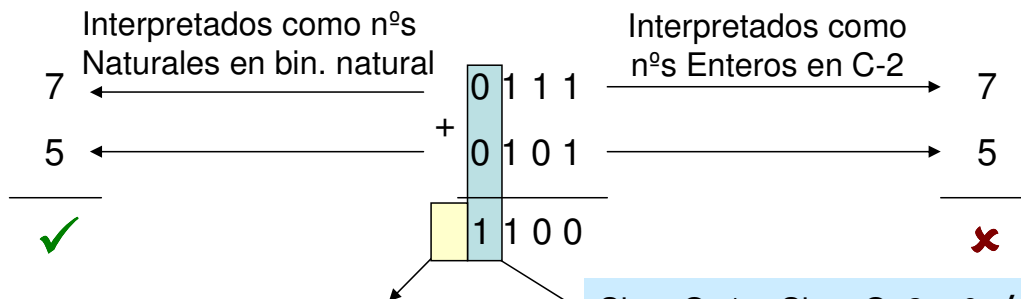
Observación: Cuando los operandos son de distinto signo, su suma nunca producirá desbordamiento.



1.4- Representaciones numéricas: comparativa

Ejemplo de detección de desbordamiento aritmético para n°s interpretados CON y SIN signo

Sup. $n = 4$ bits \rightarrow Rango de Naturales: $[0, 15]$
Rango de Enteros C-2: $[-8, 7]$



No existe Acarreo Final \rightarrow No existe desbordamiento en el caso de interpretar los como n°s Naturales \rightarrow El resultado obtenido es representable con 4 bits.

SignoOp1 = SignoOp2 = 0 \neq 1 = SignoR

En caso de interpretar los n°s como Enteros, Sí existe desbordamiento \rightarrow El resultado obtenido NO es representable con 4 bits.

Comprobación: $1100_{(2)} = 12_{(10)} = 7+5$

Comprobación: $1100_{(C-2)} = -4_{(10)} \neq 7+5$



Área de Arquitectura y Tecnología de Computadores
Departamento de Informática de la Universidad de Oviedo

Fundamentos de Computadores

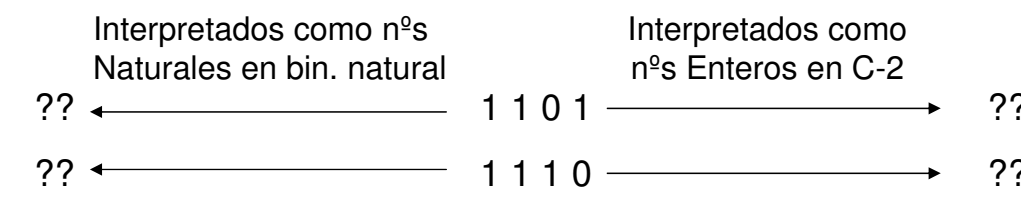
Tema 1: Información digital

35

1.4- Representaciones numéricas: ejercicio

Ejercicio propuesto

Sup. $n = 4$ bits \rightarrow Rango de Naturales: $[0, 15]$
Rango de Enteros C-2: $[-8, 7]$



¿Existe desbordamiento aritmético si se interpretan como magnitudes sin signo?

¿Existe desbordamiento aritmético si se interpretan como magnitudes con signo?



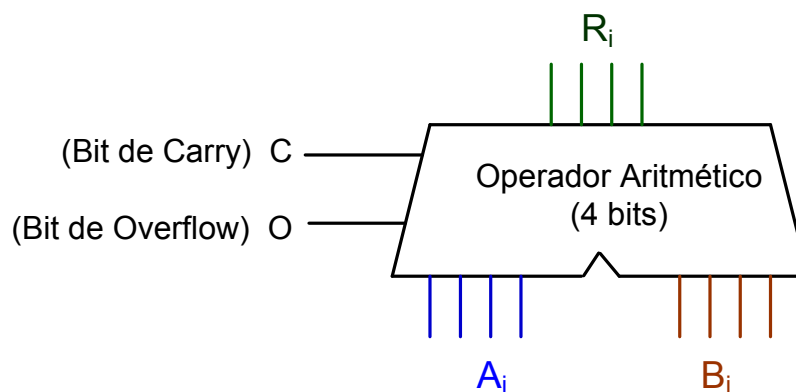
Área de Arquitectura y Tecnología de Computadores
Departamento de Informática de la Universidad de Oviedo

Fundamentos de Computadores

Tema 1: Información digital

36

Conceptos de BIT DE CARRY y BIT DE OVERFLOW



- Bit de Carry (CF): bit generado por un operador aritmético para indicar el desbordamiento cuando sus operandos de entrada (A_i y B_i) se interpretan como magnitudes sin signo (\rightarrow n° s Naturales).
- Bit de Overflow (OF): bit generado por un operador aritmético para indicar el desbordamiento cuando sus operandos de entrada (A_i y B_i) se interpretan como magnitudes con signo (\rightarrow n° s Enteros).



1.4- Representaciones numéricas: N° enteros

Formato 3: Exceso a un entero Z

- **Formato de representación:** Sumar el Exceso Z el n° a representar, y luego representar el resultado obtenido en B. Natural.

Ejemplos: Sea $n = 4$ bits. Representar los n° s 5 y -5 en EXC-8.

$$5 \xrightarrow{\text{Exc-8}} 5 + 8 = 13_{(10)} \xrightarrow{\text{B.N.}} 1101_{(2)} \quad \left| \quad -5 \xrightarrow{\text{Exc-8}} -5 + 8 = 3_{(10)} \xrightarrow{\text{B.N.}} 0011_{(2)}$$

Observación:

Para que un n° sea representable en Exceso a un entero Z con un n° de bits 'n' dado, deben cumplirse 2 condiciones:

- Después de sumarle el exceso al n° , el resultado NO puede ser negativo.
Ej. dados $n=4$ bits y EXC-5 \rightarrow Representar el -6: $-6 \rightarrow -6+5 = -1 \rightarrow$ No cumple (a)
- Después de sumarle el exceso al n° , el resultado debe ser representable en B.N. para el 'n' dado.
Ej. dados $n=4$ bits y EXC-5 \rightarrow Representar el 12: $12 \rightarrow 12+5 = 17 \rightarrow$ No cumple (b)



1.4- Representaciones numéricas: N° enteros

– **Rango de representación:** $[-Z, 2^n - 1 - Z]$, siendo 'n' el n° de bits.

Ejemplos: $n = 5$, EXC-16 \rightarrow Rango: $[-16, 2^5 - 1 - 16] = [-16, 15]$

$n = 4$, EXC-4 \rightarrow Rango: $[-4, 2^4 - 1 - 4] = [-4, 11]$

$n = 2$, EXC-7 \rightarrow Rango: $[-7, 2^2 - 1 - 7] = [-7, -4]$

NOTA:

* Si $Z = 2^{n-1} \rightarrow$ Se denomina Exceso-Central, y el rango coincidirá con el de C-2.

* Si $Z < 2^{n-1} \rightarrow$ Se podrán representar más n° s Positivos que Negativos.

* Si $Z > 2^{n-1} \rightarrow$ Se podrán representar más n° s Negativos que Positivos.

– **Consideraciones sobre Aritmética:**

Propiedad:

“Los n° s representados en Exceso-Z se ordenan de menor a mayor, de modo que al más pequeño de todos (máximo negativo) se le asigna el n° binario menor (00...0), y al más grande (máximo positivo) el n° binario mayor (11...1)”.

Esta propiedad facilita la comparación de n° s enteros.



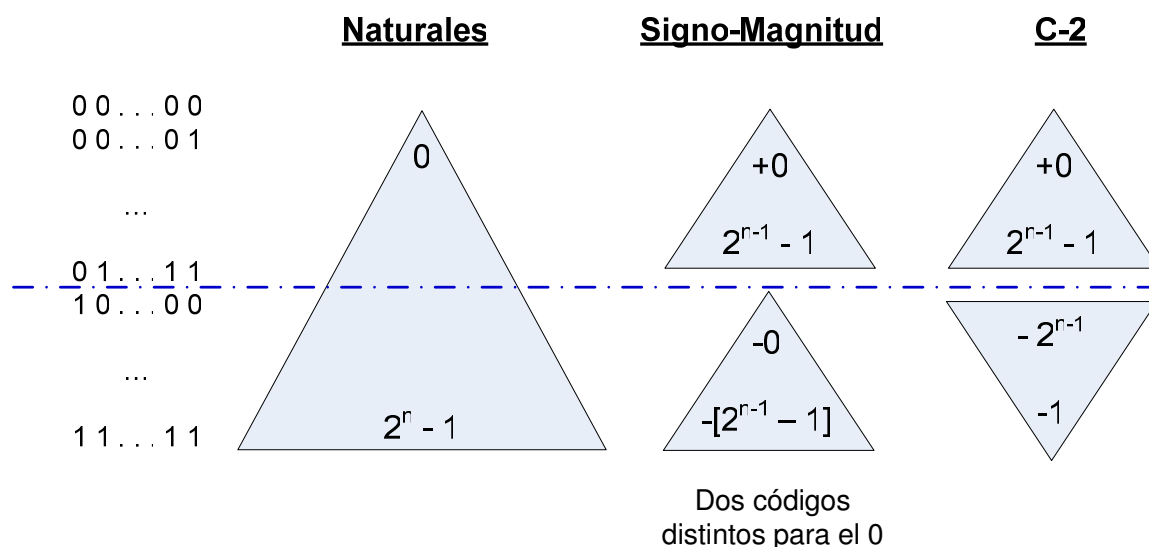
1.4- Representaciones numéricas: N° enteros

Binario	Natural	Signo-Magnitud	C-2	Exc-8	
0 0 0 0	0	0	0	-8	EXC-8 es el exceso-central para $n=4$ bits
0 0 0 1	1	1	1	-7	
0 0 1 0	2	2	2	-6	Se puede ver que el rango del EXC-8 y el del C-2 efectivamente coinciden [-8, 7]
0 0 1 1	3	3	3	-5	
0 1 0 0	4	4	4	-4	
0 1 0 1	5	5	5	-3	
0 1 1 0	6	6	6	-2	
0 1 1 1	7	7	7	-1	
1 0 0 0	8	-0	-8	0	
1 0 0 1	9	-1	-7	1	
1 0 1 0	10	-2	-6	2	
1 0 1 1	11	-3	-5	3	
1 1 0 0	12	-4	-4	4	
1 1 0 1	13	-5	-3	5	
1 1 1 0	14	-6	-2	6	
1 1 1 1	15	-7	-1	7	

Comentario: ver cómo una misma secuencia de bits puede representar cosas distintas dependiendo del formato de representación.



Esquema Resumen de los Rangos de Representación



1.4- Representaciones numéricas: ejercicios

Ejercicios

- ① En un sumador para cantidades de 5 bits se introducen el número 7 codificado en Exceso a 16 y el número -16 codificado en C-2. ¿Qué resultado se obtendrá a la salida del sumador interpretado en Exceso a 8? Contestar en decimal.

7 codificado en EXC-16 $\rightarrow 7 + 16 = 23$ Codificar en BN $\rightarrow 1\ 0\ 1\ 1\ 1$
 -16 codificado en C-2 \rightarrow Codificar el 16 $\rightarrow 1\ 0\ 0\ 0\ 0$
 Invertir a partir del 1^{er} '1': $1\ 0\ 0\ 0\ 0 \rightarrow 1\ 0\ 0\ 0\ 0$
1 0 0 1 1 1

El resultado obtenido es 7 \rightarrow Si está codificado en EXC-8 entonces: $x + 8 = 7 \rightarrow x = -1$

- ② En un sumador para cantidades de 'n' bits, con $n=4$, se introducen el mayor n° entero positivo y el menor n° entero negativo, ambos codificados en C-2. ¿Qué resultado se obtendrá a la salida del sumador interpretado como un n° entero codificado en Exceso a 2^{n-2} ? Contestar en decimal.

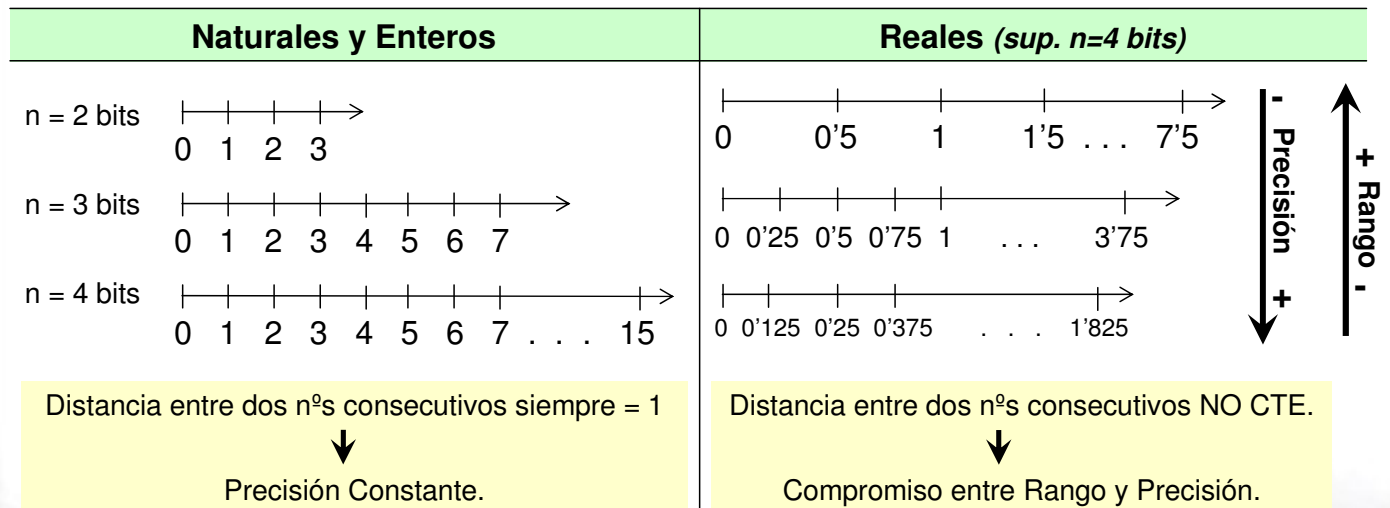
(Soluc: 1)



Consideración inicial: Conceptos de RANGO y PRECISIÓN.

Independientemente del método que se utilice para representar n°s fraccionarios, es evidente que nunca podrán representarse todos.

- Si la P_E es demasiado grande → podrán aparecer problemas de RANGO.
- Si la P_F tiene demasiados dígitos → aparecerán problemas de PRECISIÓN.



1.5- Representaciones numéricas: N° reales

- Como se acaba de explicar, son dos los conceptos a tener en cuenta: RANGO y PRECISIÓN.

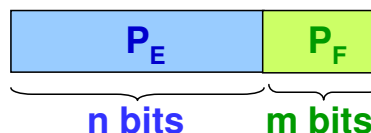
- Se verán 2 tipos de formato:
 - Coma fija
 - Coma flotante
 - a) Formato Simplificado
 - b) Formato IEEE-754 simple



Consiste en almacenar separadamente los bits que hay a la izquierda del punto fraccionario (P_E) y los que hay a la derecha del punto fraccionario (P_F).

– Formato de representación:

Dado un n° determinado de bits, $n+m$, se toman ' n ' bits para representar la P_E y ' m ' para representar la P_F .



El formato de coma fija puede utilizarse para representar n° s CON o SIN signo.

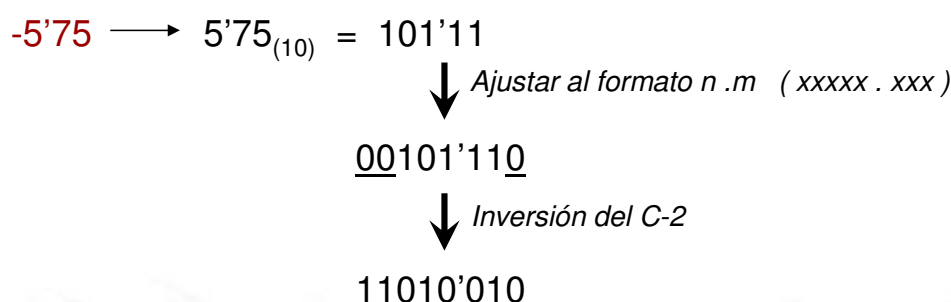
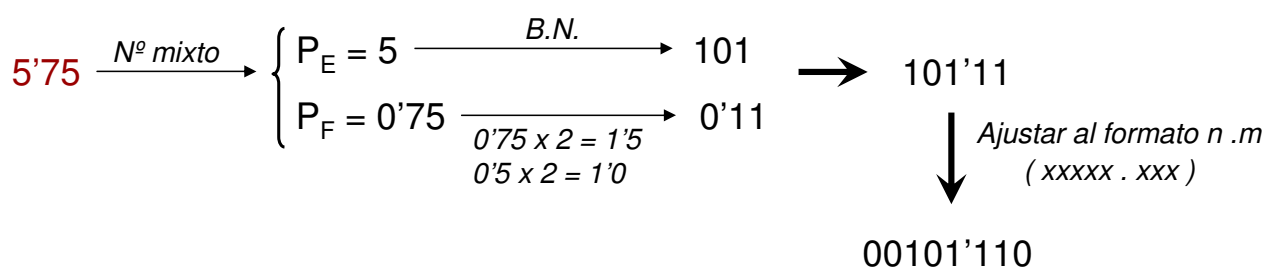
- ↗ N° s sin signo: utilizar el criterio de representación del sistema binario.
- ↘ N° s con signo: seguir el criterio de representación del C-2.



1.5- Representaciones numéricas: Coma Fija

Ejemplo:

Sup. $n=5$ bits y $m=3$ bits, representar $5'75_{(10)}$ y $-5'75_{(10)}$ en formato coma fija.



¡Ojo! No invertir los bits antes de haber ajustado al formato $n.m$



– Rango y Precisión:

Dado un n° determinado de bits ' $n+m$ ' (' n ' bits para la P_E y ' m ' para P_F), el rango y precisión del formato en coma fija depende del reparto de bits que se haga entre la P_E y la P_F .

Si $n \uparrow$ y $m \downarrow \longrightarrow$ Rango \uparrow y Precisión \downarrow

Si $n \downarrow$ y $m \uparrow \longrightarrow$ Rango \downarrow y Precisión \uparrow

El rango también depende de si se trabaja con magnitudes sin signo (sólo n° s positivos) o con magnitudes con signo (n° s positivos y negativos).



1.5- Representaciones numéricas: Coma Fija

	Magnitudes SIN signo	Magnitudes CON signo
RANGO:	$[0, 2^n - 2^{-m}]$	$[-2^{n-1}, 2^{n-1} - 2^{-m}]$
PRECISIÓN:	2^{-m}	2^{-m}

En coma fija, la **precisión es cte.** a lo largo de todo el rango de representación, y representa por tanto el máximo error de representación que se puede cometer, independientemente del n° a representar.

Ejemplo:

Sup. $n=8$ bits y $m=4$ bits, calcular la precisión del formato coma fija.

Precisión: $2^{-m} = 2^{-4} = 1/2^4 = 0'0625 \rightarrow$ Se corresponde además con el n° más pequeño representable con 8+4 bits: 00000000'0001

En este ejemplo, la precisión conseguida es del orden de las décimas.



1.5- Representaciones numéricas: Coma Fija

Deducción matemática de los Rangos del formato coma fija (p.e. para $n=5$ y $m=3$).

Magnitudes SIN signo → Rango: $[0, 2^n - 2^{-m}] \rightarrow [0, 2^5 - 2^{-3}]$

Nº más pequeño representable: 00000'000 = 0

Nº más grande representable: 11111'111 = $2^5 - 2^{-3}$

Aprender a deducirlo

$$\begin{array}{r} 11111'111 \ (\rightarrow =x) \\ + 00000'001 \ (\rightarrow =2^{-3}) \\ \hline 100000'000 \ (\rightarrow =2^5) \end{array} \longrightarrow x + 2^{-3} = 2^5 \rightarrow x = 2^5 - 2^{-3}$$

Magnitudes CON signo → Rango: $[-2^{n-1}, 2^{n-1} - 2^{-m}] \rightarrow [-2^4, 2^4 - 2^{-3}]$

Nº más pequeño representable: 10000'000 = -16 = -2^4

Nº más grande representable: 01111'111 = $2^4 - 2^{-3}$

Aprender a deducirlo

$$\begin{array}{r} 01111'111 \ (\rightarrow =x) \\ + 00000'001 \ (\rightarrow =2^{-3}) \\ \hline 10000'000 \ (\rightarrow =2^4) \end{array} \longrightarrow x + 2^{-3} = 2^4 \rightarrow x = 2^4 - 2^{-3}$$



1.5- Representaciones numéricas: Coma Fija

– Aritmética:

En el caso de nºs sin signo, es análoga a la explicada para los nºs Naturales.
En el caso de nºs con signo, es análoga a la vista en C-2.

Para detectar el desbordamiento aritmético se utiliza por tanto:

- El bit de Carry (CF) si el nº se interpreta como una magnitud sin signo.
- El bit de Overflow (OF) si el nº se interpreta como una magnitud con signo.

CONCLUSIÓN: los operadores requeridos para la aritmética en Coma Fija son idénticos a los requeridos para la aritmética de Naturales o Enteros.

	Interpretados como magnitudes sin signo		Interpretados como magnitudes con signo
7'625	←	0 0 1 1 1 ' 1 0 1	→ 7'625
28'625	←	1 1 1 0 0 ' 1 0 1	→ -3'375
<u> </u>		<u> </u>	
x		1 0 0 1 0 0 ' 0 1 0	✓
		Resultado = 4'25	

Hay Carry → Desbordamiento en el caso de interpretarlos como magnitudes sin signo → El resultado obtenido NO es representable.

No hay Overflow → No hay desbordamiento en caso de interpretarlos como Magnitudes con signo → Resultado representable.



Ejercicios

- ❶ Se dispone de 1 byte para representar n^{os} reales según un formato de **coma fija**. Los primeros 6 bits se utilizan para la parte entera y los 2 últimos para la parte fraccionaria. Los n^{os} negativos se representan en complemento a 2. Se pide:
 - a) ¿Cuál será el resultado de sumar las cantidades 12 y -2'63 expresadas en este formato? Codifica el resultado en este mismo formato y escribe la solución en hexadecimal. *(Soluc: 26₍₁₆₎)*
 - b) ¿Cuál es el n^o negativo más cercano a 0 que puede representarse en este formato? Responde en decimal. *(Soluc: -0'25)*

- ❷ Se tienen los n^{os} -8 (expresado en **complemento a 2**) y -1 (representado en un **exceso a Z**) ambos con 4 bits. Se suman ambos usando aritmética de complemento a 2 e, interpretando el resultado también en complemento a 2, se obtiene -1 en decimal y no se produce desbordamiento. ¿Cuánto valía Z? *(Soluc: 8)*



1.5- Representaciones numéricas: Coma Flotante

- Formato: $R = M \times B^e$, donde *M*: mantisa, *B*: base, *e*: exponente
- Ejemplos:

$15'25 = 15'25 \times 10^0$	
$15'25 = 1'525 \times 10^1$	$15'25 = 152'5 \times 10^{-1}$
$15'25 = 0'1525 \times 10^2$	$15'25 = 1525 \times 10^{-2}$
$15'25 = 0'01525 \times 10^3$	$15'25 = 15250 \times 10^{-3}$
...	

- Formatos Normalizados:
 - **Todo Fracción (TF)**: $R = 0'D_{-1}... \times B^e$, con D_{-1} no nulo
El dígito más significativo de la mantisa (D_{-1}) se sitúa justo a la derecha del punto fraccionario y luego se ajusta el exponente.

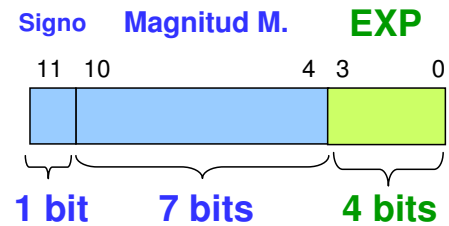
 - **Todo Entero (TE)**: $R = ...D_0'0 \times B^e$, con D_0 no nulo.
El dígito menos significativo de la mantisa (D_0) se sitúa justo a la izquierda del punto fraccionario y luego se ajusta el exponente.



1.5- Rep. numéricas: Coma flotante – Formato simplificado

i) **FORMATO:**

- Base de representación → **BINARIA**
- Normalización → **Todo Fracción**
- Mantisa → **Formato: Signo-Magnitud**
Tamaño: 8 bits (→ 1 + 7)
- Exponente → **Formato: Exceso a 8**
Tamaño: 4 bits (→ [-8,7]).
- El 0 se representa con la secuencia todo 0s.
- Error cometido: **|Nº a representar – Nº representado|**



Ejemplo: Representar $-13'625_{(10)}$ en formato simplificado.

$$13'625 \xrightarrow{\text{Convertir a binario}} 1101'101 \xrightarrow{\text{Normalizar TF}} 0'1101101 \times 2^4 \quad \begin{matrix} \text{4+8=12} \\ \downarrow \end{matrix}$$

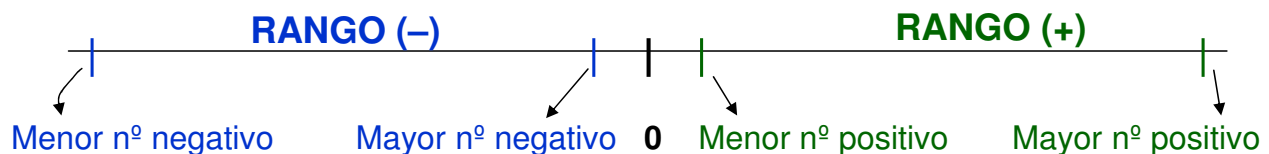
1 1101101 1100



1.5- Rep. numéricas: Coma flotante – Formato simplificado

ii) **RANGO:**

La representación de la mantisa en SIG-MAG hace que el conjunto de números representables positivos y negativos sea idéntico.



- Análisis del Rango Positivo (+):

$$\left. \begin{array}{l} \text{* Máximo nº} \rightarrow \text{Máx. mantisa: } 0'1111111 \\ \text{Máx. exponente: } 7 \end{array} \right\} \rightarrow 0'1111111 \times 2^7 = (1 - 2^{-7}) \times 2^7 = 2^7 - 1 = 127$$

$$\left. \begin{array}{l} \text{* Mínimo nº} \rightarrow \text{Mín. mantisa: } 0'1000000^{(*)} \\ \text{Mín. exponente: } -8 \end{array} \right\} \rightarrow 0'1 \times 2^{-8} = 2^{-9}$$

(*) $0'1000000$ (vs. $0'0000001$).

Sería el mismo valor aunque \uparrow nº de bits de la Mantisa.

$$\text{RANGO (+): } [2^{-9}, 2^7 - 1]$$



1.5- Rep. numéricas: Coma flotante – Formato simplificado

iii) **PRECISIÓN:**

- Análisis de n^{os} grandes:

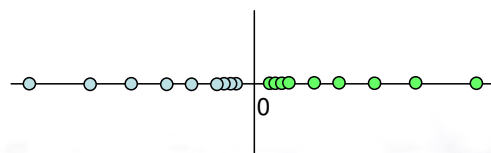
$$\left. \begin{array}{l} * \text{Máximo } n^o (+) \rightarrow 0'11111111 \times 2^7 = 127_{(10)} \\ * \text{Inmediato inferior } (+) \rightarrow 0'11111110 \times 2^7 = 126_{(10)} \end{array} \right\} \text{Dist} = 1$$

- Análisis de n^{os} pequeños:

$$\left. \begin{array}{l} * \text{Mínimo } n^o (+) \rightarrow 0'10000000 \times 2^{-8} = 2^{-9}_{(10)} \\ \text{(excluyendo el 0)} \\ * \text{Inmediato superior } (+) \rightarrow 0'10000001 \times 2^{-8} = (2^{-1} + 2^{-7}) \times 2^{-8} \\ = 2^{-9} + 2^{-15} \end{array} \right\} \text{Dist} = 2^{-15}$$

CONCLUSIÓN: la precisión varía en función del valor del exponente.

Exponente \uparrow \rightarrow Precisión \downarrow
Exponente \downarrow \rightarrow Precisión \uparrow



Dist: distancia entre dos n^{os} consecutivos.



Área de Arquitectura y Tecnología de Computadores
Departamento de Informática de la Universidad de Oviedo

Fundamentos de Computadores
Tema 1: Información digital

55

1.5- Rep. numéricas: Coma flotante – Formato simplificado

iv) **ARITMÉTICA:** (mecanismo de suma en coma flotante simplificado)

- Paso 1: representación normalizada de los operandos (TF).
- Paso 2 : igualación de exponentes
(igualar el menor exponente con el mayor, ajustando mantisas).
- Paso 3: sumar los dos operandos (ya con el mismo exponente).
- Paso 4: normalizar el resultado (TF).

Ejemplo: Sumar $12'75_{(10)}$ y $0'25_{(10)}$ en formato simplificado

$$\begin{array}{lcl} \textcircled{1} & 12'75 & \xrightarrow{\text{Convertir a binario}} 1100'11 \xrightarrow{\text{Normalizar TF}} 0'110011 \times 2^4 \\ & 0'25 & \xrightarrow{\hspace{1.5cm}} 0'01 \xrightarrow{\hspace{1.5cm}} 0'1 \times 2^{-1} \\ \\ \textcircled{2} & 0'110011 \times 2^4 & \\ & 0'1 \times 2^{-1} & \xrightarrow{\text{Igualar -1 a 4}} 0'000001 \times 2^4 \end{array} \left. \vphantom{\begin{array}{l} \textcircled{2} \end{array}} \right\} \begin{array}{l} \textcircled{3} \\ \hline 0'110011 \times 2^4 \\ 0'000001 \times 2^4 \\ \hline 0'110100 \times 2^4 \end{array}$$

$\textcircled{4}$ El resultado ya está normalizado: $0'110100 \times 2^4$



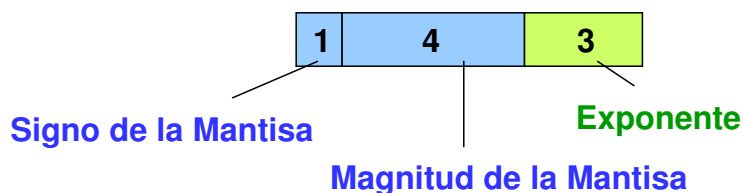
Área de Arquitectura y Tecnología de Computadores
Departamento de Informática de la Universidad de Oviedo

Fundamentos de Computadores
Tema 1: Información digital

56

1.5- Rep. numéricas: Coma Flotante – Ejercicios

- ❶ Supóngase un formato de **coma flotante** en el que la mantisa se representa en formato normalizado todo fracción y el exponente se representa en exceso a 4. El cero se representa con la secuencia todo ceros.



Se pide:

- a) Representar el $n^{\circ} 0'4_{(10)}$ (Soluc: 0 1 1 0 0 0 1 1)
- b) ¿Cuántos n° s se pueden representar con este formato? (Soluc: $2^7+1=128+1$)
- c) ¿Cuál es el error cometido al representar el $n^{\circ} 0'4$? (Soluc: 0'025)
- d) ¿Cuál es el mayor n° representable en este formato? (Soluc: $0'1111 \times 2^3 = 7'5$)
- e) ¿Cuál será el salto mínimo entre dos n° s positivos sucesivos en esta representación? (contestar el forma de potencias de 2). (Soluc: 2^{-8})



1.5- Rep. numéricas: Coma flotante – Ejercicios

- ❷ Representar en formato de coma flotante simplificado los siguientes números:

- a) $0'023_{(10)}$ (Soluc: 0 1 0 1 1 1 1 0 0 0 1 1)
- b) $130'21_{(10)}$ (Soluc: Overflow)
- c) $0'0015_{(10)}$ (Soluc: Underflow)

- ❸ Representar en formato simplificado el número $25'625_{(10)}$ y, si procede, calcular el error cometido en la representación, expresándolo tanto como valor decimal como en potencias de 2.

Soluc: 0 1 1 0 0 1 1 0 1 1 0 1

Error cometido: 0'125 (expresado como valor decimal).

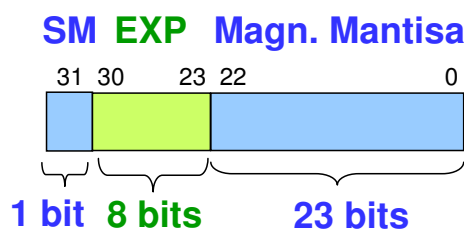
$2^{-8} \times 2^{-5} = 2^{-3}$ (expresado en potencias de 2).



1.5- Rep. numéricas: Coma flotante – Formato IEEE-754 Simple

i) FORMATO DEL ESTÁNDAR SIMPLE:

- Base de representación → **BINARIA**
- Mantisa → *Formato: Signo-Magnitud*
Tamaño: **24 bits** (→ 1 + 23)
- Exponente → *Formato: Exceso a 127*
Tamaño: **8 bits** (→ [-127,128]).
- Dos subformatos → **Normalizado**
Desnormalizado (corresponde a los exponentes 00000000 y 11111111)



Exponentes:

- * El -127 en EXC a 127 sería: $-127 + 127 = 0 \rightarrow 00000000$
 - * El 128 en EXC a 127 sería: $128 + 127 = 255 \rightarrow 11111111$
- Se reservan para representaciones desnormalizadas**

Por tanto, los exponentes disponibles para el formato normalizado son: [-126, 127]

- * El -126 en EXC a 127 sería: $-126 + 127 = 1 \rightarrow 00000001$
 - ...
 - * El 127 en EXC a 127 sería: $127 + 127 = 254 \rightarrow 11111110$
- Rango de exponentes para representaciones normalizadas**

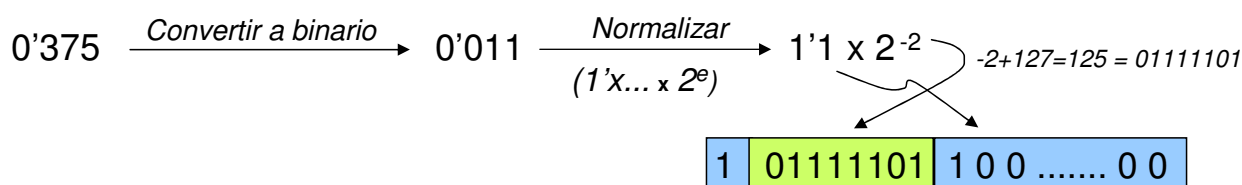


1.5- Rep. numéricas: Coma flotante – Formato IEEE-754 Simple

* Subformato Normalizado: $R = \pm 1'xxx \dots x 2^e$

Bit implícito u oculto
(no se representa)

Ejemplo: Representar $-0'375_{(10)}$ en formato IEEE-754 Simple.



Ejercicio Propuesto: Representar $13'625_{(10)}$ en formato IEEE-754 Simple.
(SOLUCIÓN: 0 10000010 1011010...0)



1.5- Rep. numéricas: Coma flotante – Formato IEEE-754 Simple

* **Subformato *Desnormalizado*:** (exponentes 00000000 y 11111111)

Exponente 00000000	{	Mantisa todo 0s	→ 0
		Mantisa distinta de 0	→ $R = 0'M \times 2^{-126}$ (M no tiene ninguna restricción de normalización)
Exponente 11111111	{	Mantisa todo 0s	→ { Si signo (+) → $+\infty$ Si signo (-) → $-\infty$
		Mantisa distinta de 0	→ Representación de errores (NaN).

Ejemplo: ¿Qué nº representa la secuencia 0 00000000 10....0 si está en formato IEEE-754 simple?

0 (00000000) 10...0

Indica formato desnormalizado → Comprobar Mantisa: todo 0s? NO →

$$R = 0'M \times 2^{-126} = 0'10...0 \times 2^{-126} = 2^{-127}$$

1.5- Rep. numéricas: Coma flotante – Formato IEEE-754 Simple

ii) **RANGO (simétrico):**

- Subformato Normalizado (+):

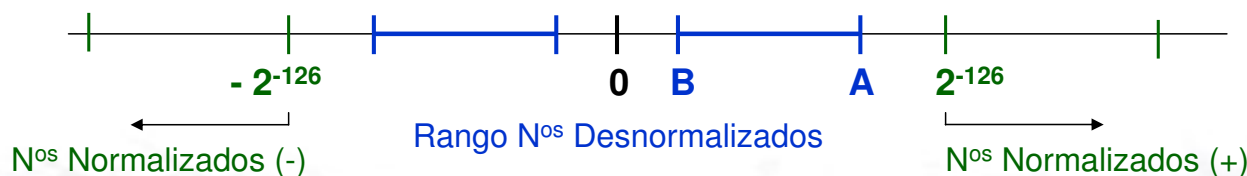
$$\begin{aligned} & \left. \begin{array}{l} * \text{ M\u00e1ximo n}^\circ \rightarrow \text{M\u00e1x. mantisa: } 1'1 \dots 1 \\ \text{M\u00e1x. exponente: } 127 \end{array} \right\} \rightarrow 1'1 \dots 1 \times 2^{127} = (2 - 2^{-23}) \times 2^{127} \\ & \quad \quad \quad (\equiv 0'1 \dots 1 \times 2^{128} = (1 - 2^{-24}) \times 2^{128}) \\ & \left. \begin{array}{l} * \text{ M\u00ednimo n}^\circ \rightarrow \text{M\u00edn. mantisa: } 1'0 \dots 0 \\ \text{M\u00edn. exponente: } -126 \end{array} \right\} \rightarrow 1'0 \dots 0 \times 2^{-126} = 2^{-126} \end{aligned}$$

RANGO (+): $[2^{-126}, (2 - 2^{-23}) \times 2^{127}]$

- Subformato Desnormalizado:

* Máximo n^o (+) $\rightarrow 0'1 \dots 1 \times 2^{-126} = (1 - 2^{-23}) \times 2^{-126} = 2^{-126} - 2^{-149}$ (A)

* Mínimo n^o (+) $\rightarrow 0'0 \dots 01 \times 2^{-126} = 2^{-23} \times 2^{-126} = 2^{-149}$ (B)



1.5- Rep. numéricas: Coma flotante – Formato IEEE-754 Simple

- iii) **PRECISIÓN:** Consideraciones análogas a las vistas en el formato de coma flotante simplificado.
- iv) **Consideración sobre la ARITMÉTICA:** Los computadores actuales ya incluyen HW especializado para operar en coma flotante.

**Cuadro Resumen
IEEE-754 simple**

		Mantisa		
		- 0	+ 0	≠ 0
Exponente	00000000	0	0	$\pm 0'M \times 2^{-126}$
	00000001	$- 1'0 \times 2^{\text{exp}-127}$	$+ 1'0 \times 2^{\text{exp}-127}$	$\pm 1'M \times 2^{\text{exp}-127}$
	...			
	11111110			
	11111111	- ∞	+ ∞	NaN

Bit oculto
 $\pm 1'M \times 2^e$,
 con $e = \text{exp}-127$



1.5- Rep. numéricas: Coma flotante – Ejercicios propuestos

1) ¿Qué número representa cada una de las siguientes secuencias binarias sabiendo que están expresadas en formato IEEE-754 simple?:

- a) 0 0 0 0 0 0 0 0 0 0 0 (Soluc: 0)
- b) 0 0 1 1 1 1 1 1 1 0 0 0 (Soluc: $1'0 \times 2^0 = 1$)
- c) 0 0 0 0 0 0 0 0 1 0 0 (Soluc: 2^{-127})
- d) 1 0 0 0 0 0 0 0 1 1 0 0 (Soluc: $-1'1 \times 2^{-126}$)
- e) 1 1 0 0 0 0 0 0 1 0 0 0 (Soluc: $-1'0 \times 2^2 = -4$)

2) La cantidad C 1 C 0 0 0 0 0 representa un nº real expresado en formato IEEE-754. ¿Con que nº decimal se corresponde? (Soluc: -24)



- Problema: intercambio de información entre distintos dispositivos (computadores, periféricos, ...)
- Solución: Utilización de un estándar para la representación de la información.
- Estándares para codificación de caracteres: ASCII, ASCII extendido, ISO 8859, UNICODE y muchos otros menos difundidos que no se estudian.



1.6- Rep. de caracteres: El estándar ASCII (I)

- Acrónimo de **American Standard Code for Information Interchange** (publicado en 1963).
- Utiliza 7 bits $\rightarrow 2^7 = 128$ caracteres.
- Los caracteres se dividen en dos grupos:
 - (0-31) Caracteres de control no imprimibles.
 - (32-127) Caracteres imprimibles (símbolos, dígitos, letras mayúsculas y minúsculas)

Caracteres imprimibles en orden:

! " # \$ % & ' () * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K
L M N O P Q R S T U V W X Y Z [\] ^ _ ` a b c d e f g h i j k l m n o p q r s t u
v w x y z { | } ~

- Problema: Sólo se ajusta al alfabeto anglosajón.



1.6- Rep. de caracteres: El estándar ASCII (II)

TABLA DE CÓDIGOS ASCII

BITS	b6 b5 b4				0 0		0 0		0 1		0 1		1 0		1 0		1 1		1 1																								
					0		1		0		1		0		1		0		1																								
					CONTROL				SÍMBOLOS NÚMEROS				MAYÚSCULAS				MINÚSCULAS																										
b3 b2 b1 b0																																											
0 0 0 0				0	NUL				16	DLE				32	SP				48	0				64	@				80	P				96	'				112	p			
				0	0				10	20				20	40				30	60				40	100				50	120				60	140				70	160			
0 0 0 1				1	SOH				17	DC1				33	!				49	1				65	A				81	Q				97	a				113	q			
				1	1				11	21				21	41				31	61				41	101				51	121				61	141				71	161			
0 0 1 0				2	STX				18	DC2				34	"				50	2				66	B				82	R				98	b				114	r			
				2	2				12	22				22	42				32	62				42	102				52	122				62	142				72	162			
0 0 1 1				3	ETX				19	DC3				35	#				51	3				67	C				83	S				99	c				115	s			
				3	3				13	23				23	43				33	63				43	103				53	123				63	143				73	163			
0 1 0 0				4	EOT				20	DC4				36	\$				52	4				68	D				84	T				100	d				116	t			
				4	4				14	24				24	44				34	64				44	104				54	124				64	144				74	164			
0 1 0 1				5	ENQ				21	NAK				37	%				53	5				69	E				85	U				101	e				117	u			
				5	5				15	25				25	45				35	65				45	105				55	125				65	145				75	165			
0 1 1 0				6	ACK				22	SYN				38	&				54	6				70	F				86	V				102	f				118	v			
				6	6				16	26				26	46				36	66				46	106				56	126				66	146				76	166			
0 1 1 1				7	BEL				23	ETB				39	'				55	7				71	G				87	W				103	g				119	w			
				7	7				17	27				27	47				37	67				47	107				57	127				67	147				77	167			



1.6- Rep. de caracteres: El estándar ASCII (III)

1 0 0 0	8	BS	24	CAN	40	(56	8	72	H	88	X	104	h	120	x
1 0 0 1	9	HT	25	EM	41)	57	9	73	I	89	Y	105	i	121	y
1 0 1 0	10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
1 0 1 1	11	VT	27	ESC	43	+	59	;	75	K	91	[107	k	123	{
1 1 0 0	12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	
1 1 0 1	13	CR	29	GS	45	-	61	=	77	M	93]	109	m	125	}
1 1 1 0	14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
1 1 1 1	15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

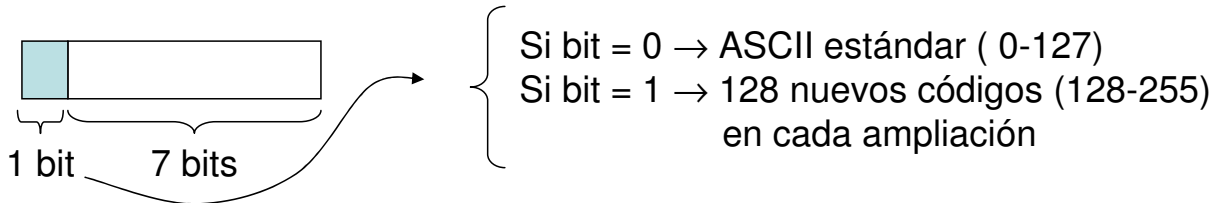
LEYENDA:

dec	CHAR
hex	oct



1.6- Rep. de caracteres: El ASCII extendido (I)

- Se circunscribe al ámbito del computador PC y del sistema operativo MS-DOS para dar cabida a otros alfabetos.
- Se definen varias ampliaciones (códigos de página) para los distintos alfabetos (latino, griego, etc.)
- Se forma con 8 bits, añadiendo 1 bit al código ASCII estándar.



- Aportaciones del juego ampliado de caracteres:
 - Caracteres extranjeros (ñ, Ñ, á, é, ¿, ¢, ...)
 - Caracteres de dibujo (L, ⊥, T, ...)
 - Caracteres científicos ($\frac{3}{4}$, ±, ², ³, ¹, ...)
- Problema: mal organizado e insuficiente.



1.6- Rep. de caracteres: El ASCII extendido (II)

Código ASCII extendido (IBM-PC)

DEC	HX	C	DEC	HX	C	DEC	HX	C	DEC	HX	C	DEC	HX	C	DEC	HX	C	DEC	HX	C	DEC	HX	C
128	80	Ç	144	90	É	160	A0	á	176	B0	⋮	192	C0	⌞	208	D0	⌞	224	E0	α	240	F0	≡
129	81	ü	145	91	æ	161	A1	î	177	B1	⋮	193	C1	⌞	209	D1	⌞	225	E1	β	241	F1	±
130	82	é	146	92	Æ	162	A2	ó	178	B2	⋮	194	C2	⌞	210	D2	⌞	226	E2	Γ	242	F2	≥
131	83	â	147	93	ô	163	A3	ú	179	B3	⋮	195	C3	⌞	211	D3	⌞	227	E3	Π	243	F3	≤
132	84	ä	148	94	ö	164	A4	ñ	180	B4	⋮	196	C4	⌞	212	D4	⌞	228	E4	Σ	244	F4	∫
133	85	ä	149	95	ò	165	A5	Ñ	181	B5	⋮	197	C5	⌞	213	D5	⌞	229	E5	σ	245	F5	∫
134	86	ä	150	96	û	166	A6	æ	182	B6	⋮	198	C6	⌞	214	D6	⌞	230	E6	μ	246	F6	÷
135	87	ç	151	97	ù	167	A7	æ	183	B7	⋮	199	C7	⌞	215	D7	⌞	231	E7	τ	247	F7	≈
136	88	ê	152	98	ÿ	168	A8	¿	184	B8	⋮	200	C8	⌞	216	D8	⌞	232	E8	φ	248	F8	°
137	89	ë	153	99	ö	169	A9	¸	185	B9	⋮	201	C9	⌞	217	D9	⌞	233	E9	θ	249	F9	•
138	8A	è	154	9A	Ü	170	AA	¸	186	BA	⋮	202	CA	⌞	218	DA	⌞	234	EA	η	250	FA	·
139	8B	ï	155	9B	¢	171	AB	½	187	BB	⋮	203	CB	⌞	219	DB	⌞	235	EB	δ	251	FB	√
140	8C	î	156	9C	£	172	AC	¾	188	BC	⋮	204	CC	⌞	220	DC	⌞	236	EC	∞	252	FC	²
141	8D	ï	157	9D	¥	173	AD	¸	189	BD	⋮	205	CD	⌞	221	DD	⌞	237	ED	φ	253	FD	³
142	8E	ä	158	9E	℞	174	AE	«	190	BE	⋮	206	CE	⌞	222	DE	⌞	238	EE	€	254	FE	⁴
143	8F	â	159	9F	f	175	AF	»	191	BF	⋮	207	CF	⌞	223	DF	⌞	239	EF	η	255	FF	⁵



- Creado por ISO (***I**nternational **S**tandardization **O**rganization*).
- Son códigos de 8 bits (formato similar al ASCII extendido).
- Se trata de una serie de ampliaciones del ASCII estándar mejor organizadas que el ASCII extendido. Ejemplos:
 - ISO 8859-1 (ISO Latin 1): ampliación para Europa occidental (incluido el castellano).
 - ISO 8859-2 (ISO Latin 2): ampliación para Europa del este.
 - ISO 8859-8 (ISO Latin/Hebreo): ampliación para el hebreo hablado en Israel.
- Windows adaptó las versiones del formato ISO 8859 anteriores. Por ejemplo la versión modificada del ISO Latin 1 es Windows-1252.
- Problemas:
 - Juegos diferentes para distintos alfabetos.
 - Un mismo carácter puede tener distinto código en cada alfabeto.
 - No tiene lenguas orientales (chino, japonés, etc.).



1.6- Rep. de caracteres: El ISO 8859 – Latin 1

Código ISO-Latin-1

A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
	í	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	­	®	¯
B0	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î
D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ
E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î
F0	ï	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	ÿ

LEYENDA:

HEX
CAR

HEX: código hexadecimal

CAR: carácter



Código ISO-Latin-2

A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
	À		Ł	Ŕ	Ĺ	Š	Š		Š	Š	Ť	Ž	-	Ž	Ž
B0	á		ł		İ	š			š	š	ť	ž		ž	ž
C0	Ř	Ā	Ā	Ā	Ā	Ĺ	Č	Č	Č	Ě	Ě	Ě	Ě	Ě	Ě
D0	Đ	Ň	Ň	Ō	Ō	Ō	×	Ř	Ů	Ú	Ů	Ü	Ü	Ÿ	Ÿ
E0	ř	ā	ā	ā	ā	ĭ	č	č	č	ě	ě	ě	ě	ě	ě
F0	đ	ň	ň	ô	ô	ö	÷	ř	ů	ú	ů	ü	ü	ý	ý

LEYENDA:

HEX
CAR

HEX: código hexadecimal
CAR: carácter



1.6- Propiedades deseables de los códigos alfanuméricos

- Los códigos correspondientes a las letras del alfabeto deben ser consecutivos y organizados según la ordenación del alfabeto.
- Las letras mayúsculas y las minúsculas deben diferenciarse en un solo bit.

ASCII estándar	ASCII extendido	ISO Latin 1
e → 65 ₍₁₆₎ → 1100101	é → 82 ₍₁₆₎ → 10000010	é → E9 ₍₁₆₎ → 11101001
E → 45 ₍₁₆₎ → 1000101	É → 90 ₍₁₆₎ → 10010000	É → C9 ₍₁₆₎ → 11001001
Bit 5	NO CUMPLE	Bit 5

- Los códigos que representan a los dígitos deben estar juntos y ordenados en forma consecutiva; además deben ser fácilmente transformables en la cantidad que representan.

0 → 0110000

9 → 0111001

- Los códigos de control deben estar unificados en una zona determinada de la tabla.



1.6- Representación de caracteres: Unicode I

- Es un estándar universal para la representación de texto.
- Objetivo: proporcionar un sistema consistente para la representación de texto de múltiples lenguas.
- Características:
 - Asigna números a caracteres de escritura.
 - 90.000 asignados aproximadamente, ejemplo 'e': U+0065 (número asignado en hexadecimal).
 - Códigos que utilizan hasta 31 bits.
 - Los códigos se organizan en bloques correspondientes a un determinado alfabeto: latino, griego, hebreo, etc.
 - Los 256 primeros se corresponden con el ISO Latin 1.
- Algunos sistemas de escritura cubiertos:
 - Latino, árabe, griego, ..., egipcio, maya, cuneiforme.



1.6- Representación de caracteres: Unicode II

- Unicode asigna números a caracteres pero no indica la forma de realizar la codificación.
 - Problema: codificación del carácter 'e' ($U+0065 = 65_{(16)}$):
 - (a) (8 bits) 0110 0101
 - (b) (16 bits) 0000 0000 0110 0101
 - (c) (32 bits) 0000 0000 0000 0000 0000 0000 0110 0101
- ¿Ventajas e inconvenientes de (a), (b) y (c)?
- Solución adoptada: creación de diversos métodos de codificación con distinto número de bits e incluso con número variable de bits.
 - Métodos de codificación: UCS (longitud fija, 2 y 4 bytes) y UTF (longitud variable, de 1 a 6 bytes).
 - Solución adoptada por MS Windows, Java y MacOSX: UTF-16.
 - Solución adoptada mayoritariamente en el desarrollo web y en las implementaciones del Sistema operativo UNIX: UTF-8.



1.6- Representación de caracteres: Unicode III – UTF 8

- Desarrollado en 1992.
- Codificación de longitud variable (de 1 a 6 bytes).
- La codificación se realiza en función del número asignado a cada carácter según la siguiente tabla:

Rango de números asignados	Codificación en UTF-8	Nº de bits
U+00000000 – U+0000007F	0xxxxxxx	7
U+00000080 – U+000007FF	110xxxxx 10xxxxxx	11
U+00000800 – U+0000FFFF	1110xxxx 10xxxxxx 10xxxxxx	16
U+00010000 – U+0010FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	21
U+00200000 – U+03FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx	26
U+04000000 – U+7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx	31



1.6- Representación de caracteres: Unicode IV – UTF 8

- Características de UTF-8:
 - Los primeros 128 caracteres se codifican con 1 byte, los siguientes 1920 con 2 bytes.
 - El bit más significativo de un carácter codificado con un solo byte es siempre 0.
 - Los bits más significativos de un carácter codificado con varios bytes determinan el tamaño. Si empieza por n bits ($n > 1$) con valor igual a 1 quiere decir que el tamaño de la codificación del carácter en bytes es n (por ejemplo: 110 indica 2 y 1110 indica 3).
 - El resto de bytes en una secuencia de múltiples bytes codificando un carácter comienzan por 10, lo cual permite la sincronización de una secuencia de bytes.

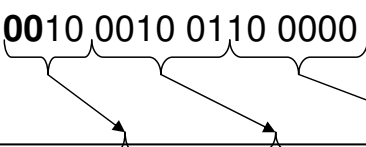
Ejercicio: determinar cuántos caracteres hay codificados en la siguiente secuencia de bytes: 01010101 11010110 10100010 00000101

(Solución: 3)



1.6- Representación de caracteres: Unicode V – UTF 8

- Pasos para codificar en UTF-8

Paso	Descripción	Ejemplo
1	Determinar el código Unicode	“no igual” (≠) (U+2260)
2	Pasar a binario	10 0010 0110 0000
3	Determinar el número mínimo de bits necesarios	14 bits
4	Determinar el número mínimo de bytes UTF-8 necesarios	3 bytes
5	Plantear el esquema de codificación y ver cuántos bits utiliza para el número	1110xxxx 10xxxxxx 10xxxxxx (16 bits)
6	Rellenar el número binario del paso 2 con ceros por la izquierda hasta tener el número de bits necesario	0010 0010 0110 0000 
7	Introducir los bits en el esquema	11100010 10001001 10100000



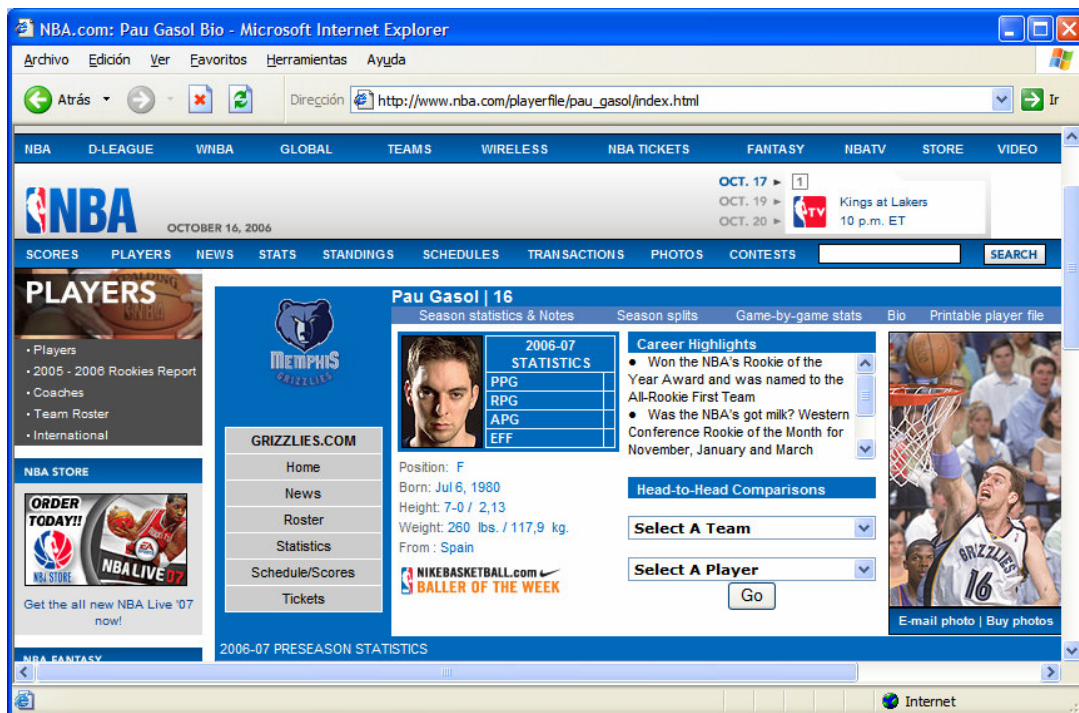
1.6- Representación de caracteres: Unicode VI – UTF 8

- Ejemplo 1: **codificación del carácter alef (א) (U+05D0):**
 - Solución: 11010111 10010000 (D7 90₍₁₆₎)
- Ejemplo 2: **codificación del copyright (©) (U+00A9):**
 - Solución: 11000010 10101001 (C2 A9₍₁₆₎)
- Ejemplo 3: **codificación de la cadena de caracteres "ña" sabiendo que el código unicode de la "ñ" es U+00F1 y el de la "a" es U+0061**
 - Solución: 11000011 10110001 01100001 (C3 B1 61₍₁₆₎)



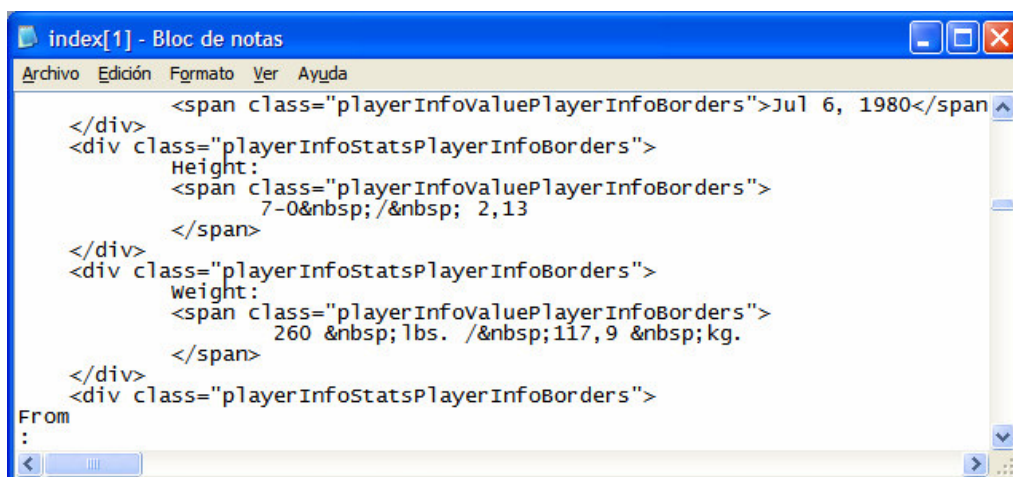
1.6- Ejemplos de codificación de caracteres I

- ¿Qué es una página web?



1.6- Ejemplos de codificación de caracteres II

- ¿Qué es una página web?
 - Es un documento de texto con información.
 - Está escrito en lenguaje HTML.
 - Un programa lo transfiere desde un servidor en Internet y lo visualiza de forma gráfica (p.e. Internet Explorer).

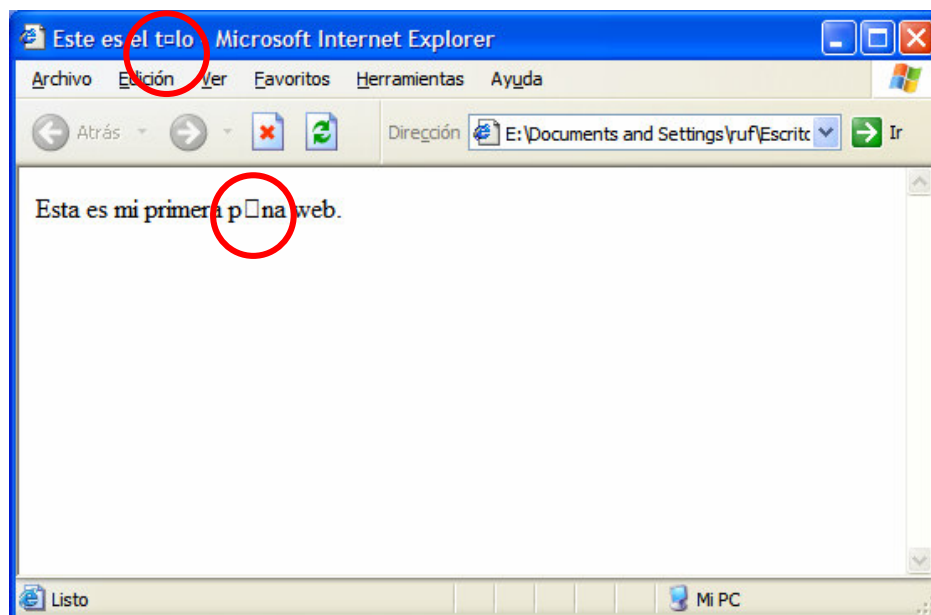


- Un ejemplo sencillo de página web

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head>
    <title>
      Este es el título
    </title>
  </head>
  <body>
    Esta es mi primera página web.
  </body>
</html>
```

1.6- Ejemplos de codificación de caracteres IV

- Su visualización (Internet Explorer) produce el siguiente resultado:



- ¿Qué ocurre?

1.6- Ejemplos de codificación de caracteres V

- Problema:
 - El programa que visualiza la página necesita saber cómo están codificados los caracteres. Si no lo sabe lo intenta adivinar y a veces falla.

Contenido del fichero de texto

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head>
    <title>
      Este es el título
    </title>
  </head>
  <body>
    Esta es mi primera página web.
  </body>
</html>
```



1.6- Ejemplos de codificación de caracteres VI

- Problema:
 - El programa que visualiza la página necesita saber como están codificados los caracteres. Si no lo sabe lo intenta adivinar y a veces falla.

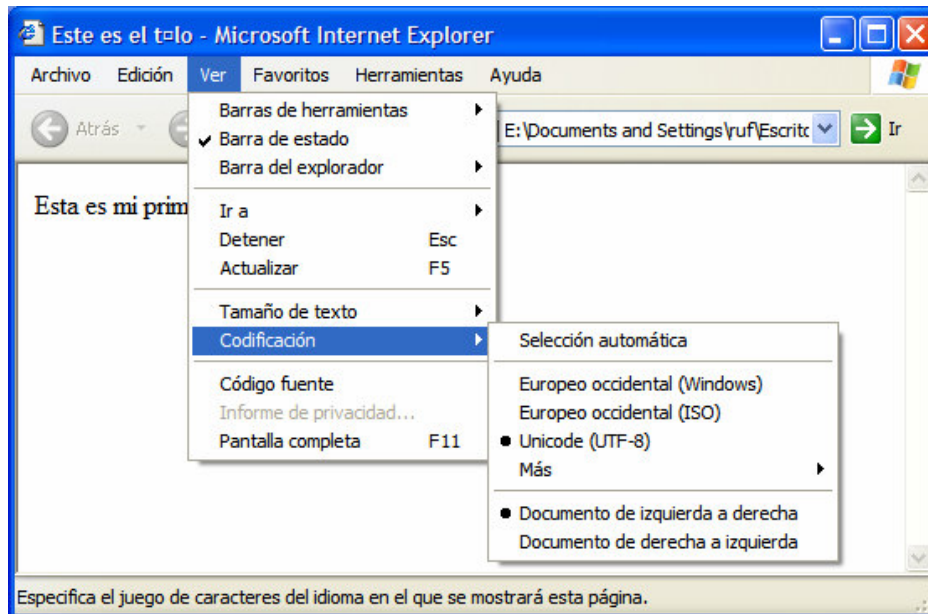
Codificación del fichero de texto

```
...
0011110001101000011101000110110101101100
0011111011011101101010101001100100111100
0110100001100101011000010110010000111110
1101110110101010100110011001100100111100
0111010001101001011101000110110001100101
0011111011011101101010101001100110011001
1001100101000101011100110111010001100101
...
```

- ¿Cómo se interpreta? ¿ISO-Latin1? ¿UTF-8? ¿Otros?



1.6- Ejemplos de codificación de caracteres VII



- Lo interpretó como si estuviera codificado en UTF-8.
- En realidad estaba codificado en ISO-Latin1.



1.6- Ejemplos de codificación de caracteres VIII

- Solución:
 - Indicar en la página web la codificación

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head>
    <meta http-equiv="Content-Type"
          content="text/html;
          charset=iso-8859-1"/>
    <title>
      Este es el título
    </title>
  </head>
  <body>
    Esta es mi primera página web.
  </body>
</html>
```



- Solución:
 - Indicar en la página web la codificación.

