Tema 1: Información digital

1.1- Introducción.

1.2- Sistemas de numeración.

1.3- Representaciones numéricas: nº Naturales.

1.4- Representaciones numéricas: nº Enteros.

1.4.1 – Signo-Magnitud.

1.4.2 – Complemento a 2.

1.4.3 – Exceso a un entero Z.

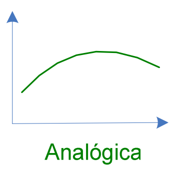
1.5- Representaciones numéricas: nº Reales.

1.5.1 – Coma fija.

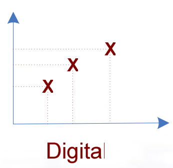
1.5.2 – Coma flotante: Formato simplificado y Formato IEEE-754 simple.

1.6- Representación de caracteres.

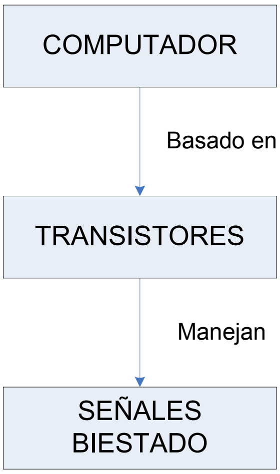
- Los computadores son *sistemas electrónicos*, capaces de almacenar, mover y procesar información.
- Los dispositivos electrónicos manejan la información mediante el uso de *señales eléctricas*.
- Las señales eléctricas pueden ser de 2 tipos:
  - Analógicas: pueden tomar cualquier valor dentro de su 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.



- 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 ⟷ 2^n unidades

- Unidades de información comúnmente manejadas:
  - BYTE (BinarY octeTE): 8 bits (→ 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.

Secuencia de bits	*Objeto representado*
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?

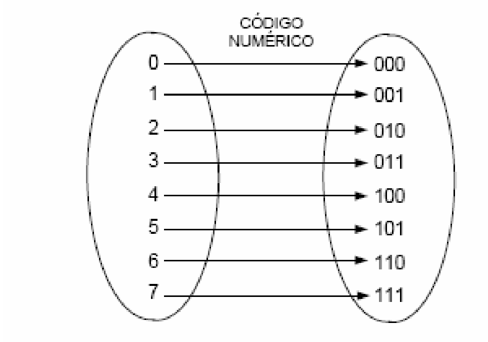
2 bits → $2^2 = 4 < 5$ → No vale

3 bits → $2^3 = 8 > 5$ → Vale

Secuencia de bits	***Objeto representado***
0 0 0	a
0 0 1	e
0 1 0	i
0 1 1	o
1 0 0	u

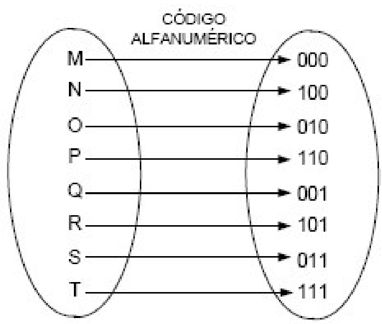
Códigos Binarios

Códigos Numéricos



- **Naturales**
- **Enteros**
- **Reales**

Códigos Alfanuméricos



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

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 x 10^3 + 0 x 10^2 + 2 x 10^1 + 3 x 10^0 = 1000 + 0 + 20 + 3 = 1023

10'23 = 1 x 10^1 + 0 x 10^0 + 2 x 10^{-1} + 3 x 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.

BASE	*Dígitos disponibles*
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:

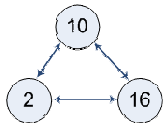
$1111_{(2)} = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 4 + 2 + 1 = 15$

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

DEC	*Binario*	*Octal*	*Hexadecimal*
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

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.

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**

```
61 |2
01  30 |2
 1  10  15 |2
     0   1   7 |2
             1   3 |2
                 1   1
```

Orden de Composición del nº

1 1 1 1 0 1

$\boxed{61_{(10)} = 111101_{(2)}}$

10 → 16 **Pasar $123_{(10)}$ a Hexadec.**

```
123 |16
 11   7
 B
```

$\boxed{123_{(10)} = 7B_{(16)}}$

1.2- Sistemas de numeración: conversión entre bases

b) Paso de cualquier base a Base 10:

Aplicar el sistema posicional sobre el nº que se desea convertir.

2 → 10 **Pasar $111101_{(2)}$ a Decimal**

$$111101_{(2)} = 1x2^5 + 1x2^4 + 1x2^3 + 1x2^2 + 1x2^0$$
$$= 32 + 16 + 8 + 4 + 1$$
$$= 61_{(10)}$$

16 → 10 **Pasar $7B_{(16)}$ a Decimal**

$$7B_{(16)} = 7x16^1 + Bx16^0 = 112 + 11 = 123_{(10)}$$

1.2- Sistemas de numeración: conversión entre bases

c) Cambios entre las bases 2 y 16:

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

Relleno

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

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

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

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

Caso 2: los números a convertir son fraccionarios puros.

a) Paso de Base 10 a cualquier base:

Se multiplica el nº a convertir por el valor de la base destino, hasta que la parte fraccionaria sea nula.

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

0'8125 x 2	=	1'625
0'625 x 2	=	1'25
0'25 x 2	=	0'5
0,5 x 2	=	1'0

Orden de Composición del nº

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

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

0'703125 x 16	=	11'25
0'25 x 16	=	4'0

0'703125$_{(10)}$ = 0'B4$_{(16)}$

b) Paso de cualquier base a Base 10:

Aplicar al nº que se desee convertir el sistema posicional.

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

$0'101101_{(2)} = 1x2^{-1} + 1x2^{-3} + 1x2^{-4} + 1x2^{-6} = 0'703125_{(10)}$

16 → 10 **Pasar 0'B4$_{(16)}$ a Decimal.**

$0'B4_{(16)} = 11x16^{-1} + 4x16^{-2} = 0'703125_{(10)}$

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

Relleno

0, 1011 0100$_{(2)}$ = 0,B4$_{(16)}$

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

16 → 2 **Pasar 0'1A9$_{(16)}$ a Binario → 0' 0001 1010 1001$_{(2)}$**

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

0'2 x 2 = 0'4
0'4 x 2 = 0'8
0'8 x 2 = 1'6
0'6 x 2 = 1'2
0'2 x 2 = 0'4
. . .

0'2$_{(10)}$ = 0'**0011**0011… $_{(2)}$

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

0'8 x 16 = 12'8
0'8 x 16 = 12'8
. . .

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 → 2

```
14 |2
 0   7 |2
     1   3 |2
         1   1
```

b) Parte fraccionaria *(aplicar Caso 2)* → 10 → 2

0'375 x 2 = 0'75
0'75 x 2 = 1'5
0'5 x 2 = 1'0

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

- ARITMÉTICA BÁSICA (suma y resta) EN DIFERENTES BASES

Ejemplos Suma

	Sin acarreo	Con acarreo
Decimal	6 8 + 2 1 8 9	4 5 9 + 3 7 8 8 3 7
Binario	01000000 + 00010101 01010101	111001011 +101111010 1101000101
Hexadecimal	4 0 + 1 5 5 5	1 C B + 1 7 A 3 4 5

Ejemplos Resta

	Sin acarreo	Con acarreo
Decimal	4 5 9 - 3 1 8 1 4 1	5 2 5 - 3 8 7 1 3 8
Binario	10100110 - 00100100 10000010	01100101 - 01011010 00001011
Hexadecimal	A 6 - 2 4 8 2	7 3 1 C - 1 A 4 6 5 8 D 6

***Recordatorio*:** **A** → 10 **B** → 11 **C** → 12 **D** → 13 **E** → 14 **F** → 15

Ejercicios:

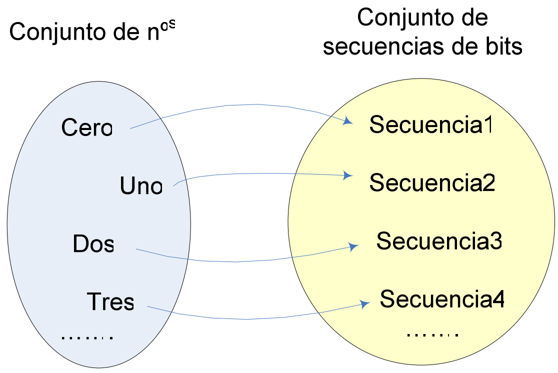
```
  01011101        01011101
+ 00101110      - 00101110
  --------        --------
  10001011        00101111
```

```
  731C        C3AB        A11C
+ 1A46      + 9EF3      - 1A48
  ----        -----       ----
  8D62        1629E       86D4
```


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	$6699'0469_{(10)}$
$110101101100010 1011_{(2)}$	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.

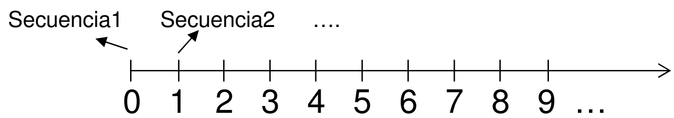


Problema: Conjuntos numéricos infinitos frente a secuencias de bits finitas (el computador tiene un límite de almacenamiento).

↓

Aparición del concepto de *RANGO*.

Planteamiento:



Pasos para crear el código:

a) Determinar el nº de objetos a representar. *(p.e. [0-7])*

b) Determinar el nº de bits necesarios para representar dichos objetos.
En general: n bits ←→ 2^n elementos *(p.e. para representar 8 objetos → 3 bits)*

c) Asignar a cada objeto una secuencia de bits.
El criterio universalmente aceptado es asignar a cada secuencia de bits el nº natural que representa en base 2. A este formato se le denomina **BINARIO NATURAL**.

En nuestro ejemplo:

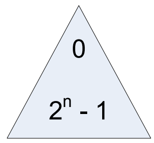
0 → 000
1 → 001
. . .
7 → 111

Esquema de representación de este formato numérico:

– ***Formato de representación***: ***Binario natural*** *(BN).*

– ***Rango de representación***: **[0, 2^n - 1]**, siendo 'n' el nº de bits.

Ejemplos:
n = 3 → Rango: [0, $2^3 – 1$] = [0, 7]
n = 4 → Rango: [0, $2^4 – 1$] = [0, 15]
n = 8 → Rango: [0, $2^8 – 1$] = [0, 255]
n = 16 → 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 nºs, cada uno representable con 'n' bits, el resultado de la suma no lo sea → Concepto de desbordamiento aritmético.

✓

	0 0 1 1	*(3)*
+	1 0 0 1	*(9)*
	1 1 0 0	*(12)*

	1 0 0 0	*(8)*
+	1 0 1 1	*(11)*
Acarreo final	1 0 0 1 1	*(3)*

✘

- Los números enteros pueden ser positivos o negativos → El signo es un elemento más a tener en cuenta.

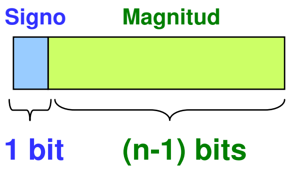
- Hay varios tipos de soluciones:
  - Forzar la codificación del signo en un bit extra.
    P.e. 0=positivo, 1=negativo (método denominado signo-magnitud).
  - Convertir los negativos en positivos sumándoles un constante Z
    (método denominado Exceso a un entero Z).
  - Complemento a 2.

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

Ej: n = 4 bits
$5_{(10)}$ → 0101
$-5_{(10)}$ → 1101

<u>**NOTA**</u>: En este formato hay una doble representación del 0 (*p.e. n=4 bits→ 0000 y 1000*).

– **Rango: [- (2^{n-1} - 1) , 2^{n-1} - 1]**, siendo ‘n’ el nº de bits *(→ Rango Simétrico)*

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

– **Aritmética: no se utiliza.**

Formato 2: Complemento a 2 (C-2)

Nºs positivos → MSB=0
Nºs negativos→ MSB=1

– **Formato de representación (para ‘n’ bits):**

a) Números Positivos → Se representan en Binario Natural

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

b) Números Negativos → 3 métodos para obtener su representación:

*** Representar en BN el valor 2^n - |-x|, *con ‘n’ = nº de bits, –x = nº a representar***

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

*** Representar en BN el valor absoluto del nº.**
Invertir todos los bits.
Sumarle 1 al nº obtenido en el paso anterior.

P.e. n = 4 bits, nº = -5 → |-5| = $5_{(10)}$ $\xrightarrow{BN}$ $0101_{(2)}$ $\xrightarrow{Invertir}$ $1010_{(2)}$ $\xrightarrow{+1}$ $1011_{(C-2)}$

*** Representar en BN el valor absoluto del nº 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 → $1101_{(2)}$ = -(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, $1101_{(2)}$ → 1100 → 0011 → -3

*** Empezando por el bit menos significativo, ir copiando los bits de drcha. a izqda. hasta encontrarel 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, $1101_{(2)}$ → 0011 → -3

– **Rango: $[- 2^{n-1}, \ 2^{n-1} - 1]$**, siendo 'n' el nº de bits.

Ej. Si n = 4 bits → 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 nºs binarios naturales, genera un resultado correcto con signo siempre y cuando dicho resultado pertenezca al rango de los nºs 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.

-7	→	$1001_{(C-2)}$
+ 5	→	$0101_{(C-2)}$
(-2)		$1110_{(C-2)}$ → -2 ✓

7	→	$0111_{(C-2)}$
+ 5	→	$0101_{(C-2)}$
(12)		$1100_{(C-2)}$ → -4 ✘

– **Detección de desbordamiento aritmético para Nº Enteros**

Interpretados como nºs Naturales		Interpretados como nºs Enteros
10	1 0 1 0	-6
7	0 1 1 1	7
17	$\boxed{1}\,\underbrace{0\,0\,0\,1}_{\text{Result = 1}}$	1
No es representable con 4 bits		Sí es representable con 4 bits

Acarreo final:

- **Para nºs Naturales indica desbordamiento.**
- **Para nºs Enteros no es indicativo de desbordamiento.**

<u>Conclusión</u>: el mecanismo para detectar desbordamiento en los en los nºs Naturales NO sirve para los nºs Enteros.

Para detectar el desbordamiento aritmético de nºs 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

***<u>Observación</u>*: Cuando los operandos son de distinto signo, su suma nunca producirá desbordamiento.**

1.4- Representaciones numéricas: comparativa

Ejemplo de detección de desbordamiento aritmético para nºs interpretados CON y SIN signo

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

Interpretados como nºs Naturales en bin. natural		Interpretados como nºs Enteros en C-2
7 ←	0 1 1 1	→ 7
5 ←	+ 0 1 0 1	→ 5
✓	1 1 0 0	✘

No existe Acarreo Final → No existe desbordamiento en el caso de interpretarlos como nºs Naturales → El resultado obtenido es representable con 4 bits.

SignoOp1 = SignoOp2 = 0 ≠ 1 = SignoR

En caso de interpretar los nºs como Enteros, SÍ existe desbordamiento → El resultado obtenido NO es representable con 4 bits.

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

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

1.4- Representaciones numéricas: ejercicio

Ejercicio propuesto

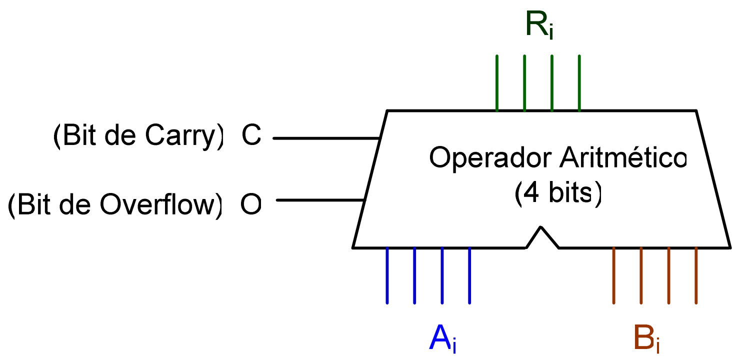
Sup. n = 4 bits → Rango de Naturales: [0, 15]
Rango de Enteros C-2: [-8, 7]

Interpretados como nºs Naturales en bin. natural		Interpretados como nºs Enteros en C-2
?? ←	1 1 0 1	→ ??
?? ←	1 1 1 0	→ ??

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

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

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 (→ nºs 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 (→nºs Enteros).

Formato 3: Exceso a un entero Z

– ***Formato de representación***: *Sumar el Exceso Z el nº a representar, y luego representar el resultado obtenido en B. Natural.*

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

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

Observación:

Para que un nº sea representable en Exceso a un entero Z con un nº de bits 'n' dado, deben cumplirse 2 condiciones:

a) Después de sumarle el exceso al nº, el resultado NO puede ser negativo.
   Ej. dados n=4 bits y EXC-5 → Representar el -6: -6 → -6+5 = -1 → No cumple (a)

b) Después de sumarle el exceso al nº, el resultado debe ser representable en B.N. para el 'n' dado.
   Ej. dados n=4 bits y EXC-5 → Representar el 12: 12 → 12+5 = 17 → No cumple (b)

– ***Rango de representación***: **[-Z , 2^n – 1 – Z]**, siendo 'n' el nº de bits.

Ejemplos: n = 5, EXC-16 → Rango: [-16, 2^5 – 1 – 16] = [-16, 15]

n = 4, EXC-4 → Rango: [-4, 2^4 – 1 – 4] = [-4, 11]

n = 3, EXC-7 → Rango: [-7, 2^2 – 1 – 7] = [-7, -4]

NOTA:

* Si Z = 2^{n-1} → Se denomina Exceso-Central, y el rango coincidirá con el de C-2.
* Si Z < 2^{n-1} → Se podrán representar más nºs Positivos que Negativos.
* Si Z > 2^{n-1} → Se podrán representar más nºs Negativos que Positivos.

– ***Consideraciones sobre Aritmética***:

Propiedad:

"Los nºs representados en Exceso-Z se ordenan de menor a mayor, de modo que al más pequeño de todos (máximo negativo) se le asigna el nº binario menor (00..0), y al más grande (máximo positivo) el nº binario mayor (11…1)".

Esta propiedad facilita la comparación de nºs enteros.

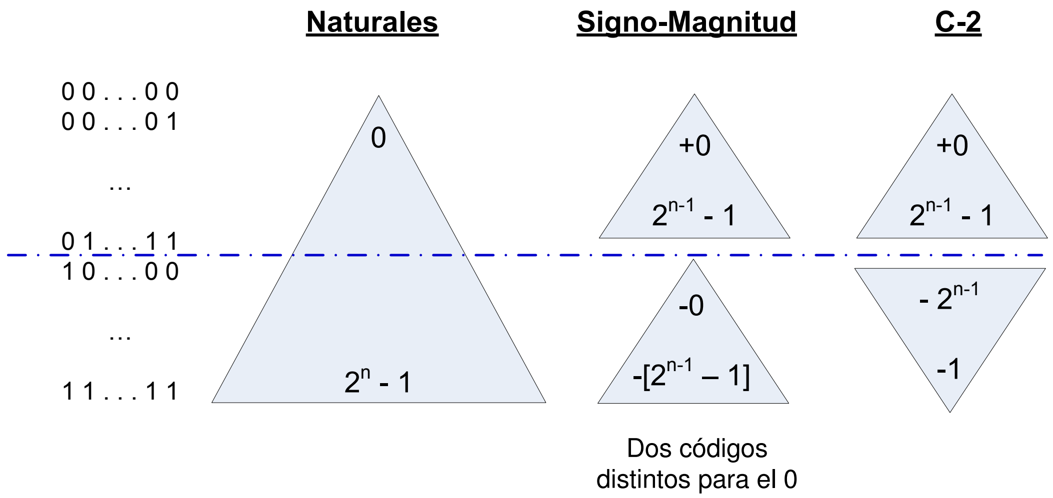
Binario	*Natural*	*Signo-Magnitud*	*C-2*	*Exc-8*
0 0 0 0	0	0	0	-8
0 0 0 1	1	1	1	-7
0 0 1 0	2	2	2	-6
0 0 1 1	3	3	3	-5
0 1 0 0	4	4	4	-4
0 1 0 1	5	5	5	-3
0 1 1 0	6	6	6	-2
0 1 1 1	7	7	7	-1
1 0 0 0	8	-0	-8	0
1 0 0 1	9	-1	-7	1
1 0 1 0	10	-2	-6	2
1 0 1 1	11	-3	-5	3
1 1 0 0	12	-4	-4	4
1 1 0 1	13	-5	-3	5
1 1 1 0	14	-6	-2	6
1 1 1 1	15	-7	-1	7

EXC-8 es el exceso-central para n=4 bits

Se puede ver que el rango del EXC-8 y el del C-2 efectivamente coinciden [-8, 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



Ejercicios

❶ 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 → 7 + 16 = 23 —Codificar en BN→ 1 0 1 1 1

-16 codificado en C-2 → Codificar el 16 → 1 0 0 0 0

Invertir a partir del 1er '1': 1 0 0 0 0 → 1 0 0 0 0

1 | 0 0 1 1 1

El resultado obtenido es 7 → Si está codificado en EXC-8 entonces: x + 8 = 7 → **x = -1**

❷ En un sumador para cantidades de 'n' bits, con n=4, se introducen el mayor nº entero positivo y el menor nº entero negativo, ambos codificados en C-2. ¿Qué resultado se obtendrá a la salida del sumador interpretado como un nº 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 nºs 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.

Naturales y Enteros	**Reales** ***(sup. n=4 bits)***
n = 2 bits: 0 1 2 3	0 0'5 1 1'5 . . . 7'5
n = 3 bits: 0 1 2 3 4 5 6 7	0 0'25 0'5 0'75 1 . . . 3'75
n = 4 bits: 0 1 2 3 4 5 6 7 . . . 15	0 0'125 0'25 0'375 . . . 1'825
Distancia entre dos nºs consecutivos siempre = 1 ↓ Precisión Constante.	Distancia entre dos nºs consecutivos NO CTE. ↓ Compromiso entre Rango y Precisión.

- Precisión +
+ Rango -

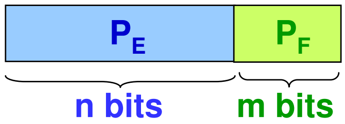
- Como se acaba de explicar, son dos los conceptos a tener en cuenta: RANGO y PRECISIÓN.

- 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 nº 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 nºs CON o SIN signo.

↗ Nºs sin signo: utilizar el criterio de representación del sistema binario.

↘ Nºs con signo: seguir el criterio de representación del C-2.

Ejemplo:

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

$$5'75 \xrightarrow{\text{Nº mixto}} \begin{cases} P_E = 5 \xrightarrow{\text{B.N.}} 101 \\ P_F = 0'75 \xrightarrow[\substack{0'75 \times 2 = 1'5 \\ 0'5 \times 2 = 1'0}]{} 0'11 \end{cases} \rightarrow 101'11$$

$\downarrow$ *Ajustar al formato n .m (xxxxx . xxx)*

$\underline{00}101'11\underline{0}$

$-5'75 \longrightarrow 5'75_{(10)} = 101'11$

$\downarrow$ *Ajustar al formato n .m (xxxxx . xxx)*

$\underline{00}101'11\underline{0}$

$\downarrow$ *Inversión del C-2*

11010'010

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

– **Rango y Precisión:**

Dado un nº 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 ↑ y m ↓ ⟶ Rango ↑ y Precisión ↓

Si n ↓ y m ↑ ⟶ Rango ↓ y Precisión ↑

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

	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 nº a representar.

Ejemplo:

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

Precisión: $2^{-m} = 2^{-4} = 1/2^4 = 0'0625$ → Se corresponde además con el nº más pequeño representable con 8+4 bits: 00000000'0001

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

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

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

Nº más pequeño representable: 00000'000 = 0

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

Aprender a deducirlo

11111'111 (→ =x)
\+ 00000'001 (→ $=2^{-3}$)
100000'000 (→ $=2^5$)

→ $x + 2^{-3} = 2^5$ → $x = 2^5 - 2^{-3}$

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

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

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

Aprender a deducirlo

01111'111 (→ =x)
\+ 00000'001 (→ $=2^{-3}$)
10000'000 (→ $=2^4$)

→ $x + 2^{-3} = 2^4$ → $x = 2^4 - 2^{-3}$

– **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:

a) El bit de Carry (CF) si el nº se interpreta como una magnitud sin signo.
b) 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 nºs Naturales	Operación	Interpretados como nºs Enteros
7'625	0 0 1 1 1 ' 1 0 1	7'625
28'625	+ 1 1 1 0 0 ' 1 0 1	-3'375
✘	1 0 0 1 0 0 ' 0 1 0	✓

Resultado = 4'25

Existe Carry Final → Overflow en el caso de interpretarlos como nºs Naturales → El resultado obtenido NO es representable.

SignoOp1 ≠ SignoOp2 → No hay Overflow 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 nº 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)

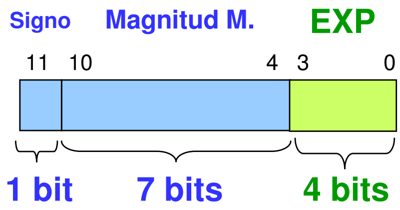
- *Formato*: **R = M x B^e** , *donde M: mantisa, B: base, e: exponente*
- *Ejemplos:*

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

- *Formatos Normalizados*:
  - Todo Fracción (TF): $\mathbf{R = 0'D_{-1}... \times B^e}$, 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): $\mathbf{R = ...D_0'0 \times B^e}$, 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.

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**|



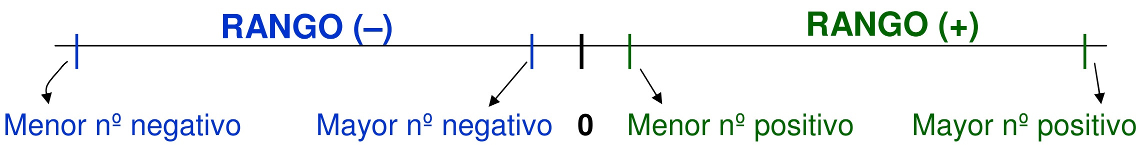
Ejemplo: Representar $-13'625_{(10)}$ en formato simplificado.

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

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

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 (+):

* Máximo nº → *Máx. mantisa:* 0'1111111; *Máx. exponente:* 7 → $0'1111111 \times 2^7 = (1-2^{-7}) \times 2^7 = 2^7-1 = 127$

* Mínimo nº → *Mín. mantisa:* 0'1000000 (*); *Mín. exponente:* -8 → $0'1 \times 2^{-8} = 2^{-9}$

(*) 0'1000000 (vs. 0'0000001).
Sería el mismo valor aunque ↑ nº de bits de la Mantisa.

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

iii) ***PRECISIÓN***:

- Análisis de nos grandes:

  * Máximo nº (+) → $0'1111111 \times 2^7 = 127_{(10)}$
  * Inmediato inferior (+) → $0'1111110 \times 2^7 = 126_{(10)}$

  Dist = 1

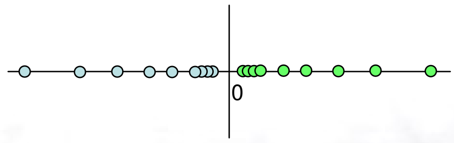
- Análisis de nos pequeños:

  * Mínimo nº (+) (excluyendo el 0) → $0'1000000 \times 2^{-8} = 2^{-9}_{(10)}$
  * Inmediato superior (+) → $0'1000001 \times 2^{-8} = (2^{-1} + 2^{-7}) \times 2^{-8} = 2^{-9} + 2^{-15}$

  Dist = 2^{-15}

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

Exponente ↑ ⟶ Precisión ↓
Exponente ↓ ⟶ Precisión ↑



Dist: distancia entre dos nos consecutivos.

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

❶ $12'75 \xrightarrow{\text{Convertir a binario}} 1100'11 \xrightarrow{\text{Normalizar TF}} 0'110011 \times 2^4$
$0'25 \longrightarrow 0'01 \longrightarrow 0'1 \times 2^{-1}$

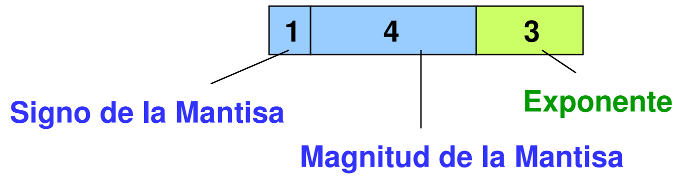
❷ $0'110011 \times 2^4$
$0'1 \times 2^{-1} \xrightarrow{\text{Igualar } -1 \text{ a } 4} 0'000001 \times 2^4$

❸
$$\begin{array}{r} 0'110011 \times 2^4 \\ 0'000001 \times 2^4 \\ \hline 0'110100 \times 2^4 \end{array}$$

❹ El resultado ya está normalizado: $0'110100 \times 2^4$

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.



Se pide:

a) Representar el nº $0'4_{(10)}$ *(Soluc: 0 1 1 0 0 0 1 1)*

b) ¿Cuántos nºs se pueden representar con este formato? *(Soluc: $2^7+1=128+1$)*

c) ¿Cuál es el error cometido al representar el nº 0'4? *(Soluc: 0'025)*

d) ¿Cuál es el mayor nº representable en este formato? *(Soluc: $0'1111 x 2^3 = 7'5$)*

e) ¿Cuál será el salto mínimo entre dos nºs 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.

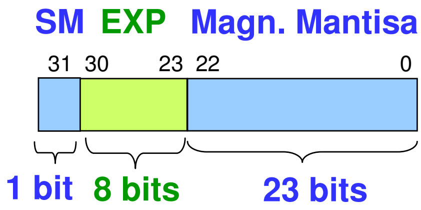
Soluc: 0 1 1 0 0 1 1 0 1 1 0 1

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 → 00000000
* El 128 en EXC a 127 sería: 128 + 127 = 255 → 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 → 00000001
...
* El 127 en EXC a 127 sería: 127 + 127 = 254 → 11111110
Rango de exponentes para representaciones normalizadas

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

* **Subformato Normalizado**: **R = ± 1 'xxx ... x 2^e^**

Bit implícito u oculto
(no se representa)

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

0'375 $\xrightarrow{\text{Convertir a binario}}$ 0'011 $\xrightarrow[\text{(1'x... x } 2^e)]{\text{Normalizar}}$ 1'1 x 2^{-2} *-2+127=125 = 01111101*

1	01111101	1 0 0 .... 0 0

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

(SOLUCIÓN: 0 10000010 1011010...0)

* **<u>Subformato *Desnormalizado*</u>:** *(exponentes 00000000 y 11111111)*

Exponente 00000000
- Mantisa todo 0s → 0
- Mantisa distinta de 0 → R = 0'M x 2^{-126}
  (M no tiene ninguna restricción de normalización)

Exponente 11111111
- Mantisa todo 0s →
  - Si signo (+) → + ∞
  - Si signo (-) → - ∞
- Mantisa distinta de 0 → Representación de errores (NaN).

<u>Ejemplo</u>: ¿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 x 2^{-126} = 0'10...0 x 2^{-126} = 2^{-127}

ii) ***<u>RANGO (simétrico):</u>***

- <u>Subformato Normalizado (+)</u>:

  * Máximo nº → *Máx. mantisa:* 1'1 ... 1
  Máx. exponente: 127
  → 1'1 ... 1 x 2^{127} = (2- 2^{-23}) x 2^{127}
  (≡ 0'1 ... 1 x 2^{128} = (1- 2^{-24}) x 2^{128})

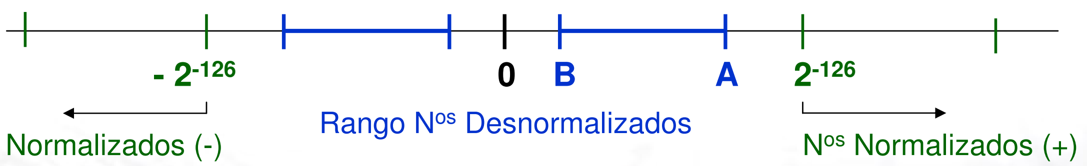
  * Mínimo nº → *Mín. mantisa:* 1'0 ... 0
  Mín. exponente: -126
  → 1'0 ... 0 x 2^{-126} = 2^{-126}

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

- <u>Subformato Desnormalizado</u>:

  * Máximo nº (+) → 0'1 ... 1 x 2^{-126} = (1 – 2^{-23}) x 2^{-126} = 2^{-126} - 2^{-149} **(A)**

  * Mínimo nº (+) → 0'0 ... 01 x 2^{-126} = 2^{-23} x 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 x 2^{-126}**
Exponente	**00000001 ... 11111110**	**- 1'0 x $2^{exp-127}$**	**+ 1'0 x $2^{exp-127}$**	**$\pm$ 1'M x $2^{exp-127}$**
Exponente	**11111111**	**- ∞**	**+ ∞**	**NaN**

Bit oculto

$\pm$ 1 'M x 2^e,

con e = 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 ...... 0 *(Soluc: 0)*
b) 0 0 1 1 1 1 1 1 1 0 0 ...... 0 *(Soluc: 1'0 x 2^0 = 1)*
c) 0 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 x 2^{-126})*
e) 1 1 0 0 0 0 0 0 1 0 0 ...... 0 *(Soluc: -1'0 x 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.

- Acrónimo de ***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange* (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.

TABLA DE CÓDIGOS ASCII

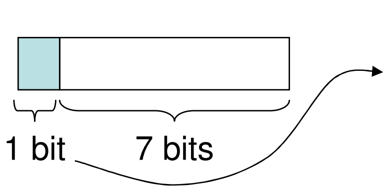
BITS b6 b5 b4 → / b3 b2 b1 b0 ↓	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		SÍMBOLOS NÚMEROS		MAYÚSCULAS		MINÚSCULAS	
0 0 0 0	0 NUL 0 0	16 DLE 10 20	32 SP 20 40	48 0 30 60	64 @ 40 100	80 P 50 120	96 ‘ 60 140	112 p 70 160
0 0 0 1	1 SOH 1 1	17 DC1 11 21	33 ! 21 41	49 1 31 61	65 A 41 101	81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

1 0 0 0	8 BS 8 10	24 CAN 18 30	40 (28 50	56 8 38 70	72 H 48 110	88 X 58 130	104 h 68 150	120 x 78 170
1 0 0 1	9 HT 9 11	25 EM 19 31	41) 29 51	57 9 39 71	73 I 49 111	89 Y 59 131	105 i 69 151	121 y 79 171
1 0 1 0	10 LF A 12	26 SUB 1A 32	42 * 2A 52	58 : 3A 72	74 J 4A 112	90 Z 5A 132	106 j 6A 152	122 z 7A 172
1 0 1 1	11 VT B 13	27 ESC 1B 33	43 + 2B 53	59 ; 3B 73	75 K 4B 113	91 [5B 133	107 k 6B 153	123 { 7B 173
1 1 0 0	12 FF C 14	28 FS 1C 34	44 , 2C 54	60 < 3C 74	76 L 4C 114	92 \ 5C 134	108 l 6C 154	124 | 7C 174
1 1 0 1	13 CR D 15	29 GS 1D 35	45 – 2D 55	61 = 3D 75	77 M 4D 115	93] 5D 135	109 m 6D 155	125 } 7D 175
1 1 1 0	14 SO E 16	30 RS 1E 36	46 . 2E 56	62 > 3E 76	78 N 4E 116	94 ^ 5E 136	110 n 6E 156	126 ~ 7E 176
1 1 1 1	15 SI F 17	31 US 1F 37	47 / 2F 57	63 ? 3F 77	79 O 4F 117	95 _ 5F 137	111 o 6F 157	127 DEL 7F 177

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.



Si bit = 0 → ASCII estándar (0-127)
Si bit = 1 → 128 nuevos códigos (128-255) en cada ampliación

- Aportaciones del juego ampliado de caracteres:
  - Caracteres extranjeros (ñ, Ñ, á, é, ¿, ç,…)
  - Caracteres de dibujo (└, ┴, ┬,…)
  - Caracteres científicos (¾,±,^{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	í	177	B1	▒	193	C1	┴	209	D1	╤	225	E1	ß	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	à	149	95	ò	165	A5	Ñ	181	B5	╡	197	C5	┼	213	D5	╒	229	E5	σ	245	F5	⌡
134	86	å	150	96	û	166	A6	ª	182	B6	╢	198	C6	╞	214	D6	╓	230	E6	µ	246	F6	÷
135	87	ç	151	97	ù	167	A7	º	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	Ω	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	¼	188	BC	╝	204	CC	╠	220	DC	▄	236	EC	∞	252	FC	^{n}
141	8D	ì	157	9D	¥	173	AD	¡	189	BD	╜	205	CD	═	221	DD	▌	237	ED	φ	253	FD	^{2}
142	8E	Ä	158	9E	₧	174	AE	«	190	BE	╛	206	CE	╬	222	DE	▐	238	EE	ε	254	FE	■
143	8F	Å	159	9F	ƒ	175	AF	»	191	BF	┐	207	CF	╧	223	DF	▀	239	EF	∩	255	FF	

1.6- Rep. de caracteres: El ISO 8859

- 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 ª	AB «	AC ¬	AD –	AE ®	AF ¯
B0 °	B1 ±	B2 ²	B3 ³	B4 ´	B5 µ	B6 ¶	B7 ·	B8 ¸	B9 ¹	BA º	BB »	BC ¼	BD ½	BE ¾	BF ¿
C0 À	C1 Á	C2 Â	C3 Ã	C4 Ä	C5 Å	C6 Æ	C7 Ç	C8 È	C9 É	CA Ê	CB Ë	CC Ì	CD Í	CE Î	CF Ï
D0 Ð	D1 Ñ	D2 Ò	D3 Ó	D4 Ô	D5 Õ	D6 Ö	D7 ×	D8 Ø	D9 Ù	DA Ú	DB Û	DC Ü	DD Ý	DE Þ	DF ß
E0 à	E1 á	E2 â	E3 ã	E4 ä	E5 å	E6 æ	E7 ç	E8 è	E9 é	EA ê	EB ë	EC ì	ED í	EE î	EF ï
F0 ð	F1 ñ	F2 ò	F3 ó	F4 ô	F5 õ	F6 ö	F7 ÷	F8 ø	F9 ù	FA ú	FB û	FC ü	FD ý	FE þ	FF ÿ

LEYENDA:

HEX
CAR

HEX: código hexadecimal
CAR: carácter

Código ISO-Latin-2

A0	A1 Ą	A2 ˘	A3 Ł	A4 ¤	A5 Ľ	A6 Ś	A7 §	A8 ¨	A9 Š	AA Ş	AB Ť	AC Ź	AD -	AE Ž	AF Ż
B0 °	B1 ą	B2 ˛	B3 ł	B4 ´	B5 ľ	B6 ś	B7 ˇ	B8 ¸	B9 š	BA ş	BB ť	BC ź	BD ˝	BE ž	BF ż
C0 Ŕ	C1 Á	C2 Â	C3 Ă	C4 Ä	C5 Ĺ	C6 Ć	C7 Ç	C8 Č	C9 É	CA Ę	CB Ë	CC Ě	CD Í	CE Î	CF Ď
D0 Đ	D1 Ń	D2 Ň	D3 Ó	D4 Ô	D5 Ő	D6 Ö	D7 ×	D8 Ř	D9 Ů	DA Ú	DB Ű	DC Ü	DD Ý	DE Ţ	DF ß
E0 ŕ	E1 á	E2 â	E3 ă	E4 ä	E5 ĺ	E6 ć	E7 ç	E8 č	E9 é	EA ę	EB ë	EC ě	ED í	EE î	EF ď
F0 đ	F1 ń	F2 ň	F3 ó	F4 ô	F5 ő	F6 ö	F7 ÷	F8 ř	F9 ů	FA ú	FB ű	FC ü	FD ý	FE ţ	FF ˙

LEYENDA: HEX / CAR

HEX: código hexadecimal
CAR: carácter

- 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.

- 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, arábico, griego, …, egipcio, maya, cuneiforme.

- 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	**00**10 0010 0110 0000
7	Introducir los bits en el esquema	1110 0010 10 001001 10 100000

1.6- Representación de caracteres: Unicode VI – UTF 8

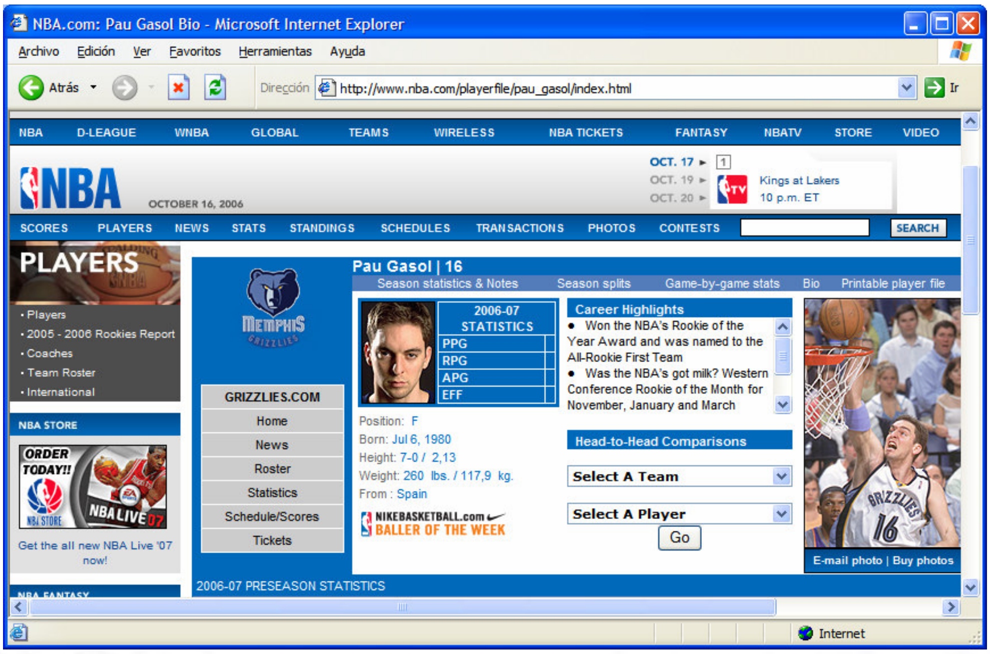
- Ejemplo 1**: codificación del carácter alef (א) (U+05D0):**
  - Solución: 110**10111** 10**010000** (D7 $90_{(16)}$)

- Ejemplo 2**: codificación del copyright (©) (U+00A9):**
  - Solución: 110**00010** 10**101001** (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: 110**00011** 10**110001** 01**100001** (C3 B1 $61_{(16)}$)

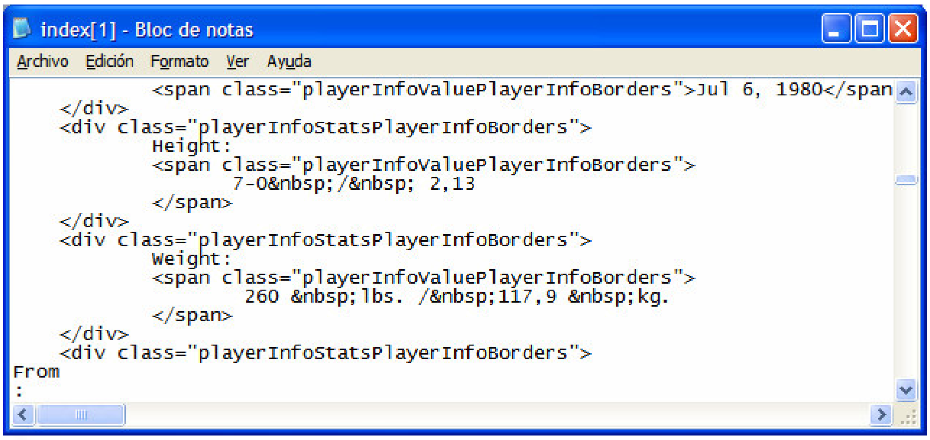
1.6- Ejemplos de codificación de caracteres I

- ¿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).

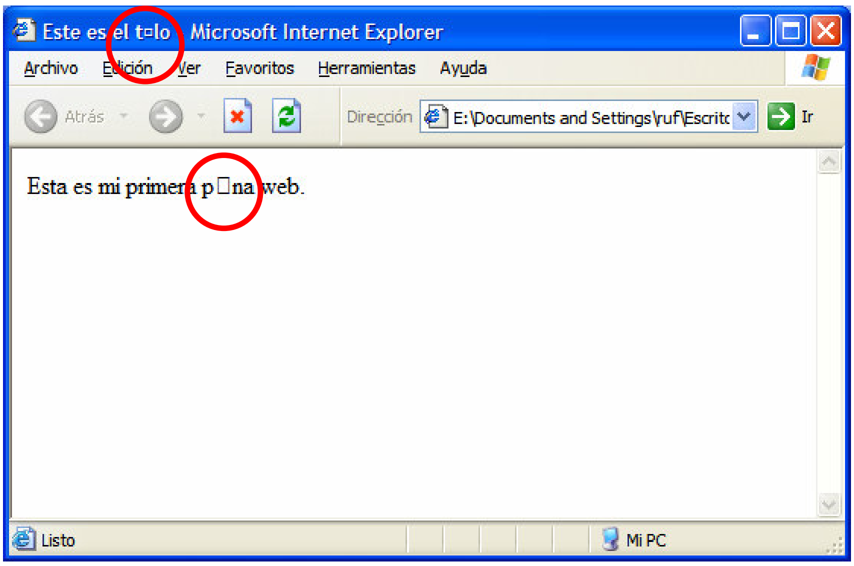



```
            <span class="playerInfoValuePlayerInfoBorders">Jul 6, 1980</span
    </div>
    <div class="playerInfoStatsPlayerInfoBorders">
            Height:
            <span class="playerInfoValuePlayerInfoBorders">
                7-0 /  2,13
            </span>
    </div>
    <div class="playerInfoStatsPlayerInfoBorders">
            Weight:
            <span class="playerInfoValuePlayerInfoBorders">
                260  lbs. / 117,9  kg.
            </span>
    </div>
    <div class="playerInfoStatsPlayerInfoBorders">
From
:
```

- 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>
```


- 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>
```

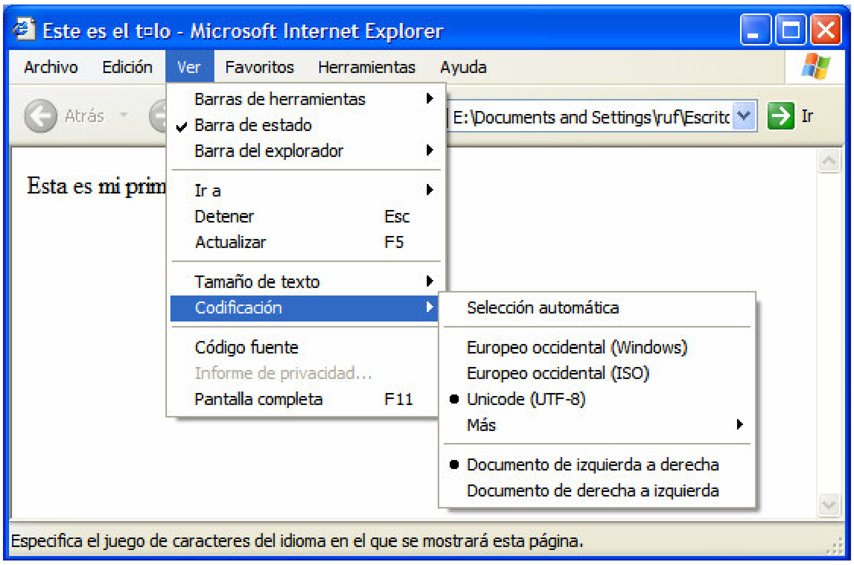

1.6- Ejemplos de codificación de caracteres VI

- 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

```
...
00111100011010000111010001101101011011000
00111110110111011010101010011001001111000
01101000011001010110000101100100001111100
11011101101010101001100110011001001111000
01110100011010010111010001101100011001010
00111110110111011010101010011001100110010
10011001010001010111001101110100011001010
...
```

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



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

- 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.