Tema 2: Representación Digital de la Información

Jorge Juan Chico <jjchico@dte.us.es>, Julián Viejo Cortés <julian@dte.us.es> 2011-17
Departamento de Tecnología Electrónica
Universidad de Sevilla

Contenidos

- Introducción a la codificación digital
- Unidades digitales
- Representación de números naturales
- Representación de números enteros
- Representación de números reales/racionales

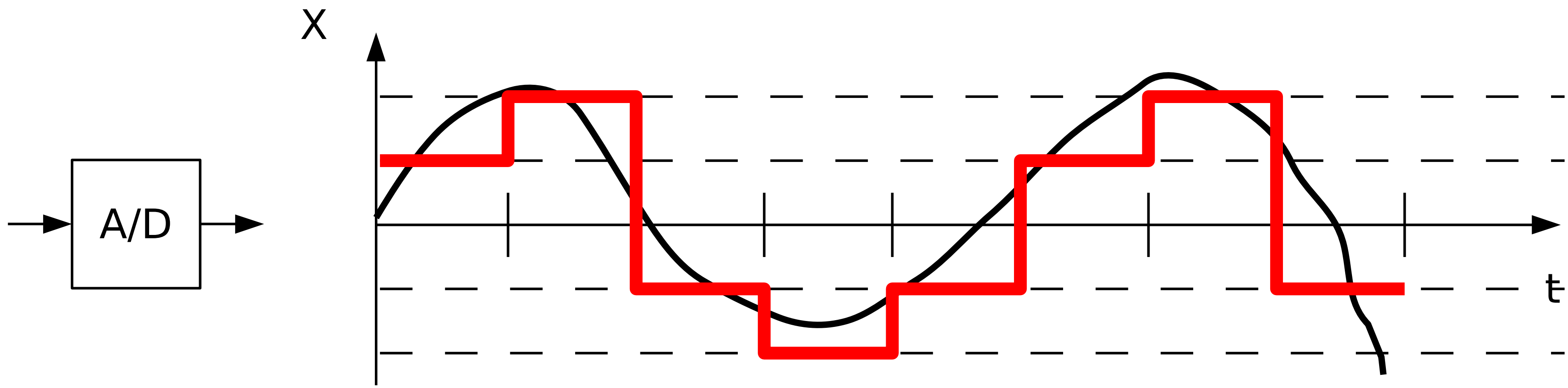
Bibliografía

- Complementaria
  - Apuntes representación en complemento a 2
    - Demostraciones de las propiedades de la representación posicional y en complemento a 2.

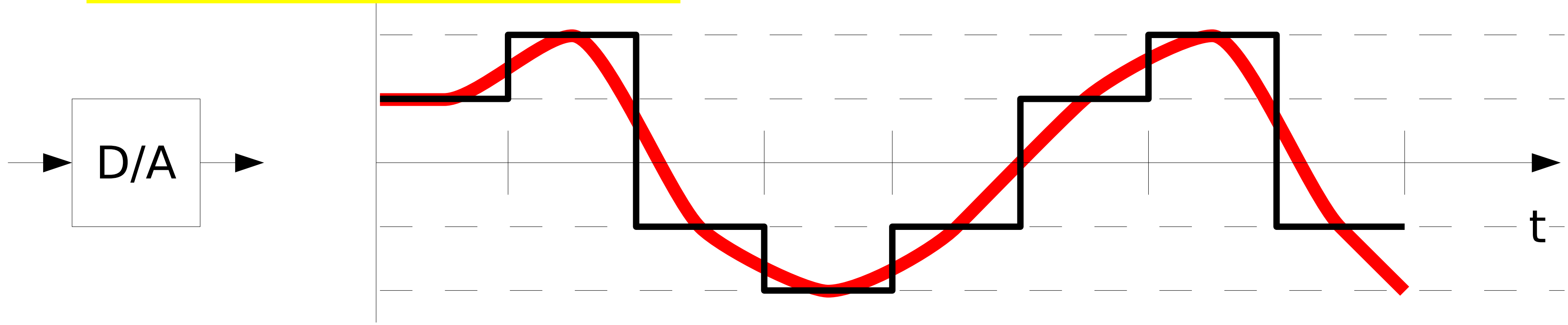
Introducción
Codificación digital

- Los circuitos digitales con los que se construyen los ordenadores trabajan con señales bivaluadas
  - Valores posibles en el conjunto {0,1}
- Los computadores se emplean para almacenar todo tipo de información:
  - números enteros, reales, texto, gráficos, audio, video, etc.
- Esta información ha de traducirse a los símbolos del conjunto {0,1} para poder ser procesada por un ordenador
- Codificación digital:
  - Proceso por el cual cualquier tipo de información se representa numéricamente.
  - Posteriormente estos números se codifican con {0,1}

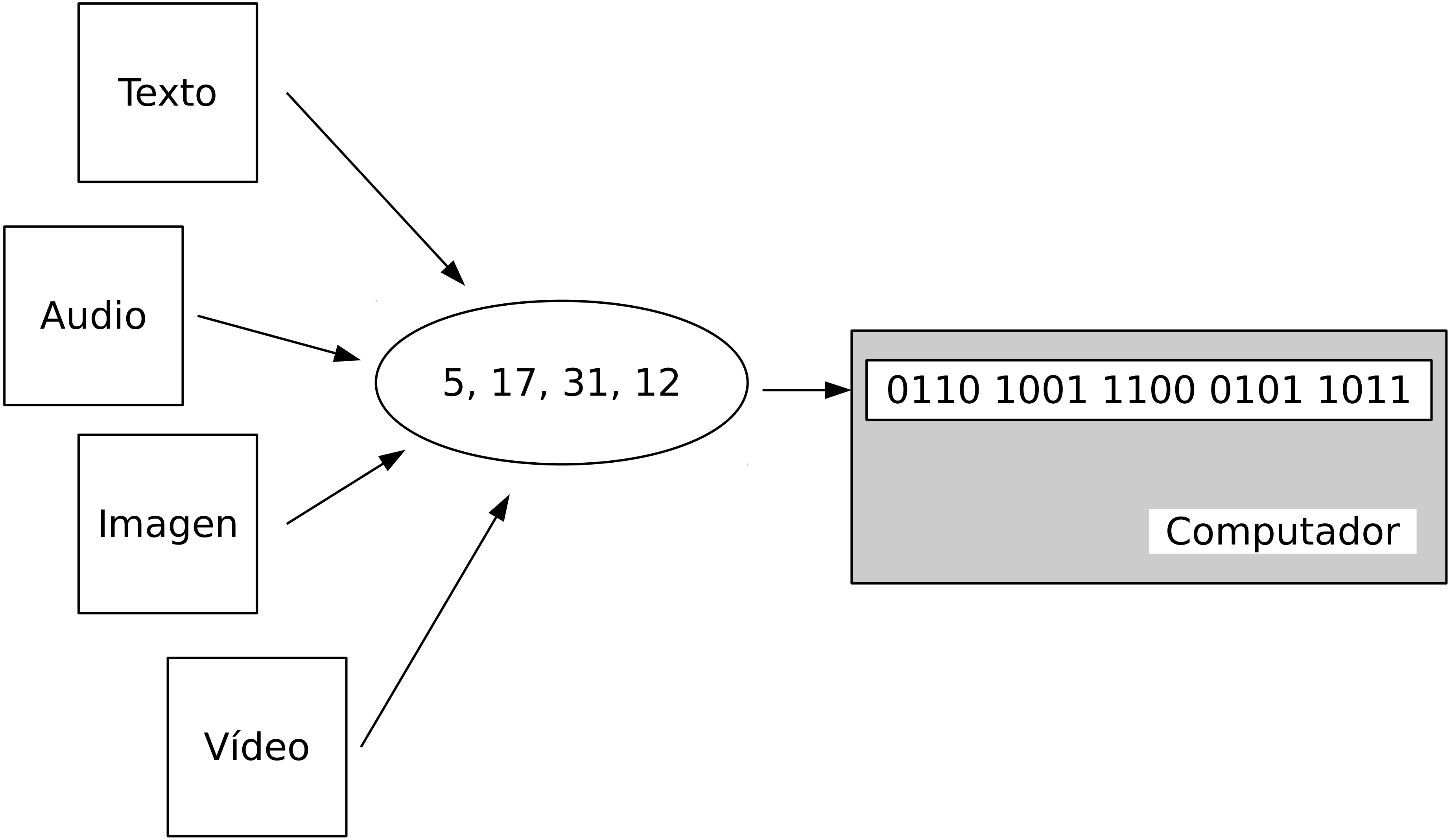
Conversión A/D y D/A



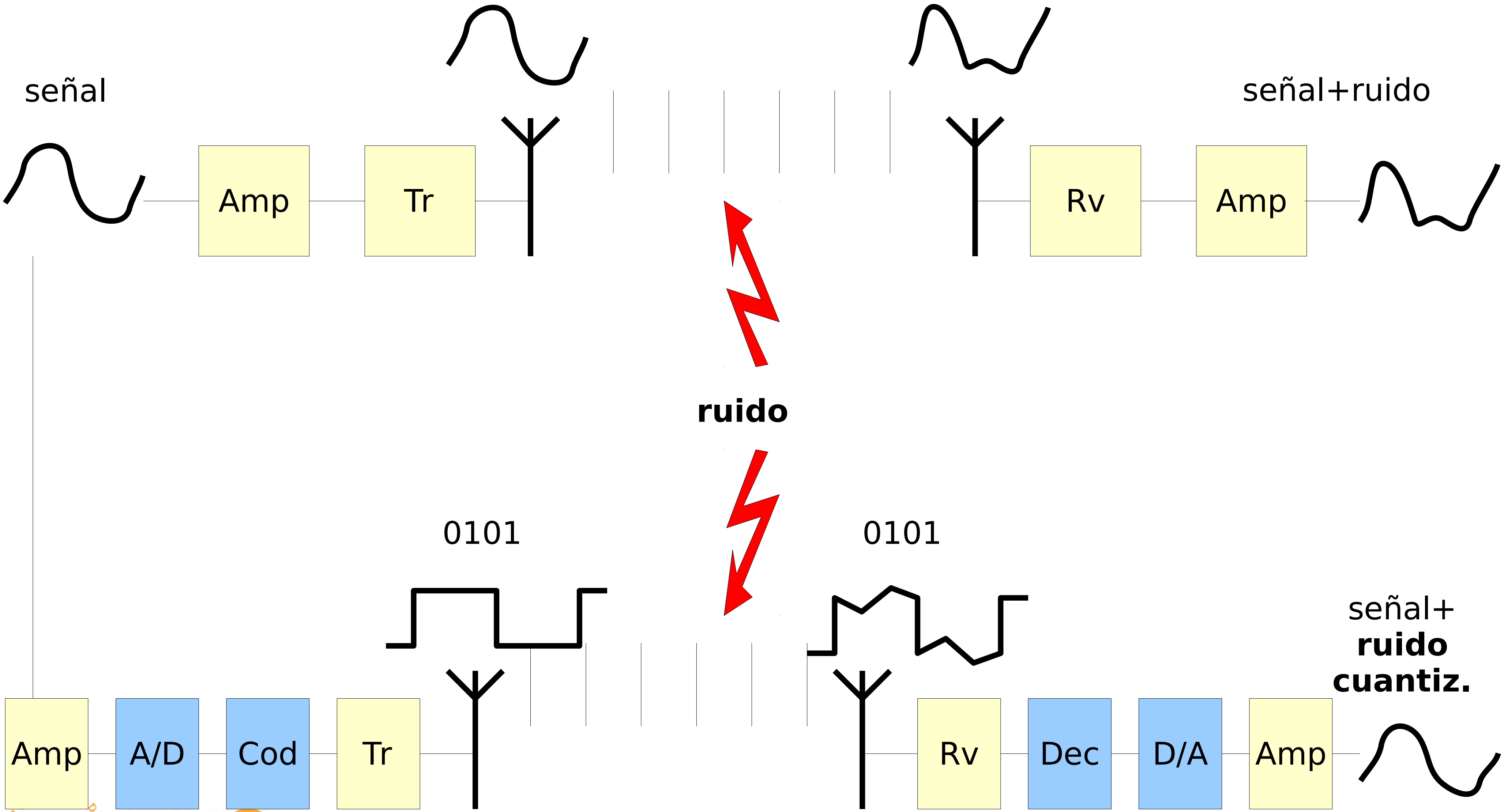
Error de cuantización



Codificación digital



Ejemplo


señal
señal+ruido
Amp
Tr
Rv
Amp
ruido
0101
0101
señal+
ruido
cuantiz.
Amp
A/D
Cod
Tr
Rv
Dec
D/A
Amp

Introducción
Codificación digital

- Ventajas de la codificación digital
  - Mayor facilidad de diseño y fabricación de equipos
  - Posibilidad de transmisión sin pérdida de calidad (salvo cuantización)
  - Más opciones de tratamiento de la información:
    - compresión, detección/corrección de errores, almacenamiento, etc.
  - Tratamiento homogéneo de la información: sonido, imagen, texto, etc.
- Inconvenientes
  - Error de cuantización (pero es controlable)
  - Necesario etapas de conversión:
    - Analógico -> Digital
    - Digital -> Analógico

Unidades digitales

- BIT (b) (BInary digiT)
  - Símbolo del conjunto {0,1}
  - Unidad mínima de información
- Palabra
  - Conjunto de 'n' bits, típicamente 4, 8, 16, 32 o 64.
  - Los ordenadores operan con palabras completas
- Nibble – Cuarteto (¿quién usa esto?)
  - Palabra de 4 bits
- Byte – Octeto (B)
  - Palabra de 8 bits
  - Unidad base en computación y telecomunicaciones

Unidades digitales

- Tradición: Unidades del SI con significado ligeramente modificado (potencias de 2 en vez de 10)
- Uso no uniforme de unidades digitales:
  - Diskette: 1.44MB = 1000 KB = 1000x1024B
  - Discos 160GB = 160000 MB = 160x1000x1024x1024B
  - DVD 4,7GB = 4700MB = 4,7x1000x1024x1024B
- Estándar para unidades digitales binarias (no muy usado):
  - IEC, IEEE-1541-2002

SI			**Binary**	**IEC**	
kilo	k	10^3	2^{10}	kibi	Ki
mega	M	10^6	2^{20}	mebi	Mi
giga	G	10^9	2^{30}	gibi	Gi
tera	T	10^{12}	2^{40}	tebi	Ti
peta	P	10^{15}	2^{50}	pebi	Pi
exa	E	10^{18}	2^{60}	exbi	Ei
zetta	Z	10^{21}	2^{70}	zebi	Zi

Número naturales. Sistemas de numeración. Bases

- El sistema decimal común es un sistema de numeración posicional que emplea 10 símbolos y donde la base es 10:
  - Símbolos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

$$1327 = 1 \times 10^3 + 3 \times 10^2 + 2 \times 10^1 + 7 \times 10^0$$

Pesos:	1000	100	10	1
Símbolos:	1	3	2	7
Valor:	1000	300	20	7

Suma

1327

Número naturales. Base 2

- Los sistemas digitales pueden representar de forma "natural" números en base 2, usando los símbolos {0,1}
- Ej: 1101

$$1101 = 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0$$

Pesos:	8	4	2	1
Símbolos:	1	1	0	1
Valor:	8	4	0	1

Suma

13

Números naturales. Base “b”

- Lo mismo aplicado a una base genérica “b”
  - x: magnitud,   b: base
  - n: número de cifras,   {xi}: cifras

$$x = x_{n-1} \times b^{n-1} + \ldots + x_1 \times b^1 + x_0 \times b^0$$

- Mayor número representable con n cifras: b^n-1
- El cambio de base b a base 10 se realiza aplicando directamente la fórmula anterior con las cifras del número en base b.

Números naturales. Cambio de base 10 a base “b”

- El cambio de base 10 a una base cualquiera b puede realizarse dividiendo sucesivamente la magnitud por la base y extrayendo los restos

$$123_{(10} \longrightarrow 234_{(7}$$

1 2 3 | 7

5 3   1 7 | 7

4   3   2

Octal y hexadecimal

- Base 8 (octal):
  - {0, 1, 2, 3, 4, 5, 6, 7}
- Base 16 (hexadecimal):
  - {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
- Formas compactas de representar números binarios
  - 1 cifra octal = 3 cifras binarias
  - 1 cifra hexadecimal = 4 cifras binarias

Octal y hexadecimal

B-8	B-2
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

B-10	B-16	B-2
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Octal y hexadecimal

1	2	7	2		$\rightarrow$	$1272_{(8}$

1 0 1 0 1 1 1 0 1 0

2	B	A		$\rightarrow$	$2BA_{(16}$

$2BA_{(16} = 2BA_{h} = 2BA_{hex}$ = #2BA = $2BA = 0x2BA = 10'h2BA

Números enteros. Representación signo-magnitud

signo	magnitud								
0	1	0	1	1	0	1	0	→	$+90_{(10}$
1	0	1	0	1	0	0	1	→	$-41_{(10}$

- Signo: 0(+), 1(-)
- Representable con n bits: 2^n-1
- Representaciones del “0”: 00000000, 10000000

$$-\left(2^{n-1}-1\right)\leqslant x\leqslant 2^{n-1}-1$$

Números enteros
Representación en exceso

- Se representa en base 2 el resultado de sumar al número el valor del “exceso” o “sesgo”.
- El resultado de sumar el “exceso” debe ser un entero positivo. Esto define el rango de números representables.
- Ej: exceso 2n-1 (números de n bits, ej: 8 bits)
  - $-35_{(10} \rightarrow -35 + 128 = 93 = 01011101_{(exc\text{-}128}$

$$-2^{n-1} \leqslant x \leqslant 2^{n-1} - 1$$

Números enteros
Rep. Complemento a 2

- El primer bit indica el signo: 0(+), 1(-)
- Una sola representación del cero: 00000...0

$$x = -x_{n-1} \times b^{n-1} + \ldots + x_1 \times b^1 + x_0 \times b^0$$

Pesos:	-8	4	2	1
Símbolos:	1	1	0	1
Valor:	-8	4	0	1

$\rightarrow$ Suma: -3

$$\boxed{-2^{n-1} \leqslant x \leqslant 2^{n-1} - 1}$$

Números enteros. Ca2: extensión del signo

- Al extender el número de bits de un número codificado en Ca2, los bits extendidos toman todos el mismo valor que el antiguo bit de signo.

-8	4	2	1	
1	1	0	1	
-8	4	0	1	→ -3

-16	8	4	2	1	
1	1	1	0	1	
-16	8	4	0	1	→ -3

-128	64	32	16	8	4	2	1	
1	1	1	1	1	1	0	1	
-128	64	32	16	8	4	0	1	→ -3

Números enteros. Ca2: Propiedad 1

- Si en la representación en Ca2 de una cantidad entera x se complementan todos los bits y, tratando el resultado como un número binario sin signo, se le suma 1, el resultado es la representación en Ca2 de -x

Números enteros. Ca2: Propiedad 2

- Si las representaciones en Ca2 de dos cantidades enteras x e y se suman, tratándolas como enteros binarios sin signo y despreciando el posible acarreo, el resultado es la representación en Ca2 de la cantidad x+y, salvo que se produzca desbordamiento.

El mismo sumador de binarios naturales sirve para enteros representados en Ca2

Números enteros. Ca2: Propiedad 3

- (Regla de desbordamiento): Si dos cantidades binarias representadas en Ca2, ambas con el mismo signo, se suman tratándolas como enteros binarios sin signo, se produce desbordamiento si el signo del resultado, interpretado en Ca2 es distinto al signo de las cantidades sumadas.

Números enteros
Ca2: Ejemplos

```
 1001 = -7          1100 = -4          0011 = +3
 0101 = +5          0100 = +4          0100 = +4
-----              -----              -----
 1110 = -2         10000 =  0          0111 = +7


 1100 = -4          0101 = +5          1001 = -7
 1111 = -1          0100 = +4          1010 = -6
-----              -----              -----
11011 = -5          1001 = -7         10011 = +3
```

¡Desbordamiento!

Números enteros. Resumen

x	s-m	Ca2	exc. 2^{n-1}
-8	-	1000	0000
-7	1111	1001	0001
-6	1110	1010	0010
-5	1101	1011	0011
-4	1100	1100	0100
-3	1011	1101	0101
-2	1010	1110	0110
-1	1001	1111	0111
0	0000/1000	0000	1000
1	0001	0001	1001
2	0010	0010	1010
3	0011	0011	1011
4	0100	0100	1100
5	0101	0101	1101
6	0110	0110	1110
7	0111	0111	1111

Números fraccionarios

$$x = x_{n-1} \times b^{n-1} + \ldots + x_0 \times b^0 + x_{-1} \times b^{-1} + \ldots + x_{-m} \times b^{-m}$$

$$\overbrace{0\quad 1\quad 1\quad 0\quad 1}^{\text{parte entera}} \bullet \overbrace{1\quad 0\quad 1}^{\text{parte fraccionaria}}$$

Números fraccionarios

- Conversión de base *b* a base 10:
  - Directamente: basta operar en base 10

$$10{,}101_2 = 1\times 2^1 + 0\times 2^0 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3}$$
$$2 + 1/2 + 1/8 = 2{,}625_{10}$$

- Conversión de base 10 a base *b*:
  - Parte entera: como con números enteros
  - Parte fraccionaria: multiplicaciones sucesivas por la base objeto. Se toma la parte entera del resultado

Números fraccionarios

- Ejemplo: $12{,}3_{(10}$
  - $12_{(10} = 1100_{(2}$
  - 0,3 x 2 = 0,6 → "0"
  - 0,6 x 2 = 1,2 → "1"
  - 0,2 x 2 = 0,4 → "0"
  - 0,4 x 2 = 0,8 → "0"
  - 0,8 x 2 = 1,6 → "1"
  - 0,6 x 2 = 1,2 → "1" (primer bit repetido)
- $12{,}3_{(10} = 1100{,}0100110011001\ldots_{(2} = 1100{,}0\overline{1001}_{(2}$
- Al cambiar de base, un número puede pasar de tener un número finito a infinito de cifras (y viceversa).
- ¿Podrían las cifras no repetirse periódicamente?
- ¿Cuándo un número tendrá un número infinito de cifras en una base dada?

Números fraccionarios

- Representación en punto fijo
  - Número constante de cifras para la parte fraccionaria.
  - Problemas para representar números muy grandes o muy pequeños.
- Representación en punto flotante
  - Se representan las cifras significativas del número.
  - La posición del punto (coma decimal) la determina un exponente.
  - Representación de números muy pequeños o muy grandes a costa de tener un precisión no uniforme.

Rep. en punto flotante

$$1.23 \times 10^{12} \qquad\qquad x = M \times B^{E}$$

- Equivalente a la notación científica decimal:
  - M: mantisa, B: base, E: exponente
- Flexibilidad: permite representar números muy grandes y muy pequeños
- Introduce error de cuantización
  - La precisión depende del valor del número representado: a mayor valor, menor la precisión.

Notación IEEE-754 (parcial)

0	1 ... 8	9 ... 31
1	8	23
signo	exponente (E*)	mantisa (M*)

- Base: 2
- Signo (1 bit): 0 -> +, 1 -> -
- Exponente (8 bits): sesgado, con un sesgo de 127
  - E*: valor almacenado E*=1...254
  - E: exponente real E=E*-127=-126...127

Notación IEEE-754 (parcial)

0	1 … 8	9 … 31
1	8	23
signo	exponente (E*)	mantisa (M*)

- Mantisa (23 bits): normalizada, parte entera = 1
  - M = 1,bbb...b
  - M* = bbb...b (23 bits)

Notación IEEE-754 (parcial)

- Mayor número representable: $\left(2-2^{-23}\right)\times 2^{127}$
- Menor número representable: $-\left(2-2^{-23}\right)\times 2^{127}$
- Menor número positivo representable: 2^{-126}
- Mayor número negativo representable: -2^{-126}

Notación IEEE-754 (parcial)

- Algunos casos especiales:
  - cero: E*=0, M*=0
  - Infinito: E*=255, M*=0
    - s=0 -> +Inf
    - s=1 -> -Inf
  - números no normalizados: E*=0, M*$\neq$0
    - E = -126
    - M=M*

Punto flotante. Paso a base 10

0 1	8 9	31
1	8	23
signo	exponente (E*)	mantisa (M*)

- Se obtiene M a partir de M* y el signo
- Se obtiene E a partir de E*:
  - E = E*-127
- Se hacen las operaciones

$$x = M \times 2^{E}$$

Punto flotante. Paso a base 10

- Ejemplo:

  0 10010100 10100010000000000000000

  - signo: 0 -> +
  - Mantisa (M) = $1{,}1010001_{(2} = 1{,}6328125_{(10}$
  - $E^* = 10010100_{(2} = 148_{(10}$; $E = E^* - 127 = 21$

$$x = 1{,}6328125 \times 2^{21} = 3424256$$

Punto flotantes. Paso desde base 10

- Se obtiene una estimación del exponente: E'
- Se elige como exponente (E) la parte entera de E'
- Se calcula el valor de la mantisa (M).
- Se calculan E* y M* y se pasan a binario.
- Se construye la palabra binaria.

$$x = M \times 2^{E}$$

$$E' = \log_2 x$$

$$E = ent(E')$$

$$M = \frac{x}{2^{E}}$$

Punto flotantes. Paso desde base 10

Ejemplo: +3424256

$$E' = \log_2 3424256 = 21.707$$

$$E = ent(21.707) = 21$$

$$M = \frac{3424256}{2^{21}} = 1.6328125$$

- signo + : “0”
- E* = 127+21 = 148 = $10010100_{(2}$
- M = 1,6328125 = $1.1010001_{(2}$
- M* = “101001000...00”

0 10010100 10100100000000000000000

Números reales. Implicaciones para los sistemas digitales

- Con un número limitado de cifras no es posible representar número reales irracionales de forma exacta.
- No podemos obtener una representación exacta en base 2 de muchos números que sí se pueden representar exactamente en base 10.
- Potencial fuente de graves errores, incluso a nivel software.
- Ejemplo: representación en punto fijo con 8 bits y 4 bits para la parte fraccionaria:

$$12{,}3_{10} = 1100{,}0\,\widehat{1001}_2 \approx 1100{,}0100_2 = 12{,}25_{10}$$

Números reales. Implicaciones para los sistemas digitales

```
$ python3

>>> x = 12.3     # el valor se almacena internamente en b. 2
>>> y = 3 * x    # las operaciones se realizan en b. 2
>>> z = y / 3

>>> x == z       # !?
False

>>> z - x
1.7763568394002505e-15

>>> from decimal import Decimal

>>> Decimal(x)   # rep. en b.10 del valor almacenado de x
Decimal('12.300000000000000710542735760100185871124267578125')
>>> Decimal(z)
Decimal('12.3000000000000024868995751603506505489349365234375')

>>> if x != z:   # kaboooom!
...     print("Destruir el mundo!!!")
...
Destruir el mundo!!!
```

Anexo: cambios de bases en Python

```
$ python3

>>> x = 215
>>> hex(x)          # repr. hexadecimal
'0xd7'
>>> bin(x)          # repr. base 2
'0b11010111'

>>> y = 0b1011      # valor binario
>>> y
11
>>> z = 0x2c        # valor hexadecimal
>>> z
44

>>> x + y
226
>>> hex(x+y)
'0xe2'
>>> bin(x+y)
'0b11100010'

>>> x - z
171
>>> hex(x-z)
'0xab'
>>> bin(x-z)
'0b10101011'
```

```
>>> int('321',7)          # base 7
162

# 321(7 + 121(3
>>> x = int('321',7) + int('212',3)

>>> x
185
>>> hex(x)
'0xb9'
>>> bin(x)
'0b10111001'

# repr. en base n?

>>> from numpy import base_repr
>>> x = 52
>>> base_repr(x, 2)
'110100'
>>> base_repr(x, 3)
'1221'
>>> base_repr(x, 4)
'310'
>>> base_repr(x, 5)
'202'
>>> base_repr(x, 6)
'124'
>>> base_repr(x, 7)
'103'
```