Bloque I. Introducción.
Tema 2. Sistema binario de representación numérica

Grado en Ingeniería Informática.
Fundamentos de Computadores.

Universidad Rey Juan Carlos

Mª Jesús Algar Díaz

Índice

1. Sistemas de numeración de base fija

- Sistema numérico: Conjunto ordenado de símbolos (cifras o dígitos) + Reglas para +, -, x, /
- Base: nº de cifras o dígitos que utiliza un sist. numérico.
- Ejemplos:
  - Sist. Decimal (base=10) → {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
  - Sist. Binario (base = 2) → {0, 1}
  - Sist. Hexadecimal (base = 16) → {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
  - Sist. Octal (base = 8) → {0, 1, 2, 3, 4, 5, 6, 7}

1. Sistemas de numeración de base fija

- En cualquier sist. de numeración:

$$N = (a_{p-1}\ a_{p-2} \ldots a_1\ a_0\ ,\ a_{-1}\ a_{-2} \ldots a_{-q})_r$$

a_i son los dígitos,
p es el número de dígitos enteros,
q es el número de dígitos fraccionarios,
a_{p-1} es el dígito más significativo,
a_{-q} es el dígito menos significativo.

- Ejemplo:
  - $(107,45)_{10}$
  - $(1A3,B)_{16}$
  - $(101110,11)_2$
  - $(1473,13)_8$

- **Sist. de numeración posicional**.

1. Sistemas de numeración de base fija

$$(123{,}54)_{10} = 1\cdot 10^2 + 2\cdot 10^1 + 3\cdot 10^0 + 5\cdot 10^{-1} + 4\cdot 10^{-2}$$

- La posición de cada dígito tiene un peso (r^i).
- Valor de a_i (dígito x peso): $a_i \cdot r^i$
- Número = suma de valores de todos sus dígitos

$$N = \sum_{i=-q}^{p} a_i \cdot r^i$$

- Representación **única.**
  - MSD, LSD
- **Notación polinomial**.

2. Conversión entre bases

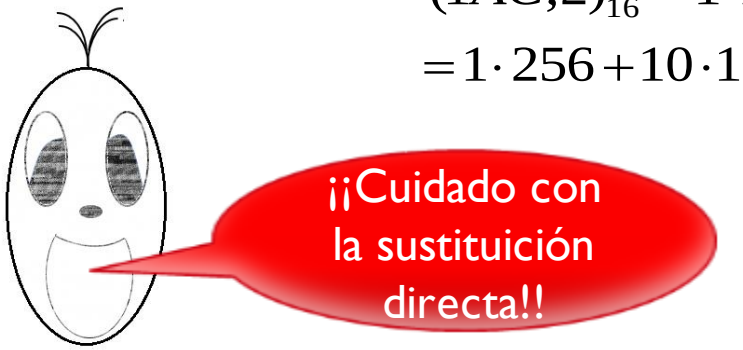
- Dos métodos:
  - Sustitución en serie: nº en base *r* a base d (decimal).

•Convertir el número $(101,01)_2$ a decimal.

$$(101,01)_2 = 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 0\cdot 2^{-1} + 1\cdot 2^{-2} =$$
$$= 4 + 0 + 1 + 0 + 0,25 = (5,25)_{10}$$

•Convertir el número $(1AC,2)_{16}$ a decimal.

$$(1AC,2)_{16} = 1\cdot 16^2 + A\cdot 16^1 + C\cdot 16^0 + 2\cdot 16^{-1} =$$
$$= 1\cdot 256 + 10\cdot 16 + 12\cdot 1 + 2\cdot 0,0625 = (428,125)_{10}$$



$(1AC,2)_{16} \neq 1+10+12+0,2 = (23,2)_{10}$

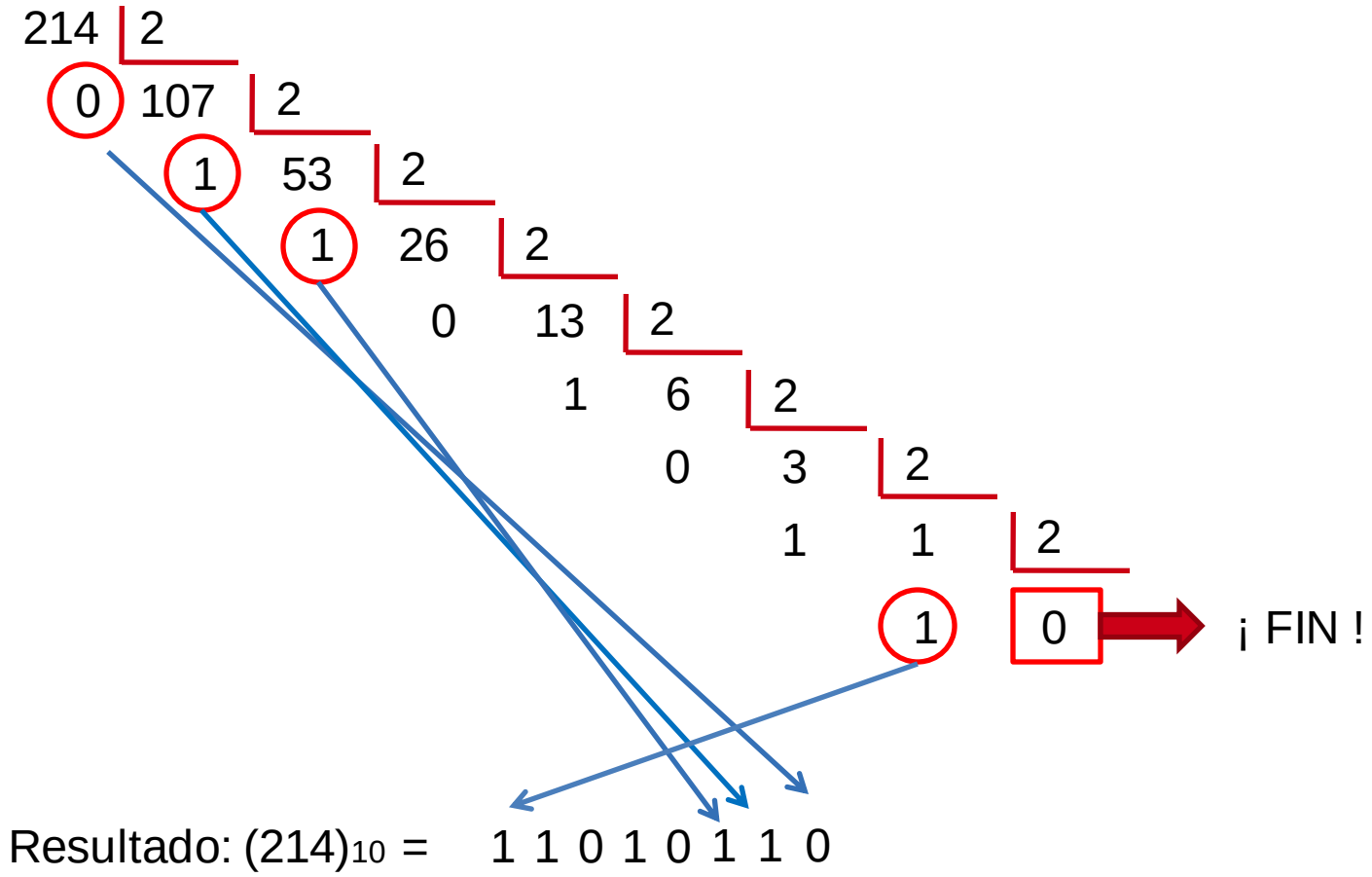
2. Conversión entre bases

- División/multiplicación por la base: nº en base d (decimal) a base s.

  - Procedimiento parte entera:
    1. Dividir parte entera entre *s* → resto = dígito a_0 de la parte entera.
    2. Dividir cociente entre *s* → resto = dígito a_1 de la parte entera (repetir paso 1).
    3. cociente = 0 ⇨ FIN
  - Procedimiento parte fraccionaria:
    1. Multiplicar parte fraccionaria por s = nuevo nº con parte entera (a_{-1}) y fraccionaria.
    2. Multiplicar nueva fracción (repetir paso 1).
    3. Parte fraccionaria = 0 o suficientes dígitos ⇨ FIN

2. Conversión entre bases

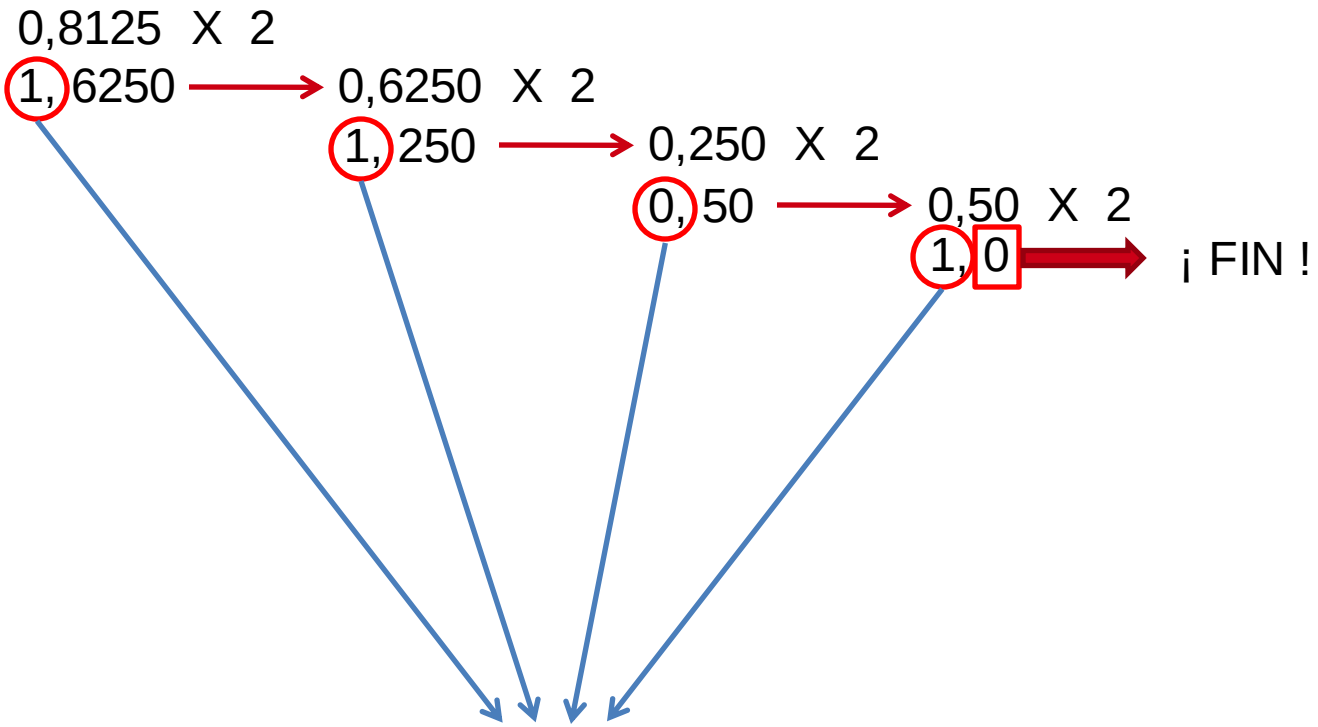
- Ejemplo: convertir el número $(214)_{10}$ a binario.



Resultado: $(214)_{10}$ = 1 1 0 1 0 1 1 0

2. Conversión entre bases

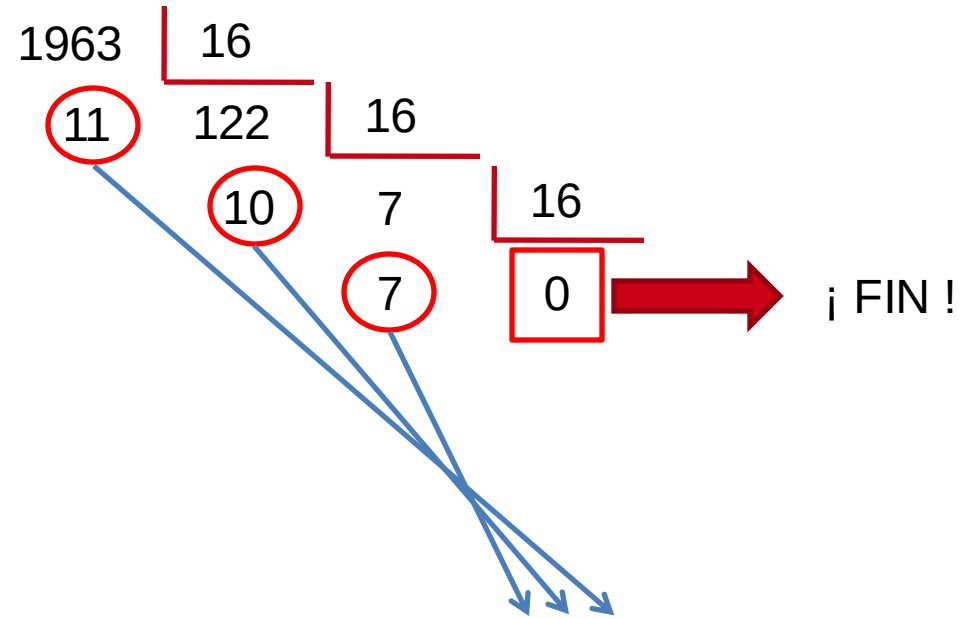
- Ejemplo: convertir el número $(0{,}8125)_{10}$ a binario.



Resultado: $(0{,}8125)_{10} = 0,1\,1\,0\,1$

2. Conversión entre bases

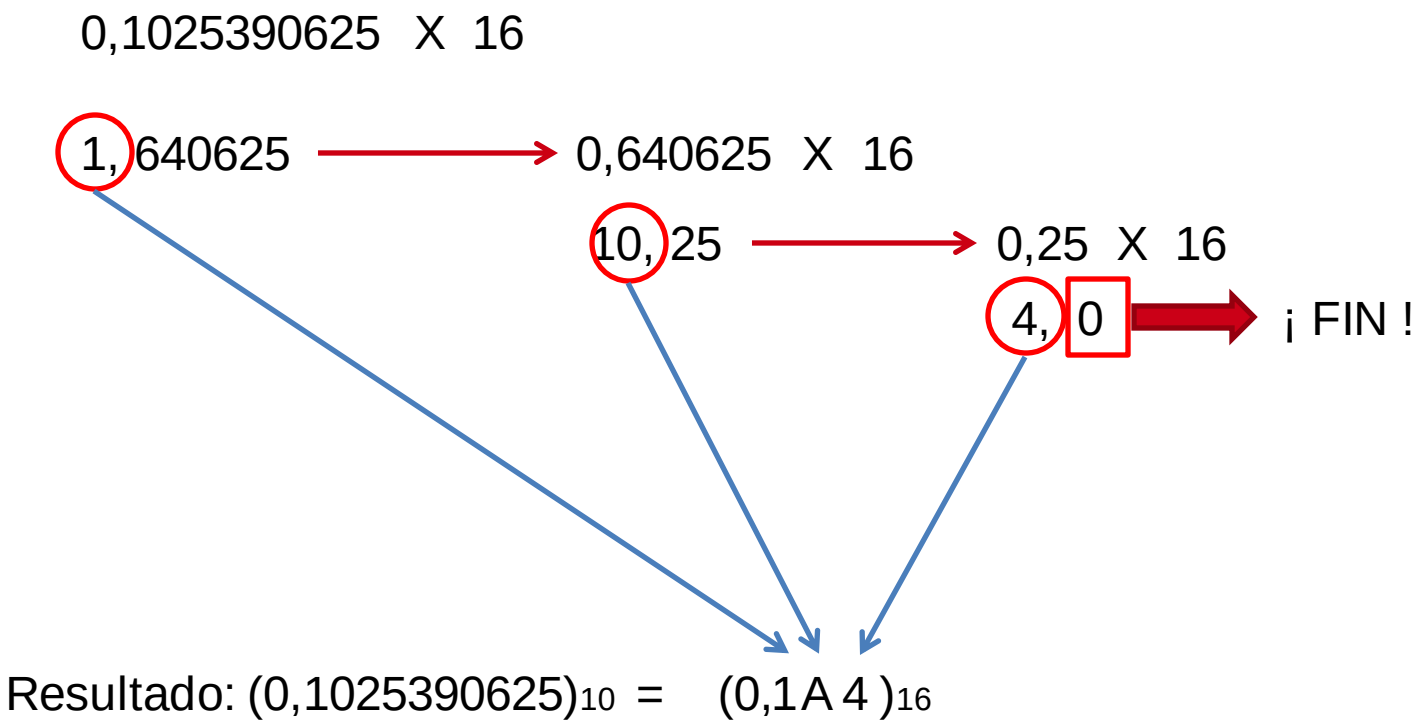
- Ejemplo: convertir el número $(1963)_{10}$ a hexadecimal.



Resultado: $(1963)_{10} = (7\ A\ B)_{16}$

2. Conversión entre bases

- Ejemplo: convertir el número $(0{,}1025390625)_{10}$ a hexadecimal.



Resultado: $(0{,}1025390625)_{10} = (0{,}1\,A\,4\,)_{16}$

2. Conversión entre bases

- Correspondencia de los 15 primeros número en distintas bases:

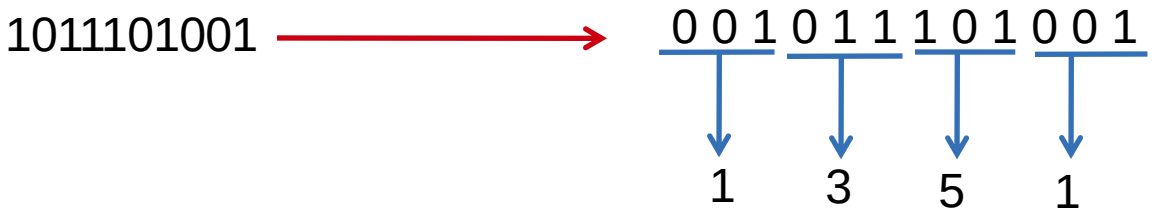
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binario	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111
Octal	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

2. Conversión entre bases

- Conversión a/de binario:
  - *Binario* ⇨ *hexadecimal*:

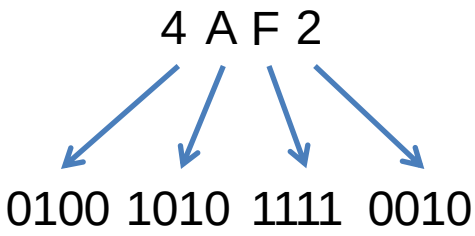
    1. Completar con '0' hasta tener nº de bits múltiplo de 4
    2. Agrupar en grupos de 4
    3. Convertir cada grupo a hexadecimal.

  - *Binario* ⇨ *octal*:

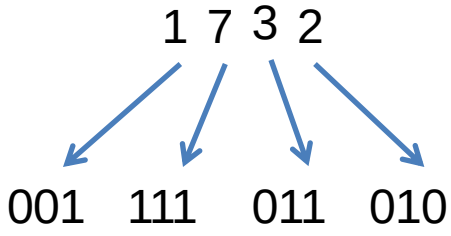


2. Conversión entre bases

- Conversión a/de binario:
  - *Hexadecimal* ➡ *binario*: sustituir cada dígito hexadecimal por su correspondiente binario con 4 bits.



  - *Octal* ➡ *binario*: sustituir cada dígito octal por su correspondiente binario con 3 bits.



3. Aritmética binaria

- ¿Por qué sistema binario?
  - Cualquier nº se representa utilizando '0' y '1'.
  - Operaciones aritméticas son muy sencillas de implementar.
- Para referirse a los dígitos binarios: bit (binary digit).
  - En los circuitos del computador:
    - "valor alto" de tensión (5V) → 1 lógico
    - "valor bajo" de tensión(0V) → 0 lógico
- Operaciones aritméticas básicas:

+	0	1
0	0	1
1	1	10

x	0	1
0	0	0
1	0	1

RESTA BINARIA (-)	0	1
0	0	1
1	1 1	0

3. Aritmética binaria

Sumar: 1001 + 1011

```
  1   1 1      acarreo
    1 0 0 1
+   1 0 1 1
-----------
  1 0 1 0 0
```

Restar: 101101 - 011011

```
    1 0 1 1 0 1
-   0 1 1 0 1 1
---------------
      1     1      acarreo
---------------
    0 1 0 0 1 0
```

Sumar: 1011 + 0011

```
      1 1      acarreo
    1 0 1 1
+   0 0 1 1
-----------
    1 1 1 0
```

Restar: 101101 - 011110

```
    1 0 1 1 0 1
-   0 1 1 1 1 0
---------------
    1 1 1 1      acarreo
---------------
  0 0 1 1 1 1
```

3. Aritmética binaria

Multiplicar: 1100 X 101

```
      1 1 0 0
    x   1 0 1
-------------
      1 1 0 0
    0 0 0 0
  1 1 0 0
-------------
  1 1 1 1 0 0
```

Dividir: 1101001 / 101

```
1 1 0 1 0 0 1 | 1 0 1
1 0 1         -------
-----           1 0 1 0 1   ➡ Cociente
0 0 1 1
  0 0 0
  -----
    1 1 0
    1 0 1
    -----
    0 0 1 0
      0 0 0
      -----
        1 0 1
        1 0 1
        -----
        0 0 0   ➡ Resto
```

4. Representación de enteros en el computador

- Tamaño datos dependen de:
  - Arquitectura del computador
  - Circuitería interna utilizada
- Maneja datos de tamaño fijo: 8bits, 16 bits, 32 bits…
- ‘n’ bits ⟹ 2^n nº o valores diferentes
- Criterios para elegir la forma de representar nº enteros:
  - Implementación de operaciones aritméticas lo más sencilla posible.
  - Overflow o desbordamiento: al realizar alguna operación aritmética, el resultado no es representable con el nº de bits disponible.

4. Representación de enteros en el computador

- Representación de enteros sin signo:
  - 'n' bits, rango: desde 0 a 2^n-1.
  - Operaciones:
    - Suma: *nº de 'n' bits* **+** *nº de 'n' bits* = hasta *'n+1' bits* para representar el resultado ⇨ *Overflow*.

```
  1 0 1 1
+ 0 1 1 1
---------
1 0 0 1 0
```

overflow

    - Resta: puede dar un resultado < 0 ⇨ *Overflow*
    - La suma y resta son diferentes. Resulta interesante que se realicen de la misma forma ⇨ utilizar **complemento a la base** para las restas.
    - Multiplicación: *nº de 'n' bits* **x** *nº de 'n' bits* = hasta *'2n' bits* para representar el resultado ⇨ *Overflow*.

4. Representación de enteros en el computador

- Complemento a la base de un número N de ‘n’ dígitos en base ‘r’, C_rN:

$$C_rN = r^n - N$$

- Complemento a 2 (cuando r=2, caso binario)
  - Ejemplo: N = 1001 de 4 bits

$$C_2N = 2^4 - N$$

$$C_2(1001) = 10000 - 1001 = 0111$$

  - Permite realizar la operación de resta como una suma:
    - A y B nº enteros sin signo de ‘n’ bits

$$A + C_2B = A + 2^n - B = (A - B) + 2^n$$

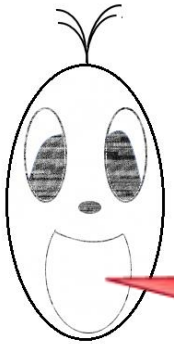
4. Representación de enteros en el computador

- Para C_2N necesitamos una resta 2^n-N ¿entonces?
- Complemento restringido a la base: complemento a 1 de N

$$C_1N = 2^n - N - 1 = C_2N - 1$$

$$C_2N = C_1N + 1$$

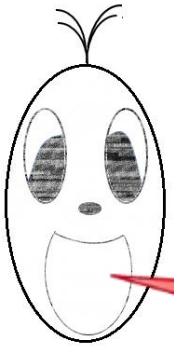
- Podemos calcular el complemento a 2 de un nº utilizando una suma en lugar de una resta.
- Cálculo del complemento a 1 de un nº N representado en binario:
  - Cambiar los '1' por '0' y los '0' por '1'.
  - $C_1(1011) = 0100$



Se pueden hacer restas utilizando sólo operaciones de suma. SE SIMPLIFICAN LOS CIRCUITOS que realizan estas operaciones

4. Representación de enteros en el computador

- Representación de enteros con signo: signo y magnitud.
  - Con ‘n’ bits podemos representar nº enteros con signo:
    - MSD ⟹ bit de signo
    - ‘n-1’ bits ⟹ magnitud
  - Rango de nº entre $2^{n-1}-1$ y $-(2^{n-1}-1)$
  - Dos codificaciones para el ‘0’: con signo, sin signo.
  - Sumas y restas:
    - Tener en cuenta los signos de los operandos.
    - Ordenar los módulos si hay que restar.
    - Calcular el signo del resultado en función de los signos de los operandos y sus módulos.

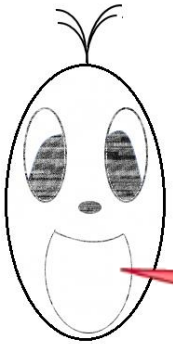


Circuitos que tengan en cuenta todas las posibilidades de signos y módulos para sumar y restar correctamente

NO SE SUELE USAR ESTA REPRESENTACIÓN EN LOS COMPUTADORES

4. Representación de enteros en el computador

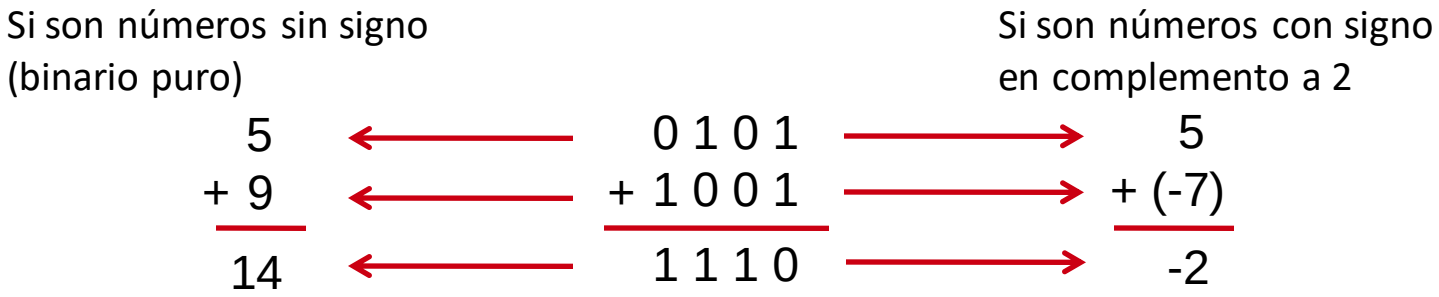
- Representación de enteros con signo: complemento a 2
  - Solo se necesitan 'n' bits:
    - Nº +: en binario natural.
    - Nº -: como el C_2 del positivo correspondiente.
    - Para cambiar el signo de un número, se hace el C2 de dicho número.
  - Ejemplo:
    - 6_{10} => 0110
    - -6_{10} => $C_2(6_{10}) = C_2(0110_2) = 1010$
  - 1º bit: signo ('0' ⟶ +, '1' ⟶ -)
  - Rango de nº desde -2^{n-1} hasta $2^{n-1}-1$
  - Representación única para '0'.
  - En las sumas y restas el bit de signo NO se trata de forma diferente (se opera)



ES LA REPRESENTACIÓN HABITUAL EN LOS COMPUTADORES

4. Representación de enteros en el computador

- Ejemplos de operaciones ('n' = 4 bits):



Detección de overflows:

- Binario puro: desbordamiento en sumas o restas si se produce acarreo superior
- Complemento a 2: desbordamiento si el signo del resultado es inesperado

5. Bibliografía

- Román Hermida, Ana Mº del Corral, Enric Pastor, Fermín Sánchez
  "Fundamentos de Computadores" , cap 1
  Editorial Síntesis
- Thomas L. Floyd
  "Fundamentos de Sistemas Digitales", cap 2
  Editorial Prentice Hall
- Daniel D. Gajski
  "Principios de Diseño Digital", cap 2
  Editorial Prentice Hall
- M. Morris Mano
  "Diseño Digital", cap 1
  Editorial Prentice Hall