Universidad Rey Juan Carlos

Repaso de Introducción a la Informática

Grado de Ingeniería del Software

REPRESENTACIÓN DE LA INFORMACIÓN EN LOS COMPUTADORES

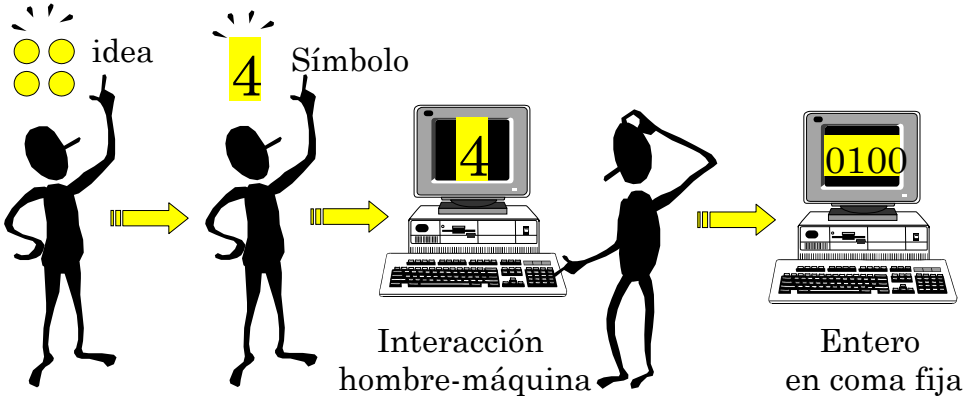
Ángel Serrano Sánchez de León
Luis Rincón Córcoles
Pablo Huerta Pellitero

Índice

- Introducción
- Sistemas de numeración de coma fija
  - Binario puro, octal y hexadecimal
  - Magnitud y signo
  - Complemento a 1
  - Complemento a 2
  - Exceso a M
- Representación en coma flotante: IEEE 754
- Representación de la información alfanumérica

Introducción

- Para representar ideas, los **seres humanos** (al menos los occidentales) utilizamos **cadenas de símbolos alfanuméricos** de un alfabeto definido.
- En el mundo de los **computadores** la información consiste en señales eléctricas cuyos valores se asocian a los conceptos lógicos **Verdadero** y **Falso**, lo que permite utilizar el cuerpo matemático de la lógica bivaluada. Esto es así, porque:
  - ➲ Los dispositivos electrónicos **distinguen solo dos estados** de forma precisa y controlada ⇒ dos dígitos ⇒ 0 y 1.
  - ➲ El **área** de diseño es **finita y acotada** ⇒ la memoria, la CPU, los buses y la E/S tienen **tamaños fijos.**



Sistema de numeración de coma fija

- Un sistema numérico consta de:
  - Un conjunto ordenado de símbolos (cifras o dígitos).
  - Relaciones definidas para la suma, resta, multiplicación y división.
- Se denomina **base *b*** de un sistema numérico al número de cifras o dígitos que utiliza.
- Ejemplos:
  - Sistema decimal (base = 10).
    - Dígitos: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
  - Sistema binario (base = 2).
    - Dígitos: {0, 1}
  - Sistema hexadecimal (base = 16).
    - Dígitos: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
  - Sistema octal (base = 8).
    - Dígitos: {0, 1, 2, 3, 4, 5, 6, 7}

SISTEMA DE NUMERACIÓN DE COMA FIJA

- Los números en un sistema de numeración consisten en una secuencia de dígitos que pueden tener **parte entera** y **parte fraccionaria** separadas por una coma inmóvil:

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

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

- Ejemplo:
  - $107{,}45_{10}$
  - $1A3, B_{16}$
  - $101110{,}11_2$
  - $1473{,}13_8$
- Esta forma de representar un número se conoce como **notación posicional**.

SISTEMA DE NUMERACIÓN DE COMA FIJA

- Se define el **peso** del dígito a_i como b^i .
- Cada dígito tiene asociado un valor en función de su peso y es el producto de dicho dígito por el peso: $a_i \cdot b^i$
- Un número se puede representar como la suma de los valores de todos sus dígitos:

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

- Ejemplo: el número en notación posicional $123{,}54_{10}$ se puede representar como:

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

- Esta forma de representar un número se conoce como **notación polinomial**.

Binario puro

- Sirve para representar números positivos o el cero. No admite negativos.
- Notación (según contexto): $1011_2 \equiv 1011_{\text{BP}} \equiv 1011\text{b}$
- Rango $= [0, 2^p - 2^{-q}]$.
- Rango (enteros con $n = p, q = 0$) $= [0, 2^n - 1]$.
- Ejemplo de representación de enteros para $n = 4$:

0	1	2	3	4	5	6	7
0000	0001	0010	0011	0100	0101	0110	0111

8	9	10	11	12	13	14	15
1000	1001	1010	1011	1100	1101	1110	1111

HEXADECIMAL

- La base 16 es **muy** utilizada en Informática porque permite una notación muy compacta (4 veces menos dígitos que el binario).
- Notación (según contexto):
  FA9016 → $FA90_{hex}$ ≡ 0xFA90 → FA90h

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

8	9	10	11	12	13	14	15
8	9	A	B	C	D	E	F

Conversión a base decimal

- **Sustitución en serie:** se utiliza para pasar un número en base b_{original} a base b_{destino}, utilizando las operaciones de la base b_{destino}. En este caso, $b_{\text{destino}} = 10$.
  - Solo hay que evaluar la notación polinomial del número utilizando las operaciones de la base b_{destino}.
  - Ejemplo: convertir el número $101{,}01_2$ a decimal.

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

  - Ejemplo: convertir el número $1\text{AC}, 2_{16}$ a decimal.

$$1\text{AC}, 2_{16} = 1 \cdot 16^2 + 10 \cdot 16^1 + 12 \cdot 16^0 + 2 \cdot 16^{-1}$$
$$= 256 + 160 + 12 + 0{,}125 = 428{,}125_{10}$$

CONVERSIÓN DESDE BASE DECIMAL

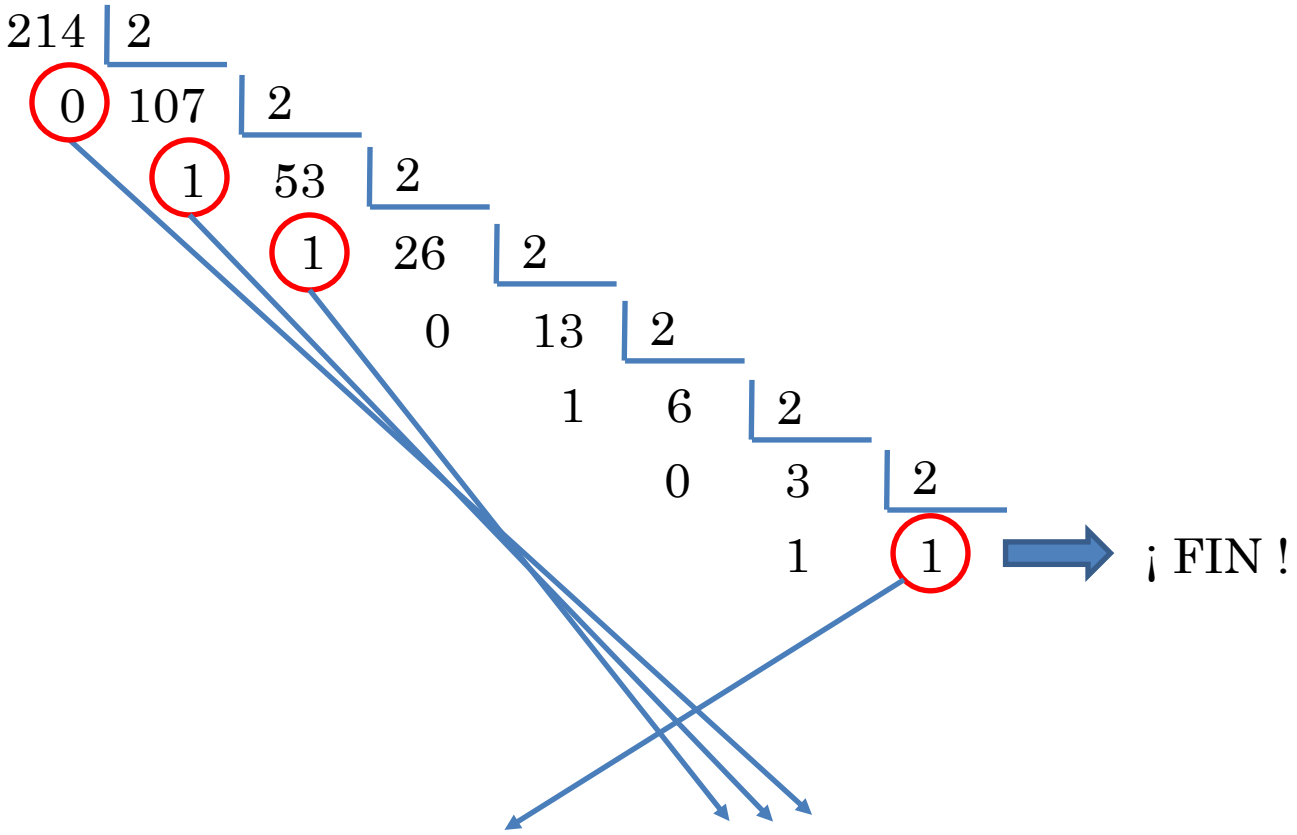
- **División/multiplicación por la base destino:** se utiliza para pasar un número en base b_{original} a base b_{destino} utilizando las operaciones de la base b_{original}. En este caso, $b_{\text{original}} = 10$.

  - **Para la parte entera:**
    - Se divide la parte entera entre la base b_{destino}. El resto obtenido será el dígito a_0 de la parte entera.
    - El cociente se divide de nuevo entre la base b_{destino}, obteniéndose un nuevo resto que será a_1 .
    - Se continúa realizando divisiones sucesivas obteniendo nuevos dígitos, hasta que el cociente sea menor que b_{destino}.
    - Se toma el último cociente y todos los restos en sentido inverso.

  - **Para la parte fraccionaria:**
    - Se multiplica la parte fraccionaria por la base b_{destino}, obteniéndose un nuevo número que tendrá parte entera y parte fraccionaria. La parte entera será el dígito a_1.
    - La nueva fraccionaria se vuelve a multiplicar por la base b_{destino}, obteniendo un nuevo dígito a_2.
    - Se continúa hasta que se obtiene una parte fraccionaria igual a 0 o se tienen suficientes dígitos.

EJEMPLOS

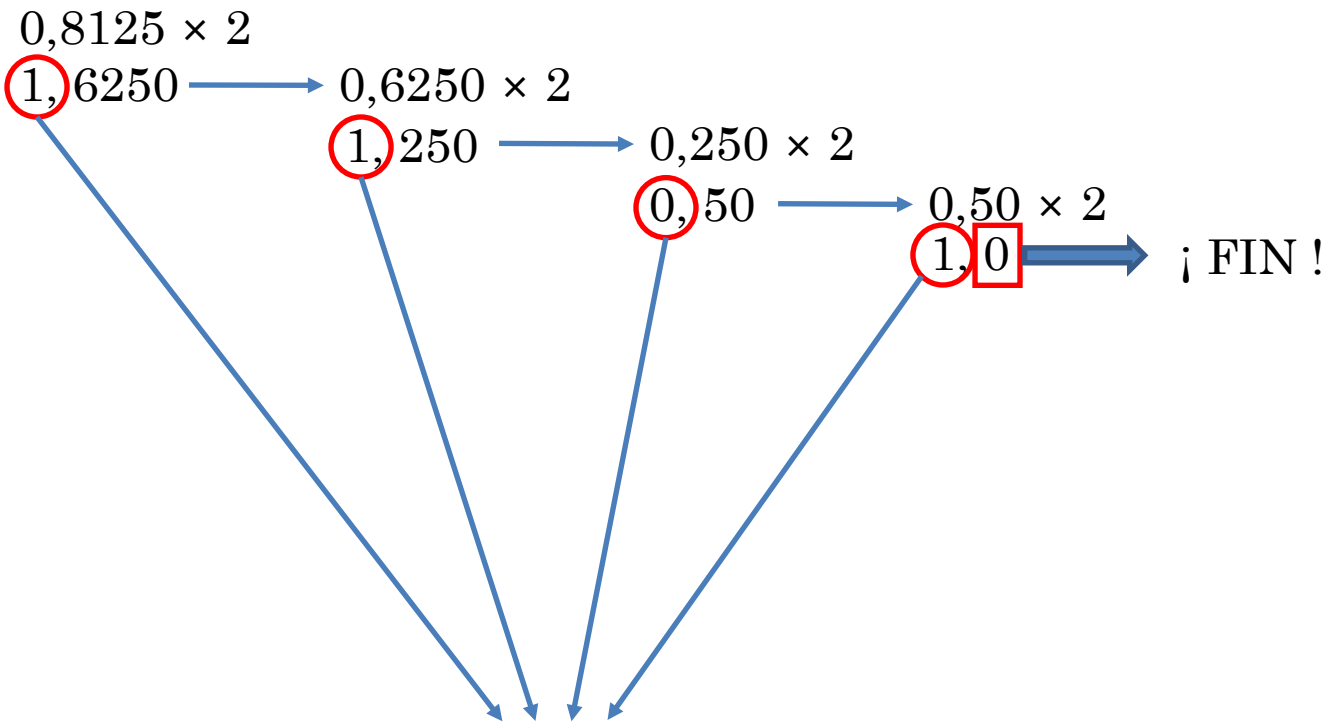
- Ejemplo: convertir el número 214_{10} a binario puro.



Resultado: $214_{10} = 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0_{2}$

EJEMPLOS

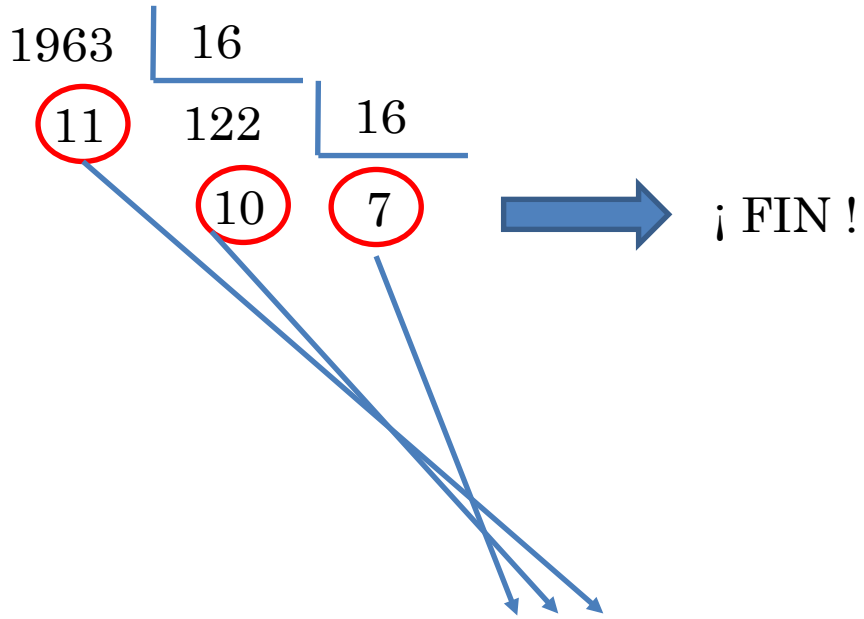
- Ejemplo: convertir el número $0{,}8125_{10}$ a binario puro.



Resultado: $0{,}8125_{10} = 0{,}1\,1\,0\,1_2$

EJEMPLOS

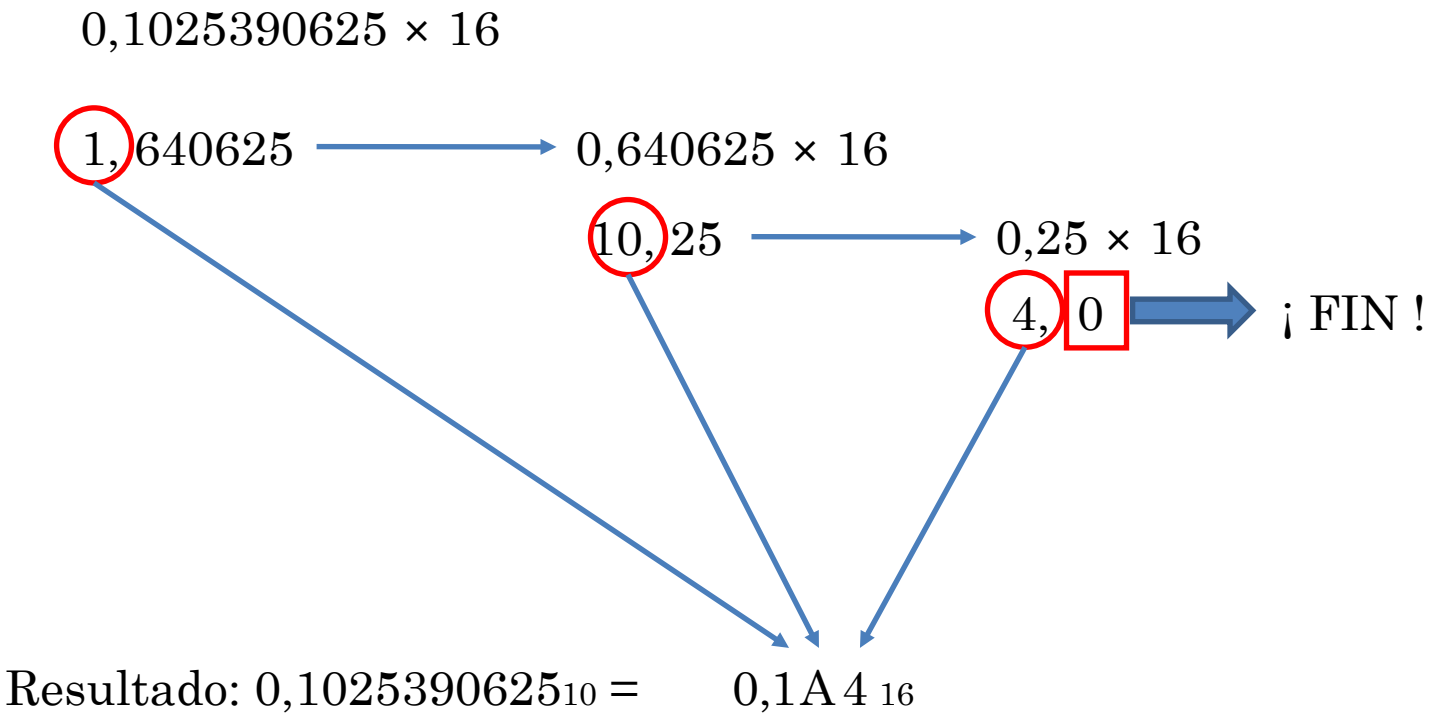
- Ejemplo: convertir el número 1963_{10} a hexadecimal.



Resultado: $1963_{10} = 7\ A\ B_{16}$

EJEMPLOS

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



Resultado: $0{,}1025390625_{10} = 0{,}1A4_{16}$

Resumen

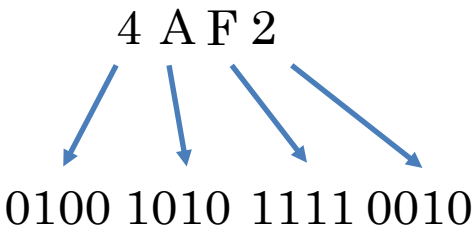
- Correspondencias de los 16 primeros números enteros en distintas bases:

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
Hexa-decimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

CONVERSIONES RÁPIDAS

- Para convertir de binario a hexadecimal y viceversa, así como de binario a octal, existe un método sencillo.
- **Binario ➔ hexadecimal:** se completa con '0' por la izquierda hasta que se tenga un número de bits múltiplo de 4. Se agrupan los bits de 4 en 4 y cada grupo de 4 bits se convierte directamente a hexadecimal:

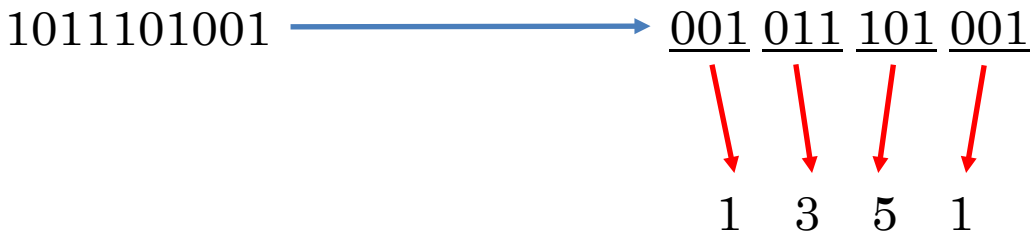
- **Hexadecimal ➔ binario:** cada dígito hexadecimal se sustituye por su correspondiente valor en binario con 4 bits:



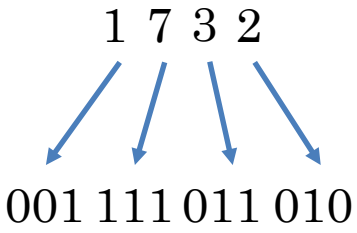
Conversiones rápidas

- Para pasar de binario a octal y viceversa el proceso es el mismo, pero agrupando los bits de 3 en 3:

Binario ➔ octal



Octal ➔ binario



Binario con Magnitud y Signo

- Reserva el **bit más significativo a_{p-1}** para indicar el signo del número: $+ \equiv 0$, $- \equiv 1$. A este bit se le llama **bit de signo**.

$$\text{signo} = \left(1 - 2 \cdot a_{p-1}\right)$$

- Los $n - 1$ bits restantes representan la magnitud (o módulo) del número.

$$N = \left(1 - 2 \cdot a_{p-1}\right) \cdot \sum_{i=-q}^{p-2} a_i \cdot 2^i$$

- Ejemplo con $p = 4$, $q = 0$:

$$0110_{\text{MS}} = (1 - 2 \cdot 0)(1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0) = +6_{10}$$

$$1110_{\text{MS}} = (1 - 2 \cdot 1)(1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0) = -6_{10}$$

Binario con Magnitud y Signo

- Rango $= [-(2^{p-1} - 2^{-q}), 2^{p-1} - 2^{-q}]$
- Rango (enteros con $n = p, q = 0$) $= [-(2^{n-1} - 1), 2^{n-1} - 1]$
- **Operación cambio de signo:** simplemente se invierte el bit de signo.
- Los números positivos se representan igual en binario puro y en magnitud y signo (en la parte del rango que coincide que ambos sistemas de representación).
- Los números negativos se representan perfectamente en magnitud y signo (imposible en binario puro).
- **Doble representación del cero:** Ejemplo para $n = 4$: $0000_{\text{MS}} = 1000_{\text{MS}} = 0_{10}$

Binario con Complemento a 1

- También llamado complemento restringido a la base $b - 1$.
- Si queremos representar un número positivo, se hace igual que en binario puro. El bit más significativo (bit de signo) debe ser 0.
- Si es un número negativo, se halla la representación de su valor absoluto en binario puro y, a continuación, **se invierten todos los bits**. En este caso, el bit de signo es 1. A esta operación se la llama **calcular el complemento a 1**.
- Ejemplo con $p = 4$, $q = 0$:

$$+6_{10} = 0110_{\mathrm{BP}} = 0110_{\mathrm{C1}}$$

$$¿-6_{10}? \Rightarrow \mathrm{C1}(0110) = 1001_{\mathrm{C1}}$$

Binario con Complemento a 1

- Rango $= [-(2^{p-1} - 2^{-q}), 2^{p-1} - 2^{-q}]$
- Rango (enteros con $n = p, q = 0$) $= [-(2^{n-1} - 1), 2^{n-1} - 1]$
- Ejemplo con $n = 4$:

$$+8_{10} = 1000_{\text{BP}} = ¿\, 1000_{\text{C1}}?$$

¡ Fuera de rango con 4 bits!: $+8_{10} = 01000_{\text{C1}}$

- Doble representación del cero. Ejemplo para $n = 4$: $0000_{\text{C1}} = 1111_{\text{C1}} = 0_{10}$

Binario con Complemento a 2

- Es el sistema de representación más habitual de enteros con signo en los computadores.
- Si queremos representar un número positivo, se hace igual que en binario puro. El bit más significativo (bit de signo) debe ser 0.
- Si es un número negativo, se halla la representación de su valor absoluto en binario puro y luego **se calcula el complemento a 2**.
- Ejemplo con $n = 4$:

$$+6_{10} = 0110_{BP} = 0110_{C2}$$

Binario con Complemento a 2

- Manera 1 de calcular el complemento a 2: Si estamos usando n bits, se resta 2^n (en binario, un uno seguido de n ceros) menos el número en cuestión.
- Ejemplo con $n = 4$:

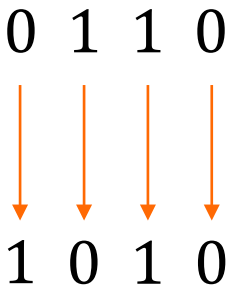
$$+6_{10} = 0110_{C2}$$

$$\begin{array}{r} 10000 \\ -\ 0110 \\ \hline 1010 \end{array} \qquad \begin{array}{r} 16 \\ -\ 6 \\ \hline 10 \end{array}$$

$$-6_{10} = 1010_{C2}$$

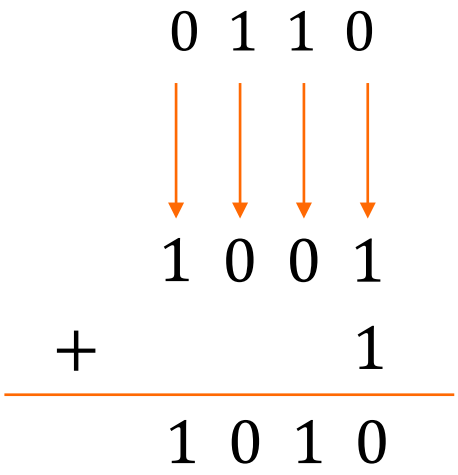
Binario con Complemento a 2

- Manera 2 de calcular el complemento a 2: Se copia bit a bit de derecha a izquierda hasta encontrar el primer 1 (que se deja tal cual). A partir de ahí, se invierten los bits.



Binario con Complemento a 2

- Manera 3 de calcular el complemento a 2: Se calcula el complemento a 1 (invirtiendo todos los bits) y al resultado se le suma 1.



- Recordad este método porque lo usaremos en el tema 7 (diseño y construcción de una ALU).

Binario con Complemento a 2

- Conversión rápida a base decimal (misma fórmula que en binario puro, pero el bit más significativo lleva un signo menos):

$$N = -a_{p-1} \cdot 2^{p-1} + \sum_{i=-q}^{p-2} a_i \cdot 2^i$$

- Conversión indirecta a base decimal:
  - Si el bit de signo es 0, es un número positivo y se aplica la fórmula habitual de binario puro.
  - Si es el bit de signo es 1, es un número negativo. Primero le aplicamos la operación complemento para volverlo positivo, calculamos su valor con la fórmula del binario puro y finalmente recuperamos el signo menos.
- Rango $= [-2^{p-1}, 2^{p-1} - 2^{-q}]$
- Rango (enteros con $n = p, q = 0$) $= [-2^{n-1}, 2^{n-1} - 1]$
- Sin ambigüedad en el cero.

¡¡CUIDADO!!

- Tanto la expresión "complemento a 1" como "complemento a 2" tienen DOS significados:
  - Hacen referencia a un sistema de representación numérica, al igual que los sistemas del binario puro, magnitud y signo, etc.
  - Se refieren también a una operación matemática (el cálculo del "complemento").
    - En el "sistema de representación de complemento a 2", se aplica la "operación de complemento a 2".
    - En el "sistema de representación de complemento a 1", se aplica la "operación de complemento a 1".
    - En ambos casos, el complemento es la operación de **cambio de signo**.

Binario con Exceso a M

- Un número N se representa en **exceso a M** igual que se representaría el número $N + M$ **en binario puro**.
  - M es el **sesgo** (**exceso**, ***bias***) de la representación y suele valer $\mathbf{2^{n-1}}$ o $\mathbf{2^{n-1} - 1}$.
  - Se suele usar **solo para enteros** ($n = p$, $q = 0$).

- Ejemplos:
  - Representar 3_{10} con $n = 4$ y exceso a 8:

$$3_{10} = (3 + 8)_{10,\text{Exc8}} = 1011_{\text{Exc8}}$$

$$11_{10} = 1011_{\text{BP}}$$

  - Representar 53_{10} con $n = 8$ y exceso a 127:

$$53_{10} = (53 + 127)_{10,\text{Exc127}} = 10110100_{\text{Exc127}}$$

$$180_{10} = 10110100_{\text{BP}}$$

Binario con Exceso a M

- Conversión a base decimal:

$$N = \sum_{i=0}^{n-1} a_i \cdot 2^i - M$$

- Rango (enteros con $n = p, q = 0$) $= [-M, 2^n - 1 - M]$

Resumen de coma fija

- Equivalencia entre la representación binario en magnitud y signo, complemento a 2, complemento a 1 y exceso a 2^{n-1} construidas para una representación de enteros con $n = 4$:

Decimal	Magnitud y signo	Complemento a 2	Complemento a 1	Exceso a 2^3
7	0111	0111	0111	1111
6	0110	0110	0110	1110
5	0101	0101	0101	1101
4	0100	0100	0100	1100
3	0011	0011	0011	1011
2	0010	0010	0010	1010
1	0001	0001	0001	1001
0	0000	0000	0000	1000
-0	1000	----	1111	----
-1	1001	1111	1110	0111
-2	1010	1110	1101	0110
-3	1011	1101	1100	0101
-4	1100	1100	1011	0100
-5	1101	1011	1010	0011
-6	1110	1010	1001	0010
-7	1111	1001	1000	0001
-8	----	1000	----	0000

Enteros con $n = 4$	BP	MS	C2	C1	Exceso a M $M = 2^{n-1}$
Definición	Divisiones sucesivas	signo | magnitud	$C_2N = 2^n - N$ De dcha a izqda: copiar hasta el primer 1 y luego invertir bits	$C_1N = 2^n - N - 1$ Invertir bits	$N_{\text{ExcM}} = N + M_{\text{BP}}$
Positivos	Todos	0 | magnitud	0 | magnitud	0 | magnitud	No evidente
Negativos	No hay	1 | magnitud	C_2N (N positivo)	C_1N (N positivo)	No evidente
Rango	$[0, 2^n - 1]$	$[-(2^{n-1} - 1), 2^{n-1} - 1]$	$[-2^{n-1}, 2^{n-1} - 1]$	$[-(2^{n-1} - 1), 2^{n-1} - 1]$	$[-M, 2^n - 1 - M]$
Cambio de signo	Imposible	Invertir bit de signo	C_2N	C_1N	No evidente
Conversión a base 10	$\sum_{i=0}^{n} a_i \cdot b^i$	$(1 - 2 \cdot a_{n-1}) \cdot \sum_{i=0}^{n-2} a_i \cdot 2^i$	$-a_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} a_i \cdot 2^i$	No evidente	$\sum_{i=0}^{n-1} a_i \cdot 2^i - M$
Cero	0000	0000 y 1000	0000	0000 y 1111	1000

Representación en coma flotante: IEEE 754

- **Estándar IEEE 754:** adoptado en 1985 y revisado por última vez en 2008 (versión IEEE 754-2008).
- Equivale a la notación científica en los computadores para representar valores muy grandes (en valor absoluto) o muy próximos a cero.
- La coma flotante **no** es una representación exacta para **números reales**, sino que únicamente permite representar los números reales en forma **aproximada**.
- El valor del exponente debe ajustarse para que solo haya un dígito no nulo a la izquierda de la coma.

$$\pm 1, xxxxxxxx_{\mathrm{BP}} \cdot 2^{yyy}$$

$1, xxxxxxxx$ = Mantisa (1=bit implícito)

$xxxxxxxx$ = Parte fraccionaria

yyy = exponente expresado en exceso

- Calculadora online: https://babbage.cs.qc.cuny.edu/IEEE-754/

Representación en coma flotante: IEEE 754 precisión simple

- Formato binary32, "precisión simple" (32 bits) o "float":

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	EXPONENTE								MANTISA																						

  - Signo (1 bit) y Mantisa (23 bits + 1 bit implícito), expresados en binario magnitud-signo. El bit implícito siempre vale 1 y no se almacena.
  - Exponente (8 bits), expresado en binario con exceso a 127.
  - Rango:

$$[-3{,}4028 \times 10^{38}, -1{,}4013 \times 10^{-45}] \cup \{0\} \cup [1{,}4013 \times 10^{-45}, 3{,}4028 \times 10^{38}]$$

Hueco del cero (underflow)

Representación en coma flotante: IEEE 754 precisión simple

$$N = \begin{cases} (-1)^s \times 1,m \times 2^{\exp-127} & \text{si } 0 < \exp < 255 \\ (-1)^s \times 0,m \times 2^{-126} & \text{si } \exp = 0 \text{ (caso desnormalizado)} \\ 0 & \text{si } \exp = m = 0 \\ +\infty & \text{si } \exp = 255, m = 0, s = 0 \\ -\infty & \text{si } \exp = 255, m = 0, s = 1 \\ \text{NaN (“Not a Number”)} & \text{si } \exp = 255, m \neq 0 \end{cases}$$

donde:

- s = signo de la mantisa
- m = módulo de la mantisa almacenada (sin bit implícito)
- exp = exponente almacenado (sin descontar exceso)

EJEMPLO: BASE 10 ➔ IEEE 754 (PREC. SIMPLE)

$$N = -0,7510$$

1. Expresarlo en binario: $-0,75_{10} = -0,11_2$
2. Normalizar: $-0,11_2 = -1,1_2 \times 2^{-1}$
3. Cálculo del exponente en exceso: $\exp = -110 + 12710 = 126_{10,Exc127} = 01111110_{2,Exc127}$
4. Cálculo de mantisa: $m = \mathbf{1},1000\ 0000\ 0000\ 0000\ 0000\ 000$ (el bit implícito no se almacena)
5. Bit de signo: $s = 1$ (n.º negativo)
6. Representación en precisión simple:

1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S	EXPONENTE								MANTISA																						

7. Representación en notación compacta hexadecimal:

$$N = \mathbf{1011\ 1111\ 0}100\ 0000\ 0000\ 0000\ 0000\ 0000_{2,IEEE754,s}$$
$$= \mathbf{BF\ 40\ 00\ 00}_{16,IEEE754,s}$$

Ejemplo: IEEE 754 (prec. simple) ➔ Base 10

$$\boldsymbol{N = C0\ A0\ 00\ 00_{16,IEEE754,s}}$$

$$= \underbrace{\mathbf{1100\ 0000\ 1010\ 0000\ 0000\ 0000\ 0000\ 0000}}_{\text{s}\quad\text{exp}\qquad\qquad\qquad\text{m}}\mathbf{_{2,IEEE754,s}}$$

1. Bit de signo: $s = 1$ (n.º negativo)
2. Exponente: $\exp = 10000001_{2,Exc127} = 129_{10,Exc127} = 129_{10} - 127_{10} = 2_{10}$
3. Mantisa: $\mathbf{1},0100\ 0000\ 0000\ 0000\ 0000\ 000_2 = 2^0 + 2^{-2} = 1{,}25_{10}$
4. Resultado: $N = -1{,}25 \times 22 = \mathbf{-51_0}$

Representación en coma flotante: IEEE 754 precisión doble

- Formato binary64, "precisión doble" (64 bits) o "double":

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	EXPONENTE											MANTISA																			

MANTISA (CONTINUACIÓN)

  - Signo (1 bit) y Mantisa (52 bits + 1 bit implícito), expresados en binario magnitud-signo. El bit implícito siempre vale 1 y no se almacena.
  - Exponente (11 bits), expresado en binario exceso a 1023.
  - Rango:

$$[-1{,}7977 \times 10^{308}, -4{,}9406 \times 10^{-324}] \cup \{0\} \cup [4{,}9406 \times 10^{-324}, 1{,}7977 \times 10^{308}]$$

Hueco del cero (underflow)

Representación en coma flotante: IEEE 754 precisión doble

$$N = \begin{cases} (-1)^s \times 1,m \times 2^{\exp-1023} & \text{si } 0 < \exp < 2047 \\ (-1)^s \times 0,m \times 2\text{^} - 1022 & \text{si } \exp = 0 \text{ (caso desnormalizado)} \\ 0 & \text{si } \exp = m = 0 \\ +\infty & \text{si } \exp = 2047, m = 0, s = 0 \\ -\infty & \text{si } \exp = 2047, m = 0, s = 1 \\ \text{NaN (“Not a Number”)} & \text{si } \exp = 2047, m \neq 0 \end{cases}$$

donde:

- s = signo de la mantisa
- m = módulo de la mantisa almacenada (sin bit implícito)
- exp = exponente almacenado (sin descontar exceso)

EJEMPLO: BASE 10 ➔ IEEE 754 (PREC. DOBLE)

$$N = -0,7510$$

1. Expresarlo en binario: $-0,75_{10} = -0,11_2$
2. Normalizar: $-0,11_2 = -1,1_2 \times 2^{-1}$
3. Cálculo del exponente en exceso: $\exp = -110 + 102310 = 1022_{10,\text{Exc1023}} = 01111111110_{2,\text{Exc1023}}$
4. Cálculo de mantisa: $m =$ **1**,1000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 (el bit implícito no se almacena)
5. Bit de signo: $s = 1$ (n.º negativo)
6. Representación en precisión simple:

1	0	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S	EXPONENTE											MANTISA																			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MANTISA (CONTINUACIÓN)																															

7. Representación en notación compacta hexadecimal:

$$N = \mathbf{1011\ 1111\ 1110}\ 1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{2,IEEE754,\text{d}}$$
$$= \mathbf{BF\ E8\ 00\ 00\ 00\ 00\ 00\ 00}_{\mathbf{16,IEEE754,d}}$$

Ejemplo: IEEE 754 (prec. doble) ➔ Base 10

$$N = \mathbf{C0\ 14\ 00\ 00\ 00\ 00\ 00\ 00}_{16,IEEE754,d}$$
$$= \mathbf{1100\ 0000\ 0001\ 0100\ 0000\ 0000\ 0000\ \ldots\ 0000}_{2,IEEE754,d}$$

s exp m

1. Bit de signo: $s = 1$ (n.º negativo)
2. Exponente: $\exp = 10000000001_{2,Exc1023} = 1025_{10,Exc1023} = 1025_{10} - 1023_{10} = 2_{10}$
3. Mantisa: $\mathbf{1},0100\ 0000\ 0000\ \ldots 0_2 = 2^0 + 2^{-2} = 1{,}25_{10}$
4. Resultado: $N = -1{,}25 \times 22 = \mathbf{-51_0}$

Representación de la información alfanumérica

- **Codificaciones alfanuméricas:**
  - Son las codificaciones encargadas de representar los caracteres alfabéticos, numéricos, signos de puntuación y signos de control mediante cadenas de dígitos binarios.
  - Al menos deben representar 26 letras del alfabeto y 10 dígitos, es decir 36 caracteres, luego necesitan un mínimo de 6 bits. En realidad, se necesitan más caracteres, de forma que las codificaciones más utilizadas emplean 7 y 8 bits: **ASCII** y **EBCDIC** (**E**xtended **BCD** **I**nterchange **C**ode). Solo se comenta el **ASCII**.

- **ASCII:**
  - El **Código estándar americano para el intercambio de información** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) es el código alfanumérico más extendido.
  - El código **ASCII original utilizaba 7 bits** para representar **128** caracteres (**0** hasta **7F** en hexadecimal (0111 1111)) donde los primeros 32 caracteres son de control (no gráficos o invisibles) y los restantes son gráficos (visibles).

Representación de la información alfanumérica

- **ASCII** (continuación):
  - En la actualidad ASCII es un **código de 8 bits**, también conocido como **ASCII extendido**, que aumenta su capacidad hasta **256** caracteres introducidos por IBM para los PC (estándar de facto).
  - Estos caracteres adicionales van del **80** (1000 0000) hasta el **FF** (1111 1111) e incluyen caracteres alfabéticos no ingleses, símbolos no ingleses, letras griegas, símbolos matemáticos, caracteres para gráficos, caracteres gráficos de barra y caracteres sombreados.
  - Los códigos ASCII extendidos presentan variaciones nacionales. El código más usado en nuestra zona geográfica es el **Latin-1**.
  - https://cs.stanford.edu/people/miles/iso8859.html

- **Unicode:**
  - Existe otro código para caracteres denominado **Unicode**, que emplea **16 bits** por carácter. Es utilizado en Java. Unicode no requiere variaciones nacionales, puesto que con 16 bits permite representar **32768** caracteres distintos, incluidos símbolos musicales, matemáticos, emojis, etc.
  - https://home.unicode.org/