T2. Representación de la información

1.1 De cualquier base a decimal:

n = *nº dígitos de la parte entera*

k = *nº dígitos de la parte decimal*

d = *número en la base original*

b = *base*

$$\sum_{i=-k}^{n-1} d_i \; b^i$$

- Base 5 a decimal:

$$130.14 = 1 \cdot 5^2 + 3 \cdot 5^1 + 0 \cdot 5^0 + 1 \cdot 5^{-1} + 4 \cdot 5^{-2} = 25 + 15 + 0.2 + 0.16 = 40.36$$

1.2 Base decimal a binario

Se deben convertir la parte decimal y la entera por separado.

- **Parte entera:**

Dividimos el número por la base hasta que el cociente sea 0. La parte entera será la sucesión de los restos de las divisiones, empezando por el último.

- **Parte decimal:**

Multiplicamos por la base hasta llegar a un número entero o alcanzar las cifras deseadas.

$$0.7 * 2 = 1.4$$
$$0.4 * 2 = 0.8$$
$$0.8 * 2 = 1.6$$
$$0.6 * 2 = 1.2$$
$$0.2 * 2 = 0.4$$
$$0.4 * 2 = 0.8$$
$$0.8 * 2 = 1.6$$

$$\text{Parte decimal} = \widehat{101100}$$

1.3 Hexadecimal

Los números en hexadecimal coinciden con su versión en decimal, sin embargo, del 10 al 15, usamos las primeras 6 letras:

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

Es posible que tengamos que añadir 0 a la izquierda para agrupar los bits en 4 y pasar a hexadecimal. Siempre de derecha a izquierda: 100 1100 → 0100 1100 = 0x4C

1.4 Codificación de números enteros con signo

Signo – magnitud

Los valores se representan mediante un bit de signo (0: positivo y 1: negativo) y tantos bits representando el número:

El rango de valores con *n* bits es $[-2^{(n-1)}+1, 2^{(n-1)}-1]$

Entonces, los números posibles con 4 bits son:

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

Valor	-0	-1	-2	-3	-4	-5	-6	-7
Binario	1000	1001	1010	1011	1100	1101	1110	1111

Es importante no usar el último bit para otra cosa que no sea el signo. Además, existen 2 representaciones para el 0

Complemento a-1

- Números **positivos**: igual que en signo–magnitud
- Números **negativos**: se invierten todos los bits del número positivo (cambiamos 1 por 0 y viceversa)

El rango de valores con *n* bits es $[-2^{(n-1)}+1, 2^{(n-1)}-1]$

Valor	-3	-2	-1	-0	0	1	2	3
Binario	100	101	110	111	000	001	010	011

Operaciones (sumamos bit a bit):

1 + 2 = 001 + 010 = 011 = **3**

3 – 2 = 3 + (-2) = 011 + 101 = ¹000 (nos llevamos 1)

000 + 1 = 001 = **1**

3 + 3 = 011 + 011 = 110 = **-1 (** no es correcto por desbordamiento)

Si al sumar dos números positivos nos da un número negativo, ha habido desbordamiento. Esto tiene sentido porque 3 + 3 = 6 (110) pero con 3 bits solo podemos representar números en el intervalo [-3 , 3].

Para 4 bits:

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

Para los números negativos haríamos lo mismo, pero invirtiendo los bits:

Valor	-0	-1	-2	-3	-4	-5	-6	-7
Binario	1111	1110	1101	1100	1011	1010	1001	1000

Complemento a-2

- Números **positivos**: igual que en complemento a-1
- Números **negativos**: se invierten todos los bits del número positivo y le sumamos 1.

El rango de valores con *n* bits es $[-2^{(n-1)}, 2^{(n-1)}-1]$. Ejemplo para 4 bits:

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

Valor	-0	-1	-2	-3	-4	-5	-6	-7	-8
Binario	0000	1111	1110	1101	1100	1011	1010	1001	1000

Operaciones : (Ojo, **se descarta el acarreo**)

3 + 2 = 0011 + 0010 = 0101 = 5
3 – 2 = 3 + (-2) = 0011 + 1110 = ^{1}0001 → 0001 = 1
- 3 – 2 = (-3) + (-2) = 1101 + 1110 = ^{1}1011 → 1011 = -5

Exceso a $2^{(n-1)}-1$

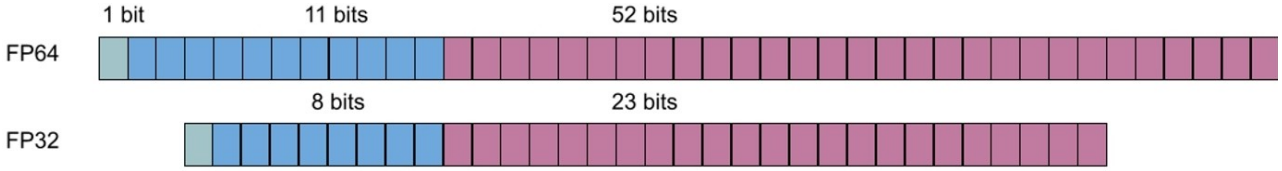
Se suma $2^{(n-1)}-1$ antes de pasarlo a binario. Números de 4 bits en exceso a 7:

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

Valor	1	2	3	4	5	6	7	8
Binario	1000	1001	1010	1011	1100	1101	1110	1111

1.5 Codificación de números reales – IEEE745

En esta representación, se necesita almacenar una mantisa y un exponente. La mantisa se representa con signo-magnitud y el exponente en exceso a $2^{(E-1)}-1$, siendo *E* el número de bits dedicados al exponente. El número de bits para cada formato:



En la mantisa, solo se incluyen los dígitos de la parte fraccionaria

1. Convertimos el número a binario 23.25 = 10111.01
2. Lo normalizamos, dejando solo 1 a la izquierda de la coma: $\pm 1.M * 2^{e}$
   10111.01 = 1.011101 * 2^{4}
3. Representamos la mantisa (parte decimal del número normalizado) y rellenando 0
4. Al exponente (4) le añadimos un exceso, en FP32: $2^{8-1}-1$ = 127
   4 + 127 = 131 = 10000011

El número final es :

0 10000011 01110100000000000000000

En doble precisión, solo tenemos que sumar un exceso de $2^{11-1}-1 = 1023$ al exponente.

IEE754 a decimal

Calculamos el valor respecto a la expresión:

$$(-1)^S * 1.M * 2^{E-127}$$

Ejemplo:

X = 0 10000011 *11100000000000000000000*

1. $(-1)^0 = +1$
2. Exponente:

   10000011 – 011111111 (131 – 127) = 4
3. Mantisa:

   1.M = 1.11100000000000000000000 =
   $1 + 1\cdot2^{-1} + 1\cdot2^{-2} + 1\cdot2^{-3} = 1{,}875$

X = $+1.875 \cdot 2^4$ = **30**

Casos especiales

- **Cero**: todo 0 (bit de signo 0 o 1)
- **Numeros desnormalizados**: exponente todo 0, mantisa distinta de cero

$(-1)^S \times 0.M \times 2^{-126}$

- **Más o menos infinito:** exponente todo 1, mantisa todo 0
- **NaN:** exponente todo 1, mantisa distinta de 0

1.6 Codificación de caracteres

ASCII

ASCII es un esquema binario de codificación de caracteres basado en el orden del alfabeto en inglés.

ISO-8859

Es un esquema de codificación de caracteres. Utiliza 8 bits para codificar cada carácter.

UTF-8

Esquema basado en símbolos de longitud variable (1-4 bytes por carácter). Los bits más significativos del primer byte indican la longitud de la secuencia.

- Con 1 byte: caracteres incluidos en US-ASCII. Primer bit siempre a 0. Por tanto, tenemos 2^7 = 128 caracteres.
- Con 2 bytes: el primer byte empieza por 110 y el segundo por 10. Por tanto, tenemos, $2^{(16-3-2)} = 2^{11}1$ = **2048** caracteres.
- Con 3 bytes: el primer byte empieza por 1110, segundo y tercero por 10. Tenemos $2^{(24-4-2-2)}$ = **655536** caracteres.
- Con 4 bytes: el primer byte empieza por 11110, segundo, tercer y cuarto byte por 10. Por tanto, tenemos $2^{(32-5-2-2-2)}= 2^{21}$ = **2097152** caracteres.