

# Sistemas Digitales

Nad Garraz y comunidad (ojalá)  
Facultad de Ciencias Exactas y Naturales  
UBA

Choose your destiny:

(doubleclick en los ejercicio para saltar)

- [Notas teóricas](#)

- Ejercicios de la guía:

[1.](#)

[6.](#)

[11.](#)

[2.](#)

[7.](#)

[12.](#)

[3.](#)

[8.](#)


[13.](#)

[4.](#)

[9.](#)

[5.](#)

[10.](#)

El repo en [github](#)  para descargar las guías con los últimos updates.



<https://github.com/nad-garraz/sistemasDigitales>

La Guía 1 se actualizó por última vez: 20/08/2024 @ 11:34

Guía 1



<https://github.com/nad-garraz/sistemasDigitales/blob/main/1-guia/1-sol.pdf>

Si querés mandar un ejercicio o avisar de algún error, lo más fácil es por

Telegram .



<https://t.me/joinchat/DS9ZukGbZgIOIaHgdBlavQ>

## Notas teóricas:

▣ *Complejidad:*

Es lo que nos da la necesidad de abstraer.

▣ *Abstracción:*

Para lidiar con la *complejidad* que tienen los sistemas usamos la abstracción.

De menor a mayor abstracción:

electrones → transistores → circuitos analógicos → circuitos digitales → circuitos de lógica →  
micro-arquitectura → arquitectura → sistema operativo → aplicaciones de software

Lo más abstracto es más fácil de controlar e implementar que lo menos abstracto.

▣ *Sistemas numéricos:*▣ *Decimal:*

$$9745_{10} = 9 \times 10^3 + 7 \times 10^2 + 4 \times 10^1 + 5 \times 10^0$$

El rango de un número *decimal* con  $n$  dígitos mayor o igual a cero es  $10^n : 0, 1, \dots, 10^n - 1$

▣ *Binario:*

$$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13_{10}.$$

El rango de un número *binario* con  $n$  dígitos mayor o igual a cero es  $2^n : 0, 1, \dots, 2^n - 1$ . Si tengo solo un bit, *binary digit*, puedo obtener o bien 0 o bien 1. Si tengo dos bits, entonces puedo tener  $2^2$  dígitos, 00, 01, 10, 11. (el rango es después de todo 4).

▣ *Hexadecimal:*

$$2AD_{16} = 2 \times 16^2 + A \times 16^1 + D \times 16^0 = 685_{10}$$

El rango de un número *hexadecimal* con  $n$  dígitos mayor o igual a cero es  $16^n : 0, 1, \dots, 16^n - 1$ . Cada dígito de un número en base hexadecimal corresponde a un número binario de 4-bits. Después de todo un número  $i_{16}$  de *un solo dígito* tiene un rango de 16:  $0, \dots, 15$  y un número binario de 4-bits tiene un rango de  $2^4 = 16$  ✓

$$2AD_{16} = \underbrace{0010}_2 \underbrace{1010}_A \underbrace{1101}_D_2$$

▣ *Operaciones entre números binarios:* Se poné picante según la representación usada.

- ▣ Sumar es fácil si los números son positivos. Tengo **overflow** si el resultado tiene más cifras que bits disponibles para almacenar dicho resultado.

▣ *Números binarios con signo:*

Hay distintas *representaciones* para hacer esto, acá están las 2 más usadas:

▣ **signo+magnitud**

El bit más significativo, el de más a la izquierda, marca el **signo**. Un número con  $n$  bits en esta representación tiene un rango:  $[-2^{n-1} + 1, 2^{n-1} - 1]$ . Por ejemplo con 3-bits:

El rango es  $[-3, 3]$

Base 2	→	Base 10
000	→	0
001	→	1
010	→	2
011	→	3
100	→	-0
101	→	-1
110	→	-2
111	→	-3

Sumar normalmente en esta representación no tiene sentido. Representa en total  $2^n - 1$  elementos porque el **-0** está usando un lugar al pedo.

▣ complemento a 2 :

▣ Menos intuitivo, pero más útil. Si tengo un número de n-bits, voy a tener siempre:

$0_{10}$	→	$00...00_2$	→ <i>weird number</i>
$-1_{10}$	→	$11...11_2$	
$-2^{n-1}_{10}$	→	$100...00_2$	
$2^{n-1} - 1_{10}$	→	$011...11_2$	

Teniendo al 0,  $-1$ ,  $-2^{n-1}$  y  $(2^{n-1} - 1)$ , puedo encontrar el resto de la *representación*:

▣ Para encontrar el opuesto a un número, se cambian los 0 por 1 y bicerveza <sup>1</sup>, luego se le suma 1 a eso, ej:

$$5_{10} = 0101_2 \xrightarrow[-5]{\text{busco}} 1010_2 + 0001_2 = 1011_2 = -5_{10}.$$

Cosa que no funciona con el *weird number*, porque su complemento te da a él mismo **A**.

▣ El rango es de  $[-2^{n-1}, 2^{n-1} - 1]$ ,  $2^n$  elementos. Hay un elemento negativo más que positivos.

▣ Ejemplito: En 3-bits me encuentro todo el conjunto,  $[-4, 3]$ :

$000_2$	→	$0_{10}$
$001_2$	→	$1_{10}$
$010_2$	→	$2_{10}$
$011_2$	→	$3_{10}$
$100_2$	→	$-4_{10}$
$101_2$	→	$-3_{10}$
$110_2$	→	$-2_{10}$
$111_2$	→	$-1_{10}$

▣ *Suma en complemento a 2* : Si sumo dos números de distinto signo no voy a tener **overflow** !

$$4 \text{ bits: } -4_{10} + 5_{10} = 1100_2 + 0101_2 = \cancel{1}0001_2 = 1_{10}$$

▣ *Sign extension*: Para encontrar la representación de un número conocido con más bits.

Copio el signo al resto de los númerosengo que mandar el bit del signo hacia el dígito más significativo, :

$$4 \text{ bits: } 6_{10} = 0110_2 \text{ y } -5_{10} = 1011_2$$

$$\text{Extendido a 8 bits: } 6_{10} = (0000 \ 0110)_2 \text{ y } -5_{10} = (1111 \ 1011)_2$$

▣ Exceso m:

▣ Se desplaza el 0 a la posición m. Es así que si la representación es **exceso 4** en 6-bits base 3:  
 $0_{10} = (000 \ 0011)_3$

<sup>1</sup>chiste: bicerveza = **DD**. Se escribe viceversa **DD**.

▣ Comparación representaciones del mismo Dato:

Posición	Dato	unsigned	signo+magnitud	Exceso m (m=4)	complemento a 2
0	(0000) <sub>2</sub>	0	0	-4	0
1	(0001) <sub>2</sub>	1	1	-3	1
2	(0010) <sub>2</sub>	2	2	-2	2
3	(0011) <sub>2</sub>	3	3	-1	3
4	(0100) <sub>2</sub>	4	4	0	4
5	(0101) <sub>2</sub>	5	5	1	5
6	(0110) <sub>2</sub>	6	6	2	6
7	(0111) <sub>2</sub>	7	7	3	7
8	(1000) <sub>2</sub>	8	-0	4	-8
9	(1001) <sub>2</sub>	9	-1	5	-7
10	(1010) <sub>2</sub>	10	-2	6	-6
11	(1011) <sub>2</sub>	11	-3	7	-5
12	(1100) <sub>2</sub>	12	-4	8	-4
13	(1101) <sub>2</sub>	13	-5	9	-3
14	(1110) <sub>2</sub>	14	-6	10	-2
15	(1111) <sub>2</sub>	15	-7	11	-1

## Ejercicios de la guía:

## Ejercicio 1

- Utilizando el método del cociente, expresar en bases 2, 3 y 5 los números  $33_{10}$  y  $511_{10}$ .
- Expresar en decimal los números  $1111_2$ ,  $1111_7$  y  $CAFE_{16}$ .
- Expresar  $17_8$  en base 5 y  $BABA_{13}$  en base 6.
- Pasar  $(1010\ 1110\ 1010\ 1101)_2$ ,  $(1111\ 1011\ 0010\ 1100\ 0111)_2$ ,  $(0\ 0110\ 0010\ 1001)_2$ , a base 4, 8 y 16 agrupando bits.
- Expresar en decimal los números  $0x142536$ ,  $0x142536$  y  $0xFCD9$ , y pasar a base 16 los números  $7848_{10}$  y  $46183_{10}$ .

- Fijarse si el número es cercano a una potencia de la base buscada.*

$$33_{10} = (10\ 0001)_2 = 1020_3 = 113_5$$

$$511_{10} = (1111\ 1111)_2 = (20\ 0221)_3 = 4021_5$$

- ¿Tiene truco?

$$1111_2 = 15_{10}$$

$$1111_7 = 400_{10}$$

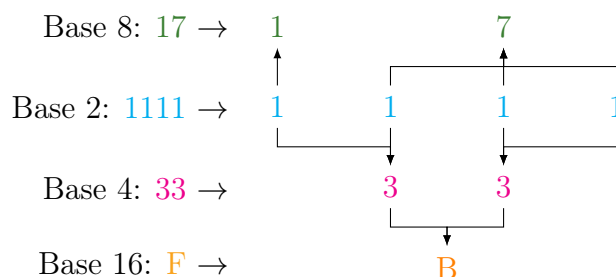
$$CAFE_{16} = C \times 16^3 + A \times 16^2 + F \times 16^1 + E \times 16^0 = 12 \times 16^3 + 10 \times 16^2 + 15 \times 16^1 + 14 \times 16^0 = 51966_{10}$$

- ¿Hay truco para no pasar por base 10?

$$17_8 = 15_{10} = 30_5$$

$$BABA_{13} = 11 \times 13^3 + 10 \times 13^2 + 11 \times 13^1 + 10 \times 13^0 = 26180_{10} = 321112_6$$

- Como las bases se tiene que cambiar a otras bases que son potencia de 2, se puede hacer agrupando los bits.



$$(1010\ 1110\ 1010\ 1101)_2 = (22\ 32\ 22\ 31)_4 = 5555_8 = AEAD_{16}$$

$$(1111\ 1011\ 0010\ 1100\ 0111)_2 = (33\ 23\ 02\ 30\ 13)_4 = (373\ 1307)_8 = FB2CD_{16}$$

$$(0\ 0110\ 0010\ 1001)_2 = (12\ 02\ 21)_4 = 3051_8 = C29_{16}$$

- $0x142536 = 142536_{16} = 1 \times 16^5 + 4 \times 16^4 + 2 \times 16^3 + 5 \times 16^2 + 3 \times 16^1 + 6 \times 16^0 = (132\ 0246)_{10}$

$$0xFCD9 = FCD9_{16} = 15 \times 16^3 + 12 \times 16^2 + 13 \times 16^1 + 9 \times 16^0 = (6\ 4729)_{10}$$

$$7848_{10} = 0x1EA8$$

$$(4\ 6183)_{10} = 0xB467$$

**Ejercicio 2** Realizar las siguientes sumas de precisión fija, sin convertir a decimal. Indicar en cada caso si hubo acarreo.

$$\begin{array}{r} 100001_2 \\ a) + 011110_2 \\ \hline \text{-----}2 \end{array}$$

$$\begin{array}{r} 100001_2 \\ b) + 011111_2 \\ \hline \text{-----}2 \end{array}$$

$$\begin{array}{r} 9999_{16} \\ c) + 1111_{16} \\ \hline \text{----}2 \end{array}$$

$$\begin{array}{r} F0F0_2 \\ d) + FOCA_2 \\ \hline \text{----}2 \end{array}$$

$$\begin{array}{r} 100001_2 \\ a) + 011110_2 \\ \hline 111111_2 \end{array}$$

$$\begin{array}{r} 9999_{16} \\ c) + 1111_{16} \\ \hline AAAA_2 \end{array}$$

$$\begin{array}{r} \text{Acarreo } 111111 \\ b) \quad \quad 100001_2 \\ \quad + \quad 011111_2 \\ \hline 1000000_2 \end{array}$$

$$\begin{array}{r} \text{Acarreo } 11 \\ d) \quad \quad F0F0_2 \\ \quad + \quad FOCA_2 \\ \hline 1E1BA_2 \end{array}$$

Precisión fija? Tengo **overflow** en el b y en el d? Se tira el decimal de más? Cómo se expresa el resultado?

**Ejercicio 3** ¿Puede suceder en alguna base que la suma de dos números de precisión fija tenga un acarreo mayor que 1? Exhibir un ejemplo o demostrar lo contrario.

**Ejercicio 4** Sean los siguientes numerales binarios de ocho dígitos:

$$r = (1011 \ 1111)_2, \ s = (1000 \ 0000)_2 \text{ y } t = (1111 \ 1111)_2.$$

¿Qué números representan si asumimos que son codificaciones de enteros en complemento a 2? ¿Y si fueran codificaciones en signo+magnitud?

En la teoría están cuales son los valores *amigos* de la representación complemento a 2

- Notar que si  $r$  es de complemento:

$$r + 64 = (1011 \ 1111)_2 + (0100 \ 0000)_2 = (1111 \ 1111)_2 = -1_{10} \rightarrow r = -65_{10}$$

Puedo llegar a lo mismo usando la técnica para encontrar el complemento de  $r$ :

$$\bar{r} = \overline{10111111}_2 \xrightarrow[1 \leftrightarrow 0]{\text{cambio}} (0100 \ 0000)_2 \xrightarrow[1]{\text{sumo}} (0100 \ 0001)_2 = 1 = 65_{10} = \bar{r}, \text{ entonces } r = -65_{10}$$

$$\text{Si fuese de signo+magnitud: } r = (1011 \ 1111)_2 = -63_{10}$$

Sumar normalmente en la representación de signo+magnitud es para problemas.

- Para complemento a 2:  $s = (1000 \ 0000)_2 = -128_{10}$  es el *weird number*

$$\text{Y en signo+magnitud: } s = (1000 \ 0000)_2 = -0_{10}$$

- Para complemento a 2:  $t = (1111 \ 1111)_2 = -1_{10}$

$$\text{Y en signo+magnitud: } t = (1111 \ 1111)_2 = -127_{10}$$

**Ejercicio 5** Codificar los siguientes números en base 2, usando la precisión y forma de representación indicada en cada caso. Comparar los resultados.

- <sub>a</sub>  $0_{10} \rightarrow$  usando 8 bits, notación signo+magnitud y notación complemento a 2.
- <sub>b</sub>  $-1_{10} \rightarrow$  usando 8 y 16 bits, en ambos casos notación signo+magnitud y notación complemento a 2.
- <sub>c</sub>  $255_{10} \rightarrow$  usando 8 bits notación sin signo y 16 bits notación complemento a 2.
- <sub>d</sub>  $-128_{10} \rightarrow$  usando 8 y 16 bits, en ambos casos notación complemento a 2.
- <sub>e</sub>  $128_{10} \rightarrow$  usando 8 bits, notación sin signo y 16 notación complemento a 2.

Leer la [técnica para encontrar los números con complemento a 2](#).

- <sub>a</sub> signo+magnitud :  $0_{10} = (0000\ 0000)_2 \stackrel{?}{=} (1000\ 0000)_2$ . abuso de notación?  
complemento a 2 :  $0_{10} = (0000\ 0000)_2$ .
- <sub>b</sub> signo+magnitud :  
 $-1_{10} = (1000\ 0001)_2$ .  
 $-1_{10} = (1000\ 0000\ 0000\ 0001)_2$ .  
 complemento a 2 :  
 $-1_{10} = (1111\ 1111)_2$ .  
 $-1_{10} = (1111\ 1111\ 1111\ 1111)_2$ .
- <sub>c</sub> Sin Signo:  
 $255_{10} = (1111\ 1111)_2$ .  
 complemento a 2 :  
 $255_{10} = (0000\ 0000\ 1111\ 1111)_2$ .
- <sub>d</sub> complemento a 2 :  
 $-128_{10} = (1000\ 0000)_2$ .  
 $-128_{10} \rightarrow 128_{10} = (0000\ 0000\ 1000\ 0000)_2 \xrightarrow[\text{opuesto}]{\text{busco}} (1111\ 1111\ 1000\ 0000)_2 = -128_{10}$ .
- <sub>e</sub> Sin Signo:  
 $128_{10} = (1000\ 0000)_2$ .  
 $128_{10} = (0000\ 0000\ 1000\ 0000)_2$ .

¿Qué se puede interpretar de esto?

**Ejercicio 6** ¿Puede alguna cadena binaria de k dígitos, interpretada en complemento a 2, representar un número que no puede ser representado por una cadena de la misma longitud, interpretada en signo+magnitud? ¿Y al revés?

De al teoría de [complemento a 2](#), tengo que el rango es distinto en las distintas interpretaciones. Hay un número más en la de complemento a 2.



Si tengo  $k$  dígitos  $(-2^{k-1})_{10} = (\underbrace{100\dots 0}_{k\text{-bits}})_2$ . Número que no puedo representar en signo+magnitud porque el menor número es  $(-2^{k-1} + 1)_{10}$ .

La representación complemento a 2 contiene a todos los números de la representación signo+magnitud.  
**¿Hay algo que aprender? O es solo eso?**

---

**Ejercicio 7** Interpretar los operadores y resultados de las sumas del ejercicio 2 como representaciones de enteros en complemento a 2 y, para cada una de ellas, indicar cuáles son correctas, cuáles no. ¿Se evidencia **overflow** en alguna?

---

**Ejercicio 8** 🙄... hay que hacerlo! 🙄

Si querés mandarlo: Telegram → , o mejor aún si querés subirlo en L<sup>A</sup>T<sub>E</sub>X → .

---

**Ejercicio 9** 🙄... hay que hacerlo! 🙄

Si querés mandarlo: Telegram → , o mejor aún si querés subirlo en L<sup>A</sup>T<sub>E</sub>X → .

---

**Ejercicio 10** 🙄... hay que hacerlo! 🙄

Si querés mandarlo: Telegram → , o mejor aún si querés subirlo en L<sup>A</sup>T<sub>E</sub>X → .

---

**Ejercicio 11** 🙄... hay que hacerlo! 🙄

Si querés mandarlo: Telegram → , o mejor aún si querés subirlo en L<sup>A</sup>T<sub>E</sub>X → .

---

**Ejercicio 12** 🙄... hay que hacerlo! 🙄

Si querés mandarlo: Telegram → , o mejor aún si querés subirlo en L<sup>A</sup>T<sub>E</sub>X → .

---

**Ejercicio 13** 🙄... hay que hacerlo! 🙄

Si querés mandarlo: Telegram → , o mejor aún si querés subirlo en L<sup>A</sup>T<sub>E</sub>X → .

---