Sistemas Digitales

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

Choose your destiny:

(dobleclick en los ejercicio para saltar)

- Notas teóricas
- Ejercicios de la guía:

1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.

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: 22/08/2024 @ 20:04

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 necesida 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:*

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

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

  - *Binario:*

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

    El rango de un número *binario* con $\texttt{n}$ dígitos mayor o igual a cero es $2^{\texttt{n}} : 0, 1, \ldots, 2^{\texttt{n}} - 1$. 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:*

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

    El rango de un número *hexadecimal* con $\texttt{n}$ dígitos mayor o igual a cero es $16^{\texttt{n}} : 0, 1, \ldots, 16^{\texttt{n}} - 1$. 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: $0, \ldots 15$ y un número binario de 4-bits tiene un rango de $2^4 = 16$ ✓

    $$\texttt{2AD}_{16} = \underbrace{\texttt{0010}}_{\texttt{2}}\underbrace{\texttt{1010}}_{\texttt{A}}\underbrace{\texttt{1101}}_{\texttt{D}}\ {}_{2}$$

- *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 $\texttt{n}$ bits en esta representación tiene un rango: $[-2^{\texttt{n}-1} + 1, 2^{\texttt{n}-1} - 1]$. 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}	→	$00\ldots00_2$	
-1_{10}	→	$11\ldots11_2$	
$-2^{n-1}{}_{10}$	→	$100\ldots00_2$	→ *weird number*
$2^{n-1} - 1_{10}$	→	$011\ldots11_2$	

    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 [1] , luego se le suma 1 a eso, ej:

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

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

$$\texttt{4 bits: } 6_{10} = 0110_2 \text{ y } -5_{10} = 1011_2$$
$$\text{Extendido a 8 bits: } 6_{10} = (0000\ 0110)_2 \text{ y } -5_{10} = (1111\ 1011)_2$$

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

---

[1] chiste: bicerveza = 🍺🍺. Se escribe viceversa 👍😊.

- *Comparación representaciones del mismo* Dato:

Posición	Dato	unsigned	signo+magnitud	Exceso m (m=4)	complemento a 2
0	$(0000)_2$	0	0	-4	0
1	$(0001)_2$	1	1	-3	1
2	$(0010)_2$	2	2	-2	2
3	$(0011)_2$	3	3	-1	3
4	$(0100)_2$	4	4	0	4
5	$(0101)_2$	5	5	1	5
6	$(0110)_2$	6	6	2	6
7	$(0111)_2$	7	7	3	7
8	$(1000)_2$	8	-0	4	-8
9	$(1001)_2$	9	-1	5	-7
10	$(1010)_2$	10	-2	6	-6
11	$(1011)_2$	11	-3	7	-5
12	$(1100)_2$	12	-4	8	-4
13	$(1101)_2$	13	-5	9	-3
14	$(1110)_2$	14	-6	10	-2
15	$(1111)_2$	15	-7	11	-1

Ejercicios de la guía:

Ejercicio 1

a) Utilizando el método del cociente, expresar en bases 2, 3 y 5 los números $\texttt{33}_{10}$ y $\texttt{511}_{10}$.

b) Expresar en decimal los números $\texttt{1111}_2$, $\texttt{1111}_7$ y $\texttt{CAFE}_{10}$.

c) Expresar $\texttt{17}_8$ en base 5 y $\texttt{BABA}_{13}$ en base 6.

d) Pasar $(\texttt{1010 1110 1010 1101})_2$, $(\texttt{1111 1011 0010 1100 0111})_2$, $(\texttt{0 0110 0010 1001})_2$, a base 4, 8 y 16 agrupando bits.

e) Expresar en decimal los números `0x142536`, `0x142536` y `0xFCD9`, y pasar a base 16 los números $\texttt{7848}_{10}$ y $\texttt{46183}_{10}$.

---

a) *Fijarse si el número es cercado a una potencia de la base buscada.*

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

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

b) ¿Tiene truco?

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

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

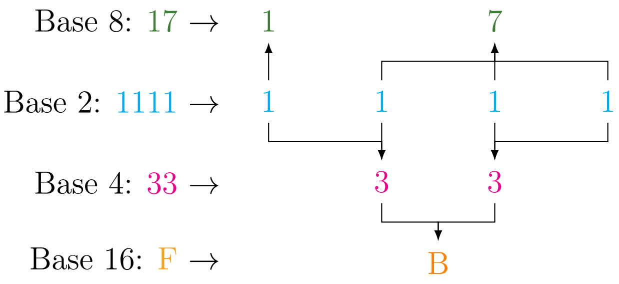
$\texttt{CAFE}_{16} = \texttt{C}\times 16^3 + \texttt{A}\times 16^2 + \texttt{F}\times 16^1 + \texttt{E}\times 16^0 = \texttt{12}\times 16^3 + \texttt{10}\times 16^2 + \texttt{15}\times 16^1 + \texttt{14}\times 16^0 = \texttt{51966}_{10}$

c) ¿Hay truco para no pasar por base 10?

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

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

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



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

$(\texttt{1111 1011 0010 1100 0111})_2 = (\texttt{33 23 02 30 13})_4 = (\texttt{373 1307})_8 = \texttt{FB2CD}_{16}$

$(\texttt{0 0110 0010 1001})_2 = (\texttt{12 02 21})_4 = \texttt{3051}_8 = \texttt{C29}_{16}$

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

$\texttt{0xFCD9} = \texttt{FCD9}_{16} = \texttt{15}\times 16^3 + \texttt{12}\times 16^2 + \texttt{13}\times 16^1 + \texttt{9}\times 16^0 = (\texttt{6 4729})_{10}$

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

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

`Los culpables de que esto haya sucedido:`

`Nad Garraz`

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

a) $\begin{array}{r} \texttt{100001}_2 \\ + \quad \texttt{011110}_2 \\ \hline \texttt{------}_2 \end{array}$ b) $\begin{array}{r} \texttt{100001}_2 \\ + \quad \texttt{011111}_2 \\ \hline \texttt{------}_2 \end{array}$ c) $\begin{array}{r} \texttt{9999}_{16} \\ + \quad \texttt{1111}_{16} \\ \hline \texttt{----}_2 \end{array}$ d) $\begin{array}{r} \texttt{F0F0}_2 \\ + \quad \texttt{F0CA}_2 \\ \hline \texttt{----}_2 \end{array}$

---

a) $\begin{array}{r} \texttt{100001}_2 \\ + \quad \texttt{011110}_2 \\ \hline \texttt{111111}_2 \end{array}$ c) $\begin{array}{r} \texttt{9999}_{16} \\ + \quad \texttt{1111}_{16} \\ \hline \texttt{AAAA}_2 \end{array}$

b) $\begin{array}{lr} \texttt{Acarreo} & \texttt{11111} \\ & \texttt{100001}_2 \\ + & \texttt{011111}_2 \\ \hline & \texttt{1000000}_2 \end{array}$ d) $\begin{array}{lr} \texttt{Acarreo} & \texttt{1} \\ & \texttt{F0F0}_2 \\ + & \texttt{F0CA}_2 \\ \hline & \texttt{1E1BA}_2 \end{array}$

La precisión fija y la representación `signo+magnitud` al tener el acarreo que agrega un bit generan `overflow` . Los resultados de los items b) y d) no tienen representación en `signo+magnitud` .

`Los culpables de que esto haya sucedido:`

`Nad Garraz`

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.

En cualquier base `B` se da que el último símbolo que puede reprentar es `B-1`. Entonces cuando se sumen 2 dígitos:

$$\begin{array}{c} \texttt{B - 1} \\ \texttt{B - 1} \\ \hline \texttt{2B -2} \end{array}$$

La expresión de un número en base `B` que tiene un 2 de peso:

$$x = 2\ B^{n+1},$$

y otro que tiene como peso a la suma de los mayores pesos de la base:

$$y = (2B - 2)\ B^{n}$$

Pero, pero, pero:

$$\begin{array}{c} 2\ B^{n+1} \overset{?}{\geq} (2B - 2)\ B^{n} \\ 0 \overset{\checkmark}{\geq} -\ 2\ B^{n} \end{array}$$

Por lo que se estaría complicando para el `team acarreo mayor a 1`

`Los culpables de que esto haya sucedido:`

`Nad Garraz`

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`:

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

  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*

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

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

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

Los culpables de que esto haya sucedido:

Nad Garraz

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

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

Leer la técnica para encontrar los números con `complemento a 2` .

- a. `signo+magnitud` : $0_{10} = (0000\ 0000)_2 \overset{?}{=} (1000\ 0000)_2.$ abuso de notación?

  `complemento a 2` : $0_{10} = (0000\ 0000)_2.$

- b. `signo+magnitud` :

  $-1_{10} = (1000\ 0001)_2.$

  $-1_{10} = (1000\ 0000\ 0000\ 0001)_2.$

`complemento a 2` :

$-1_{10} = (1111\ 1111)_2$.

$-1_{10} = (1111\ 1111\ 1111\ 1111)_2$.

- ■$_{c}$ Sin Signo:

  $255_{10} = (1111\ 1111)_2$.

  `complemento a 2` :

  $255_{10} = (0000\ 0000\ 1111\ 1111)_2$.

- ■$_{d}$ `complemento a 2` :

  $-128_{10} = (1000\ 0000)_2$.

  $-128_{10} \to 128_{10} = (0000\ 0000\ 1000\ 0000)_2 \xrightarrow[\text{opuesto}]{\text{busco}} (1111\ 1111\ 1000\ 0000)_2 = -128_{10}$.

- ■$_{e}$ Sin Signo:

  $128_{10} = (1000\ 0000)_2$.

  $128_{10} = (0000\ 0000\ 1000\ 0000)_2$.

¿Qué se puede interpretar de esto?

Los culpables de que esto haya sucedido:

Nad Garraz

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 $(-2^{k-1})_{10} = (\underbrace{100...0}_{k-bits})_2$. 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 número 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 ¿Cómo acomodaría esta suma de números `hexadecimales` de 4 dígitos en notación `complemento a 2` , para que en ningún momento se produzca `overflow` ?

$$7744_{16} + 5499_{16} + 6788_{16} + AB68_{16} + 88BD_{16} + 9878_{16} = 0003_{16}$$

$$7744_{16} + 5499_{16} + 6788_{16} + AB68_{16} + 88BD_{16} + 9878_{16} = 0003_{16}$$
$$7744_{16} + 5499_{16} + 6788_{16} + AB68_{16} + 88BD_{16} + 9878_{16} = 0003_{16}$$

Ejercicio 9 Dar ocho pares de numeros tales que la suma de las representaciones de cada par en complemento a dos de 4 bits provoque lo siguiente:

1) No se produzca acarreo ni `overflow` .
2) Se produzca acarreo pero no `overflow` .
3) Se produzca acarreo y `overflow` .
4) No se produzca acarreo pero si `overflow` .
5) Se produzca acarreo y el resultado sea cero.
6) No se produzca acarreo y el resultado sea cero.
7) El resultado sea negativo y se produzca `overflow` .
8) El resultado sea negativo y no se produzca `overflow` .

---

1) $(0000,\ 0000)_2$ o $(0101,\ 1010)_2$
2) $(1111,\ 0001)_2$ o $(1100,\ 0100)_2$
3) $(1111,\ 1111)_2$
4) $(0100,\ 0100)_2$
5) $(0001,\ 1111)_2$
6) $(0000,\ 0000)_2$
7) $(1111,\ 1111)_2$
8) $(1111,\ 0000)_2$ o $(1111,\ 1001)_2$ o $(1111,\ 1100)_2$

Nota: Los incisos 3) y 7) están mal. Los dejamos ahí para testearte y hacerte saber que lograste esto. Cuando usamos la representación que usemos, el `overflow` tiene que ver con el sentido que le damos a cierto número. Por ejemplo:

$$\begin{array}{lr} \texttt{Acarreo} & 11111 \\ & 100001_2 \\ + & 011111_2 \\ \hline & 1000000_2 \end{array}$$

Acabo de hacer $-7_{10} + 7_{10} = 0_{10}$, el `bit` más significativo que supera la cantidad de bits de mi cuenta se descarta y el resultado es coherente con mi represntación. No hay `overflow` (hay acarreo).

Los culpables de que esto haya sucedido:

- Iñaki Frutos
- Nad Garraz

Ejercicio 10 La función $\texttt{SignExt}_\texttt{n}$ convierte números de `k-bits` en números de `k+n-bits` de la siguiente manera:

$$\texttt{SignExt}_\texttt{n}(b_{k\text{-}1}...b_0) = \begin{cases} 0...0b_{k\text{-}1}...b_0 & \texttt{si} \quad b_{k\text{-}1} = 0 \\ 1...1b_{k\text{-}1}...b_0 & \texttt{si} \quad b_{k\text{-}1} = 1 \end{cases}$$

Mostrar que para todo número `x` de `k-bits`, `x` y $\texttt{SingExt}_\texttt{n}\texttt{(x)}$ representan el mismo número si se los interpreta en notación `complemento a 2` de `k` y `k+n-bits` respectivamente.

Esto está no demostrado en las notas teóricas de `complemento a 2` . Si estoy laburando en `complemento a 2` voy a tener que el 0_{10} se representa para `k-bits` y `k-bits + n` como:

$$\begin{array}{ccc} \texttt{base10} & \texttt{k-bits} & \texttt{k-bits + n} \\ 0_{10} & (\underbrace{0...0}_{k})_2 & (\underbrace{0...0}_{n}\underbrace{0...0}_{k})_2, \end{array}$$

al $\texttt{-1}_{10}$ lo represento como:

$$\begin{array}{ccc} \texttt{base10} & \texttt{k-bits} & \texttt{k-bits + n} \\ \texttt{-1}_{10} & \texttt{(1...1)}_2 & \texttt{(1...11...1)}_2, \end{array}$$

el mayor número de la representacion también en `complemento a 2` :

$$\begin{array}{ccc} \texttt{base10} & \texttt{k-bits} & \texttt{k-bits + n} \\ (2^k\ \texttt{- 1})_{10} & \texttt{(01...1)}_2 & \texttt{(0...001...1)}_2, \end{array}$$

y el menor número de la representacion `complemento a 2` también como:

$$\begin{array}{ccc} \texttt{base10} & \texttt{k-bits} & \texttt{k-bits + n} \\ (\texttt{-}2^k)_{10} & \texttt{(10...0)}_2 & \texttt{(1...110...0)}_2, \end{array}$$

La verdad que no creo que esto sea una demostración pero voy a asumir que es trivial que para números positivos:

$$x_n = \texttt{SignExt}_\texttt{n}\texttt{(x) si } x_n > 0,$$

porque solo son `0s` a la izquierda. Y para los negativos podría buscarles el opuesto:

$$\begin{array}{cccc} \texttt{base10} & x_{10} < 0 & \xrightarrow{\text{opuesto}} & x_{10} > 0 \\ \texttt{k-bits} & \texttt{(1b...b)}_2 & \xrightarrow{\text{opuesto}} & (0\neg b...\neg b)_2 + \texttt{(0...1)}_2 \\ \texttt{k-bits + n} & \texttt{(1...11b...b)}_2 & \xrightarrow{\text{opuesto}} & (0...00\neg b...\neg b)_2 + \texttt{(0...00...1)}_2 \end{array}$$

La suma en la parte cyan para los números positivos me va a dar lo mismo. Por lo tanto bajo la *conjetura que hice* de que para los positivos es trivial la igualdad $x_n = \texttt{SignExt}_\texttt{n}(x)$ me muestra que también los negativos son iguales después de aplicarles la extensión de signo.

Los culpables de que esto haya sucedido:

Nad Garraz

Ejercicio 11 Represente los números 2_{10}, -5_{10} y 0_{10} en notación `complemento a 2` de `4-bits` de longitud. Luego:

a) Invierta los bits de cada representación obtenida e indique a que número representa en el mismo sistema.

b) A partir de lo realizado en el punto anterior, proponga un método para obtener la representación en `complemento a 2` del inverso aditivo de un número, dada la representación de ese número en el mismo sistema.

(a)
- 2_{10}
  - `complemento a 2` : 0010_2 Se necesitan 3 digitos, porque con 2, solamente se podrían representar 4 numeros en total, siendo el 1 el maximo numero posible a represntar
  - Inverso de la representación obtenida: $1101_2 = -3_{10}$
- -5_{10}
  - Complemento a dos: 1011_2
  - Inverso de la representación obtenida: $0100_2 = 4_{10}$
- 0_{10}
  - Complemento a dos: 0000_2
  - Inverso de la representación obtenida: $1111_2 = -1_{10}$

(b) Método para obtener el inverso aditivo:

- Buscar el inverso del número
- Sumar uno al inverso del número

Los culpables de que esto haya sucedido:
Iñaki Frutos

Ejercicio 12 Diremos que un sistema de representación de números como cadenas binarias de longitud fija es `biyectivo` si no admite más de una representación para cada número y toda cadena disponible es utilizada para representar algún número.
Decidir si la siguiente afirmación es verdadera o falsa:

"No es posible dar con un sistema que represente números con signo utilizando cadenas binarias de longitud fija que sea `biyectivo`, tenga una representación para el cero y donde la cantidad de números positivos y negativos representados sea la misma". Justificar.

Esto buscando algo de esta pinta:

Posición	Dato	casi 1	casi 2
0	$(000)_2$	-3	-4
1	$(001)_2$	-2	-3
2	$(010)_2$	-1	-2
3	$(011)_2$	0	-1
4	$(100)_2$	1	0
5	$(101)_2$	2	1
6	$(110)_2$	3	2
7	$(111)_2$	4	3

Y está complicado porque tengo que usar de 2^n elementos Uno para el cero, por lo que quedan:

$$2^n = 2^n - 1 \to 2^n - 1 \bmod 2 \neq 0$$

luego no podría tener igual cantidad de números positivos y negativos.

Los culpables de que esto haya sucedido:

Nad Garraz

Ejercicio 13 Dar un ejemplo de un sistema de representación `biyectivo` en el que lal cantidad de números positivos y negativos representados es la misma.

Exceso 4: 2-trits. Como tengo cantidad de impar de elementos que representar es así.

Posición	Dato	Exceso 4
0	$(00)_3$	-4
1	$(01)_3$	-3
2	$(02)_3$	-2
3	$(10)_3$	-1
4	$(11)_3$	0
5	$(12)_3$	1
6	$(20)_3$	2
7	$(21)_3$	3
8	$(22)_3$	4