# Práctica 1 - Representación de la información

**Ejercicio 1**

1. (33)10 = (100001)2 = (1020)3 = (113)5

(511)10 = (111111111)2 = (200221)3 = (4021)5

1. (1111)2 = 1 x 23 + 1 x 22 + 1 x 21 + 1 x 20 = 8 + 4 + 2 + 1 = (15)10

(1111)7 = 1 x 73 + 1 x 72 + 1 x 71 + 1 x 70 = 343 + 49 + 7 + 1 = (400)10

(CAFE)16 = 12 x 163 + 10 x 162 + 15 x 161 + 14 x 160 = 49152 + 2560 + 240 + 14 = (51966)10

1. (17)8 = 1 x 81 + 7 x 80 = 8 + 7 = (15)10 = (30)5

(BABA)13 = 11 x 133 + 10 x 132 + 11 x 131 + 10 x 130 = 24167 + 1690 + 143 + 10 = (26010)10 = (320220)6

1. (1010 1110 1010 1101)2 = (22322231)4 = (127255)8 = (AEAD)16

(1111 1011 0010 1100 0111)2 = (3323023013)4 = (3731307)8 = (FB2C7)16

(0 0110 1001)2 = (1221)4 = (151)8 = (69)16

Para hacer un cambio de base agrupando bits cuando se quiere pasar de base 2 a una base que es potencia de 2, siempre se debe empezar a agrupar desde la derecha.

* A base 4: agrupar de a 2
* A base 8: agrupar de a 3
* A base 16: agrupar de a 4

1. 0x142536 = (142536)16 = 1 x 165 + 4 x 164 + 2 x 163 + 5 x 162 + 3 x161 + 6 x 160

= 1048576 + 262144 + 8192 + 1280 + 48 + 6 = (1320246)10

0xFCD9 = (FCD9)16 = 15 x 163 + 12 x 162 + 13 x 161 + 9 x 160 = 61440 + 3072 + 208 + 9 = (64729)10

(7848)10 = (1EA8)16 = 0x1EA8

(46183)10 = (B467)16 = 0xB467

**Ejercicio 2**

No hay acarreo

Hay acarreo

No hay acarreo

No hay acarreo

Hay acarreo

**Ejercicio 3**

En el contexto de la aritmética de precisión fija, el acarreo se refiere al valor que se transfiere a la siguiente posición más significativa cuando la suma de dos dígitos excede la base en la que se está operando. Para que el acarreo sea mayor que *1*, la suma de dos dígitos en una posición debe exceder dos veces la base.

Consideremos una base *b*. Si sumamos dos números *a* y *d* en esta base, el acarreo *c* se calcula como:

*c = (a + d) / b*

Para que *c* sea mayor que *1*, *a + d* debe ser mayor que *2b*. Sin embargo, en cualquier base *b* , los valores *a* y *d* están restringidos a *0 ≤ a, d < b*. Por lo tanto, la suma *a + d* siempre será menor que *2b*, lo que implica que el acarreo *c* no puede ser mayor que *1*.

Por ejemplo, en base *10*, si sumamos los números más grandes posibles en una posición *9 + 9*, obtenemos *18*. El acarreo sería:

*c = 18 / 10 = 1*

Esto demuestra que en cualquier base, el acarreo no puede ser mayor que *1*. Por lo tanto, no es posible que la suma de dos números de precisión fija tenga un acarreo mayor que *1*.

**Ejercicio 4**

r = (1011 1111)2 es igual a:

* - (0 x 26 + 1 x 25 + 1 x 24 + 1 x 23 + 1 x 22 + 1 x 21 + 1 x 20) = - (0 + 32 + 16 + 8 + 4 + 2 + 1) = (-63)10 en signo + magnitud
* inv(1011 1111) + 0000 0001 = 0100 0000 + 0000 0001 = 0100 0001

= - (0 x 27 + 1 x 26 + 0 x 25 + 0 x 24 + 0 x 23 + 0 x 22 + 0 x 21 + 1 x 20)

= - (0 + 64 + 0 + 0 + 0 + 0 + 0 + 1) = (-65)10 en complemento a 2

s = (1000 0000)2 es igual a:

* - (0 x 26 + 0 x 25 + 0 x 24 + 0 x 23 + 0 x 22 + 0 x 21 + 0 x 20 ) = - (0 + 0 + 0 + 0 + 0 + 0 + 0) = (-0)10 en signo + magnitud
* inv(1000 0000) + 0000 0001 = 0111 1111 + 0000 0001 = 1000 0000

= - (1 x 27 + 0 x 26 + 0 x 25 + 0 x 24 + 0 x 23 + 0 x 22 + 0 x 21 + 0 x 20)

= - (128 + 0 + 0 + 0 + 0 + 0 + 0 + 0) = (-128)10 en complemento a 2

t = (1111 1111)2 es igual a:

* - (1 x 26 + 1 x 25 + 1 x 24 + 1 x 23 + 1 x 22 + 1 x 21 + 1 x 20 ) = - (64 + 32 + 16 + 8 + 4 + 2 + 1) = (-127)10 en signo + magnitud
* inv(1111 1111) + 0000 0001 = 0000 0000 + 0000 0001 = 0000 0001

= - (0 x 27 + 0 x 26 + 0 x 25 + 0 x 24 + 0 x 23 + 0 x 22 + 0 x 21 + 1 x 20)

= - (0 + 0 + 0 + 0 + 0 + 0 + 0 + 1) = (-1)10 en complemento a 2

**Ejercicio 5**

(0)10 es igual a:

* (0000 0000)2 en 8 bits y notación signo + magnitud
* (0000 0000)2 en 8 bits y notación complemento a 2

(-1)10 es igual a:

* (1000 0001)2 en 8 bits y notación signo + magnitud
* (1000 0000 0000 0001)2 en 16 bits y notación signo + magnitud
* inv(0000 0001) + 0000 0001 = 1111 1110 + 0000 0001 = (1111 1111)2 en 8 bits y notación complemento a 2
* inv(0000 0000 0000 0001) + 0000 0000 0000 0001 = 1111 1111 1111 1110 + 0000 0000 0000 0001

= (1111 1111 1111 1111)2 en 16 bits y notación complemento a 2

(255)10 es igual a:

* (1111 1111)2 en 8 bits y notación sin signo
* (0000 0000 1111 1111)2 en 16 bits y notación complemento a 2

(-128)10 es igual a:

* inv(1000 0000) + 0000 0001 = 0111 1111 + 0000 0001 = (1000 0000)2 en 8 bits y notación complemento a 2
* inv(0000 0000 1000 0000) + 0000 0000 0000 0001 = 1111 1111 0111 1111 + 0000 0000 0000 0001

= (1111 1111 1000 0000)2 en 16 bits y notación complemento a 2

(128)10 es igual a:

* (1000 0000)2 en 8 bits y notación sin signo
* (0000 0000 1000 0000)2 en 16 bits y notación complemento a 2

**Ejercicio 6**

Para una cadena binaria de k dígitos, el rango de representación en la notación signo + magnitud abarca desde - (2k - 1 - 1) hasta (2k - 1 - 1). En cambio para la notación en complemento a 2 el rango es desde - (2k - 1) hasta (2k - 1 - 1). Como se puede observar, para una misma cadena de k dígitos, la notación en complemento a 2 permite representar un número negativo más que en la notación de signo + magnitud. El valor - (2k - 1) puede ser representado en complemento a 2 pero no puede ser representado en signo + magnitud para una misma k. De manera contraria, todo valor que puede ser representado en notación signo + magnitud, también puede ser representado en notación complemento a 2.

**Ejercicio 7**

En complemento a 2 esto es la suma de -31 + 30 = -1. La operación es correcta y no hay overflow.

En complemento a 2 esto es la suma de -31 + 31 = 0. La operación es correcta. Hay acarreo, pero no hay overflow.

En complemento a 2 esto es la suma de 15 + 15 = 30. La operación no es correcta ya que al sumar 2 números positivo, me devuelve como resultado un número negativo (-2). Hay overflow.

En complemento a 2 esto es la suma de -26215 + 4369 = -21846. La operación es correcta. No hay overflow.

(9999)16 = (1001 1001 1001 1001)2 = - (inv(1001 1001 1001 1001) + 0000 0000 0000 0001)

= - (0110 0110 0110 0110 + 0000 0000 0000 0001) = - (0110 0110 0110 0111)2

= - (1 x 214 + 1 x 213 + 1 x 210 + 1 x 29 + 1 x 26 + 1 x 25 + 1 x 22 + 1 x 21 + 1 x 20)

= - (16384 + 8192 + 1024 + 512 + 64 + 32 + 4 + 2 + 1) = (-26215)10

(1111)16 = (0001 0001 0001 0001)2 = 1 x 212 + 1 x 28 + 1 x 24 + 1 x 20 = 4096 + 256 + 16 + 1 = (4369)10

(AAAA)16 = (1010 1010 1010 1010)2 = - (inv(1010 1010 1010 1010) + 0000 0000 0000 0001)

= - (0101 0101 0101 0101 + 0000 0000 0000 0001) = - (0101 0101 0101 0110)2

= - (1 x 214 + 1 x 212 + 1 x 210 + 1 x 28 + 1 x 26 + 1 x 24 + 1 x 22 + 1 x 21)

= - (16384 + 4096 + 1024 + 256 + 64 + 16 + 4 + 2) = (-21846)10

En complemento a 2 esto es la suma de -3856 - 3894 = -7750. La operación es correcta. Hay acarreo pero no hay overflow.

(F0F0)16 = (1111 0000 1111 0000)2 = - (inv(1111 0000 1111 0000) + 0000 0000 0000 0001)

= - (0000 1111 0000 1111 + 0000 0000 0000 0001) = - (0000 1111 0001 0000)2

= - (1 x 211 + 1 x 210 + 1 x 29 + 1 x 28 + 1 x 24)

= - (2048 + 1024 + 512 + 256 + 16) = (-3856)10

(F0CA)16 = (1111 0000 1100 1010)2 = - (inv(1111 0000 1100 1010) + 0000 0000 0000 0001)

= - (0000 1111 0011 0101 + 0000 0000 0000 0001) = - (0000 1111 0011 0110)2

= - (1 x 211 + 1 x 210 + 1 x 29 + 1 x 28 + 1 x 25 + 1 x 24 + 1 x 22 + 1 x 21)

= - (2048 + 1024 + 512 + 256 + 32 + 16 + 4 + 2) = (-3894)10

(E1BA)16 = (1110 0001 1011 1010)2 = - (inv(1110 0001 1011 1010) + 0000 0000 0000 0001)

= - (0001 1110 0100 0101 + 0000 0000 0000 0001) = - (0001 1110 0100 0110)2

= - (1 x 212 + 1 x 211 + 1 x 210 + 1 x 29 + 1 x 26 + 1 x 22 + 1 x 21)

= - (4096 + 2048 + 1024 + 512 + 64 + 4 + 2) = (-7750)10

**Ejercicio 8**

(7744)16 = (0111 0111 0100 0100)2 = 1 x 214 + 1 x 213 + 1 x 212 + 1 x 210 + 1 x 29 + 1 x 28 + 1 x 26 + 1 x 22

= 16384 + 8192 + 4096 + 1024 + 512 + 256 + 64 + 4 = (30532)10

(5499)16 = (0101 0100 1001 1001)2 = 1 x 214 + 1 x 212 + 1 x 210 + 1 x 27 + 1 x 24 + 1 x 23 + 1 x 20

= 16384 + 4096 + 1024 + 128 + 16 + 8 + 1 = (21657)10

(6788)16 = (0110 0111 1000 1000)2 = 1 x 214 + 1 x 213 + 1 x 210 + 1 x 29 + 1 x 28 + 1 x 27 + 1 x 23

= 16384 + 8192 + 1024 + 512 + 256 + 128 + 8 = (26504)10

(AB68)16 = (1010 1011 0110 1000)2 = - (inv(1010 1011 0110 1000) + 0000 0000 0000 0001)

= - (0101 0100 1001 0111 + 0000 0000 0000 0001) = - (0101 0100 1001 1000)2

= - (1 x 214 + 1 x 212 + 1 x 210 + 1 x 27 + 1 x 24 + 1 x 23)

= - (16384 + 4096 + 1024 + 128 + 16 + 8) = (-21656)10

(88BD)16 = (1000 1000 1011 1101)2 = - (inv(1000 1000 1011 1101) + 0000 0000 0000 0001)

= - (0111 0111 0100 0010 + 0000 0000 0000 0001) = - (0111 0111 0100 0011)2

= - (1 x 214 + 1 x 213 + 1 x 212 + 1 x 210 + 1 x 29 + 1 x 28 + 1 x 26 + 1 x 21 + 1 x 20)

= - (16384 + 8192 + 4096 + 1024 + 512 + 256 + 64 + 2 + 1) = (-30531)10

(9879)16 = (1001 1000 0111 1001)2 = - (inv(1001 1000 0111 1001) + 0000 0000 0000 0001)

= - (0110 0111 1000 0110 + 0000 0000 0000 0001) = - (0110 0111 1000 0111)2

= - (1 x 214 + 1 x 213 + 1 x 210 + 1 x 29 + 1 x 28 + 1 x 27 + 1 x 22 + 1 x 21 + 1 x 20)

= - (16384 + 8192 + 1024 + 512 + 256 + 128 + 4 + 2 + 1) = (-26503)10

(0003)16 = (0000 0000 0000 0011)2 = 1 x 21 + 1 x 20 = 2 + 1 = (3)10

(7744)16 + (5499)16 + (6788)16 + (AB68)16 + (88BD)16 + (9879)16 = (0003)16

(30532)10 + (21657)10 + (26504)10 - (21656)10 - (30531)10 - (26503)10 = (3)10

Para que no ocurra overflow al hacer la suma hay que intercalar los valores positivos y negativos, por ejemplo:

(30532)10 - (30531)10 + (21657)10 - (21656)10 + (26504)10 - (26503)10 = (3)10

**Ejercicio 9**

1. No se produce acarreo ni overflow: 1 + 4 = 5
2. Hay acarreo pero no hay overflow: 7 + (-6) = 1
3. Hay acarreo y overflow: (-4) + (-5) = -9
4. No hay acarreo pero si hay overflow: 5 + 4 = 9
5. Hay acarreo y el resultado es cero: 5 + (-5) = 0
6. No hay acarreo y el resultado es cero: 0 + 0 = 0
7. El resultado es negativo y hay overflow: 7 + 7 = 14
8. El resultado es negativo y no hay overflow: (-7) + 2 = -5

**Ejercicio 10**

La función SignExtn extiende un número de k bits a un número de k + n bits. Si es un número positivo en complemento a 2, es decir si la cifra más significativa es igual a cero, extiende el número con 0 a la izquierda. En cambio si el número es negativo en complemento a 2, es decir si la cifra más significativa es igual a uno, extiende el número con 1 a la izquierda. Si se trata de un número positivo, al extender el número con ceros, el valor no cambia. En el caso de un número negativo ocurre lo mismo, ya que al extender el número con unos, la cifra más significativa sigue siendo un uno, y como la magnitud del número se calcula invirtiendo el número y sumándole uno, el valor se mantiene igual.

**Ejercicio 11**

(2)10 = (0010)2

(-5)10 = - (0101)2 = inv(0101) + 0001 = 1010 + 0001 = (1011)2

(0)10 = (0000)2

1. inv(0010) = (1101)2 = (-3)10

inv(1011) = (0100)2 = (4)10

inv(0000) = (1111)2 = (-1)10

1. Para obtener el inverso aditivo en notación complemento a 2 hay que primero invertir los bits y luego sumar 1.

**Ejercicio 12**

Para que un sistema de representación de números con signo utilizando cadenas binarias de longitud fija sea biyectivo, debe cumplir con las siguientes condiciones:

* Biyectividad: Cada número debe tener una única representación y cada cadena binaria debe representar un número.
* Representación del cero: Debe haber una representación específica para el cero.
* Igual cantidad de números positivos y negativos: La cantidad de números positivos y negativos representados debe ser la misma.

Consideremos un sistema de longitud fija con *n* bits:

El número total de cadenas binarias posibles es *2n*.

Para incluir el cero, necesitamos una cadena específica para representarlo.

Para tener igual cantidad de números positivos y negativos, debemos dividir las cadenas restantes entre los números positivos y negativos.

Una cadena se usa para el cero, dejando *2n - 1* cadenas para los números positivos y negativos.

Para tener igual cantidad de números positivos y negativos, *2n - 1* debe ser un número par, lo cual no es posible ya que *2n - 1* siempre es impar. Por lo tanto, no es posible diseñar un sistema que cumpla con todas estas condiciones simultáneamente, haciendo que la afirmación sea verdadera.

**Ejercicio 13**

FALTA!!