SISTEMAS NUMÉRICOS.

Ing. Miguel Antonio Martínez.
PROFESOR ADJUNTO.
Dto. Electrónica. Facultad de Ingeniería (UBA).
Dto. Sistemas. Facultad de Ciencias Económicas (UBA).

SISTEMAS NUMÉRICOS.

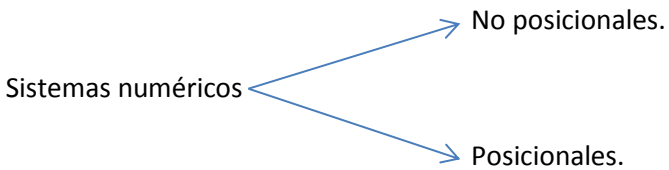
La habilidad para contar, cuantificar y representar números y cantidades es una de los pilares de la inteligencia humana. Tal es así que a lo largo de la historia civilizaciones enteras han dificultado o detenido su desarrollo científico/tecnológico debido a que no contaban con un sistema de representación de cantidades. En este capítulo se estudiarán diferentes formas de representar números según sus características de una manera adecuada para las maquinas digitales.

Definiremos un **sistema de numeración** como un conjunto de símbolos y reglas de generación que permitan construir todos los números validos del sistema. Un sistema de numeración puede representarse como:

N = S + R

- **N** es el sistema de numeración considerado.
- **S** son los símbolos permitidos del sistema.
- **R** son las reglas de generación que nos indican que números son válidos y cuales no en el sistema.

Dentro de los sistemas numéricos se puede hacer dos grandes divisiones:



No Posicionales o No Ponderados son aquellos en los cual el valor de los símbolos que componen el sistema es fijo, o sea no depende de la posición que ocupe este dentro del número, ejemplo de este tipo es el sistema romano.

Sistema Posicional o Ponderado son aquellos que el valor de los símbolos depende del valor que se les ha asignado y de la posición que ocupa el símbolo dentro del número. Podemos decir que cada símbolo tiene dos valores, uno **absoluto** que depende del valor del signo y otro **relativo** que depende de la posición que ocupa dentro del mismo. Ejemplo de este sistema es el griego.

Como se comprende los sistemas más usados son los posicionales. La costumbre, adquirida desde la infancia, de utilizar un único sistema de numeración, suele hacer que quien lo utiliza olvide que los conceptos que se utilizan a diario podrían adecuarse con facilidad a otros sistemas. Como todos sabemos el sistema a que nos referimos es al decimal. Se cree que el hombre lo empezó a utilizar por poseer diez dedos y era la forma más elemental de contar. Este sistema está formado por diez símbolos, los números del 0 al 9 conocidos como **"dígitos"**. Se define como **"dígito"** a cada uno de los símbolos diferentes que constituyen el sistema de numeración.

Así mismo se define como **"base"** del sistema de numeración a la cantidad de dígitos que lo forman. Una vez agotada la cantidad de dígitos que forman el sistema de numeración, las cantidades mayores a la base se obtienen combinando en forma adecuada los diferentes dígitos del sistema. Esto hace que cada uno de los dígitos adopte distintos valores según la posición que ocupe dentro del número representado. La existencia de este **valor relativo o peso**, que define al dígito según su posición más allá del valor absoluto, es la característica que permite definir al sistema decimal como un sistema numérico **posicional.**

Por ejemplo en el número decimal 3434, será distinto entre sí el valor de los dos dígitos 3 y lo mismo sucede para los dos números 4.

$$3434_{(10} = 3000 + 400 + 30 + 4$$

Una forma más clara es si expresamos en número en función de su base 10.

$$3434_{(10} = 3 \cdot 10^{3} + 4 \cdot 10^{2} + 3 \cdot 10^{1} + 4 \cdot 10^{0}$$

Generalizando se puede decir que cualquier número **N** de **n** dígitos, expresado en una base **B,** donde los dígitos del mismo se representan con la letra **a,** permite definir un número natural cualquiera **N** como:

$$N_B = a_0 \cdot B^0 + a_1 \cdot B^1 + a_2 \cdot B^2 + \ldots\ldots\ldots\ldots + a_{n-1} \cdot B^{n-1}$$

O en forma más abreviada:

$$N_B = \sum_{i=0}^{n-1} a_i \cdot B^i$$

También podemos representar números con decimales en un sistema posicional, siguiendo con nuestro ejemplo si queremos representar $3434.25_{(10}$, podemos desarrollarlo como:

$$3434.25_{(10} = 3 \cdot 10^3 + 4 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0 + 2 \cdot 10^{-1} + 5 \cdot 10^{-2}$$

Generalizando como en el caso anterior, un número **N** de **n** dígitos enteros y **k** dígitos fraccionarios, donde los dígitos del mismo se representan con la letra **a**, permite definir un número cualquiera **N** como:

$$N_B = a_{-k} \cdot B^{-k} + a_{-k+1} \cdot B^{-k+1} + \ldots\ldots\ldots + a_0 \cdot B^0 + \ldots\ldots\ldots + a_{n-2} \cdot B^{n-2} + a_{n-1} \cdot B^{n-1}$$

O en forma más abreviada:

$$N_B = \sum_{i=-k}^{n-1} a_i \cdot B^i$$

Por ejemplo si queremos representar con la formula anterior al número $541.25_{(10}$, tenemos en este caso k=2, n=3 y B=10, lo que queda:

$$5 \times 10^2 + 4 \times 10^1 + 1 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2} =$$
$$= (500) + (40) + (1) + (2/10) + (5/100) = (541.25)$$

Si, en forma similar se considera el numero binario $1010.01_{(2,}$ en este caso k=2, n=4 y B=2, nos queda:

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} =$$
$$= (8) + (0) + (2) + (0) + (0/2) + (1/4) = (10.25)$$

Este último procedimiento da una idea acerca de cómo convertir un número expresado en un sistema de numeración de una base cualquiera al sistema de numeración decimal, mediante la utilización de una **representación polinómica.** Se trata de multiplicar

cada dígito por el peso asignado a su posición (potencias de dos en este ejemplo) y luego sumar para obtener el número convertido.

Importante, en el ejemplo anterior pudimos hallar el equivalente decimal porque hicimos las operaciones en esa base, si hubiéramos usado otra base obtendríamos el equivalente en esa base distinta a la decimal.

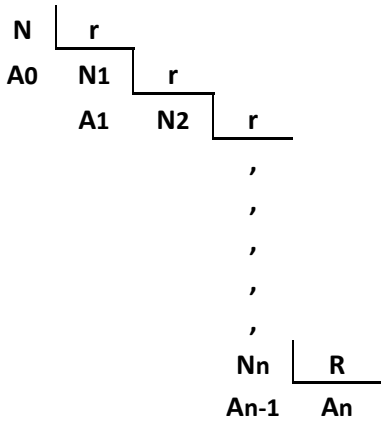
Más adelante veremos cómo usar este procedimiento para pasar de una base a otra, para ir adelantándonos diremos que este método es ventajoso cuando pasamos de **una base menor a una mayor.**

Nótese que en sistemas numéricos binarios posicionales se define el bit con el mayor peso asociado como **bit más significativo (MSB,** most significant bit) y el bit de menor peso como **bit menos significativo (LSB,** least significant bit). Por convención, el LSB es el bit a la derecha de la expresión numérica, en tanto que el MSB es el bit a la izquierda de la misma.

CONVERSION ENTRE BASES.

Supongamos que tenemos un numero **N** expresado en la base **s**. Este se puede convertir a la base **r,** mediante la siguiente secuencia de divisiones efectuadas en la base **s**. Los dígitos $\mathbf{A_i}$ son los residuos de cada división, de tal modo que $\mathbf{A_i} < r$.

La operatoria sería:



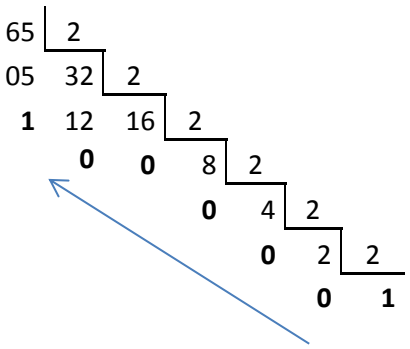
La división se puede escribir de la siguiente forma:

$$N = r \cdot N_1 + A_0$$
$$N_1 = r \cdot N_2 + A_1$$
$$\vdots$$
$$N_n = r \cdot 0 + A_n$$

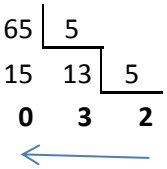
O bien como:

$$N = r \,.\, (rN_2 + A_1) + A_0$$
$$= r^2 \,.\, N_2 + r \,.\, A_1 + A_0$$
$$= r^2 \,(r.\, N_3 + A_3) + r \,.\, A_1 + A_0$$
.
.
.
.
.
.
$$= A_n \,.\, r^n + A_{n-1} \,.\, r^{n-1} + \ldots\ldots\ldots\ldots\ldots + A_1 \,.\, r + A_0$$

Ejemplo: Convertir $65_{(10}$ a base 2 y a base 5



O sea que $65_{(10} = 1000001_{(2}$
Si queremos verificarlo, hacemos: $1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 65_{(10}$



O sea que $65_{(10} = 230_{(5}$
Si queremos verificarlo, hacemos $0 \times 5^0 + 3 \times 5^1 + 2 \times 5^2 = 0 + 15 + 50 = 65_{(10}$

Ahora veremos cómo operar entre base con números fraccionarios, para ello primero ponemos la expresión general de un numero N que está en una base **s** y lo pasamos a una base **r.** La expresión de este número en la nueva base, con su parte entera y su parte fraccionaria, será:

$$N = N_E + N_F = A_n \,.\, r^n + \ldots\ldots + A_1 \,.\, r^1 + A_0 \,.\, r^0 + A_{-1} \,.\, r^{-1} + A_{-2} \,.\, r^{-2} + \ldots\ldots$$

Las incógnitas son los coeficientes A_i . Si empiezo a multiplicar por r nos queda:

$$r \,.\, N_F = A_{-1} + A_{-2} \,.\, r^{-1} + A_{-3} \,.\, r^{-2} + \ldots\ldots$$

La parte entera de $r \,.\, N_F$ es $A_{-1,}$ que es nuestra primera incógnita, se resta y proseguimos.

$$r \cdot (r \cdot N_F - A_{-1}) = A_{-2} + A_{-3} \cdot r^{-1} + A_{-4} \cdot r^{-2} + \ldots\ldots\ldots$$

Y así, sucesivamente. Ejemplo: Pasar a base 2, el numero $65.74_{(10}$. La parte real ya la hallamos, para la parte fraccionaria multiplicamos por dos y nos quedamos con la parte entera, luego restamos esta y seguimos multiplicando la parte fraccionaria:

$$0.74 \times 2 = 1.48 \quad A_{-1} = 1$$
$$0.48 \times 2 = 0.96 \quad A_{-2} = 0$$
$$0.96 \times 2 = 1.92 \quad A_{-3} = 1$$
$$0.92 \times 2 = 1.84 \quad A_{-4} = 1$$

Y así sucesivamente. La pregunta es hasta que cifra seguimos si los restos no se hacen cero. Esto lo contestaremos más adelante, pero depende del error de truncamiento que queramos tener. En este caso:

$$65.74_{(10} = 1000001.1011_{(2}$$

El tema que queremos abordar es poder calcular la cantidad de dígitos que necesito para expresar un numero en distintas bases. Por ejemplo si tengo el numero $50_{(10}$ y quiero saber cuántos dígitos binarios (bits) necesito para expresar esta cantidad, se debe cumplir que:

$$2^{n-1} - 1 < 50 \leq 2^n - 1$$

Aplicando logaritmos para hallar n, nos queda:

$$n \geq \log (50+1)/\log 2$$
$$n \geq 1.70757 / 0.30103$$
$$n \geq 5.67 \approx 6$$

Si buscamos el equivalente binario del decimal 50 nos da $110010_{(2}$. Como vemos necesitamos 6 bits. Generalizando para cualquier número M nos queda:

$$2^{n-1} - 1 < M \leq 2^n - 1$$

Si en vez de base 2 tomamos cualquier base B, nos queda:

$$B^{n-1} - 1 < M \leq B^n - 1$$

Si ahora tomamos dos bases B_1 y B_2 sería:

$$(B_1)^{n-1} - 1 < M \leq (B_1)^n - 1$$
$$(B_2)^{m-1} - 1 < M \leq (B_2)^m - 1$$

Comparando los segundos términos de cada desigualdad, podemos simplificar diciendo que:

$$\boxed{(B_1)^n = (B_2)^m}$$

Formula que me permite calcular la cantidad de dígitos que necesito en una base cuando conozco la cantidad de dígitos en otra base distinta. Tener en cuenta que siempre **se toma el entero superior.**

Ejemplo: Cuantos dígitos binarios necesito para expresar un número decimal de 3 cifras.
Resolución:

Si aplicamos la formula anterior nos queda:

$$2^x = 10^3$$

Aplicando logaritmos:

$$x.\ \log(2) = 3\ .\ \log(10)$$
$$x = 3\ .\ \log(10)/\log(2)$$
$$x = (3\ .\ 1)\ /\ 0.30103$$
$$x = 9.96 \approx \mathbf{10}$$

Otra forma sería buscar cual es número mayor de tres cifras en base 10, o sea el 999. Luego dividir sucesivamente por 2 y llegamos a $1111100111_{(2}$. Si contamos los bits nos da 10.

REPRESENTACION EN LOS SISTEMAS BINARIO, OCTAL Y HEXADECIMAL.

Si bien los números binarios reflejan la realidad de la representación interna de los números tal como se utiliza en la inmensa mayoría de las computadoras, tienen como desventaja el hecho de requerir mayor cantidad de dígitos para representar un numero dado que cualquier otro sistema de numeración posicional. Asimismo, suele ser más fácil cometer cuando se escriben números binarios debido a la gran cantidad de ceros y unos que hay que utilizar en la representación. Veremos que vinculación hay entre el **sistema de numeración octal** (sistema de base 8) y el **sistema de numeración hexadecimal** (sistema de base 16). Esta relación está dada por el hecho de ser estas bases potencias de dos. Se procederá a demostrar que la conversión entre los sistemas de numeración binario, octal y hexadecimal es trivial y que hay ventajas prácticas significativas en el uso de estos sistemas para la representación de números.

Los números binarios pueden ser considerablemente más grandes (en cantidad de dígitos) que sus equivalentes decimales. Suele resultar práctico como elemento de representación el utiliza aquellos sistemas de numeración cuyas bases son potencias de dos. La conversión entre los sistemas de numeración de bases 2, 8 y 16 en mucho más sencilla que convertir hacia y desde el sistema decimal. Los valores utilizados para los dígitos del sistema octal resultan familiares por cuanto son los primeros ocho dígitos del sistema decimal. En cambio, para el sistema hexadecimal, se requieren seis dígitos más que los que se usan en el sistema decimal. La conversión habitual para la representación de los dígitos adicionales (10, 11, 12, 13, 14 y 15) del sistema hexadecimal pasa por el uso de las seis primeras del abecedario, sean mayúsculas o minúsculas. En la siguiente figura se ven los dígitos utilizados comúnmente en los sistemas de numeración de bases 2, 8, 10 y 16.

Sistema Numérico

Decimal (Base 10)	Binario (Base 2)	Octal (Base 8)	Hexadecimal (Base 16)
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Al comparar la columna correspondiente al sistema binario con las columnas de los sistemas octal y hexadecimal surge que se requieren tres bits para representar en binario cada uno de los dígitos del sistema octal, y cuatro bits para representar en binario cada uno de los 16 dígitos del sistema hexadecimal. En general, se requieren **k** bits para representar en binario un dígito del sistema de numeración de base $\mathbf{2^k}$, siendo **k** un numero entero, por lo que el sistema de numeración de base $\mathbf{8 = 2^3}$ requiere de tres bits por dígito, en tanto que el sistema de numeración de base $\mathbf{16 = 2^4}$ requiere cuatro bits por dígito.

Para convertir un numero expresado en el sistema binario al sistema octal, se divide el número binario original en grupos de tres bits cada uno, empezando a partir de la como decimal, completando el grupo más significativo con ceros, en caso de ser necesario. Luego, cada trío de bits se convierte en forma individual al sistema octal. Para conversiones desde el sistema binario al hexadecimal se utilizan grupos de cuatro bits. Ejemplo, si se quiere convertir el número $(10110)_2$ al sistema de base 8, el procedimiento es el siguiente:

$$(10110)_2 = (010)_2\,(110)_2 = (2)_8\,(6)_8\,) = (26)_8$$

En el caso de los dos bits de mayor peso, se agregó un cero a la izquierda para completar el trío correspondiente.

Si se considera la conversión del número binario $(10110110)_2$ al sistema hexadecimal, el procedimiento similar al anterior lleva a:

$$(10110110)_2 = (1011)_2\,(0110)_2 = (B)_{16}\ (6)_{16} = (B6)_{16}$$

Si quiero pasar de base hexadecimal o base octal al sistema binario, hago el camino inverso. En vez de agrupar debo dividir cada dígito en la cantidad de bits que corresponda. Ejemplo, Si quiero pasar el número $(306.D)_{16}$ al sistema binario, hago lo siguiente:

3	0	6	.	D
0 0 1 1	0 0 0 0	0 1 1 0		1 1 0 1

O sea me queda que $306.D_{16} = 001100000110.1101_2$

Si quiero pasar a base 8 agrupo:

0 0 1	1 0 0	0 0 0	1 1 0	.	1 1 0	1 **0 0**
1	4	0	6	.	6	4

Como se ve, agregué dos ceros a la derecha para poder formar el último trío.

Si quiero pasar el binario a base 4 agrupo de a dos:

0 0	1 1	0 0	0 0	0 1	1 0	.	1 1	0 1
0	3	0	0	1	2	.	3	1

Resumiendo, en este ejemplo nos queda:

$$(306.D)_{16} = (1100000110.1101)_2 = (1406.64)_8 = (30012.31)_4$$

ARITMÉTICA EN OTRAS BASES.

Sumar y multiplicar en algunas bases nos resulta fácil y cotidiano. Esto pasa en la base decimal y hasta en la base dos también. Cuando hay que realizar estas operaciones en bases distintas a las nombradas, a veces el proceso es complejo. Para ayudar en esta tarea existen tablas de suma y multiplicación en la base que deseemos trabajar. Para empezar a interiorizarnos con esto diremos que estas tablas son un tipo de matriz donde tenemos todas las combinaciones posibles de las operaciones suma y producto. La más simple es la de base dos, en este caso tendremos una tabla con 4 combinaciones posibles, o sea podemos ubicarlas en una matriz de dos filas y dos columnas como se muestra:

suma	0	1
0	0	1
1	1	10

producto	0	1
0	0	0
1	0	1

Este ejemplo que veremos es muy simple, por ejemplo queremos sumar y multiplicar dos números binarios y los dos son 11.

	1		me llevo									
	1	1				1	1					
	1	1		x			1	1				
1	**1**	**0**				1			me llevo			
							1	1				
						1	1					
					1	**0**	**0**	**1**				

Obviamente se ve que $11_{(2}$ = 3 en decimal, lo cual el resultado de la suma y la multiplicación dan 6 y 9 respectivamente. Como dijimos en la base decimal y en la binaria no hay demasiados dramas, pero veremos algo más complejo. Por ejemplo si queremos trabajar en la base 5, empezaremos por escribir las tablas de suma y producto en esa base:

suma	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	10
2	2	3	4	10	11
3	3	4	10	11	12
4	4	10	11	12	13

prod	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	11	13
3	0	3	11	14	22
4	0	4	13	22	31

Por ejemplo quiero hacer $133_{(5}$ multiplicado por $23_{(5}$, sería:

	1	3	3	
x		2	3	
	1	1		me llevo
	3	4	4	
1	1			me llevo
2	1	1		
1	1			me llevo
4	**2**	**1**	**4**	

De la tabla vemos que la primera multiplicación es 3 x 3 = 14, por lo tanto queda 4 y me llevo 1, la segunda multiplicación es nuevamente 3 x 3 = 14, otra vez queda 4 y me llevo 1. La última multiplicación es 3 x 1 = 3, o sea queda 344 y me llevo 11. Cuando tomo el 2 y hago el producto con 3, me queda 2 x 3 = 11, queda 1 y me llevo 1. Nuevamente 2 x 3 = 11, dejo 1 y me llevo 2. Por último 2 x 1 = 2. O sea queda 211 y me llevo 11. Ahora sumo todo, el digito menos significativo es 4, después me queda 4 + 1 + 1 = 11, dejo el 1 y me llevo 1. La tercera columna es 1 + 3 + 1 + 1 + 1 = 12, dejo el 2 y me llevo 1. Por último 1 + 2 + 1 = 4.

También las tablas me permiten llevar cualquier numero expresado en una base menor a 5 a base 5, por ejemplo quiero pasar $221_{(2}$ a base 5. Primero desarrollo el 221 en potencias y coeficientes en su base (o sea 2):

$$1 \times 2^0 + 2 \times 2^1 + 2 \times 2^2$$

Cabe aclarar que a partir de este polinomio puedo ir a **cualquier base,** siempre y cuando haga las operaciones en la base a la que quiero llegar. Por demás está decir que si opero en decimal me queda $1 + 4 + 8 = 13_{(10}$. Pero este no es nuestro objetivo, si no ir a base 5.

$$1 + 4 + 2.(4) =$$
$$= 1 + 4 + 13 =$$
$$\mathbf{= 23_{(5}}$$

O sea que $221_{(2} = 13_{(10} = 23_{(5}$

Si ahora quiero pasar $321_{(4}$ a base 5 haría:

$$1 \times 4^0 + 2 \times 4^1 + 3 \times 4^2 =$$
$$= 1 + 13 + 3 \,.\, (31) =$$
$$= 1 + 13 + 143 =$$
$$\mathbf{= 212_{(5}}$$

Ahora resolveremos un ejercicio tipo examen:

(Ejercicio tipo examen).

Con ayuda de las tablas de sumar y multiplicar convertir el numero $32102_{(4}$ a base 6.

Resolución:

En este caso tenemos un número expresado en una base y queremos expresarlo en otra base **que es más grande que la original.** Recordar que cuando vamos de una base superior a una inferior lo mejor es dividir, no es nuestro caso.

Recordar que cualquier número natural se puede expresar en su base como:

$$N = \sum a_i \, b^i$$

Donde a_i son los coeficientes y b^i las distintas potencias de la base, i varía desde 0 hasta n, donde n es la cantidad de dígitos -1 que posee el número.

Si expresamos el número 32102, que nos dicen está en base 4, como potencias de su base, nos queda:

$$32102_{(4} = 2x4^0 + 0x4^1 + 1x4^2 + 2x4^3 + 3x4^4$$

Tener presente que como resolvamos este polinomio será la base a la que vamos, si lo resolvemos en decimal encontraremos el equivalente decimal del número $32102_{(4.}$

En nuestro ejemplo nos piden ir a la base 6, por lo tanto todas las operaciones las debemos hacer en esta base, o sea 6. Para ayudarnos con los cálculos escribimos las tablas de sumar y multiplicar en esta base.

	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	10
2	2	3	4	5	10	11
3	3	4	5	10	11	12
4	4	5	10	11	12	13
5	5	10	11	12	13	14

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	10	12	14
3	0	3	10	13	20	23
4	0	4	12	20	24	32
5	0	5	14	23	32	41

Empecemos por el exponente más grande, o sea realizaremos $3x4^{4.}$ Según la tabla 4 x 4 = 24 en base 6, por lo tanto para calcular 4^4 debemos hacer 24 x 24 y después multiplicarlo por tres.

	2	4		
x		2	4	
	1			me llevo
	1	2		me llevo
		2	4	producto
	1			me llevo
	4	2		producto
1	**1**	**0**	**4**	

Ahora el resultado lo multiplicamos por tres.

	1	1	0	4	
x				3	
			2		Me llevo.
	3	3	0	0	producto
	3	**3**	**2**	**0**	

O sea 3 x 4^4 da 3320 en base 6.

Ahora resolveremos 2 x $4^{3,}$ o sea hacemos 24 x 4 y después multiplicamos por 2:

	2	4	
x		4	
	1	4	4

x			2	
		1	1	me llevo
		2	2	2 producto
		3	**3**	**2**

Directamente de la tabla se ve que $1 \times 4^2 = 24$, $0 \times 4^1 = 0$ y $2 \times 4^0 = 2$.
Ahora sumamos todo (siempre recordar que estamos en **base 6**).

	1	1		me llevo
3	3	2	0	
	3	3	2	
		2	4	
			2	
4	**1**	**2**	**2**	

Quiere decir que:

$$32102_{(4} = 4122_{(6}$$

Si queremos verificar si no nos equivocamos podemos pasar los dos números a decimal:

$2 \times 4^0 + 0 \times 4^1 + 1 \times 4^2 + 2 \times 4^3 + 3 \times 4^4 = 2 + 0 + 16 + 128 + 768 =$ **914$_{(10.}$**

$2 \times 6^0 + 2 \times 6^1 + 1 \times 6^2 + 4 \times 6^3 = 2 + 12 + 36 + 864 =$ **914$_{(10.}$**

EJERCICIOS RESUELTOS.

1) Pasar a la base binaria el número 45 decimal.

Solución:

Como anteriormente la forma más sencilla de pasar de una base mayor a una menor es dividir por la nueva base, en este caso nos queda:

45	2				
05	22	2			
1	02	11	2		
	0	**1**	5	2	
			1	2	2
				0	**1**

Tomando desde el último coeficiente hasta el primero nos queda que:

$$45_{(10} = 101101_{(2}$$

2) Pasar a la base 5 el decimal 289.

Solución:

Al igual que el ejercicio anterior me conviene dividir por la nueva base, en este caso base 5. O sea:

245	5			
45	49	5		
0	**4**	9	5	
		4	1	
			1	

Al igual que en el caso anterior me queda:

$$245_{(10} = 1440_{(5}$$

3) Pasar al base binaria el numero decimal 45.125.

Solución:

La parte entera ya la resolvimos en el ejercicio número 1, para la parte fraccionaria multiplicamos sucesivamente por dos y nos quedamos con la parte entera, o sea:

$$0.125 \times 2 = 0.250 \quad \mathbf{0}$$
$$0.250 \times 2 = 0.500 \quad \mathbf{0}$$
$$0.500 \times 2 = 1.000 \quad \mathbf{1}$$

Nos queda:

$$45.125_{(10} = 101101.001_{(2}$$

4) Pasar a la base decimal el número 65 que está expresado en base 7.

Solución:

En este caso como vamos de una base menor a una mayor nos conviene expresar el numero en potencias de la base original y operar en la decimal. O sea:

$$65_{(7} = 5 \times 7^0 + 6 \times 7^1 = 5 + 42 = 47$$
$$65_{(7} = 47_{(10}$$

5) Pasar a la base decimal el número 321.12 que está expresado en base 4.

Solución:

Al igual que en el ejercicio anterior por pasar de una base menor a una mayor desarrollamos el número dado en potencias de su base:

$$3 \quad \times 4^2 + 2 \times 4^1 + 1 \times 4^0 + 1 \times 4^{-1} + 2 \times 4^{-2} =$$
$$4 \quad \times 16 + 8 + 1 + 0.25 + 2 \times 0.0625 =$$
$$64 + 8 + 1 + 0.25 + 0.125 =$$
$$73.375$$

$$321.12_{(4} = 73.375_{(10}$$

6) Sin pasar por la base decimal convertir los números $2122_{(4}$ y $12321_{(4}$ a base 8.

Solución:

Se puede resolver como el ejercicio anterior desarrollando en potencias de la base (4 en este caso) y realizando las operaciones en base 8. Otra forma es usar bases que son potencias de otras bases. En este caso 8 no es potencia de 4, pero si las dos son potencias de la base binaria. Por lo tanto puedo distribuir los dígitos en base 2, teniendo en cuenta que $2^2 = 4$.

2	1	2	2
10	01	10	10

Ahora agrupamos de a tres:

010	011	010
2	3	2

O sea que $2122_{(4} = 232_{(8}$

Ahora para el caso del $12321_{(4}$ hacemos los mismo:

1	2	3	2	1
01	10	11	10	01

110	111	001
6	7	1

O sea que $12321_{(4} = 671_{(8}$

7) Pasar a la base 3, el número 666.666 que está expresado en la base 9.

Solución:

Al igual que en los casos anteriores 9 es una potencia de 3, por lo tanto divido el número expresado en base 9 en duplas:

6	6	6	.	6	6	6
20	20	20		20	20	20

Por lo tanto $666.666_{(9} = 202020.20202_{(3}$

8) Pasar a bases 2, 4 y 8 el número hexadecimal ABCD.EF.

Solución:

Desagregamos el número hexadecimal de la siguiente forma:

A	B	C	D	.	E	F
1010	1011	1100	1101		1110	1111

Así y lo tenemos en base 2, para pasar a base 4 agrupamos de a dos y para pasar a base 8 de a tres:

101	010	111	100	.	111	011	11**0**
5	2	7	4		7	3	6

10	10	10	11	11	00	11	01	.	11	10	11	11
2	2	2	3	3	0	3	1		3	2	3	3

O sea que $ABCD.FE_{(16} = 1010101111001101.11101111_{(2} = 5274.736_{(8} = 22233031.3233_{(4}$

9) Indicar en que bases son correctas las operaciones 6 + 7 = 11 y 5 + 7 = 13.

Solución:

Para resolver este ejercicio se puede pensar de la siguiente manera, en decimal 6 + 7 da 13, habría que calcular en que base el 13 se representa como 11. Lo mismo con el 5 + 7, en decimal da 12. Por lo tanto pensaríamos en que base 12 se representa como 13. Pero la forma más académica de resolverlo es desarrollar en potencias de la base, que justamente es nuestra incógnita.

$$6.b^0 + 7.b^0 = 1.b^0 + 1.b^1$$
$$6 + 7 = 1 + b$$
$$6 + 7 - 1 = b$$
$$\mathbf{b = 12}$$

$$5.b^0 + 7.b^0 = 3.b^0 + 1.b^1$$
$$5 + 7 = 3 + b$$
$$5 + 7 - 3 = b$$
$$\mathbf{b = 9}$$

10) Las tres cifras de un numero decimal suma 18, si se le resta el que resulta de invertir el orden de sus cifras, se obtiene 594. La cifra de las decenas es media aritmética de las otras dos. Hallar dicho número.

Solución:

Diremos que buscamos un número decimal del tipo X = a b c, de acuerdo al enunciado nos queda:

$$a + b + c = 18$$
$$a \times 10^2 + b \times 10^1 + c \times 10^0 - c \times 10^2 - b \times 10^1 - a \times 10^0 = 594$$
$$b = (a + c)/2$$

Trabajando un poco la segunda ecuación nos queda:

$$100a + 10b + c - 100c - 10b - a = 594$$
$$99a - 99c = 594$$

De la primera y tercera ecuación nos queda:

$$a + b + c = 18 \Rightarrow b = 18 - a - c$$
$$a + c = 2b \Rightarrow b = (a + c)/2$$

De estas dos últimas:

$$18 - a - c = (a + c)/2$$
$$36 - 2a - 2c = a + c$$
$$36 - 3a = 3c$$
$$12 - a = c$$

Reemplazando en la segunda expresión:

$$99a - 99(12 - a) = 594$$
$$99a - 1188 + 99a = 594$$
$$198a = 1782 \Rightarrow \mathbf{a = 9}$$
$$12 - a = c$$
$$12 - 9 = c \Rightarrow \mathbf{c = 3}$$
$$b = (a + c)/2$$
$$b = (9 + 3)/2 \Rightarrow \mathbf{b = 6}$$

O sea el número buscado es:

$$\mathbf{X = 963}$$

11) Un profesor dice que hay 100 estudiantes en la clase de los cuales 24 son varones y 32 son mujeres. ¿Qué base numérica está utilizando el profesor?.

Solución:

Suponiendo que llamamos **b** a la base pedida, nos queda que:

$$100_{(b} = 24_{(b} + 32_{(b}$$

Desarrollando en potencias de la base **b**, nos queda:

$$0 \times b^0 + 0 \times b^1 + 1 \times b^2 = 4 \times b^0 + 2 \times b^1 + 2 \times b^0 + 3 \times b^1$$
$$b^2 = 4 + 2 \times b^1 + 2 + 3 \times b^1$$
$$b^2 = 6 + 5 \times b^1$$
$$b^2 - 5b - 6 = 0$$

Las raíces de esta ecuación cuadrática son b = -1 y b = 6, por lo tanto la solución pedida es **b=6.** Con esta solución la cantidad de alumnos (en decimal) son 36 estudiantes en total, de los cuales son 16 varones y 20 mujeres.

12) Realizar la suma de los números expresados en base 8, ellos son $1372_{(8}$ y $4631_{(8.}$

Solución:

$$\begin{array}{cccc} 1 & 1 & & \\ 1 & 3 & 7 & 2 \\ 4 & 6 & 3 & 1 \\ \hline \mathbf{6} & \mathbf{2} & \mathbf{2} & \mathbf{3} \end{array}$$

13) Un profesor de matemática califica los exámenes de la siguiente manera: el primer problema vale un punto, el segundo 2, el tercero 4, el cuarto 8 y así sucesivamente. Un problema, o está bien o está mal, no hay término medio. Un alumno aprueba si al menos la mitad de todos los problemas está bien. Un estudiante obtuvo en el examen de junio, que

constaba de 10 problemas, la cantidad de 581 puntos. Averigua que problemas hizo bien y si aprobó el examen.

Solución:

Como los problemas se puntúan en forma creciente en potencias de 2, la solución vendrá dada al expresar 581 puntos conseguidos en base 2, o sea:

$$581_{(10} = 1001000101_{(2}$$

De modo que hizo bien los problemas 1, 3, 7 y 10. Como resolvió solo 4 ejercicios **no aprobó el examen.**

14) Un número N se representa en una base b como 51 y en la base (b + 1) como 44. Calcular dicha base.

Solución:

$$N = 51_{(b} = 44_{(b+1}$$

$$1 \times b^0 + 5 \times b^1 = 4 \times (b+1)^0 + 4 \times (b+1)^1$$

$$1 + 5b = 4 + 4b + 4$$

$$1 + 5b = 8 + 4\,b$$

$$5b - 4b = 8 - 1$$

$$\mathbf{b = 7}$$

15) Hallar **a** para que $10a5_{(6}$ sea igual que $a2aa_{(4}$.

Solución:

Para que los dos números (que están en distintas bases) sean iguales se tiene que cumplir que:

$$6 \quad \times 6^0 + a \times 6^1 + 0 \times 6^2 + 1 \times 6^3 = a \times 4^0 + a \times 4^1 + 2 \times 4^2 + a \times 4^3$$

$$5 + 6a + 216 = a + 4a + 32 + 64a$$

$$221 + 6a = 32 + 69a$$

$$189 = 63a$$

$$\mathbf{a = 3}$$

16) Sin pasar por la base decimal convertir el número 432 que está expresado en base 5 a la base binaria.

Solución:

Como voy de una base mayor a una menor, lo más práctico es dividir el número sucesivamente por la base a la que quiero llegar. El problema principal es no olvidarse que estamos en base 5. Cuando tengo un solo dígito no hay problema, por ejemplo $4_{(5}$ es igual a $4_{(10}$, lo mismo pasa con 3 y con 2 tomados individualmente. Hacemos la primera división y la explicamos paso a paso:

432	2
0	2

Hasta acá no hay problema, dividimos como en la primaria. 4 dividido 2, da 2. 2 por 2 es igual a 4 y resto 0.

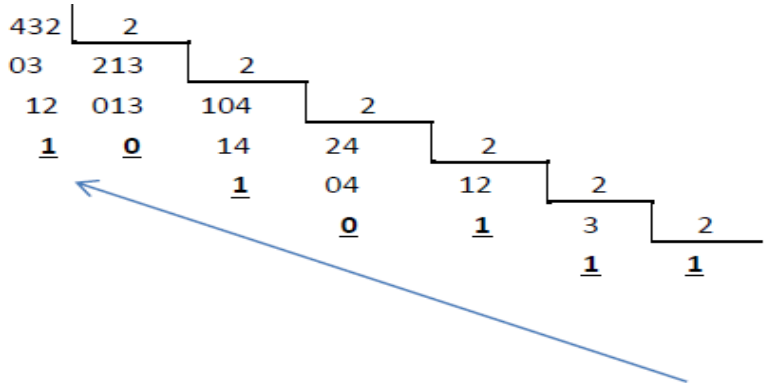
432	2
03	21

1

Bajo el 3, tampoco hay problema. 3 dividido 2 da 1. 1 por 2 es igual a 2, respecto del tres me da resto 1. Cuando bajo el 2, si hay problemas porque me queda 12. 12 en base 5 no es igual a 12 en base 10. 12 en base 5 es igual al 7 en base 10. O sea dividir $12_{(5}$ por 2 es igual a dividir $7_{(10}$ por 2. El resultado es tres y resto 1. O sea nos queda:

432	2
03	213
12	
1	

Este es el primer resto, siguiendo con este cuidado para toda la división, nos queda:



Si queremos verificar que no nos equivocamos, pasamos los dos números a la base 10:

$$432_{(5} = 2 \times 5^0 + 3 \times 5^1 + 4 \times 5^2 = 2 + 15 + 100 = 117_{(10}$$

$$1110101_{(2} = 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 + 1 \times 2^6 =$$
$$= 1 + 0 + 4 + 0 + 16 + 32 + 64 = 117_{(10}$$

Se verifica que está bien hecho el ejercicio, o sea:

$$432_{(5} = 1110101_{(2}$$

17) Ídem anterior pero para el numero $876_{(9}$, pasarlo a base 8.

Solución:

Como hicimos en ejercicio anterior, dividimos sucesivamente por 8 sin olvidarnos que estamos en base 9.

876	8		
076	108	8	
5	18	12	8
	1	**3**	**1**

O sea que $876_{(9} = 1315_{(8}$

Complemento.

Hemos visto Sistemas Numéricos trabajando solo en el campo de los números naturales. Si queremos ahora empezar a trabajar con números enteros, debemos definir como simbolizamos el signo. En el campo binario tenemos solo dos símbolos (0 y 1) así que tendremos que buscar alguna manera de indicar el signo de un número con alguno de estos dos elementos.

Adoptaremos la convención de que el bit más significativo lo usaremos para indicar el signo. Si este bit toma el valor "0" diremos que se corresponde con un decimal positivo, y si es "1" diremos que corresponde a un decimal negativo.

Si encontramos un número binario de la siguiente forma, diremos que corresponde a un decimal positivo.

n	n-1				
0	X	X	X	.....	X

En cambio si el bit en la posición n vale "1" diremos que se corresponde con un decimal negativo.

n	n-1				
1	X	X	X	.....	X

Antes de ver como se interpretan los números positivos y negativos, definiremos que se entiende por **Complemento de un número.** Además el complemento no solo me permite expresar números negativos, sino que me ayuda a realizar operaciones de resta.

Definición:

Se define el complemento de un número A de n dígitos, expresado en una base b, como:

$$C(A) = b^n - A$$

Ejemplo si tengo el número $345_{(10}$ y quiero hallar su complemento. Primero veo que está en base 10 y además consta de 3 dígitos, por lo tanto el complemento se calcula como:

$$C = 10^3 – 345 = 655$$

Complemento en binario:

Antes de definir las distintas convenciones de como complementar un numero binario recordaremos las reglas de la suma y de la resta tal cual lo aprendimos en la Escuela Primaria con la salvedad que lo haremos en binario.
Cuando sumamos tenemos la siguiente tabla:

A	B	Suma	Me llevo
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Cuando resto A - B, la tabla me queda:

A	B	Resta	Pido Prestado
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Para representar números enteros, veremos tres convenciones. Ellas son Valor Absoluto más Bit de Signo (VA+BS), Complemento a la base o complemento a 2 o complemento a la base menos 2 (C_{m-2}) y Complemento a la base menos uno o complemento a 1 (C_{m-1}).

Importante: Los números positivos representan el mismo valor en todas las convenciones, no así los negativos.

Valor Absoluto más Bit de Signo:

En esta convención se divide el número en 2 partes, el bit más significativo (n) se toma como bit de signo, y los (n -1) bits menos significativos me dan el valor absoluto del número representado.

Ejemplo:

Si me dan el número 011, veo que el bit más significativo es 0 por lo tanto representa un numero positivo, los bits restantes 11 me dan 3, por lo tanto el 011 representa el +3.

Ahora si tengo 111, el bit más significativo me indica que es negativo y los bits restantes me dan también tres, por lo tanto este número es el -3.

Complemento al módulo:

Para hallar el complemento al módulo de un número se usa la misma fórmula vista precedentemente, salvo que ahora la usamos en binario. O sea:

$$C_M (A) = 2^n - A$$

Ejemplo:

Si quiero saber el complemento al módulo de 011, vemos que tiene 3 bits, por lo tanto tengo que hacer:

$$C_m (011) = 2^3 – 011 = 1000 – 011.$$

	1	1	1		Pido Prestado
	1	0	0	0	
menos		0	1	1	
		1	**0**	**1**	

Lo mismo pasa si quiero hacer el complemento de 010, en ese caso tengo:

	1	1			Pido Prestado
	1	0	0	0	
menos		0	1	0	
		1	**1**	**0**	

Si nos fijamos qué relación hay entre 011 y su complemento 101, y también qué relación hay entre 010 y su complemento 110, en vez de estar realizando restas podemos adoptar una regla nemotécnica que consiste en:

Cuando quiero hallar el complemento al módulo de un número, recorro el mismo de derecha a izquierda, el primer uno que encuentro queda como está y después cambio unos por ceros y ceros por unos.

Ejemplo:

Quiero hallar el complemento al módulo del número 010101010. Recorro el número de derecha a izquierda hasta encontrar el primer uno, entonces queda:

Numero 010101010
Complemento 101010110

Complemento al módulo menos uno:

El complemento al módulo menos uno de un número se calcula con la siguiente fórmula:

$$C_{M-1}(A) = 2^n - 1 - A$$

Ejemplo:

Usaremos los mismos números que antes, si quiero hacer el complemento al módulo menos uno de 011, debo calcular:

$$C_{M-1}(011) = 2^3 - 1 - 011 = 1000 - 1 - 011 = 111 - 011$$

O sea:

	1	1	1
menos	0	1	1
	1	**0**	**0**

Siguiendo el ejemplo anterior, si quiero hallar el complemento al módulo menos uno de 010.

	1	1	1
menos	0	1	0
	1	**0**	**1**

Al igual que en complemento al módulo, en vez de estar restando, hallaremos una regla nemotécnica comparando en este caso el número y su complemento al módulo menos uno. Este caso es más fácil.

Cuando quiero calcular el complemento al módulo menos uno de un número cambio directamente unos por ceros y ceros por unos.

Ejemplo:

Si quiero hallar el complemento al módulo menos uno del número 010101010, nos queda:

Numero 010101010
Complemento 101010101

Importante: Cuando trabajo con Complemento al módulo o Complemento al módulo menos uno si tengo que interpretar un numero negativo (empieza con 1), todo el numero forma parte del resultado y además ese número es el **Complemento** del resultado. Para clarificar este tema haremos una tabla con números de tres bits y los interpretaremos en todas las convenciones vistas, además de interpretarlos también como si no tuvieran signo.

En la tabla siguiente pondremos en binario los números del 0 al 7 y los interpretaremos en las distintas convenciones.

Numero	Sin signo	VA + BS	C modulo	C módulo -1
000	0	0	0	0
001	1	1	1	1
010	2	2	2	2
011	3	3	3	3
100	4	-0	-4	-3
101	5	-1	-3	-2
110	6	-2	-2	-1
111	7	-3	-1	-0

Fíjense que en todas las convenciones siempre tenemos ocho representaciones, esto sucede porque en binario con 3 bits, tenemos ocho combinaciones posibles. Además se observa que en VA+BS y en Cmódulo-1, tenemos dos representaciones para el cero.

Si observamos el mayor y el menor número representado en cada convención, al abanico de números representables entre el máximo y el mínimo se le llama **rango.** Calcularemos los rangos para las cuatro representaciones que tenemos en el grafico anterior.

Empecemos con los números sin signo, la idea es encontrar el rango en función de la base (en este caso es la binaria).

Mirando la columna sin signo vemos que los números representados van del 0 al +7, poniendo esto en función de la base 2, me queda:

$$0 \longleftrightarrow (2^3 - 1)$$

Si extendemos esto para el caso de n bits, vemos que **sin signo** tenemos un **rango** como lo muestra la expresión siguiente:

$$0 \longleftrightarrow (2^n - 1)$$

En **VA+BS** como en **Complemento al módulo menos uno**, tenemos el mismo rango que va desde – 3 a + 3, siempre considerando 3 bits. O sea:

$$-(2^{(3-1)} - 1) \longleftrightarrow +(2^{(3-1)} - 1)$$

Al igual que hicimos antes extendiendo el concepto a n bits nos queda que el **rango** para estas dos convenciones usando n bits es:

$$-(2^{(n-1)} - 1) \longleftrightarrow +(2^{(n-1)} - 1)$$

De la misma forma en **Complemento al módulo menos dos,** vemos que representamos desde el -4 al +3, o sea:

$$-2^{(3-1)} \longleftrightarrow +(2^{(3-1)} - 1)$$

Extendiendo para n bits:

$$-2^{(n-1)} \longleftrightarrow +(2^{(n-1)} - 1)$$

A modo de ejemplo haremos una tabla con los rangos para distintas cantidades de bits.

	5 bits	**8 bits**	**10 bits**
Sin signo	0 ------- + 31	0 ------- +255	0 ------- + 1023
VA+BS	- 15 ----- + 15	- 127 --- +127	- 511 --- + 511
CM-1	- 15 ----- + 15	- 127 --- +127	- 511 --- + 511
CM-2	- 16 ----- + 15	- 128 --- +127	- 512 --- + 511

En el próximo ejemplo interpretaremos el número binarios dados en todas las convenciones vistas:

Numero	**Sin signo**	**VA + BS**	**CM-1**	**CM-2**
010101	21	+ 21	+ 21	+ 21
110101	53	- 21	- 10	- 11

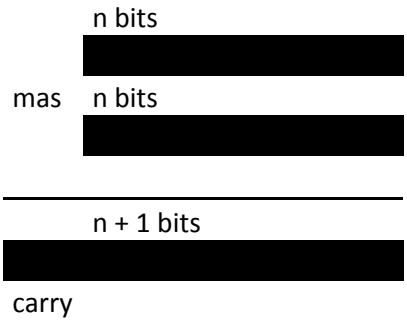
Como se ve en el último ejemplo los positivos representan **el mismo número** en todas las representaciones.

Operaciones con números binarios.

Empezaremos a ver cómo realizar sumas y restas con números binarios signados. Antes de introducirnos en el tema daremos dos definiciones muy importantes a la hora de realizar dichas operaciones.

Carry:

Cuando sumo dos números de **n bits** y el resultado no lo puedo expresar en n bits, sino que preciso **n + 1 bits,** a esto se lo llama **Carry o acarreo.** Es importante tener el concepto que el carry es **independiente** de que los operandos tengan signo o no. O sea que el carry se puede producir sumando números con signo o sin signo.



Overflow:

Este fenómeno ocurre solo cuando trabajo con números **con signo.** Sucede solo cuando sumo dos positivos o dos negativos. Se produce overflow cuando sumo dos positivos y el resultado da un numero negativo o viceversa, sumo dos negativos y el resultado me da positivo. O sea que se cambia el **bit de signo.**

Tanto el **Carry** como el **Overflow** indican que la operación salió del **rango** de los operandos involucrados.

Resta de números binarios:

Veremos cómo operar restas con números binarios solo en las convenciones de Complemento al módulo y Complemento al módulo menos uno. En cualquiera de las dos convenciones se verifica que:

$$A - B = A + C(B)$$

Esto es importantísimo en la operación de una Unidad Aritmético Lógica (ALU) ya que no tengo circuitos independientes para sumar y restar, sino que solo sumo y si necesito complemento. Esto es una gran ventaja al reducir el hardware de la ALU.

Operaciones en Complemento al módulo:

Si quiero operar con dos números binarios A y B de n bits cada uno, independientemente de los signos de cada uno, se cumple que:

$$A - B = A + C_m(B)$$

Aplicando la definición de **Complemento al módulo** nos queda que:

$$A - B = A + C_m(B) = A + 2^n - B$$

Lo que podríamos escribir como:

$$A - B = 2^n + (A - B)$$

Se nos abren tres posibilidades, que A > B, que A = B o que A < B.

A < B :

En este caso me queda 2^n menos algo, si a este algo lo llamo X, queda:

$2^n - X$ que no es otra cosa que el $C_m(X)$.

Y como definimos a X como A – B estoy encontrando el **Complemento al módulo del resultado**.

Ejemplo: Sería el caso de querer restar 3 menos 7, debo hacer 3 menos el Complemento al módulo de 7.

+ 3 = 0011
+ 7 = 0111

Debo hacer 0011 + C_m (0111).

+3 = 0011
C_m (+7) = 1001
1100

Este sería el Complemento al módulo del resultado, al tener el bit más significativo en "1" sé que el resultado representa un número decimal negativo, descomplementando queda:

0100 que corresponde al **– 4.**

A > B:

En caso me queda 2^n mas un numero positivo, ese número positivo es el resultado que estoy buscando (A – B), o sea que me queda el resultado correcto más un 2^n. Justamente este 2^n es un "1" en la posición n, o sea es un **carry.** Como solo me interesa (A – B), la forma de hallarlo es **despreciar este carry.**

Importante: Siempre que opero en Complemento al módulo y aparece un CARRY, lo deshecho.

Ejemplo: Sería el caso de querer hacer la resta entre +7 y +3, debo hacer la suma entre +7 y el Complemento al módulo de + 3

$$+ 7 = 0111$$
$$+ 3 = 0011$$

Debo hacer $0111 + C_m (0011)$

$$+ 7 = 0111$$
$$C_m (+ 3) = 1101$$
$$10100$$

Como vemos apareció un Carry, lo desecho y me queda el resultado correcto **0111 = + 4**

A = B:

En este caso me queda un 2^n seguido de todos ceros, como el caso anterior este Carry que me indica el 2^n lo desecho y me queda el resultado correcto.

Ejemplo: Sería el caso de querer hacer la resta entre + 3 y +3, debo hacer +3 más el complemento al módulo de +3, o sea:

$$+ 3 = 0011$$
$$Cm (+ 3) = 1101$$
$$10000$$

Desechando el Carry me queda el resultado correcto que es **0000= 0**

Operaciones en Complemento al módulo menos uno:

Si quiero operar con dos números binarios A y B de n bits cada uno, independientemente de los signos de cada uno, se cumple que:

$$A - B = A + C_m (B)$$

Aplicando la definición de **Complemento al módulo menos uno** nos queda que:

$$A - B = A + C_{m-1} (B) = A + 2^n - 1 - B$$

Lo que podríamos escribir como:

$$A - B = (2^n - 1) + (A - B)$$

Se nos abren tres posibilidades, que A > B, que A = B o que A < B.

A < B:

En este caso me queda 2^n menos uno menos algo, si a este algo lo llamo X, queda:

$2^n - 1 - X$ que no es otra cosa que el $C_{m-1} (X)$.

Y como definimos a X como A – B estoy encontrando el **Complemento al módulo menos uno del resultado**.

Ejemplo: Sería el caso de querer restar 3 menos 7, debo hacer 3 menos el Complemento al módulo menos uno de 7.

+ 3 = 0011
+ 7 = 0111

Debo hacer 0011 + C_{m-1} (0111).

+3 = 0011
C_{m-1} (+7) = 1000
1011

Este sería el Complemento al módulo menos uno del resultado, al tener el bit más significativo en "1" sé que el resultado representa un número decimal negativo, descomplementando queda:

0100 que corresponde al **– 4.**

A > B:

En caso me queda (2^n – 1) más un numero positivo, ese número positivo es el resultado que estoy buscando (A – B), o sea que me queda el resultado correcto más un (2^n – 1). Justamente este 2^n es un "1" en la posición n, o sea es un **carry.** Como solo me interesa (A – B), debo anular el 2^n – 1. Para ello n tendría que valer 0, como dijimos 2^n representa el carry. Para poder anular entonces el 2^n – 1, tendría que realimentar el **carry** y ponerlo en la posición n = 0. Y realizar la suma final.

Importante: Siempre que opero en Complemento al módulo menos uno y aparece un CARRY, lo realimento y lo sumo.

Ejemplo: Sería el caso de querer hacer la resta entre +7 y +3, debo hacer la suma entre +7 y el Complemento al módulo menos uno de + 3

+ 7 = 0111
+ 3 = 0011

Debo hacer 0111 + C_{m-1} (0011)

+ 7 = 0111
C_{m-1} (+ 3) = 1100
10011

Como vemos apareció un Carry, lo realimento
+ 7 = 0111

C_{m-1} (+ 3) = 1100
10011
1
0100

Como vemos realimento el Carry, lo sumo y me queda el resultado correcto **0111 = + 4**

A = B:

En este caso me queda un 2^n -1, que no es otra cosa que (n – 1) bits en valor 1, como explicamos anteriormente este sería el **complemento al módulo menos uno del resultado.** Al ser todos unos, su complemento al módulo menos uno me daría " – 0".

Ejemplo: Sería el caso de querer hacer la resta entre + 3 y +3, debo hacer +3 más el complemento al módulo menos uno de +3, o sea:

$$\begin{array}{r} +3 = 0011 \\ \text{m-1 } (+3) = 1100 \\ \hline 1111 \end{array}$$

Descomplementando, me queda el resultado correcto que es **0000= - 0**

Banderas o flags:

Las banderas o flags son bits individuales que me indican como fue el resultado de una operación. Recordar que los circuitos digitales hacen operaciones preestablecidas sin saber que convención usamos. Los circuitos directamente suman, restan, multiplican o dividen. La interpretación del resultado la hace el usuario o el programador del sistema digital. Por lo tanto necesitamos de indicadores que señalen que pasó con la cuenta. Estos son las banderas o flags. Dentro de un sistema microprocesado, estos indicadores generalmente forman parte de un registro llamado de estado o de códigos de condición. Los más importantes son:

C (Carry), se pone en uno si el resultado de la operación dio carry.
O (Overflow), se pone en uno si el resultado de la operación dio overflow.
N (Negative), se pone en uno si el resultado de la operación es negativo.
Z (Zero), se pone en uno si el resultado de la operación es cero.
P (Paridad), respecto de este tendremos que ponernos de acuerdo si usamos paridad par o impar de 1. Existen las dos. **Nosotros usaremos paridad par de unos.** Usando esta convención, este bit se pone en "1" si la cantidad de unos del resultado es **impar,** y se pone en "0", si la cantidad de unos del resultado en **par.** Esto es así para que toda la palabra completa (resultado más bit de paridad) **tenga una cantidad para de unos.**

Ejemplo:

						Paridad
0	1	1	1	1	0	0

0	1	1	1	1	1	1

Algunos ejemplos resueltos:

Vamos a ver varios ejemplos, mostrando los flags y además para las distintas combinaciones que se pueden presentar con **Carry y Overflow.** Usaremos números binarios de 6 bits. Lo primero que hacemos es calcular para las convenciones vistas, los rangos en cada una usando 6 bits.

De acuerdo a las formular vistas anteriormente, con 6 bits tenemos los siguientes rangos:

Sin signo: 0--------- + 63
Cm - 32--------- +31
Cm-1 - 31----------+ 31

Caso 1: Realizar la suma de 010011 + 101001, analizaremos los números en las tres convenciones y calcularemos los flags.

	Numero	Sin signo	Cm	Cm-1
	010011	+ 19	+ 19	+ 19
Suma	101001	+ 41	- 23	- 22
	111100	**+ 60**	**- 4**	**- 3**

Carry	0
Overflow	0
Zero	0
Negativo	1
Paridad	0

Como vemos en todas la convenciones el resultado es correcto ya que los bits de **Carry y Overflow** están en cero.

Caso 2: Realizar la suma 011010 + 101101.

	Numero	Sin signo	Cm	Cm-1
	011010	+ 26	+ 26	+ 26
Suma	101101	+ 45	- 19	- 18
	1000111	**+ 7**	**+ 7**	**+ 8**

Carry	1
Overflow	0
Zero	0
Negativo	0
Paridad	1

Como vemos el resultado es **correcto con signo** pero **incorrecto sin signo,** esto me lo indica el **carry en 1 y el overflow en cero.** Es incorrecto sin signo porque la cuenta me tendría que dar 71 y el número máximo que puedo representar en 6 bits sin signo es + 63.

Caso 3: Realizar la suma 011010 + 010101.

	Numero	Sin signo	Cm	Cm-1
	011010	+ 26	+ 26	+ 26
Suma	010101	+ 21	+21	+ 21
	101111	**+ 47**	**- 17**	**- 16**

Carry	0
Overflow	1
Zero	0
Negativo	1
Paridad	1

En este caso el resultado en **correcto sin signo e incorrecto con signo,** esto me lo indica el **carry en 0 y el overflow en 1.** + 47 está dentro del rango sin signo pero no dentro del rango con signo cuyo número máximo, para cualquiera de las dos convenciones es + 31.

Caso 4: Sumar 101111 + 100100

	Numero	Sin signo	Cm	Cm-1
	101111	+ 47	- 17	- 16
Suma	100100	+ 36	- 28	- 27
	1010011	**+ 19**	**+ 19**	**+ 20**

Carry	1
Overflow	1
Zero	0
Negativo	0
Paridad	1

En este caso el resultado es erróneo en todas las convenciones, esto me lo indica tanto el **carry como el overflow en 1.** Sin signo el resultado tendría que dar +83 pero el máximo número que puedo representar sin signo es + 63. Con signo también ya que en Complemento al módulo debería dar – 45, pero no puedo representar más allá del – 32. En Complemento al módulo menos uno pasa lo mismo, quiero expresar el – 43 y no puedo ir más allá del – 31.

Conclusión: Como vimos al principio, tanto el Overflow como el Carry (aunque son dos cosas distintas) me indican que estoy saliendo del rango de números que puedo representar.

EJERCICIOS RESUELTOS.

1) Expresar en decimal el complemento al módulo y el complemento al módulo menos uno de los números 65 y 128.

Solución:

Recordando que para un número A de n dígitos en una base B, el complemento al módulo y al módulo menos uno se calcula como:

$$C_m(A) = B^n - A$$

$$C_{m-1}(A) = B^n - 1 - A$$

En el caso del 65 nos queda:

$$C_m(65) = 10^2 - 65 = 100 - 65 = \mathbf{35}$$

$$C_{m-1}(65) = 10^2 - 1 - 65 = 100 - 1 - 35 = 99 - 65 = \mathbf{34}$$

Para el caso de 128 nos queda:

$$C_m(128) = 10^3 - 128 = 1000 - 128 = \mathbf{872}$$

$$C_{m-1}(128) = 10^3 - 1 - 128 = 1000 - 1 - 128 = 999 - 128 = \mathbf{871}$$

2) Expresar en binario el complemento al módulo y el complemento al módulo menos uno de los siguientes números decimales utilizando 8 bits: 65, 99 y 128.

Solución:

Primero escribimos los tres números en binario utilizando 8 bits:

Decimal	Binario	Comp. a dos	Comp. a uno
65	01000001	**10111111**	**10111110**
99	01100011	**10011101**	**10011100**
128	10000000	**10000000**	**01111111**

3) Un procesador opera con números de 8 bits. En un programa se realizan operaciones con signo. Escribir en decimal los siguientes números:

RA = 11111010 RB = 11111111 RC = 00000000 RD = 10000000
RE = 00000001 RF = 01110101 RH = 10000001 RL = 01111111

Solución:

Para resolver esto haremos una tabla. Recordar que los números que expresan decimales positivos (empiezan con 0), se representan en todos las convenciones con el mismo valor:

Numero	VA+BS	Comp.al Modulo	Comp. al módulo-1
11111010	**-122**	**-6**	**-5**
11111111	**-127**	**-1**	**-0**
00000000	**0**	**0**	**0**
10000000	**-0**	**-128**	**-127**
00000001	**+1**	**+1**	**+1**
01110101	**+117**	**+117**	**+117**
10000001	**-1**	**-127**	**-126**
01111111	**+127**	**+127**	**+127**

4) Realizar, previa conversión al sistema binario las siguientes operaciones. Considerando que los números expresados se representan con signo. Verificar, mediante el análisis de los indicadores, si las operaciones producen un resultado correcto. Los números decimales son:

26 + 19; 26 – 19; 19 – 26 y -19 + 26

Solución:

Lo primero que hacemos es expresar estos número (+19 y +26) en binario, nos queda que $+19_{(10} = 010011_{(2}$ y $+26_{(10} = 011010_{(2}$. Los expresamos con un cero adelante porque el enunciado dice que son números signados.

Primero haremos las operaciones en **complemento al módulo.** Nos conviene calcular el rango de números que podemos expresar, como tenemos 6 bits el rango en esta convención es:

$$-(2^{6-1}) \longleftrightarrow +\ (2^{6-1}-1)$$
$$-32 \longleftrightarrow +31$$

+ 26	010011
+19	011011
+45	101101

En este primer caso la operación no tiene acarreo al bit 7 por lo tanto el bit de carry está en cero, pero si tenemos **overflow**, porque estamos sumando dos positivos y el resultado me da un numero negativo. Esto sucede por el + 45 no lo podemos expresar en **complemento al módulo con 6 bits.** Si nos fijamos el rango no podemos representar más allá del + 31.

+ 26	+ 26	011010
- 19	Cm (19)	101101
+ 7		1000111

En este caso el carry es 1 porque hubo acarreo hacia el séptimo bit. Como en esta convención se desecha el carry, nos queda como resultado + 7 ya que el overflow es cero.

+ 19	+ 19	010011
- 26	Cm (26)	100110
- 7		111001

En este caso no hay ni carry ni overflow, además el resultado da negativo por lo tanto descomplementamos y tenemos el resultado correcto (- 7)

- 19	Cm (19)	101101
+ 26		011010
+ 7		1000111

En este caso el carry es 1 porque hubo acarreo hacia el séptimo bit. Como en esta convención se desecha el carry, nos queda como resultado + 7 ya que el overflow es cero.

Si ahora hacemos lo mismo en **complemento al módulo menos uno.** El rango en esta convención es:

$$-(2^{6-1}-1) \longleftrightarrow +\ (2^{6-1}-1)$$
$$-31 \longleftrightarrow +31$$

+ 26	010011
+19	011011
+45	101101

En este primer caso la operación no tiene acarreo al bit 7 por lo tanto el bit de carry está en cero, pero si tenemos **overflow**, porque estamos sumando dos positivos y el resultado me da un numero negativo. Esto sucede por el + 45 no lo podemos expresar en **complemento al**

módulo menos uno con 6 bits. Si nos fijamos el rango no podemos representar más allá del + 31.

+ 26		011010
- 19	Cm-1 (19)	101100
+ 7		1000110
		1
		000111

En este caso el carry es 1 porque hubo acarreo hacia el séptimo bit. Como en esta convención se realimenta el carry y se suma, nos queda como resultado + 7 ya que el overflow es cero.

- 19	Cm-1 (19)	101100
+ 26		011010
+ 7		1000110
		1
		000111

En este caso el carry es 1 porque hubo acarreo hacia el séptimo bit. Como en esta convención se realimenta el carry y se suma, nos queda como resultado + 7 ya que el overflow es cero.

+ 19		010011
-26	Cm-1 (26)	100101
- 7		111000

En este caso no hay ni carry ni overflow, además el resultado da negativo por lo tanto descomplementamos y tenemos el resultado correcto (- 7)

5) Un procesador trabaja con dos registros RA y RB de 7 bits c/u, en un instante dado los mismos contienen la siguiente información decimal: RA contiene el decimal 63 y RB el decimal 32. Suponiendo que el sistema trabaja con convención de **complemento a la base.** Indicar el resultado de las siguientes operaciones:

RA + RB = RA – RB = RB – RA =

Indicar en todos los casos los valores que toman la bandera en cada caso e interpretar el resultado en decimal.

Solución:

Primero expresamos el contenido de los registros en binario usando 7 bits como indica el enunciado.

$RA = 63_{(10} = 0111111_{(2}$ y $RB = 32_{(10} = 0100000_{(2}$

a) RA + RB

RA	0	1	1	1	1	1	1
RB	0	1	0	0	0	0	0
	1	0	1	1	1	1	1

Carry	0
Overflow	1
Zero	0
Negativo	1
Paridad	0

Como vemos estamos sumando dos positivos y el resultado da negativo. Por ese motivo el flag de Overflow da 1. Viendo la cuenta, si sumamos 63 + 32 debería dar + 95. Como en la convención de Complemento al Módulo mi límite máximo con 7 bits es + 63, no puedo expresar el + 95 con la cantidad de bits que me indica el enunciado. Mirando el resultado de la cuenta está da – 33 (obviamente resultado incorrecto).

b) Ahora hacemos RA – RB, o sea RA + Cm(RB).

RA	0	1	1	1	1	1	1
Cm(RB)	1	1	0	0	0	0	0
	1	0	0	1	1	1	1	1

Carry	1
Overflow	0
Zero	0
Negativo	0
Paridad	1

En este caso hay Carry, pero en esta convención de Complemento al Módulo si existe se desecha, por lo tanto el resultado es correcto ya que si eliminamos este bit de carry el resultado da + 31.

c) Del mismo modo si hacemos RB – RA, nos queda RB + Cm(RA).

RB	0	1	0	0	0	0	0
Cm(RA)	1	0	0	0	0	0	1
	1	1	0	0	0	0	1

Carry	0
Overflow	0
Zero	0
Negativo	1
Paridad	1

En este caso el resultado es correcto por ser ceros tanto el carry como el overflow. Al dar resultado negativo, este es el Complemento al Módulo del resultado buscado. Descomplementando nos queda – 31 que era lo buscado.