TEMA 3: ARITMÉTICA Y TIPOS DE DATOS EN EL COMPUTADOR. PARTE I

Introducción

La información en el ordenador se guarda en celdas que sólo pueden estar en dos estados posibles: encendido/apagado.

Habitualmente estos estados suelen representarse con números: encendido, uno y apagado cero. De alguna manera en cada celda podemos guardar o bien un cero o bien un uno.

Pero normalmente en un ordenador necesitaremos guardar información distinta de ceros y unos.

En esta primera parte del tema veremos la manera de representar 'cualquier' tipo de información en el ordenador.

La información básica que podremos guardar en un ordenador va a ser:

- Información booleana
- Valores enteros (positivos y negativos)
- Caracteres
- Valores reales

Información booleana

La información booleana es básicamente *Verdadero* y *Falso*

Eso equivale a dos valores, que es justamente lo que podemos guardar en el ordenador.

Generalmente el valor verdadero se asocia al 1 y el valor falso al 0 (en lenguaje de programación C, el valor falso es 0, y el valor verdadero cualquier valor distinto de cero.)

Valores enteros (positivos y negativos)

Representación binaria de valores enteros positivos

En general un número cualquiera **N** entero positivo se puede representar por un número indeterminado de símbolos que tendrán un determinado valor por sí mismos y en función de la posición que ocupan.

De esa manera hablaremos de un sistema de numeración en base **B** a la representación de un número mediante un conjunto de B símbolos de la siguiente manera:

$$N = \sum_{i=0}^{n} a_i \cdot B^i$$

y el número **N** se puede representar como

$$a_nB_na_{n-1}B_{n-1}a_{n-2}B_{n-2}...a_2B_2a_1B_1a_0B_0$$

Por ejemplo, la base con que trabajamos habitualmente es base 10 (**B**=10) y los diez símbolos con los que escribimos normalmente los números son los 10 dígitos {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} (al que se suele llamar alfabeto.)

El número 237 en realidad lo que nos esta diciendo es que el valor que tiene es:

$$2*10^2+3*10^1+7*10^0$$

Operación que, evidentemente porque nosotros trabajamos (multiplicamos y sumamos) inconscientemente en base diez o decimal, da como resultado 237.

Dentro del ordenador el alfabeto que podemos utilizar es el 0 y el 1, en total dos símbolos. De manera que la base será la base binaria (**B**=2).

Cualquier número lo representaremos como una sucesión de unos y ceros que cumplen:

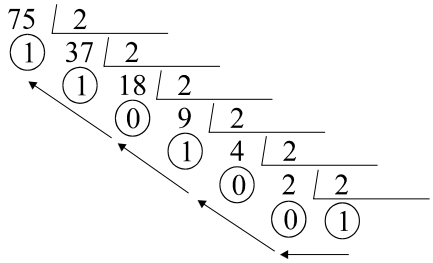
$$N = \sum_{i=0}^{n} a_i \cdot 2^i$$

Cambios de base

Paso de base decimal a base binaria:

El paso de base decimal a binaria se realiza mediante divisiones sucesivas

Ejemplo: Pasar $75)_{10}$ a base binaria



$75)_{10} = 1001011)_2$

Paso de base binaria a base decimal:

Como las operaciones de suma y multiplicación las realizamos habitualmente en base decimal, no hay mas que aplicar la fórmula $N = \sum_{i=0}^{n} a_i \cdot 2^i$

Ejemplo: Pasar $1001011)_2$ a base decimal

$1*2^0+1*2^1+0*2^2+1*2^3+0*2^4+0*2^5+1*2^6=1*1+1*2+0*4+1*8+0*16+0*32+1*64=1+2+8+64=75$

Para realizar el paso de binario a decimal es conveniente recordar las diferentes potencias de 2:

$2^0 = 1$	$2^3 = 8$	$2^6 = 64$	$2^9 = 512$
$2^1 = 2$	$2^4 = 16$	$2^7 = 128$	$2^{10} = 1024$
$2^2 = 4$	$2^5 = 32$	$2^8 = 256$	

Códigos de representación intermedios: Octal y Hexadecimal

Como en base binaria los únicos dígitos que pueden utilizarse son el cero y el uno, normalmente es engorroso y largo escribir un número en base binaria, aunque sea ésta la única base que realmente utiliza el ordenador.

Para escribir números fácilmente convertibles a binario, pero con menor número de cifras se utilizan dos tipos de códigos intermedios: la base octal y la base hexadecimal.

En la representación octal la base es ocho, y el alfabeto los dígitos entre 0 y 7.

B = 8

α = {0, 1, 2, 3, 4, 5, 6, 7}

En la representación hexadecimal la base es dieciséis, y el alfabeto, que constará de dieciséis caracteres es: los dígitos desde el 0 hasta el 9 (diez caracteres) mas las letras (generalmente mayúsculas) desde la A hasta la F (6 caracteres)

B = 16

α = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

Paso de base decimal a base octal/hexadecimal:

La manera de pasar de base decimal a octal o hexadecimal es similar al paso a base binaria, pero dividiendo en cada caso sucesivamente por ocho o dieciséis.

Ejemplo: Pasar $75)_{10}$ a base octal y a base hexadecimal

$75)_{10} = 113)_8$ $75)_{10} = 4B)_{16}$

Paso de base octal/hexadecimal a base decimal:

Al igual que en el caso binario nos limitaremos a utilizar la fórmula.

$N = \sum_{i=0}^{n} a_i \cdot 8^i$ en el caso octal y $N = \sum_{i=0}^{n} a_i \cdot 16^i$ en el caso hexadecimal

Ejemplo: Pasar $113)_8$ a base decimal

$3*8^0+1*8^1+1*8^2=3*1+1*8+1*64=3+8+64=75$

Ejemplo: Pasar $4B)_{16}$ a base decimal

$B*16^0+4*16^1=11*1+4*16=11+64=75$

Paso de base binaria a base octal/hexadecimal:

La base octal está basada en el ocho, mientras que la binaria esta basada en el dos. Ocho es dos elevado a tres. De manera que tres cifras binarias hacen una cifra octal. El paso de binario a octal se reduce a agrupar de tres en tres de derecha a izquierda las cifras binarias y evaluar su valor decimal.

Para la base hexadecimal tenemos una cosa similar pero si sabemos que dieciséis es dos elevado a cuatro, vemos que tenemos que agrupar las cifras en grupos de cuatro y evaluar, siempre recordando que valores superiores a 9 son representados mediante letras (A=10, B=11, C=12, D=13, E=14 y F=15).

Ejemplo: Pasar $1001011)_2$ a base octal y a base hexadecimal

$1001011)_2 = 001\ 001\ 011)_2 \rightarrow 001 = 1 \,/\, 001 = 1 \,/\, 011 = 3 \rightarrow 113)_8$

$1001011)_2 = 0100\ 1011)_2 \rightarrow 0100 = 4 \,/\, 1011 = 11 = B \rightarrow 4B)_{16}$

Paso de octal/hexadecimal a binario

Una vez visto el paso de binario a octal y hexadecimal, la transformación inversa es obvia: En el caso octal sólo hay que pasar cada uno de los dígitos octales a tres cifras binarias y en el caso hexadecimal a cuatro cifras binarias. Sólo habría que recodar las transformaciones básicas en cada caso.

Octal	$000)_2 = 0)_8$	$011)_2 = 3)_8$	$110)_2 = 6)_8$
	$001)_2 = 1)_8$	$100)_2 = 4)_8$	$111)_2 = 7)_8$
	$010)_2 = 2)_8$	$101)_2 = 5)_8$	

Hexadecimal	$0000)_2 = 0)_{16}$	$0110)_2 = 6)_{16}$	$1100)_2 = C)_{16}$
	$0001)_2 = 1)_{16}$	$0111)_2 = 7)_{16}$	$1101)_2 = D)_{16}$
	$0010)_2 = 2)_{16}$	$1000)_2 = 8)_{16}$	$1110)_2 = E)_{16}$
	$0011)_2 = 3)_{16}$	$1001)_2 = 9)_{16}$	$1111)_2 = F)_{16}$
	$0100)_2 = 4)_{16}$	$1010)_2 = A)_{16}$	
	$0101)_2 = 5)_{16}$	$1011)_2 = B)_{16}$	

Representación de números enteros con signo (números enteros positivos y negativos)

Existen diferentes estrategias para realizar la representación del signo en los números.

Representación en signo y magnitud

La más sencilla y obvia es realizar algo similar a la manera habitual de representar los números con signo: Dedicar un espacio para indicar el signo del número. Así si hablamos de número '75' positivo, podemos indicarlo como '+75'. Y si hablamos del número '75' negativo lo indicaremos con el símbolo '-' precediendo al '75' es decir: '-75'. De alguna manera distinguimos la magnitud o valor del número (el 75) de su signo (+/-).

En la memoria del ordenador es imposible la representación del símbolo + o del símbolo -. Sólo se pueden representar unos y ceros. De manera que lo que haremos será dedicar un espacio concreto de la representación del número al signo, que también será un cero o un uno, pero que por estar en una cierta posición no indicará magnitud sino signo.

Si hablamos de un lugar determinado donde va a estar el signo, también tendremos que hablar de un número determinado de espacios para poder localizar el signo y la magnitud. Así, de ahora en adelante, cualquier representación binaria de un número llevará pareja el número de bits en que se va a realizar la representación (y a su vez el método en que ha sido representado).

Ejemplo: Representar los números $75)_{10}$ y $-75)_{10}$ en base binaria, en signo magnitud en 8 bits (n=8)

$75)_{10} = \underset{S}{0}\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ )_{2SM}$

$-75)_{10} = \underset{S}{1}\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ )_{2SM}$

Si al realizar la representación apareciesen huecos (casillas sin rellenar) éstas se dejarán siempre a la derecha del signo y a la izquierda de la magnitud, y se rellenarán siempre con ceros.

Ejemplo: Representar los números $18)_{10}$ y $-18)_{10}$ en base binaria, en signo magnitud en 8 bits (n=8)

$18)_{10} = 10010)_2$

$18)_{10} = \underset{S}{0}\ 0\ 0\ 1\ 0\ 0\ 1\ 0)_{2SM}$

$-18)_{10} = \underset{S}{1}\ 0\ 0\ 1\ 0\ 0\ 1\ 0)_{2SM}$

El inconveniente de esta representación es básicamente la realización de operaciones (sumar/restar), en las que hay que comprobar el signo y dependiendo del signo realizar una operación u otra, comprobando cual de las magnitudes es mayor, etc.

Representación en complemento a 1

Dado x entero positivo o negativo, lo que haremos será aplicarle una transformación:

$$x` = (2^n - 1) + x$$

y en las operaciones se tiene en cuenta el acarreo.

Ejemplo: Supongamos n=4. Representar todos los números posibles.

$2^4 = 10000 \rightarrow 2^4 - 1 = 1111$

$0 \rightarrow 1111 + 0 = 1111)_{2C1}$

$1 \rightarrow 1111 + 1 = (1)0000 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0000 + 1 = 0001)_{2C1}$

$2 \rightarrow 1111 + 10 = (1)0001 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0001 + 1 = 0010)_{2C1}$

$3 \rightarrow 1111 + 11 = (1)0010 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0010 + 1 = 0011)_{2C1}$

$4 \rightarrow 1111 + 100 = (1)0011 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0011 + 1 = 0100)_{2C1}$

$5 \rightarrow 1111 + 101 = (1)0100 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0100 + 1 = 0101)_{2C1}$

$6 \rightarrow 1111 + 110 = (1)0101 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0101 + 1 = 0110)_{2C1}$

$7 \rightarrow 1111 + 111 = (1)0110 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0110 + 1 = 0111)_{2C1}$

$8 \rightarrow 1111 + 1000 = (1)0111 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0111 + 1 = 1000)_{2C1}$
(número no válido porque es igual que el –7)

$-1 \rightarrow 1111 - 1 = 1110)_{2C1}$

$-2 \rightarrow 1111 - 10 = 1101)_{2C1}$

$-3 \rightarrow 1111 - 11 = 1100)_{2C1}$

$-4 \rightarrow 1111 - 100 = 1011)_{2C1}$

$-5 \rightarrow 1111 - 101 = 1010)_{2C1}$

$-6 \rightarrow 1111 - 110 = 1001)_{2C1}$

$-7 \rightarrow 1111 - 111 = 1000)_{2C1}$

$-8 \rightarrow 1111 - 1000 = 0111)_{2C1}$ (número no válido porque es igual que el 7)

Viendo los resultados podemos ver que se cumplen algunas características.

- Los números positivos tienen como primer bit un cero, mientras que los negativos tienen un uno.
- Los números positivos tienen la misma representación que tenían pero completados con ceros a la izquierda hasta el número adecuado de bits de la representación.

- Los números negativos podríamos obtenerlos completando con ceros los números binarios positivos y cambiando ceros por unos y unos por ceros.

De esta forma podemos obtener la representación binaria en complemento a uno bien utilizando la fórmula (**`x'= (2`^{n}**`-1) + x`**) o bien siguiendo los siguientes pasos:

- Si el número a representar es positivo entonces se pasa a binario y se completa con ceros hasta la representación elegida.
- Si el número es negativo entonces se pasa su magnitud a binario, se completa con ceros hasta la representación elegida y finalmente se cambian ceros por unos y unos por ceros.

Con esta representación la suma y la resta se convierten en una sola operación: suma.

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

<u>Ejemplo 1:</u> Supongamos n=4. Sumar A=5 y B=2.

$A = 5)_{10} = 0\ 1\ 0\ 1)_{2C1}$ $\quad B = 2)_{10} = 0\ 0\ 1\ 0)_{2C1}$

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ +\ 0\ 0\ 1\ 0 \\ \hline 0\ 1\ 1\ 1 \end{array}$$

$A + B = 0\ 1\ 1\ 1)_{2C1}$

<u>Ejemplo 2:</u> Supongamos n=4. Sumar A=5 y B=-6.

$A = 5)_{10} = 0\ 1\ 0\ 1)_{2C1}$ $\quad B = -\ 6)_{10} = -\ 0\ 1\ 1\ 0)_2 = 1\ 0\ 0\ 1)_{2C1}$

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ +\ 1\ 0\ 0\ 1 \\ \hline 1\ 1\ 1\ 0 \end{array}$$

$A + (-B) = 1\ 1\ 1\ 0)_{2C1}$

<u>Ejemplo 3:</u> Supongamos n=4. Sumar A=5 y B=6.

$A = 5)_{10} = 0\ 1\ 0\ 1)_{2C1}$ $\quad B = 6)_{10} = 0\ 1\ 1\ 0)_{2C1}$

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ +\ 0\ 1\ 1\ 0 \\ \hline 1\ 0\ 1\ 1 \end{array}$$

$A + B = 1\ 0\ 1\ 1)_{2C1}$ (*Hemos obtenido un número negativo de la suma de dos números positivo. Luego existe un error (error de overflow)*)

<u>Ejemplo 4:</u> Supongamos n=4. Sumar A=-5 y B=-6.

$A = -\ 5)_{10} = -\ 1\ 0\ 1)_2 = 1\ 0\ 1\ 0)_{2C1}$ $\quad B = -\ 6)_{10} = -\ 1\ 1\ 0)_2 = 1\ 0\ 0\ 1)_{2C1}$

$$\begin{array}{r} 1\ 0\ 1\ 0 \\ +\ 1\ 0\ 0\ 1 \\ \hline (1)\ 0\ 0\ 1\ 1 \end{array}$$

$\rightarrow$ El acarreo se vuelve a sumar $\rightarrow 0\ 0\ 1\ 1 + 1 = 0\ 1\ 0\ 0$

$A + B = 0\ 1\ 0\ 0)_{2C1}$ (*Hemos obtenido un número positivo de la suma de dos números negativos. Luego existe un error (error de underflow)*)

El único pequeño inconveniente de esta representación es que el cero tiene dos representaciones de manera que tenemos representados 'sólo' 2^n-1 diferentes números, cuando podríamos tener representados 2^n.

Representación en complemento a 2

Dado x entero positivo o negativo, lo que haremos será aplicarle la misma transformación que antes pero añadiendo siempre uno, y no teniendo en cuenta luego el acarreo.

$$x` = (2^n - 1) + x + 1$$

Ejemplo: Supongamos n=4. Representar todos los números posibles.

$2^4 = 10000 \rightarrow 2^4 - 1 = 1111$

$0 \rightarrow 1111 + 0 + 1 = (1)\ 0\ 0\ 0\ 0 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow$ $0\ 0\ 0\ 0)_{2C2}$

$1 \rightarrow 1111 + 1 + 1 = (1)\ 0\ 0\ 0\ 1 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow$ $0\ 0\ 0\ 1)_{2C2}$

$2 \rightarrow 1111 + 10 + 1 = (1)\ 0\ 0\ 1\ 0 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow$ $0\ 0\ 1\ 0)_{2C2}$

$3 \rightarrow 1111 + 11 + 1 = (1)\ 0\ 0\ 1\ 1 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow$ $0\ 0\ 1\ 1)_{2C2}$

$4 \rightarrow 1111 + 100 + 1 = (1)\ 0\ 1\ 0\ 0 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow$ $0\ 1\ 0\ 0)_{2C2}$

$5 \rightarrow 1111 + 101 + 1 = (1)\ 0\ 1\ 0\ 0 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow$ $0\ 1\ 0\ 1)_{2C2}$

$6 \rightarrow 1111 + 110 + 1 = (1)\ 0\ 1\ 1\ 0 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow$ $0\ 1\ 1\ 0)_{2C2}$

$7 \rightarrow 1111 + 111 + 1 = (1)\ 0\ 1\ 1\ 1 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow$ $0\ 1\ 1\ 1)_{2C2}$

$8 \rightarrow 1111 + 1000 + 1 = (1)\ 1\ 0\ 0\ 0 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow$ $1\ 0\ 0\ 0)_{2C2}$ (número no válido porque es igual que el –8)

$-1 \rightarrow 1111 - 1 + 1 = 1\ 1\ 1\ 1)_{2C2}$

$-2 \rightarrow 1111 - 10 + 1 = 1\ 1\ 1\ 0)_{2C2}$

$-3 \rightarrow 1111 - 11 + 1 = 1\ 1\ 0\ 1)_{2C2}$

$-4 \rightarrow 1111 - 100 + 1 = 1\ 1\ 0\ 0)_{2C2}$

$-5 \rightarrow 1111 - 101 + 1 = 1\ 0\ 1\ 1)_{2C2}$

$-6 \rightarrow 1111 - 110 + 1 = 1\ 0\ 1\ 0)_{2C2}$

$-7 \rightarrow 1111 - 111 + 1 = 1\ 0\ 0\ 1)_{2C2}$

$-8 \rightarrow 1111 - 1000 + 1 = 1\ 0\ 0\ 0)_{2C2}$

$-9 \rightarrow 1111 - 1001 + 1 = 0\ 1\ 1\ 1)_{2C2}$ (número no válido porque es igual que el 7)

Al igual que antes, viendo los resultados, podemos ver que se cumplen algunas características.

- Los números positivos tienen como primer bit un cero, mientras que los negativos tienen un uno (al igual que la representación en complemento a 1.)
- Los números positivos tienen la misma representación que tenían pero completados con ceros a la izquierda hasta el número adecuado de bits de la representación.
- Los números negativos podríamos obtenerlos completando con ceros los números binarios positivos y cambiando ceros por unos y unos por ceros y sumando uno al número resultante.

De esta forma podemos obtener la representación binaria en complemento a dos bien utilizando la fórmula (**`x'= (2^n-1) + x + 1`**) o bien siguiendo los siguientes pasos:

- Si el número a representar es positivo entonces se pasa a binario y se completa con ceros hasta la representación elegida.
- Si el número es negativo entonces se pasa su magnitud a binario, se completa con ceros hasta la representación elegida y finalmente se cambian ceros por unos y unos por ceros, sumándole 1 al resultado.

Igual que antes, con esta representación la suma y la resta se convierten en una sola operación: suma.

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

Ejemplo 1: Supongamos n=4. Sumar A=5 y B=2.

$A = 5)_{10} = 0\ 1\ 0\ 1)_{2C2}$ $B = 2)_{10} = 0\ 0\ 1\ 0)_{2C2}$

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ +\ 0\ 0\ 1\ 0 \\ \hline 0\ 1\ 1\ 1 \end{array}$$

$A + B = 0\ 1\ 1\ 1)_{2C2}$

Ejemplo 2: Supongamos n=4. Sumar A=5 y B=-6.

$A = 5)_{10} = 0\ 1\ 0\ 1)_{2C2}$ $B = -6)_{10} = -\ 0\ 1\ 1\ 0)_2 = 1\ 0\ 1\ 0)_{2C2}$

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ +\ 1\ 0\ 1\ 0 \\ \hline 1\ 1\ 1\ 1 \end{array}$$

$A + (-B) = 1\ 1\ 1\ 1)_{2C2}$

Ejemplo 3: Supongamos n=4. Sumar A=5 y B=6.

$A = 5)_{10} = 0\ 1\ 0\ 1)_{2C2}$ $B = 6)_{10} = 0\ 1\ 1\ 0)_{2C2}$

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ +\ 0\ 1\ 1\ 0 \\ \hline 1\ 0\ 1\ 1 \end{array}$$

$A + B = 1\ 0\ 1\ 1)_{2C2}$ (*Hemos obtenido un número negativo de la suma de dos números positivo. Luego existe un error (error de overflow)*)

Ejemplo 4: Supongamos n=4. Sumar A=-5 y B=-6.

$A = -5)_{10} = -\ 1\ 0\ 1)_2 = 1\ 0\ 1\ 1)_{2C2}$ $B = -6)_{10} = -\ 1\ 1\ 0)_2 = 1\ 0\ 1\ 0)_{2C2}$

$$\begin{array}{r} 1\ 0\ 1\ 1 \\ +\ 1\ 0\ 1\ 0 \\ \hline (1)\ 0\ 1\ 0\ 1 \end{array}$$

$\rightarrow$ El acarreo se ignora en esta representación

$A + B = 0\ 1\ 0\ 1)_{2C2}$ (*Hemos obtenido un número positivo de la suma de dos números negativos. Luego existe un error (error de underflow)*)

Caracteres

La representación de caracteres se hará mediante una cierta correspondencia entre los caracteres que queremos representar y una serie de números binarios.

La correspondencia más conocida y usada es el código ASCII (*American Standard Code for Information Interchange*) originalmente utilizado para comunicar información entre distintas máquinas, por lo que además de los caracteres normales utilizados en la escritura, puede representar una serie de caracteres con un cierto significado especial en los ordenadores, conocidos como caracteres de control.

El código ASCII original utilizaba siete bits para la representación de la información relativa a caracteres, y utilizaba un octavo bit (bit de control de errores) para comprobar la corrección de los otros siete, por ejemplo controlando la paridad de los bits (número par o impar de unos del número binario, por ejemplo el número binario 0110101, tiene número par de unos, por lo que el bit de control sería 0, mientras que 0100101 tendría como bit de control un 1.) Con los siete bits pueden llegar a representarse hasta un máximo de 2^7 caracteres (es decir, 128.)

Actualmente, y por la alta difusión del código ASCII para la representación de caracteres, se han introducido un número mayor de caracteres para poder representar caracteres internacionales (como vocales acentuadas, la 'ñ', la 'ç', etc.) Esta extensión del código ASCII se hace a costa del octavo bit y

es específico de cada país. Una representación de uno de los códigos ASCII extendidos españoles podría ser el que se muestra en la tabla 1.

		Código ASCII estándar								Código ASCII extendido							
	Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Hex	Bin	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	NUL	DLE	SP	0	@	P	`	p	Ç	É	á	░	└	╨	α	≡
		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
1	0001	SOH	DC1	!	1	A	Q	a	q	ü	æ	í	▒	┴	╤	β	±
		1	17	33	49	65	81	97	11	129	145	161	177	193	209	225	241
2	0010	STX	DC2	"	2	B	R	b	r	é	Æ	ó	▓	┬	╥	Γ	≥
		2	18	34	50	66	82	98	114	130	146	162	178	194	210	226	242
3	0011	ETX	DC3	#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
		3	19	35	51	67	83	99	115	131	147	163	179	195	211	227	243
4	0100	EOT	DC4	$	4	D	T	d	t	ä	ö	ñ	┤	─	╘	Σ	⌠
		4	20	36	52	68	84	100	116	132	148	164	180	196	212	228	244
5	0101	ENQ	NAK	%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	σ	⌡
		5	21	37	53	69	85	101	117	133	149	165	181	197	213	229	245
6	0110	ACK	SYN	&	6	F	V	f	v	å	û	ª	╢	╞	╓	µ	÷
		6	22	38	54	70	86	102	118	134	150	166	182	198	214	230	246
7	0111	BEL	ETB	'	7	G	W	g	w	ç	ù	º	╖	╟	╫	τ	≈
		7	23	39	55	71	87	103	119	135	151	167	183	199	215	231	247
8	1000	BS	CAN	(	8	H	X	h	x	ê	ÿ	¿	╕	╚	╪	Φ	°
		8	24	40	56	72	88	104	120	136	152	168	184	200	216	232	248
9	1001	HT	EM	)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	┘	θ	•
		9	25	41	57	73	89	105	121	137	153	169	185	201	217	233	249
A	1010	LF	SUB	*	:	J	Z	j		è	Ü	¬	║	╩	┌	Ω	·
		10	26	42	58	74	90	106	122	138	154	170	186	202	218	234	250
B	1011	VT	ESC	+	;	K	[	k	{	ï	¢	½	╗	╦	█	δ	√
		11	27	43	59	75	91	107	123	139	155	171	187	203	219	235	251
C	1100	FF	FS	,	<	L	\	l	|	î	£	¼	╝	╠	■	∞	ⁿ
		12	28	44	60	76	92	108	124	140	156	172	188	204	220	236	252
D	1101	CR	GS	-	=	M	]	m	}	ì	¥	¡	╜	═	▌	∅	²
		13	29	45	61	77	93	109	125	141	157	173	189	205	221	237	253
E	1110	SO	RS	.	>	N	^	n	~	Ä	₧	«	╛	╬	▐	∈	■
		14	30	46	62	78	94	110	126	142	158	174	190	206	222	238	254
F	1111	SI	US	/	?	O	_	o	DEL	Å	ƒ	»	┐	╧	■	∩	
		15	31	47	63	79	95	111	127	143	159	175	191	207	223	239	255

Caracteres de Control — Caracteres Gráficos

Caracteres de Control:

- **NUL** ⇨ Carácter Nulo
- **SOH** ⇨ Comienzo de cabecera
- **STX** ⇨ Comienzo de texto
- **ETX** ⇨ Final de texto
- **EOT** ⇨ Fin de transmisión
- **ENQ** ⇨ Pregunta
- **ACK** ⇨ Confirmación positiva
- **BEL** ⇨ Señal acústica
- **BS** ⇨ Espacio atrás
- **HT** ⇨ Tabulador horizontal
- **LF** ⇨ Salto de línea
- **VT** ⇨ Tabulador vertical
- **FF** ⇨ Salto de página
- **CR** ⇨ Retorno de carro
- **SO** ⇨ Shift Out
- **SI** ⇨ Shift In
- **DLE** ⇨ Carácter de transparencia en tramas de datos
- **DC1** ⇨ Control dispositivo 1
- **DC2** ⇨ Control dispositivo 2
- **DC3** ⇨ Control dispositivo 3
- **DC4** ⇨ Control dispositivo 4
- **NAK** ⇨ Confirmación negativa
- **SYN** ⇨ Sincronismo
- **ETB** ⇨ Fin del bloque de texto
- **CAN** ⇨ Cancelar
- **EM** ⇨ Fin de dispositivo
- **SUB** ⇨ Sustitución
- **ESC** ⇨ Escape
- **FS** ⇨ Separador de ficheros
- **GS** ⇨ Separador de grupos
- **RS** ⇨ Separador de registros
- **US** ⇨ Separador de unidades

Otros caracteres:

- **SP** ⇨ Espacio en blanco
- **DEL** ⇨ Carácter de borrado

Valores Reales

Es importante tener en cuenta que para la representación de números reales tendremos una cantidad finita de elementos, de manera que tendremos que trabajar con una cierta precisión (no se pueden representar un número infinito de decimales.)

Un número real, en general, se puede representar de dos formas concretas: Mediante coma fija; y mediante coma flotante (o notación científica.)

Coma fija

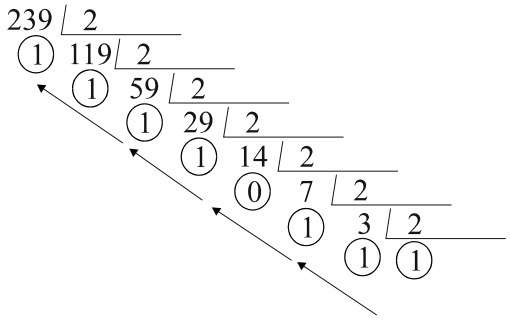
Para pasar de decimal a binario un número real, lo que hay que hacer es pasar por un lado la parte entera y por otro la parte decimal.

La parte entera se pasa como hemos visto que se transformaban los números enteros: Mediante divisiones sucesivas.

La parte decimal se pasa por multiplicaciones sucesivas, viendo la parte entera resultante de cada multiplicación.

Ejemplo: Pasar el número decimal 239'403 a binario.

Primero pasaremos la parte entera:



$$239)_{10} = 11101111)_2$$

La parte decimal mediante multiplicaciones sucesivas de las partes decimales:

0'403 * 2 = 0'806 (Obtenemos un 0)
0'806 * 2 = 1'612 (Obtenemos un 1)
0,612 * 2 = 1'224 (Obtenemos un 1)
0'224 * 2 = 0'448 (Obtenemos un 0)
0'448 * 2 = 0'896 (Obtenemos un 0)
0,896 * 2 = 1'792 (Obtenemos un 1)
0'792 * 2 = 1'584 (Obtenemos un 1)
...

$$0'403)_{10} = 0'0110011...)_2$$

El resultado será pues:

$$239'403)_{10} = 11101111'0110011...)_2$$

Importante: Un número decimal con número finito de decimales, no tiene porque tener un número finito de decimales en su representación binaria.

La respuesta a la pregunta: ¿Cuántos decimales hay que obtener? Depende de la cantidad de bits que tengamos para realizar la representación (y que comentaremos en la parte de coma flotante.)

El número en coma fija, es difícil representarlo en el ordenador, básicamente porque no sabemos la posición en que va a quedar la coma, y como esta rodeada de unos y de ceros, no podríamos distinguirla de ellos (ya que sólo podemos representar unos y ceros en la memoria del ordenador.)

Coma flotante

La representación en coma flotante se basa en colocar la coma en una posición determinada, de manera que siempre estará en esa posición y en la representación binaria no tendremos que preocuparnos por ella. El hecho de mover la coma, supondrá la aparición de una base (en nuestro caso 2) elevada a un exponente, que también tendremos que representar.

En general cualquier número real en cualquier base puede ponerse en coma flotante moviendo la coma a la izquierda de la primera cifra significativa del número y multiplicando el número por la base elevada al exponente adecuado.

Por ejemplo:

$239'403)_{10} = 0'239403 * 10^3$

$0'00000345)_{10} = 0'345 * 10^5$

$16'76)_8 = 0'1676 * 8^2$

$11101111'011001)_2 = 0'11101111011001 * 2^8$

Una vez tenemos el número binario en coma flotante la representación se realizará separando por un lado la representación del número que contiene la coma (o mantisa) y del exponente que eleva la base.

La mantisa se escribe representando los valores que aparecen a la derecha de la coma (ya que el 0 y la coma siempre están en ese lugar.)

Al igual que en la representación de enteros, aquí también nos tendrán que informar del número de bits que debemos utilizar para la representación.

La representación binaria de un número real en coma flotante siempre (o casi siempre) estará representada por:

signo/mantisa/exponente

El signo con la mantisa son similares a un entero, de manera que la mantisa con el signo puede representarse también como hemos representado los números enteros (complemento a dos) pero la forma habitual es mediante signo y magnitud.

El exponente es un número entero, de manera que cualquier método de representación de enteros serviría para representar el exponente. Pero es habitual que en la representación de números reales en coma flotante el exponente se represente con sesgo (es decir con un desplazamiento o exceso.)

La representación de binarios con *n* bits representa de forma natural números entre 0 y 2^n (todos positivos.) Con el sesgo o desplazamiento lo que se hace es tener una aplicación que a los números negativos les haga corresponder números positivos. La transformación que hay que aplicar es:

$$X' = X + 2^{n-1}$$

Con esta transformación podemos representar números en el rango $-(2^{n-1}-1), ...-1, 0, 1, ... 2^n$

Ejemplo: Representa en coma flotante, con ocho bits para la mantisa (en signo magnitud) y 5 bits para el exponente (sesgado) el número $239'403)_{10}$

$239'403)_{10} = 11101111'0110011)_2 = 0'11101111011011 * 2^8$

Mantisa: Signo: Positivo (0)

Magnitud: 11101111011011

Como el signo con la magnitud ocupan más de 8 bits, tendremos truncamiento (perdida de información menos significativa.) Es decir, sólo podremos representar realmente el `0'1110111`, en vez de `0'11101111011011`

Exponente: Valor: $8)_{10} = 1000)_2$

Sesgo: $2^{n-1})_{10} = 10000)_2$

Exponente con sesgo será: 1000 + 10000 = 11000

Resumiendo:

$$\underset{s}{0}\ \ 1\ \ 1\ \ 1\ \ 0\ \ 1\ \ 1\ \ 1\ \ /\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0$$

Operaciones en coma flotante: Multiplicación

Supongamos dos números A y B representados en coma flotante. En esa representación tendremos que:

$A = mA * 2^{eA}$ $B = mB * 2^{eB}$

Si lo que queremos hacer es multiplicar A y B, lo que haremos será:

$$A * B = (mA * 2^{eA}) * (mB * 2^{eB}) = (mA * mB) * 2^{eA+eB}$$

Ejemplo: Multiplicar los números 1'19 y 0'36 en coma flotante (8 bits para la mantisa en signo/magnitud y 6 para el exponente sesgado)

$A = 1'19)_{10} = 1'0011)_2 = 0'1001100... * 2^1$

$$A = \underset{s}{0}\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ /\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1$$

$B = 0'36)_{10} = 0'01011100001)_2 = 0'1011100... * 2^{-1}$

$$B = \underset{s}{0}\ \ 1\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ /\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1$$

```
   0'1 0 0 1 1
 x 0'1 0 1 1 1
     1 0 0 1 1
   1 0 0 1 1
 1 0 0 1 1
1 0 0 1 1 0
0'0 1 1 0 1 1 0 1 0 1
```

$A * B = 0'0110110101 * 2^{-1+1} = (0'10110101 * 2^{-1}) * 2^0 = 0'10110101 * 2^{-1}$

$$A * B = \underset{s}{0}\ \ 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0\ \ /\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1$$

Operaciones en coma flotante: Suma

La suma es un poco más complicada. La manera de sumar valores que están multiplicados por un número es buscar que el número que multiplica sea el mismo para poder sacarlo factor común y luego sumar lo que nos queda:

$A = mA * 2^{eA}$ $B = mB * 2^{eB}$

Si **`eA > eB`** entonces significa que **`eB = eA - eaux`**

$$A + B = (mA * 2^{eA}) + (mB * 2^{eB}) = (mA * 2^{eA}) + (mB * 2^{eA-eaux}) =$$
$$= (mA + mB * 2^{-eaux}) * 2^{eA}$$

Ejemplo: Sumar los números 1'19 y 0'36 en coma flotante (8 bits para la mantisa en signo/magnitud y 6 para el exponente sesgado)

$A = 1'19)_{10} = 1'0011)_2 = 0'1001100... * 2^1$

$B = 0'36)_{10} = 0'01011100001)_2 = 0'1011100... * 2^{-1} = (0{,}1011100 * 2^{-1} * 2^{+1}) * 2^{-1} = 0{,}001011100 * 2^{+1}$

```
  0'1 0 0 1 1 0 0
+ 0'0 0 1 0 1 1 1
  0'1 1 0 0 0 1 1
```

$A + B = 0'1100011 * 2^1$

$$A + B = \underset{s}{0}\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ /\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1$$