Guía de trabajos prácticos de TD2

M. Estefanía Pereyra

15 de junio de 2019

Trabajo Práctico: bases numéricas

Conversión de Binario a Decimal/Decimal a Binario

Los números binarios se representan mediante la unión de bits que pueden tomar el valor de 0 0 1. Cada columna de un número binario tiene dos veces el peso de la columna previa, por lo tanto los números binarios tienen *base* 2. Los pesos de las columnas de un número binario (de derecha a izquierda) son: $1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, \cdots$. Un número binario de $N - bits$ representa una de las 2^N posibilidades: $0, 1, 2, 3, \cdots, 2^N - 1$.

- Binario a decimal: para convertir un número binario a decimal se debe realizar la sumatoria del producto de cada bit por el peso de la columna correspondiente, esto es:

$$10110_2 = 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0 = 22_{10}$$

- Decimal a Binario: la conversión de un número decimal a binario se puede realizar de dos formas, comenzando desde la columna izquierda del número binario resultante o desde la derecha. Comenzando desde el bit mas significativo, se comienza con la mayor potencia de 2 menor o igual al numero decimal, esto es:

$$84_{10} \rightarrow binario$$

la mayor potencia de 2 menor a 84 es 64,

$$
\begin{aligned}
84 \geq 64 &\rightarrow 1 \text{ (en la columna } 2^6) \\
84 - 64 = 20 & \\
20 < 32 &\rightarrow 0 \text{ (en la columna } 2^5) \\
20 \geq 16 &\rightarrow 1 \text{ (en la columna } 2^4) \\
20 - 16 = 4 & \\
4 < 8 &\rightarrow 0 \text{ (en la columna } 2^3) \\
4 \geq 4 &\rightarrow 1 \text{ (en la columna } 2^2) \\
4 - 4 = 0 & \\
0 < 2 &\rightarrow 0 \text{ (en la columna } 2^1) \\
0 < 1 &\rightarrow 0 \text{ (en la columna } 2^0) \\
84_{10} = 1010100_2 &
\end{aligned}
$$

Comenzando desde el bit menos significativo del número binario, se divide repetidamente el número decimal por 2. El resto va en cada columna. Ejemplo:

$$
\begin{aligned}
84_{10} \rightarrow \textit{binario} & \\
84/2 = 42 &\rightarrow \text{resto} = 0 \text{ (en la columna } 2^0) \\
42/2 = 21 &\rightarrow \text{resto} = 0 \text{ (en la columna } 2^1) \\
21/2 = 10 &\rightarrow \text{resto} = 1 \text{ (en la columna } 2^2) \\
10/2 = 5 &\rightarrow \text{resto} = 0 \text{ (en la columna } 2^3) \\
5/2 = 2 &\rightarrow \text{resto} = 1 \text{ (en la columna } 2^4) \\
2/2 = 1 &\rightarrow \text{resto} = 0 \text{ (en la columna } 2^5) \\
1/2 = 0 &\rightarrow \text{resto} = 1 \text{ (en la columna } 2^6) \\
84_{10} = 1010100_2 &
\end{aligned}
$$

- Ejemplos:

- Representar 421_{10} en base 2,

$$
\begin{aligned}
421_{10} &\rightarrow \textit{binario} \\
421/2 = 210 &\rightarrow \text{resto} = 1 \text{ (en la columna } 2^0) \\
210/2 = 105 &\rightarrow \text{resto} = 0 \text{ (en la columna } 2^1) \\
105/2 = 52 &\rightarrow \text{resto} = 1 \text{ (en la columna } 2^2) \\
52/2 = 26 &\rightarrow \text{resto} = 0 \text{ (en la columna } 2^3) \\
26/2 = 13 &\rightarrow \text{resto} = 0 \text{ (en la columna } 2^4) \\
13/2 = 6 &\rightarrow \text{resto} = 1 \text{ (en la columna } 2^5) \\
6/2 = 3 &\rightarrow \text{resto} = 0 \text{ (en la columna } 2^6) \\
3/2 = 1 &\rightarrow \text{resto} = 1 \text{ (en la columna } 2^7) \\
1/2 = 0 &\rightarrow \text{resto} = 1 \text{ (en la columna } 2^8) \\
421_{10} &= 110100101_2
\end{aligned}
$$

- Representar 11010111_2 en base 10,

$$
\begin{aligned}
11010111_2 &= 1 * 2^7 + 1 * 2^6 + 0 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 \\
11010111_2 &= 128 + 64 + 0 + 16 + 0 + 4 + 2 + 1 = 215_{10}
\end{aligned}
$$

- Representar $1101{,}101_2$ en base 10,

$$
\begin{aligned}
1101{,}101_2 &= 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} \\
1101{,}101_2 &= 8 + 4 + 0 + 1 + 0{,}5 + 0 + 0{,}125 = 13{,}625_{10}
\end{aligned}
$$

- Representar $23{,}43_{10}$ en base 2 con 8 dígitos después de la coma,

$$
\begin{aligned}
23{,}43_{10} &\rightarrow \textit{binario} \\
&\text{Parte entera:} \\
23/2 = 11 &\rightarrow \text{resto} = 1 \text{ (en la columna } 2^0) \\
11/2 = 5 &\rightarrow \text{resto} = 1 \text{ (en la columna } 2^1) \\
5/2 = 2 &\rightarrow \text{resto} = 1 \text{ (en la columna } 2^2) \\
2/2 = 1 &\rightarrow \text{resto} = 0 \text{ (en la columna } 2^3) \\
1/2 = 0 &\rightarrow \text{resto} = 1 \text{ (en la columna } 2^4) \\
&\text{Parte decimal:} \\
0{,}43 * 2 = 0{,}86 &\rightarrow \text{entero} = 0 \text{ (en la columna } 2^{-}1) \\
0{,}86 * 2 = 1{,}72 &\rightarrow \text{entero} = 1 \text{ (en la columna } 2^{-}2) \\
0{,}72 * 2 = 1{,}44 &\rightarrow \text{entero} = 1 \text{ (en la columna } 2^{-}3) \\
0{,}44 * 2 = 0{,}88 &\rightarrow \text{entero} = 0 \text{ (en la columna } 2^{-}4) \\
0{,}88 * 2 = 1{,}76 &\rightarrow \text{entero} = 1 \text{ (en la columna } 2^{-}5) \\
0{,}76 * 2 = 1{,}52 &\rightarrow \text{entero} = 1 \text{ (en la columna } 2^{-}6) \\
0{,}52 * 2 = 1{,}04 &\rightarrow \text{entero} = 1 \text{ (en la columna } 2^{-}7) \\
0{,}04 * 2 = 0{,}08 &\rightarrow \text{entero} = 0 \text{ (en la columna } 2^{-}8) \\
23{,}43_{10} &= 10111{,}01101110_2
\end{aligned}
$$

- Representar $10{,}27_{10}$ en base 2 con un error máximo de 0,1 %,

$$e = 10{,}27 * 0{,}1\,\% = 0{,}01027$$

Determinamos el número de bits necesarios para representar ese valor:

$$
\begin{aligned}
2^n &= 0{,}01027 \\
n &= \tfrac{ln(0{,}01027)}{ln(2)} = -6{,}605
\end{aligned}
$$

tomamos el entero menor,

$$n = -7$$

por lo tanto, para obtener un error menor al 0,1 % debemos utilizar $7 - bits$ para representar la parte decimal.

Parte entera:

$10_{10} = 1010_2$

Parte decimal:

$0{,}27 * 2 = 0{,}54 \rightarrow$ entero $= 0$ (en la columna 2^-1)

$0{,}54 * 2 = 1{,}08 \rightarrow$ entero $= 1$ (en la columna 2^-2)

$0{,}08 * 2 = 0{,}16 \rightarrow$ entero $= 0$ (en la columna 2^-3)

$0{,}16 * 2 = 0{,}32 \rightarrow$ entero $= 0$ (en la columna 2^-4)

$0{,}32 * 2 = 0{,}64 \rightarrow$ entero $= 0$ (en la columna 2^-5)

$0{,}64 * 2 = 1{,}28 \rightarrow$ entero $= 1$ (en la columna 2^-6)

$0{,}28 * 2 = 0{,}56 \rightarrow$ entero $= 0$ (en la columna 2^-7)

$23{,}43_{10} = 1010{,}0100010_2$

Comprobación:

$1010{,}010001_2 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2} + 0 * 2^{-3} + 0 * 2^{-4} + 0 * 2^{-5} + 1 * 2^{-6}$

$1010{,}010001_2 = 10{,}265625_{10}$

Conversión de Hexadecimal a Binario y Decimal

Los números *hexadecimales* están formados por grupos de 4 bits ($2^4 = 16$ posibilidades) y su *base* es 16. Los números hexadecimales usan los dígitos del 0 al 9 junto con las letras A a F. Cada columna en base 16 tiene pesos: $1, 16, 256, 4096, \cdots$.

- Hexadecimal a binario: cada dígito hexadecimal se corresponde con 4 dígitos binario (la conversión de 0 a 9 es directa, luego A = 1010, B = 1011, C = 1100, D = 1101, E = 1110 y F = 1111).

$$2ED_{16} \rightarrow binario$$
$$2_{16} \rightarrow 0010_2$$
$$E_{16} \rightarrow 1110_2$$
$$D_{16} \rightarrow 1101_2$$
$$2ED_{16} = 001011101101_2$$

- Hexadecimal a decimal: se debe realizar la sumatoria del producto de cada dígito hexadecimal (equivalente decimal) por el peso de la columna correspondiente, esto es.

$$2ED_{16} = 2 * 16^2 + E * 16^1 + D * 16^0$$
$$2ED_{16} = 2 * 16^2 + 14 * 16^1 + 13 * 16^0$$
$$2ED_{16} = 749_{10}$$

- Binario a hexadecimal: Se toman de a 4 bits comenzando desde la derecha y convirtiendo a su equivalente en hexadecimal.

$$1111010_2 \rightarrow hexadecimal$$
$$1010_2 = A_{16}$$
$$111_2 = 7_{16}$$
$$1111010_2 = 7A_{16}$$

- Decimal a hexadecimal: la conversión se hace de forma similar a la vista de decimal a binario. Comenzando desde el dígito a la izquierda, se comienza con la mayor potencia de 16 menor o igual al numero decimal, esto es:

$$333_{10} \rightarrow hexadecimal$$

la mayor potencia de 16 menor a 333 es 256 (16^2),

$$333 \geq 256 \rightarrow 1 \text{ (en la columna } 16^2)$$
$$333 - 256 = 77$$
$$77/16 = 4 \rightarrow 4 \text{ (en la columna } 16^1)$$
$$77 - (16 * 4) = 13$$
$$13 \rightarrow D \text{ (en la columna } 16^0)$$
$$33310 = 14D_{16}$$

Comenzando desde el dígito de la derecha, se divide repetidamente el número decimal por 16. El resto va en cada columna. Ejemplo:

$$333_{10} \rightarrow hexadecimal$$
$$333/16 = 20 \rightarrow \text{resto} = 13_{10} = D_{16} \text{ (en la columna } 16^0)$$
$$20/16 = 1 \rightarrow \text{resto} = 4_{10} = 4_{16} \text{ (en la columna } 16^1)$$
$$1/16 = 0 \rightarrow \text{resto} = 1_{10} = 1_{16} \text{ (en la columna } 16^2)$$
$$333_{10} = 14D_{16}$$

Número binarios con signo

Existen muchos esquemas para representar números con signo, los más comunes son *Signo+Magnitud*, *Complemento a 1*, *Complemento a 2* y *Exceso de 7*.

- Representación *Signo+Magnitud*: un número de N-bits representado por signo+magnitud usa el bit más significativo como bit de signo y los restantes $N-1$ bits como magnitud (en valor absoluto). Un bit de signo 0 indica signo positivo y 1 negativo.

  Rango: posee un rango simétrico $[-2^{N-1}+1, 2^{N-1}-1]$. Posee doble representación del cero $+0_{10} = 0000_2$ y $-0_{10} = 1000_2$, para $N = 4$ bits.

- Representación *Complemento a 1*: el complemento a uno de un número binario implica la inversión de unos por ceros y ceros por unos. Un número de N-bits representado en complemento a uno usa un bit para representar el signo y los $N-1$ bits restantes para representar la magnitud del número en valor absoluto para el caso de números positivos, o bien, el complemento a uno del valor absoluto, en caso de ser negativo.

  Rango: posee un rango simétrico $[-2^{N-1}+1, 2^{N-1}-1]$. Posee doble representación del cero $+0_{10} = 0000_2$ y $-0_{10} = 1111_2$, para $N = 4$ bits.

- Representación *Complemento a 2*: los números en complemento a 2 son idénticos a los números binarios sin signo excepto que el bit más significativo tiene peso -2^{N-1} en lugar de 2^{N-1}. En complemento a 2 el cero tiene una única representación, $0_{10} = 000\cdots000_2$. El mayor número positivo tiene un 0 en el bit más significativo y todos $1s$ en los restantes, $01\cdots111_2 = 2^{N-1}-1$. El menor número negativo tiene un 1 en el bit mas significativo y todos $0s$ en los restantes, $10\cdots000_2 = -2^{N-1}$. El -1 se representa con todos unos: $11\cdots111_2$ El proceso de tomar el complemento a 2 de un número consiste en invertir todos los bits del número, luego sumarle 1 al bit menos significativo. Esto permite encontrar la representación de un número negativo o determinar la magnitud de un número negativo. Rango: para $N-bit$, $[-2^{N-1}, 2^{N-1}-1]$. Existe un número negativo más que positivos ya que no existe el -0.

- Representación en *Exceso a K, K=7*: en esta representación a cada número se le suma un valor fijo, por lo tanto cada número está en *exceso* por dicho valor. Este formato es habitual para la representación del exponente en números en punto flotante. Por ejemplo para $N = 4$, la representación en exceso a $2^{N-1}-1 = 7$ permite representar los $2^4 = 16$ números en exceso a 7. Si queremos representar -4_{10} en exceso a 7, debemos sumar el exceso y luego convertir a binario, $-4_{10} + 7_{10} = 3_{10} \rightarrow 0011_2$.

  Rango: posee rango asimétrico $[-2^{N-1}+1, 2^{N-1}]$.

Comparación de sistemas numéricos:

Representación	*Rango*
Sin signo	$[0, 2^N - 1]$
Signo+Magnitud	$[-2^{N-1} + 1, 2^{N-1} - 1]$
Complemento a 1	$[-2^{N-1} + 1, 2^{N-1} - 1]$
Complemento a 2	$[-2^{N-1}, 2^{N-1} - 1]$
Exceso 7	$[-2^{N-1} + 1, 2^{N-1}]$

Ejemplos de representación de números binarios de 4 bits:

Decimal	*Valor absoluto*	*Sig+Mag*	*Complemento a 1*	*Complemento a 2*	*Exceso a 7*
+8	1000	-	-	-	1111
+7	0111	0111	0111	0111	1110
0	0000	0000 / 1000	0000 /1111	0000	0111
−3	0011	1011	1100	1101	0100
−7	0111	1111	1000	1001	0000
−8	1000	-	-	1000	–

Aritmética con enteros complemento a dos

La suma de dos números en la representación de complemento a dos se realiza de tratando por igual a todos los bits y si hay un bit de acarreo más allá del final de la palabra, éste se ignora.

$$\begin{array}{rcl} 1001 & (-7) & \\ +\ 0101 & (+5) & \\ \hline 1110 & = & -2 \end{array} \quad \begin{array}{rcl} 1100 & (-4) & \\ +\ 0100 & (+4) & \\ \hline 10000 & = & 0 \end{array} \quad \begin{array}{rcl} 0011 & (+3) & \\ +\ 0100 & (+4) & \\ \hline 1111 & = & 7 \end{array} \quad \begin{array}{rcl} 1100 & (-4) & \\ +\ 1111 & (-1) & \\ \hline 11011 & = & -5 \end{array}$$

Los sistemas digitales usualmente operan con un número fijo de dígitos. Si el resultado es demasiado grande que no entra en el número fijo de dígitos, entonces se dice que la adición desbordó (*overflow*). Por ejemplo, un número de $4 - bits$ tiene un rango de $[0, 15]$. La adición de $4 - bits$ desborda si el resultado es mayor a 15. El quinto bit es descartado produciendo un resultado de $4 - bits$ erróneo. La suma de dos números positivos o negativos de $N - bits$ puede causar *overflow* si el resultado es mayor que $2^{N-1} - 1$ o menor a -2^{N-1}. La suma de un número positivo con un número negativo nunca produce *overflow*. A diferencia de los números sin signo, un acarreo en del bit más significativo no implica *overflow*. Sí existe *overflow* cuando los dos números a sumar tienen el mismo signo y el resultado tiene signo opuesto. A continuación se muestran ejemplos de casos de *overflow*.

$$\begin{array}{rcl} 0101 & (+5) & \\ +\ 0100 & (+4) & \\ \hline 1001 & = & -7 \end{array} \quad \begin{array}{rcl} 1001 & (-7) & \\ +\ 1010 & (-6) & \\ \hline 10011 & = & 3 \end{array}$$

La sustracción de dos números puede considerarse como la suma del primer número y el opuesto del segundo. El opuesto de un número en la representación de complemento a dos, se obtiene mediante

la *transformación complemento a dos*. Esto es, obteniendo el *complemento a 1* del número original y sumándole 1 al binario obtenido. Por lo tanto, para sustraer un número (el sustraendo) de otro (el minuendo), se forma el complemento a dos del sustraendo y se le suma al minuendo. A continuación se muestran ejemplos de sustracción de números con signo de $4-bits$ representados en complemento a dos.

$$\begin{array}{rrcc} & 0010 & (+2) & \\ + & 1001 & - & (+7) \\ \hline & 1011 & = & -5 \end{array} \quad \begin{array}{rrcc} & 0101 & (+5) & \\ + & 1110 & - & (+2) \\ \hline & 10011 & = & 3 \end{array} \quad \begin{array}{rrcc} & 0101 & (+5) & \\ + & 0010 & - & (-2) \\ \hline & 0111 & = & 7 \end{array} \quad \begin{array}{rrcc} & 1011 & (-5) & \\ + & 1110 & - & (+2) \\ \hline & 11001 & = & -7 \end{array}$$

Casos de desbordamiento:

$$\begin{array}{rrcc} & 0111 & (+7) & \\ + & 0111 & - & (-7) \\ \hline & 1110 & = & -2 \end{array} \quad \begin{array}{rrcc} & 1010 & (-6) & \\ + & 1100 & - & (+4) \\ \hline & 10110 & = & 6 \end{array}$$

Sistemas Numéricos

Las computadoras operan tanto con enteros como con fracciones. Hasta ahora solo consideramos las representaciones de *enteros* con signo y sin signo. Ahora introduciremos la representación de números racionales mediante los sistemas numéricos de *punto fijo* y *punto flotante*. Los números en punto fijo son semejantes a los decimales; algunos bits representan la parte entera y los restantes representan la parte fraccional. Por otra parte, los números en punto flotante son análogos a la notación científica, donde se tiene una mantisa y un exponente.

Punto fijo

- Notación: la notación en punto fijo tiene implícito un punto binario entre los bits que representan la parte entera y los bits que representan la parte fraccional, de forma similar al punto decimal que existe entre la parte entera y fraccional de un número decimal cualquiera. Se dice *implícito* ya que al igual que cualquier número binario, los números en punto fijo son una colección de bits y no existe una forma de conocer la existencia del punto binario salvo a través de un acuerdo entre las personas que interpretan el número. Por ejemplo, un número en punto fijo con cuatro bits para representar enteros (*high word*) y cuatro bits para la fracción (*low word*) es:

  01101100

  con el punto binario implícito:

  0110,1100

  cuyo equivalente decimal es:

$$2^2 + 2^1 + 2^{-1} + 2^{-2} = 6{,}75_{10}$$

- Signo: los números en punto fijo signados pueden usar la notación de signo+magnitud o de complemento a dos. Ejemplo de representación del número $-2{,}375$, 4 bits parte entera y 4 bits parte fraccional:

  Valor absoluto:

  $$0010{,}0110$$

  Signo y magnitud:

  $$1010{,}0110$$

  Complemento a dos:

  $$1101{,}1010$$

- Aritmética en punto fijo: la aritmética en punto fijo es similar a lo visto con enteros. Usando la representación en complemento a dos, en la suma y la resta se debe hacer coincidir las posiciones de la coma y luego realizar la operación aritmética como el número fuera entero. Ejemplo: Sumar $0{,}75 + (-0{,}625)$ usando números de punto fijo.

$$\begin{aligned} 0{,}75_{10} &= 0000{,}1100_2 \\ -0{,}625_{10} &\rightarrow 0000{,}1010_2 \text{ Magnitud} \\ &\quad 1111{,}0101_2 \text{ Complemento a uno} \\ &\quad 1111{,}0110_2 \text{ Complemento a dos} \end{aligned}$$

$$\begin{array}{rrcr} & 0000{,}1100 & & (+0{,}750) \\ + & 1111{,}0110 & & (-0{,}625) \\ \hline & 10000{,}0010 & = & +0{,}125 \end{array}$$

Punto flotante

Los números en punto flotante son análogos a la notación científica. Como en el caso de números de base decimal, también es necesario en algunos casos trabajar con números muy grandes o pequeños. Para ello resulta mas cómodo poder expresarlos en un formato que permita operar con números de diferente posición de la como. A esto se le denomina números de coma flotante. Al igual que la notación científica, los números en punto flotante tienen un *signo*, *mantisa* (M), *base* (B) y *exponente* (e).

$$\pm M * B^e$$

Se usan 32 bits para representar 1 bit de signo, 8 bits de exponente y 23 bits de mantisa.

- Exponente: el exponente de un número de punto flotante es modificado antes de su almacenamiento con el fin de poder representar exponentes tanto positivos como negativos. Por lo cual, el número a almacenar en la posición del exponente será igual al exponente original sumado a una constante de *polarización* (representación en *exceso k*):

  $$e_{ieee} = e + 2^{m-1} - 1$$

  Donde e será el exponente original y m es el número de bits del exponente en el formato elegido. El punto flotante de 32 bits usa 8 bits para el exponente y suma 127 al exponente original.

- Mantisa: la mantisa debe cumplir la condición, $1 \leq mantisa < 2$. Por lo tanto la parte entera será siempre 1 y no necesita ser almacenada. Solo se almacenan los bits de la parte decimal.

- Signo: en el caso del punto flotante, en la mantisa se guarde siempre el valor absoluto de la misma. El bit de signo identificará que tipo de número es,

  $1 \rightarrow$ Negativo
  $0 \rightarrow$ Positivo

Formatos

- Precisión simple: son número en punto flotante de 32 bits, también denominados *single-precision*, *single* o *float*.

- Precisión doble: son números en punto flotante de 64 bits, también denominados *doubles*, que proveen mayor precisión y rango.

Formato	*Número de bits*	*Signo*	*Exponente*	*Mantisa (parte decimal)*
Simple	32	1	8	23
Doble	64	1	11	52

Ejemplo, representación en punto flotante del número decimal 228:
Primero se debe convertir el número decimal a binario,
$228_{10} = 11100100_2 = 1{,}11001_2 x 2^7$
Luego, codificar el bit de signo, en este caso será 0. El exponente, de 8 bits y valor $e = 7$, convertido al estándar IEEE 754 se debe sumar 127. $e_{ieee} = 7 + 127 = 134 = 10000110_2$. La mantisa, de 23 bits será la parte decimal del número, $M = 11001000000000000000000$.

1 bit	8 bits	23 bits
0	10000110	11001000000000000000000
Signo	Exponente	Mantisa (parte decimal)

Adición en punto flotante

La suma con números en punto flotante no es tan simple como la suma con números en complemento a dos. A continuación se muestran los pasos para sumar números en punto flotante con el mismo signo:

- Extraer los bits del exponente y parte decimal.
- Anteponer el bit 1 para formar la mantisa.
- Comparar los exponentes.
- Si es necesario desplazar la mantisa menor.
- Sumar mantisas.
- Normalizar mantisas y si es necesario ajustar el exponente.
- Redondear el resultado.
- Formar el resultado en punto flotante con su exponente y parte decimal.

Ejemplo, sumar los números en punto flotante 7,875$(1{,}11111x2^2)$y 0,1875$(1{,}1x2^-3)$.

7,875	0	10000001	111	1100	0000	0000	0000	0000
0,1875	0	01111100	100	0000	0000	0000	0000	0000

		Exponente	Parte decimal						
Paso 1		10000001	111	1100	0000	0000	0000	0000	
		01111100	100	0000	0000	0000	0000	0000	
Paso 2		10000001	1,111	1100	0000	0000	0000	0000	
		01111100	1,100	0000	0000	0000	0000	0000	
Paso 3		10000001	1,111	1100	0000	0000	0000	0000	
	-	01111100	1,100	0000	0000	0000	0000	0000	
		101							
Paso 4		10000001	1,111	1100	0000	0000	0000	0000	
		10000001	0,000	0110	0000	0000	0000	0000	00000
Paso 5		10000001	1,111	1100	0000	0000	0000	0000	
		10000001	0,000	0110	0000	0000	0000	0000	
			10,000	0010	0000	0000	0000	0000	
		10000001	10,000	0010	0000	0000	0000	0000	>> 1
Paso 6	+	1							
		10000010	1,000	0001	0000	0000	0000	0000	
Paso 7		No necesita redondeo							
Paso 8	0	10000010	000	0001	0000	0000	0000	0000	

Redondeo

La norma IEEE-754 contempla cuatro modalidades de redondeo,

- Redondeo al más cercano (al par en caso de empate)
- Redondeo a más infinito (por exceso)
- Redondeo a menos infinito (por defecto)
- Redondeo a cero (truncamiento)

Todas estas modalidades de redondeo requieren el uso de bits adicionales de precisión, denominados *bits de guarda* y se encuentran a la derecha del bit menos significativo. Los bits de guarda se conservan temporalmente y son empleados en la normalización del resultado y durante el redondeo.

La norma IEEE-754 contempla tres bits de guarda, denominados guard (G), round (R) y sticky (S).

- Roun (R) y sticky (S): la siguiente tabla resume el rol de los bits R y S en los distintos esquemas de redondeo (donde p_0 es el valor del dígito menos significativo del resultado). El bit S se calcula haciendo OR entre los bits descartados (los que no forman parte del resultado ni tampoco de G ni R).

Tipo de Redondeo	*resultado* ≥ 0	*resultado* < 0
más próximo	+1 LSB si (R AND p_0) OR (R AND S)	+1 LSB si (R AND p_0) OR (R AND S)
hacia $+\infty$	+1 LSB si (R OR S)	
hacia $-\infty$		+1 LSB si (R OR S)
hacia 0	se descartan los bits desplazados a la derecha	se descartan los bits desplazados a la derecha

Ejemplo:

Suma y redondeo,

$$\begin{array}{rrcl} & 1{,}001000 & x & 2^3 \\ + & 1{,}101001 & x & 2^{-1} \\ \hline & \Downarrow & & \end{array}$$

$$\begin{array}{rrcl} & 1{,}001000 & x & 2^3 \\ + & 0{,}0001101001 & x & 2^3 \\ \hline & 1{,}00111011 & x & 2^3 \end{array}$$

$$\text{RS} \rightarrow \quad \text{R = 1, S = (0 OR 0 OR 1)}$$

Resultado de redondeo al más próximo,

$$\begin{array}{rrcl} & 1{,}001110 & x & 2^3 \\ + & 1 & & \\ \hline & 1{,}001111 & x & 2^3 \end{array}$$

Resultado de redondeo a cero: $1{,}001110x2^3$

- Guard (G): el bit G se usa al normalizar el resultado preliminar de las operaciones aritméticas, esto es, antes de aplicar el redondeo. En los casos en que no haga falta usar el bit G (cuando no sea necesario normalizar el resultado) se deben ajustar los valores de los bits R y S de la siguiente manera:

$R_{nuevo} = G_{anterior}$
$S_{nuevo} = R_{anterior}$ OR $S_{anterior}$

Ejercicio 1:

Representar los siguientes números en la base solicitada,

1. 1745_8 en decimal

2. 18493_{10} en Binario
3. 10101011_2 en Decimal
4. 15626_{10} en Hexadecimal
5. 3432_8 en Binario
6. 1746_{10} en Octal
7. 134882_{16} en Binario

Ejercicio 2:

Representar los siguientes número en base 10 a binario, tomando 8 bit después de la coma.

1. $8{,}125_{10}$
2. $0{,}3_{10}$
3. $0{,}42_{10}$

Ejercicio 3:

Representar los siguientes número en base 10 a binario, con un error máximo de 0,1 %.

1. $14{,}45_{10}$
2. $5{,}955_{10}$
3. $0{,}47_{10}$

Ejercicio 4:

Representar los siguientes números en base 2 a decimal,

1. $1011{,}001101_2$
2. $11{,}1010101_2$
3. $0{,}110111001_2$

Ejercicio 5:

Representar en base 2 los siguientes números con la longitud indicada,

Número Decimal	1 byte	16 bits o medio Word
-51		
-130		
-32125		

Indicar cual o cuales no pueden ser representados por 1 byte.

Ejercicio 6:

Realizar las siguientes operaciones aritméticas en base 2,

1. Sumar $00011101 + 10110010$
   a) Suponiendo que son números sin signo.
   b) Suponiendo que son números con signo (8 bits).
2. Restar $00101101 - 11100101$
   a) Suponiendo que son números sin signo.
   b) Suponiendo que son números con signo (8 bits).

solución:

1. a)

$$\begin{array}{rcc} & 00110000 & \\ & 00011101 & (29) \\ + & 10110010 & (178) \\ \hline & 11001111 & 207 \end{array}$$

b)

$$\begin{array}{rcc} & 00110000 & \\ & 00011101 & (29) \\ + & 10110010 & (-78) \\ \hline & 11001111 & -49 \end{array}$$

$C_2(10110010) = 01001110 = 78$
$C_2(11001111) = 00110001 = 49$

2.

a) se agrega un 0 a la izquierda para evitar desborde,

$$\begin{array}{rll} & 000101101 & (45) \\ - & 011100101 & (229) \end{array}$$

se saca el complemento a 2 del sustraendo y se suman,
$C_2(011100101) = 100011011 = -229$

$$\begin{array}{rll} & 000111111 & \\ & 000101101 & (45) \\ + & 100011011 & (-229) \\ \hline & 101001000 = & -184 \end{array}$$

para saber el valor absoluto del resultado que fue negativo se saca el complemto a dos del mismo,
$C_2(101001000) = 010111000 = 184$

b) se agrega un bit más igual al bit de signo a la izquierda para evitar desborde,

$$\begin{array}{rll} & 000101101 & (45) \\ - & 111100101 & (-27) \end{array}$$

se saca el complemento a 2 del sustraendo y se suman,
$C_2(111100101) = 000011011 = 27$

$$\begin{array}{rll} & 000111111 & \\ & 000101101 & (45) \\ + & 000011011 & (27) \\ \hline & 001001000 = & 72 \end{array}$$

el resultado fue positivo.

Ejercicio 7:

Exprese los siguientes números decimales en el formato 16 bits punto fijo signo+magnitud con ocho bits enteros y ocho decimales. Exprese la respuesta en hexadecimal.

1. -13.5625

2. 42.3125
3. -17.15625

Solución:

1. -13.5626

   Parte entera valor absoluto: $13_{10} = 00001101_2$

   Parte decimal valor absoluto: $0{,}5626_{10} = 0{,}10010000_2$

   Número en formato signo+magnitud: $-13{,}5625_{10} = 10001101{,}10010000_2 = 0x8D90_{16}$

2. $42{,}3125_{10} = 0x2A50_{16}$
3. $-17{,}15625_{10} = 0x9128_{16}$

Ejercicio 8:

Exprese los siguientes números decimales en el formato 12 bits punto fijo complemento a dos con seis bits enteros y seis decimales. Exprese la respuesta en hexadecimal.

1. -30.5
2. 16.25
3. -8.078125

Ejercicio 9:

Exprese los números decimales del ejercicio 8 en el formato de punto flotante de simple precisión IEEE 754. Exprese la respuesta en hexadecimal. *Solución:*

1. -30.5

   Valor en binario: $-30{,}5_{10} = -11110{,}1_2$

   Valor normalizado: $-30{,}5_{10} = -1{,}11101x2^4$

   Signo: 1

   Exponente polarizado ($e_{ieee} = e + 2^{m-1} - 1$): $e_{ieee} = 4 + 127 = 131_{10} = 10000011_2$

   Número en formato punto flotante de simple precisión IEEE 754:

   $1\ 10000011\ 11101000000000000000000_2 = 0xC1F40000_{16}$

2. $16{,}25_{10} = 0\ 10000011\ 00000100000000000000000_2 = 0x41820000$
3. $-8{,}078125_{10} = 1\ 10000010\ 00000010100000000000000_2 = 0xC1014000$

Ejercicio 10:

Sumar los siguientes números en punto flotante de simple precisión IEEE 754.

1. $C0123456 + 81C564B7$
2. $D0B10301 + D1B43203$
3. $5EF10324 + 5E039020$

Solución:

1. $C0123456 + 81C564B7 = C0123456_{16}$
2. $D0B10301 + D1B43203 = D1E072C3_{16}$

$D0B10301$	1	10100001	011	0001	0000	0011	0000	0001
$D1B43203$	1	10100011	011	0100	0011	0010	0000	0011

		Exponente	Parte decimal						
Paso 1		10100001	011	0001	0000	0011	0000	0001	
		10100011	011	0100	0011	0010	0000	0011	
Paso 2		10100001	1,011	0001	0000	0011	0000	0001	
		10100011	1,011	0100	0011	0010	0000	0011	
Paso 3		10100001	1,011	0001	0000	0011	0000	0001	
	-	10100011	1,011	0100	0011	0010	0000	0011	
		10							
Paso 4		10100011	0,010	1100	0100	0000	1100	0000	01
		10100011	1,011	0100	0011	0010	0000	0011	
Paso 5		10100011	0,010	1100	0100	0000	1100	0000	
		10100011	1,011	0100	0011	0010	0000	0011	
			1,110	0000	0111	0010	1100	0011	

Paso 6 No necesita normalizar mantisa.

Paso 7 Con redondeo por truncamiento.

Paso 8	1	10100011	110	0000	0111	0010	1100	0011

3. $5EF10324 + 5E039020 = 5F19659A_{16}$

Ejercicio 11:

Sumar los siguientes números en punto flotante de simple precisión IEEE 754, haciendo uso del redondeo al más próximo y redondeo a cero.

$x_1 = 567FFFFF$, $x_2 = 4A520000$.