Representación de numeros reales

Dr. Roberto Gómez Cárdenas

16 de marzo de 1999

Resumen

La importancia que tienen los números reales hoy en día fue la principal motivación para escribir este documento. Los números reales tienen un gran uso en diferentes áreas de la ingeniería y de la ciencia. Es importante conocer como las computadoras representan y manejan estos números para poder aprovechar mejor las facilidades de cálculo que nos ofrecen estas máquinas.

1 Introducción

La representación de un número entero, a excepción de los números negativos decimal es su conversión a base dos. En este documento se abarcan tres formas de representar números reales en un sistema digital.

El presente documento se ecuentra dividido de la siguiente forma, la primera sección presenta los aspectos a considerar para representar un número real; la segunda el formato normalizado de representación, después se presenta el formato sesgado y por último se explica el formato IEEE-754 de representación de números reales.

2 Representando numeros reales

En los cálculos no siempre intervienen valores enteros, debido a la naturaleza de muchos problemas es necesario manejar números reales o de punto flotante. El problema aparece con el inconveniente de como representar en binario el punto para manejar cantidades fraccionarias. El punto podría representarse con un 1 o con un 0 dentro de la cadena de bits que representa el número binario; sin embargo el punto no tiene una posición fija, sino variable dentro de la cadena:

```
111 = 11.1 = 1.11
```

La solución consiste en eliminar el punto de la secuencia e indicar en alguna forma el lugar que ocupa dentro del número. Es por esto que los números se representan de acuerdo al siguiente formato:

$$M \times B^E$$

donde:

M: la mantisa representa el número

E: el exponente la posición del punto dentro del número

B: la base la base en la que esta representado el número

Esta notación también recibe el nombre de notación científica, y es bastante utilizada en el área de ingeniería. Debido a la naturaleza de la notación es posible representar un mismo número de diferentes formas. A continuación se listan tres formas de representar el número 3.5:

$3.5 = 3.5 \times 10^0$

$3.5 = 35 \times 10^{-1}$

$3.5 = 0.35 \times 10^{+1}$

En este caso el punto flota, no ocupa un lugar fijo, de ahí el nombre de punto flotante. Sin embargo muchas veces el termino número punto flotante y el termino número real se usan indistintamente, pero es importante establecer la diferencia entre uno y otro.

La diferencia clave entre los números de punto flotante y los reales esta en su densidad. Entre dos numeros reales, x e y, siempre hay otro número real, no importa que tan cercanos esten x e y. Esto se puede comprobar por el hecho de que para todo par de números reales diferentes, x e y $z = (x + y)/2$, siendo z otro número real situado entre ambros. Es por esto quse puede afirmar que los los números de punto flotante forman un continuo.

Por otro lado, los números en punto flotante no forman un continuo. Supongamos que utilizamos tres dígitos para representar la mantisa, dos para el exponente y dos signos, (+/-). Tomando en cuenta los datos anteriores hay exactamente hay 179100 números positivos, 179100 negativos y el cero (que puede representarse de muchas formas) dando un toal de $358,201$ números. Del número infinito de números reales que se encuentran entre $i - 1 \times 10^{+100}$ y $+0.999 \times 10^{99}$, solamaente es posible representar $358,201$. Si algún cálculo arroja un resultado entre ese rango y no se puede expresar en el sistema de representación usado, es evidente que se debe utilizar el número expresable más cercano. A este proceso se le conoce como *redondeo*.

2.1 Las fracciones normalizadas

Para evitar confusiones acerca de la posición del punto dentro de un número se utilizan las fracciones normalizadas.

Una fracción normalizada se caracteriza por que el punto esta a la izquierda del dígito más significativo y distinto de cero de la mantisa.

Ejemplos de fracciones normalizadas son:

$$\begin{aligned} 12.23 &= 0.1223 \times 10^{+2} \\ 5218.23 &= 0.521823 \times 10^{+4} \\ 387.6721 &= 0.38767 \times 10^{+3} \end{aligned}$$

Como se puede constatar con los ejemplos anteriores, es necesario recorrer el punto hasta que a la izquierda de él no exista ningún dígito diferente de cero. Al recorrer el punto es necesario incrementar el valor del exponente. En algunas ocasiones se tiene que recorrer el punto a la izquierda y decrementar el valor del exponente, por ejemplo:

$$0.0000345 = 0.345 \times 10^{-4}$$

$$0.0021651 = 0.21651 \times 10^{-2}$$

La utilidad de los números normalizados es que estos proporcionan una forma estándar para representar los números reales.

2.2 Los rangos de los números reales

La memoria de una computadora es finita. El número de bits que esta utiliza también lo es, por lo que no es posible representar todos los números reales. Existen números para los cuales el número de bits disponibles no es suficiente. La recta real se divide en 7 regiones, la cual establece los límites de representación de números. Consideremos que se utilizan tres dígitos para representar la mantisa, dos para el exponente y los signos +/-, entonces los límites de la recta son los siguientes:

1. Números negativos grandes, menores que -0.999×10^{99}
2. Números negativos entre -0.999×10^{99} y -0.100×10^{-99}
3. Números negativos pequeños, de modulo menor que $0.100 \times 10^{-}99$
4. El cero.

5. Números positivos pequeños, menores que 0.100×10^{-99}
6. Números positivos entre 0.100×10^{-99} y 0.99×10^{99}
7. Números positivos grandes, mayores que 0.999×10^{99}

Las regiones uno y siete representan números muy grandes grandes que no es posible representar, mientras que las regiones tres y cinco son números muy pequeños. En el diagrama de abajo se muestran todas las regiones anteriores. Cuando el número es muy grande (ya sea positivo o negativo) y el número de bits usados para representar el exponente no es suficiente decimos que hay un desbordamiento (overflow en inglés) positivo o negativo según sea el caso.

```
      (1)             (2)              (3)           (4)          (5)             (6)           (7)
   dddddddddd | xxxxxxxxxx |uuuuuuuuuuuuuuuuuu|   0   |uuuuuuuuuuuuuuuuuuu| xxxxxxxxx | dddddddddd

desbordamiento    numeros     subdesordamiento  CERO    subdesordamiento    numeros    desbordamiento
   negativo      negativos        negativo                  positivo        positivos      positivo
                expresables                                                expresables
```

Por otro lado, cuando el número es muy pequeño (positivo o negativo muy cercano al cero) y el número de bits del exponente no es suficiente para representar un exponente negativo, se dice que hay un subdesbordamiento (underflow en inglés) negativo o positivo según sea el caso.

Otro detalle ocurre en los bits de la mantisa. Si estos no son suficientes es posible que el número real no pueda ser representado. En este caso existen dos opciones: que el número sea truncado (de 0.76325198456 pase a 0.763251) o que se diga que el número no es posible representarlo. Lo mismo ocurre en el caso de los números muy pequeños (0.352×10^{-999}) ya que es posible señalar un subdesbordamiento o considerar el número como cero. Esto último se hace en algunos cálculos aritméticos.

2.3 La representación binaria de los numeros reales

Existen diferentes formas de representar un número real con 1's y 0's. Todos se caracterizan por utilizar dos entidades numéricas:

1. la mantisa (fracción normalizada y con signo)
2. el exponente (número entero y con signo)

La base es conocida, base dos, por lo que no aparece representada. El exponente indica la posición del punto en la mantisa y los signos `+ -`, que acompañan al exponente, indican su desplazamiento a la derecha o izquierda con respecto del origen.

Los memoria del procesador almacena el valor de la mantisa y el exponente. El origen del punto y la base son previamente determinados por un convenio (fracción normalizada y base dos) y conocidos por los programas que utilizan los números.

Se consideran formatos con m bits para la mantisa y n bits para el exponente. A partir de esto podemos definir el rango de representación de números reales que siempre es finito. Algunos números reales son representables en el formato y otros quedan afuera del mismo, (overflow o underflow según sea el caso).

Existen tres formatos de representación de números reales, el formato normalizado, el formato sesgado y el formato IEEE-754. En las siguientes secciones se expondrán las características de cada uno de ellos.

3 Formato normalizado

El primero de los formatos asume m bits para la representación del signo y e bits para la representación del exponente. El número cumple con el formato siguiente:

```
 ---------------------------------------
|                  |                    |
|   bits mantisa   |   bits exponente   |
|__________________|____________________|
```

La mantisa representa un número binario decimal, es decir los bits tienen asignados los siguientes pesos:

bit 1	bit 2	bit 3	bit 4	...	bit m
2^0	2^{-1}	2^{-2}	2^{-3}	...	$2^{-(m-1)}$

El exponente se representa como un número en base dos, utilizando complementos a dos (en el formato estándar, aunque es posible utilizar complementos a uno) si este es negativo.

Es importante hacer notar que el primer bit de la mantisa nunca es usado, tan solo sirve para indicarnos el signo de esta. Si tiene un valor de 1 la mantisa es negativa y hay obtener su complemento a dos, si es cero es positiva y el número se convierte directamente.

3.1 Ejemplo conversión formato normalizado a número real

Como ejemplo consideremos un formato con siete bits para representar la mantisa ($m = 7$) y cinco bits para representar el exponente ($e = 5$). Ahora bien se desea saber el valor del número:

```
1010000 00101
```

Empezemos por calcular el valor de la mantisa `1010000`.

El primer bit es un 1 por lo que es número negativo, entonces hay que obtener su complemento a dos (cambiar unos por ceros, ceros por unos y sumarle un uno):

```
1010000
0101111
     +1
-------
0110000
```

Ahora hay que convertir el número a base dos, tomando en cuenta el peso de los bits:

$$\texttt{0110000} = (0\times2^0)+(1\times2^{-1})+(1\times2^{-2})+(0\times2^{-3})+ (0\times2^{-4})+(0\times2^{-5})+(0\times2^{-6})$$
$$\texttt{0110000} = 0.5 + 0.25 = 0.75$$

No olvidemos que es un número negativo por lo que el valor de la mantisa es de:

$$M = -0.75$$

Ahora calculemos el valor del exponente `00101`.

El primer bit es 0, lo que implica que se trata de un número positivo. La conversión consiste en calcular el valor del número binario en base 10:

$$\texttt{00101} = (1 \times 2^0) + (0 \times 2^2) + (1 \times 2^3) + (0 \times 2^4)+ (0 \times 2^5)$$
$$\texttt{00101} = 5$$

Entonces el valor del exponente es:

$$E = +5$$

Estamos en base dos, por lo que el valor de la base es dos, ($B = 2$).
El número se construye en base a lo calculado anteriormente:

$$M \times B^E = -0.75 \times 2^{+5} = -0.75 \times 32 = 24_{10}$$

Por lo que:

$$\texttt{1010000 00101} \;\; = +24_{10}$$

Supongamos ahora un formato con 4 bits de mantisa y 4 de exponente. Se deja como ejercicio al lector la verificación de las siguientes igualdades:

- `1001 1000` $-(0.875) \times 2^{-8} = -0.003417$
- `1111 1010` $-(0.125) \times 2^{-6} = -0.00195$
- `1010 0111` $-(0.75) \times 2^{7} = -96$
- `0101 0110` $+(0.625) \times 2^{6} = 40$
- `0111 0111` $+(0.875) \times 2^{7} = 112$

3.2 Ejemplo conversión número real a formato normalizado

Ahora se trata el caso contrario. Se trata de calcular como se representaría el número 27.5625 con un formato normalizado con seis bits de mantisa y diez de exponente.
Lo primero es convertir el número a binario:

$$27.5625 = 11011.1001$$

Después hay que normalizar el número anterior:

$$\begin{aligned} 11011.1001 &= 11011.1001 \times 2^0 \\ &= 0.110111001 \times 2^5 \end{aligned}$$

El orden en que se hagan las dos operaciones anteriores no es importante. Es posible primero normalizar el número en base diez y después convertir a base dos, sin embargo se considera que es más fácil convertir el número antes de normalizarlo.
El siguiente paso es insertar el bit de signo en la mantisa; si este es positivo hay que insertar un 0, si es negativo es un 1. El resto de los bits se recorren. Si el número de bits no fuera suficiente se deben de perder los bits que se encuentren más a la derecha, esto repercutirá en una perdida de exactitud en el número (ver último parrafo de la sección 2).
Ya que 27.5625 es positivo hay que insertar un cero; entonces la mantisa quedaría:

$$\texttt{110111001} = \texttt{0110111001}$$

Una vez calculada la mantisa es necesario calcular el exponente. Lo cual se hace convirtiendo el exponente a base dos (con el número de bits especificado). Como el exponente es positivo la conversión se hace directamente por lo que:

$$5 = \texttt{000101}$$

Por último se junta la mantisa y el exponente, lo cual da como resultado:

$$27.5625 = \;\; \texttt{0110111001 000101}$$

Ahora consideremos un número negativo, calculemos la representación del número -9.125 con ocho bits de mantisa y cuatro de exponente.

Primero se debe convertir a binario el valor absoluto del número:

$$9.125 = 1001.001$$

Después se normaliza el número convertido:

$$1001.001 \times 2^0 = 0.1001001 \times 2^4$$

Ya que es un número negativo es necesario obtener el complemento a dos de la mantisa:

```
1001001
0110110
     +1
-------
0110111
```

Se inserta el bit signo (1 ya que es negativo) en la mantisa:

$$0110111 = \texttt{10110111}$$

Se convierte el exponente:

$$4 = \texttt{0100}$$

y por último se juntan las dos partes:

$$-9.125 = \texttt{10110111} \quad \texttt{0100}$$

Algunos comentarios extras

Si al obtener el valor de la mantisa esta utiliza menos bits que los establecidos por el formato, es necesario completar el resto. Ya sea que el número es positivo o negativo, se tienen que completar insertando bits con valor cero a la derecha de la mantisa. Por ejemplo si se tiene una mantisa de 01101 y se tienen ocho bits, entonces el valor final de esta es de:

$$m = \texttt{01101000}$$

Como ejemplo de número negativo tomemos el número `10110`, el valor de la mantisa a 8 bits es de:

$$m = \texttt{10110000}$$

En el caso de que falten bits para representar el exponente positivo de un número se declara un *overflow*, si faltan bits para expresar el exponente negativo de otro número se declara un *underflow*.

3.3 Rangos representación del formato normalizado

El obtener estos rangos esquivale a encontrar el valor del último número real, antes de caer en las zonas uno, tres, cinco y siete. En esta sección calcularemos esos cuatro números, para lo cual se asume un sistema que utiliza seis bits de mantisa y seis de exponente, así como complementos a dos para representar los números negativos,

La zona uno la componen números negativos grandes, por lo que es necesario calcular el valor del exponente positivo más grande que se puede representar con seis bits y el valor negativo más grande que se puede representar con seis bits para la mantisa.

Empezemos con el exponente, con seis bits y complementos a dos el valor positivo más grande que se puede representar es cuando todos los bits son 1 excepto el más significativo. Entonces el valor del exponente es `011111`.

Si se tratara de un número positivo el valor de la mantisa sería (`011111`) pero como es negativo hay que obtener el complemento a dos de este número:

```
011111
100000
    +1
------
100001
```

Por lo que el valor de la mantisa es `100001`. Si juntamos los los resultados anteriores el número negativo más grande es:

$$
\begin{aligned}
\texttt{100001 0111111} &= -(0.96875 \times 2^{+31}) \\
\texttt{100001 0111111} &= -2,080,374,784
\end{aligned}
$$

Un detalle que se puede prestar a confusión es el hecho de que se utilizan complementos a dos para la mantisa. Esto puede llevar a pensar que el valor negativo más grande es `100000`. Sin embargo esto se aplica a enteros no a decimales. No hay que olvidar que en este formato el primer bit de la mantisa solo es usado para el signo.

La zona tres contempla los números negativos muy pequeños cercanos al cero. En este caso es necesario calcular el valor del exponente negativo más grande y el valor negativo más chico de la mantisa.

El exponente utiliza complementos a dos, por lo que el valor negativo más grande es el negativo más grande que se puede representar con seis bits y complementos a dos. Este valor es `100000`.

La mantisa positiva más chica es `000001`, pero como se requiere un valor negativo es necesario obtener el complementos a dos de este número:

```
000001
111110
    +1
------
111111
```

Entonces el valor de la mantisa es `111111`, que juntandolo con el del exponente nos da el número negativo más chico:

$$
\begin{aligned}
\texttt{111111 100000} &= -((0 \times 2^{-1}) + (0 \times 2^{-2})... + (1 \times 2^{-5})) \times (2^{-32}) \\
\texttt{111111 100000} &= -((1 \times 2^{-5}) \times 2^{-32}) \\
\texttt{111111 100000} &\simeq -1 \times 2^{-37}
\end{aligned}
$$

La zona cinco representa los números positivos muy pequeños cercanos a un valor de cero. Para calcular este número debemos de obtener el valor del exponente negativo más chico y del valor de la mantisa positivo más chico.

El valor del exponente negativo más chico ya se calculó y es `100000`, la mantisa positiva más chica es `000001`. Por lo que el número positivo más chico es:

$$
\begin{aligned}
\texttt{000001 100000} &= ((0 \times 2^{-1}) + (0 \times 2^{-2})... + (1 \times 2^{-5})) \times (2^{-32}) \\
\texttt{000001 100000} &= ((1 \times 2^{-5}) \times 2^{-32}) \\
\texttt{000001 100000} &\simeq 1 \times 2^{-37}
\end{aligned}
$$

La última zona a calcular (zona siete) es la de los número positivos muy grandes. Este cálculo implica conocer el valor positivo más grande del exponente y la mantisa. En ambos casos, ya que se usa complementos a dos, este número se representa cuando todos los bits son 1 excepto el primer bit izquierdo (el del signo) que toma el valor de 0. El valor del positivo más grande es:

$$\texttt{011111 011111} = (0.5 + 0.25 + 0.125 + 0.0625 + 0.03125) \times (2^{+31})$$
$$\texttt{011111 011111} = (0.96875 \times 2^{+31})$$
$$\texttt{011111 011111} = 2,080,374,784$$

Resumiendo los rangos del formato normalizado para la representación de número reales son:

- Número negativo más grande: `100001 0111111` $= -2,080,374,784$
- Número positivo más grande: `011111 0111111` $= -1 \times 2^{-37}$
- Número negativo más chico: `111111 1000000` $= 1 \times 2^{-37}$
- Número positivo más chico: `000001 1000000` $= 2,080,374,784$

4 Formato sesgado

El formato sesgado no utiliza complementos a dos, sino que utiliza un bit para designar el signo que se tiene y dos campos para representar la mantisa y el exponente. La representación del exponente se hace de acuerdo a un formato conocido como exceso 2^{m-1}.

Antes de explicar las características del formato sesgado de números reales se explicará en que consiste la representación exceso 2^{m-1}.

4.1 Representación exceso a 2^{m-1}

Este tipo de representación se usa para representar números de m bits, los números pueden ser positivos o negativos. El número es el equivalente binario (con m bits) del resultado de la suma de este y 2^{m-1}.

Tomemos como ejemplo un sistema que utiliza 8 bits ($m = 8$). En este caso el sistema es exceso 128 ($2^{8-1} = 2^7 = 128$), y para representar un número en este sistema se tendrá que sumar dicho número con 128.

Por ejemplo, para representar los números -3 y $+8$:

$$-3 = -3 + 128 = 125 = \texttt{01111101}$$
$$+8 = +8 + 128 = 136 = \texttt{10001000}$$

Los números de -128 a +127 se corresponden con los números de 0 a 255, los cuales pueden expresarse como enteros de 8 bits.

Hay dos detalles a remarcar, el primero de ellos es que el sistema tiene un comportamiento idéntico al de complementos a dos con el signo invertido. El -3 en complementos a dos es el `11111011` y en exceso 128 es `01111101`, como puede observarse es el mismo número pero con el primer bit (el de signo) invertido.

El otro detalle es que para representar un número positivo en formato sesgado basta con asignarle un valor de 1 al bit más significativo y representar el resto del número con los $m-1$ bits restantes. Por ejemplo para representar el +15 basta con "prender" el bit más significativo y representar el 15 con siete bits (`0001111`). El número es `1 0001111`.

4.2 El formato sesgado de representación de números reales

Este formato se divide en tres partes: signo, exponente y mantisa. El orden en que aparecen dentro de una palabra se encuentra en el diagrama de abajo.

```
 ----------------------------------------------
|   |                 |                          |
| s | bits exponente |   bits para la mantisa   |
|___|_________________|__________________________|

1 bit    8 bits                   23 bits
```

El primer bit (`s` en el diagrama) es el del signo de todo el número, si este tiene un valor de 0 el número es positivo y si es 1 el número es negativo.

Se tienen 8 bits para representar el exponente, el cual utiliza una representación exceso 2^{m-1}, es decir exceso 128. Como consecuencia de lo anterior el exponente puede tomar valores entre -128 y +127.

En este caso la mantisa cambia de nombre y se conoce como *significante*, y utiliza 23 bits para su representación. la diferencia entre una y otra, es que el significante asume que el primer bit (izquierda) es 1 y no se pone. La rezón de esto es que cualquier número binario normalizado siempre tiene un valor de uno en el el primer bit, por ejemplo:

$$\begin{array}{rcl} 11.0110 \times 2^0 & = & 0.110110 \times 2^{+2} \\ 1010.01 \times 2^3 & = & 0.101001 \times 2^{+7} \\ 0.00011 \times 2^2 & = & 0.11 \times 2^{-1} \end{array}$$

También se debe tomar en cuenta que se representa el valor absoluto del número, ya que el primer bit del formato representa el signo.

4.3 Ejemplo conversión formato sesgado a número real

Para poder ejemplificar lo anterior consideremos la representación del número 21.1875

Lo primero a realizar, al igual que en el formato normalizado, es convertir el número a base dos:

$$21.1875 = \texttt{10101.0011}$$

Después hay que normalizar el número binario:

$$10101.0011 \times 2^0 = 0.101010011 \times 2^{+5}$$

La mantisa tiene un valor de 101010011 pero el significante del formato sesgado asume que el primer bit siempre es 1, por lo que el significante tendrá un valor de:

```
significante = 01010011
```

Sin embargo no olvidemos que se utilizan 23 bits para el significante, por lo que el valor final es:

```
significante = 01010011000000000000000
```

Para calcular el exponente se tiene que obtener la representación de este número en exceso 128:

$$+5 = 5 + 128 = 133 = \texttt{10000101}$$

Por lo que el valor de exponente es:

```
exponente = 10000101
```

El signo del número es positivo por lo que el bit de signo (`s`) dene tener un valor de 1:

`signo = 1`

Lo último a realizar es juntar los tres campos (signo, exponente y significante en ese orden). El número es:

$$21.1875_{10} = \texttt{1 10000101 01010011000000000000000}$$

Para evitar confusiones en los bits, en algunas ocasiones se representa el número en hexadecimal:

$$21.1875_{10} = \texttt{C2A98000}_{16}$$

4.4 Ejemplo conversión número real a formato sesgado

Para ejemplificar el caso contrario, se analizará el valor en base diez del número:

$$\texttt{1 01110110 10111010000000000000000} \quad (\texttt{BB5D0000}_{16})$$

Empezemos en el orden del formato. El signo es representado por el primer bit que tiene un valor de `1`, lo que significa que el número es negativo.

El exponente esta represntado por `01110110`, que se encuentra en exceso 128, por lo que para encontrar su valor, hay que encontrar el equivalente en base 10 del número y restarle 128. Por lo que el exponente es:

$$\texttt{01110110} = 118 - 128 = -10$$

El significante es `10111010000000000000000` pero no hay que olvidar que el primer bit tiene un valor de 1 y esta implíto. Por lo que hay que incluirlo para calcular el valor del significante:

$$\begin{aligned}\texttt{1101110100...0} &= (1\times2^{-1})+(1\times2^{-2})+(0\times2^{-3})+(1\times2^{-4})+(1\times2^{-5})+(1\times2^{-6})+\\ &\quad (0\times2^{-7})+(1\times2^{-8})+(0\times2^{-9})+...+(0\times2^{-24})\\ \texttt{1101110100...0} &= 0.86328125\end{aligned}$$

Reuniendo los resultados anteriores, y tomando en cuenta que se esta trabajando en base dos, podemos decir que el número representado es el siguiente:

$$\texttt{1 01110110 10111010...0} = 0.86328125 \times 2^{-10} = 0.000843048_{10}$$

es decir:

$$\texttt{C2A98000}_{16} = 0.000843048_{10}$$

En algunas ocasiones el número de bits utilizados para representar el exponente y el significante es menor que el estándar. El principio del formato no cambia, lo que no hay que olvidar es el orden en que se encuentran, que el exponente utiliza exceso 2^{m-1} para su representación y que el significante tiene un 1 implícito al principio.

4.5 Rangos representación del formato sesgado

Ahora calcularemos hasta donde se puede representar con el formato sesgado. Al igual que en el formato normalizado se calularán cuatro números que son el límite entre un número representable y un overflow o underflow.

Para calcular el número negativo más grande necesitamos calcular el valor positivo más grande que pueda tomar el exponente y la mantisa. El valor más grande del exponente y de la mantisa es cuando todos los bits de los dos campos son 1. Como se quiere el número negativo, el bit de signo debe tener un valor de 1. Debido a lo anterior el número negativo más grande es:

$$\texttt{1 11111111 1111111...11} = -(0.5+0.25+0.125+0.0625+...+(2^{-24}))\times(2^{+127})$$
$$\texttt{1 11111111 1111111...11} = -(0.999999999\times 2^{+127})$$
$$\texttt{1 11111111 1111111...11} \simeq -1.7014118\times 10^{38}$$

En el caso del número negativo más pequeño hay que calcular el valor negativo más grande que pueda tomar el exponente y el valor más pequeño que pueda tomar la mantisa. Para el exponente esto equivale a que todos los bits tengan un valor de cero ($0-128=-128$). En el caso de la mantisa no hay que olvidar que hay un 1 implícito, por lo que el número más chico en la mantisa es cuando todos los bits tienen un valor de 0. Por último el valor del bit de signo es 1 ya que se trata de un número negativo. Entonces el número es:

$$\texttt{1 00000000 0000000...00} = -((1\times 2^{-1})+(0\times 2^{-2})...+(0\times 2^{-24}))\times(2^{-128})$$
$$\texttt{1 00000000 0000000...00} = -(1\times 2^{-1})\times 2^{-128}$$
$$\texttt{1 00000000 0000000...00} \simeq -1\times 2^{-129}$$

El número positivo más pequeño es el mismo que el negativo; lo único que cambia es el signo. Entonces lo único diferente es el bit de signo, el número es:

$$\texttt{0 00000000 0000000...00} \simeq +1\times 2^{-152}$$

Para el número positivo grande se aplica el mismo principio que para el negativo, solo cambia el signo, por lo que el número es:

$$\texttt{0 11111111 1111111...11} \simeq +1.7014118\times 10^{38}$$