Representación Computacional de Números Reales

Diego Feroldi
feroldi@fceia.unr.edu.ar

Arquitectura del Computador
Departamento de Ciencias de la Computación
FCEIA-UNR

Octubre de 2018

Índice

1. Introducción

Para representar números reales es necesario utilizar una coma o punto para separar la parte entera de la parte fraccionaria, independientemente del sistema de representación con que se trabaje (decimal, binario, etc.). Existen dos formas de resolver el problema:

- Se considera la coma o punto en cierta posición fija.
- Se almacena la posición que la coma o punto ocupa en el número (posición flotante).

2. Representación de números reales con punto fijo

Este método considera que el punto está siempre en una misma posición. En los sistemas digitales se utilizan registros para almacenar los datos y entonces se adoptan dos posiciones posibles para el punto:

1. En el extremo izquierdo con lo cual el valor almacenado en el registro representa la parte fraccionaria del número.
2. En el extremo derecho del registro con lo cual el valor almacenado representa la parte entera del número.

En ninguno de los dos casos el punto existe realmente, pero su presencia se supone porque el número almacenado en el registro se trata como una fracción o un entero.

Ejemplo:

Trabajando con 8 bits de los cuales hemos fijado y reservado 5 para la parte entera y 3 para la fraccionaria:

$$(11011.011)_2 = 1\times 2^4 + 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2} + 1\times 2^{-3} = (27.375)_{10}$$

En este ejemplo se puede observar que existe una limitación en cuanto al rango de los números que se pueden representar:

- Menor número representable: $(00000.001)_2 = 2^{-3} = (0.125)_{10}$
- Mayor número representable: $(11111.111)_2 = 2^5 - 2^{-3} = (31.875)_{10}$

En forma general:

$$2^{-m} \leq \text{Rango} \leq 2^n - 2^{-m}$$

donde n es la cantidad de bots de la parte entera y m es la cantidad de bits de la parte fraccionaria.

Ventajas: La aritmética de punto fijo es relativamente simple.

Desventajas: El rango de representación, es decir el número de cantidades a representar, es muy limitado.

3. Representación de números reales con punto flotante

La representación anterior tiene importantes limitaciones. En efecto, viendo el ejemplo anterior se deduce que números muy grandes y fracciones muy pequeñas no pueden ser representadas. En números decimales esta limitación puede superarse utilizando la notación científica que permite representar números muy grandes o muy pequeños utilizando relativamente pocos dígitos.

3.1. Notación científica normalizada

Para la representación de números reales sobre un amplio rango de valores con menos dígitos se emplea la notación científica. Por ejemplo, el número 976000000000000 se puede representar como 0.976×10^{15} mientras que el número 0.000000000000000976 se puede representar como 0.976×10^{-15}. En esta notación la coma o punto decimal se mueve dinámicamente a una posición conveniente y se utiliza el exponente en base 10 para registrar la posición del punto decimal.

Sin embargo, observemos que este tipo de notación tiene cierta ambigüedad dado que un mismo número puede ser representado de diferentes maneras. En el ejemplo anterior, el número 976000000000000 también puede escribirse como 9.76×10^{14}, etc. Por lo tanto es necesario definir una normalización.

Definición

Todo número real no nulo se puede escribir en forma única en la **notación científica normalizada** como:

$$(-1)^s 0.a_1 a_2 a_3 \ldots a_t \ldots \times 10^e$$

siendo el dígito $a_1 \neq 0$. Por lo tanto, en el ejemplo anterior la forma 0.976×10^{-15} es normalizada.

De forma general, todo número real no nulo puede representarse en forma única respecto a la base β de la siguiente forma:

$$(-1)^s 0.a_1 a_2 a_3 \ldots a_t \ldots \times \beta^e,$$

donde los "dígitos" a_i son enteros positivos tales que $1 \leq a_1 \leq \beta - 1$, $0 \leq a_i \leq \beta - 1$ para $i = 2, 3, \ldots$ y constituyen la parte fraccional o **mantisa** del número, en tanto que e es el **exponente**, el cual indica la posición del punto correspondiente a la base β.

Si m es la la fracción decimal correspondiente a $(0.a_1 a_2 a_3 \ldots)_\beta$, es decir $m = a_1 \times \beta^{-1} + a_2 \times \beta^{-2} + a_3 \times \beta^{-3} + \ldots$, entonces el número representado corresponde al número decimal $(-1)^s \times m \times \beta^e$, siendo $\beta^{-1} \leq m < 1$.

Ejemplo

En este ejemplo, un número N se representa en sistema binario (base $\beta = 2$) con un bit para el signo (s), una mantisa fraccionaria de 8 bits (m) y un exponente con signo de 6 bits (e). El punto o coma fijo de la fracción se encuentra inmediatamente después del bit de signo, pero no está representada realmente en el registro.

Número decimal	Número binario	Signo	Mantisa	Exponente
+3.3750	+11.011	1	00011011	000101

Por lo tanto:

$$N = (-1)^s \times m \times \beta^e = (-1)^0 \times (.00011011)_2 \times 2^{+5}$$

Sin embargo, este número no está normalizado. Un número con punto flotante está normalizado si el dígito más significativo de la mantisa es distinto de cero. Por lo tanto, normalizando tenemos:

$$N = (-1)^0 \times (.11011000)_2 \times 2^{+2}$$

Mantisa (normalizada)	Exponente (normalizado)
11011000	000010

Nota: Al realizar la normalización se debe tener en cuenta la corrección del exponente.

Rango de los números representables

Es importante notar que en todo dispositivo el número de dígitos posibles para representar la mantisa es finito y el exponente también varía en un rango finito:

$$L \leq e \leq U, \quad \text{con } L < 0 \text{ y } U > 0.$$

Además, la mantisa también puede representar un número acotado de valores distinto. Por lo tanto, esto implica que solo un rango finito de números puede ser representado. En el ejemplo anterior tenemos $-31 \leq e \leq 32$ y $2^{-1} \leq m \leq (1 - 2^{-8})$. Por lo tanto:

$$2^{-1}2^{-31} \leq |N| \leq (1 - 2^{-8})2^{32}.$$

Propiedades

Sea el conjunto de números de punto flotante $\mathbb{F}(\beta, t, L, U)$, donde β es la base, t es la cantidad de dígitos significativos de la mantisa, L es valor mínimo del exponente y U es el valor máximo del exponente:

1. El número de elementos del conjunto $\mathbb{F}$ es $2(\beta-1)\beta^{t-1}(U-L+1)+1$ dado que existen dos posibles elecciones del signo, $\beta - 1$ elecciones posibles para el dígito principal de la mantisa, β elecciones para cada uno de los restantes dígitos de la mantisa, $U - L + 1$ posibles valores para el exponente y un valor para representar el cero.

2. El cero no puede ser representado como punto flotante normalizado y se representa como caso particular.

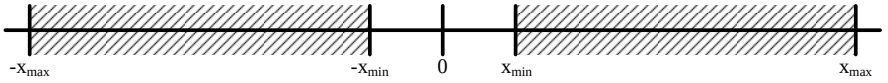
3. Si $x \in \mathbb{F}$, entonces su opuesto $-x \in \mathbb{F}$.

4. El conjunto está acotado tanto superior como inferiormente:

$$x_{min} = \beta^L \beta^{-1} \leq |x| \leq x_{max} = \beta^U (1 - \beta^{-t})$$

5. Hay cinco regiones excluidas para los números del conjunto $\mathbb{F}$:

   - Los números negativos menores que $-x_{max}$ ("overflow" negativo).
   - Los números negativos mayores que $-x_{min}$ ("underflow" negativo).
   - El cero.
   - Los números positivos menores que x_{min} ("underflow" positivo).
   - Los números positivos mayores que x_{max} ("overflow" positivo).



6. Los números en punto flotante no están igualmente espaciados sobre la recta real, si no que están más próximos cerca del origen y más espaciados a medida que nos alejamos del origen (ver Sección 6).

7. Una cantidad de gran importancia es el denominado **Epsilon de máquina**:

$$\epsilon_m = \beta^{1-t}, \tag{1}$$

   la cual representa la distancia entre el número 1 y el número en punto flotante siguiente más próximo: $1 \oplus \epsilon_m > 1$, donde el operador $\oplus$ indica una suma en el sistema de números en flotante.

8. Para evitar la proliferación de diversos sistemas de punto flotante se desarrolló la norma IEEE 754. Como se verá en la Sección 4, esta norma define los siguientes formatos ($\mathbb{F}(\beta, t, L, U)$):

   - $\mathbb{F}(2, 24, -126, 127)$ precisión simple de 32 bits (`float` de C).
   - $\mathbb{F}(2, 53, -1022, 1023)$ precisión doble de 64 bits (`double` de C).
   - $\mathbb{F}(2, 64, -16382, 16383)$ precisión extendida de 80 bits (`long double` de C).
   - $\mathbb{F}(2, 113, -16382, 16383)$ precisión extendida de 128 bits (`__float128` de C).

3.2. Redondeo de un número real

Dado que sólo los valores del conjunto $\mathbb{F}$ pueden ser representados, entonces los números reales deben ser aproximados a un valor perteneciente al conjunto, al cual denotaremos $fl(x)$. La manera usual de proceder consiste en aplicar el redondeo simétrico a t dígitos a la mantisa de representación de punto flotante (de longitud infinita) de x. Esto es, a partir de

$$x = (-1)^s 0.a_1 a_2 \ldots a_t a_{t+1} \ldots \times \beta^e.$$

Si el exponente está en el rango $L \leq e \leq U$ obtenemos $fl(x)$ como

$$fl(x) = (-1)^s 0.a_1 a_2 \ldots \tilde{a}_t \times \beta^e$$

con

$$\tilde{a}_t = \begin{cases} a_t & \text{si} \quad a_{t+1} < \beta/2 \\ a_t + 1 & \text{si} \quad a_{t+1} \geq \beta/2 \end{cases} .$$

El error que resulta se denomina error de redondeo. Todo número real x dentro del rango de los números de punto flotante puede ser representado con un error relativo $\delta_r(x)$:

$$\delta_r(x) = \frac{|x - fl(x)|}{|x|} \leq \frac{1}{2}\beta^{1-t}.$$

Aquí se observa la importancia del Epsilon de máquina, dado que recordando la expresión (1) podemos expresar el error relativo cometido por redondeo de la siguiente manera:

$$\delta_r(x) = \frac{|x - fl(x)|}{|x|} \leq \frac{1}{2}\epsilon_m.$$

Ejemplos:

Si queremos redondear utilizando 5 dígitos, $t = 5$. Por lo tanto, los errores resultan:

$$\begin{array}{lcll} x = (0.4567689010\ldots)_{10} \times 10^3 & \rightarrow & fl(x) = (0.4567\mathbf{7})_{10} \times 10^3 & \rightarrow \quad \delta_r(x) \leq 0.5 \times 10^{-4} \\ x = (0.110101010\ldots)_2 \times 2^4 & \rightarrow & fl(x) = (0.1101\mathbf{1})_2 \times 2^4 & \rightarrow \quad \delta_r(x) \leq 0.5 \times 2^{-4} \end{array}$$

3.3. Representación computacional

Para poder utilizar los números en punto flotante en sistemas de cómputo, sus valores de exponente y mantisa deben ser almacenados en registros. Las principales características de la representación de números en punto flotante son las siguientes:

- Mantisa normalizada, convención magnitud y signo, asumiendo que la misma es una fracción (el punto está implícito).
- Al exponente se le asigna un tipo de representación sesgada o polarizada. Se le suma un valor fijo o sesgo (o *bias*) para obtener un valor final que es siempre positivo aunque el valor que representa puede ser negativo (este tema se verá en detalle).
- La base del sistema con que se trabaja se asume y no se se representa.

Ejemplo:

En este ejemplo se representan números en punto flotante utilizando 24 bits con la siguiente distribución:

S: Signo

S	Exponente con sesgo	Mantisa
1 bit	7 bits	16 bits

7 bits en el exponente implica 128 valores distintos en el rango $[0, 127]$. Esto rango puede asociarse a un rango $[-63, 64]$ utilizando un sesgo de 63 para poder contar con exponentes negativos y positivos. Por lo tanto, dada la siguiente representación no normalizada:

$$\underbrace{0}_{s}\underbrace{1010100}_{exponente}\underbrace{0000000000011011}_{mantisa}$$

Signo: $s = 0$
Exponente: $E = (1010100)_2 = 84 \longrightarrow e = E - sesgo = 84 - 63 = 21$
Mantisa: $0000000000011011 \longrightarrow m = 1 \times 2^{-12} + 1 \times 2^{-13} + 1 \times 2^{-15} + 1 \times 2^{-16}$

Por lo tanto:

$$\underbrace{0}_{s}\underbrace{1010100}_{exponente}\underbrace{0000000000011011}_{mantisa} = (-1)^s \times 2^e \times m = 864$$

Para normalizar la representación se debe mover la fracción 11 lugares hacia la izquierda y por lo tanto restar 11 al exponente:

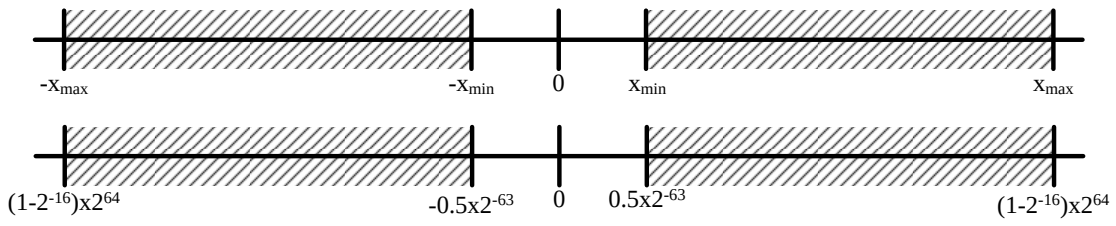
- Signo: $s = 0$
- Exponente: $e = 21 - 11 = 10 \longrightarrow$ se almacena $E = e + sesgo = 10 + 63 = 73 = (01001001)_2$
- Mantisa: $(.1101100000000000)_2 \longrightarrow m = 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-4} + 1 \times 2^{-5} = 0.84375$

Por lo tanto:

$$\underbrace{0}_{s}\underbrace{1001001}_{exponente}\underbrace{1101100000000000}_{mantisa} = 2^{10}(1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-4} + 1 \times 2^{-5}) = 864$$

Analizando el ejemplo anterior se pueden realizar las siguientes observaciones:

- Con 24 bits se pueden representar 2^{24} números distintos
- Se pueden representar los siguientes rangos:
  - Número negativos entre $-(1 - 2^{-16}) \times 2^{64}$ y -0.5×2^{-63}.
  - Números positivos entre 0.5×2^{-63} y $(1 - 2^{-16}) \times 2^{64}$



Por lo tanto, cuando una magnitud es demasiado pequeña para ser representada se produce un desbordamiento a cero mientras que cuando una operación resulta en un número con exponente mayor que 64 se produce un desbordamiento ("*overflow*").

Es importante notar que si bien con 24 bits se pueden representar 2^{24} valores distintos, estos valores están divididos en dos intervalos (uno positivo y uno negativo). Además, al contrario que en punto fijo estos números no están distribuidos en forma equiespaciada. Si se incrementa el número de bits del exponente (a expensas de la cantidad de bits en la mantisa) aumenta el rango pero decrece la precisión dado que la cantidad de números representables depende de la cantidad de bits totales. Por lo tanto, como se ampliará más adelante, existe un compromiso entre rango y precisión. Este tema se desarrollará en detalle en la Sección 6. Notar además que el cero tiene muchas representaciones posibles (mantisa nula con distintos exponentes) o ninguna representación si exigimos normalización, dado que en este caso la mantisa será siempre no nula. Por lo tanto, es necesario contar con una normalización.

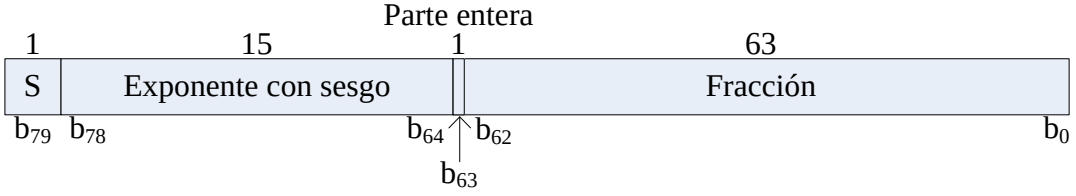
4. Standard IEEE 754 para números en punto flotante

El Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) creó un comité para formular una norma en 1985 (Standard IEEE 754) que define los siguientes formatos estándar para números en punto flotante:

1. Precisión simple con 32 bits y sesgo 127 (`float` de c):

2. Precisión doble con 64 bits y sesgo 1023 (`double` de C):

3. Precisión doble extendida con 80 bits y sesgo 16383 (`long double` de c):

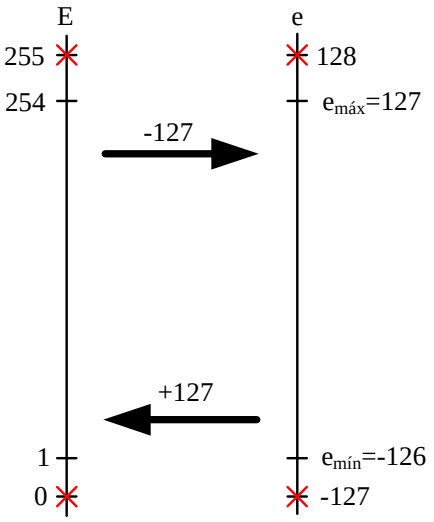


   Notar que a diferencia los formatos simple y doble precisión, en este formato no se utiliza un bit implícito. Por el contrario, el bit 63 contiene la parte entera del significante y los bits 0-62 contienen la parte fraccional. El bit 63 será uno en todos los números normalizados.

4. Precisión cuádruple con 128 bits y sesgo 16383 (`__float128` de C):

4.1. Exponente sesgado

Los valores mínimos (0) y máximo del exponente (255, 2047 y 32767, según el formato) no se utilizan para número normalizados sino que tienen usos especiales como se verá más adelante. Además, se utilizada una representación sesgada para el exponente. Es decir, al exponente (e) se le suma un sesgo o "bias" para poder representar exponentes negativos. De esta manera, el exponente codificado (E) está desplazado con respecto al verdadero exponente. La siguiente figura esquematiza el concepto de representación de números con exponente sesgado según norma IEEE 754 simple precisión (Sesgo=-127):



Por lo tanto, los valores mínimos y máximos de exponente realmente efectivos resultan:

- Precisión simple (sesgo 127):
  - $e_{min} = 1 - 127 = -126$
  - $e_{max} = 254 - 127 = 127$
- Precisión doble (sesgo 1023):
  - $e_{min} = 1 - 1023 = -1022$
  - $e_{max} = 2046 - 1023 = 1023$
- Precisión extendida y cuádruple (sesgo 16383):
  - $e_{min} = 1 - 16383 = -16382$
  - $e_{max} = 32766 - 16383 = 16383$

4.2. Significante

Como se mencionó en la sección anterior, una fracción normalizada binaria comienza siempre con un punto binario seguido por un bit igual a 1 y luego el resto de los bits de la representación. Entonces, ese bit igual a 1 no necesita ser almacenado ya que puede asumirse su presencia. Por lo tanto el estándar IEEE 754 define esta fracción de una manera distinta a la usual y para evitar confusiones la llama **significante**. Este significante consta de un 1 implícito seguido del punto y luego $t = p - 1$ bits, donde t es la cantidad de bits en el significante incluyendo el "1" implícito (Ver Figura 1). Si todos los bits son cero el significante vale 1.0, mientras que si todos son uno el valor del mismo es ligeramente inferior a 2. Notar que esta forma de normalización tampoco permite la representación del cero.

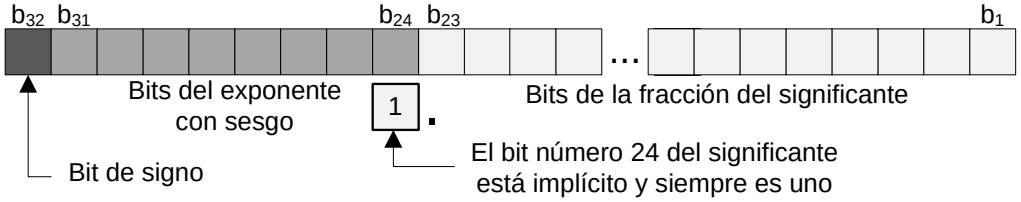


Figura 1: Formato IEEE 754 para números en punto flotante en precisión simple.

Por lo tanto, el valor de un número en formato IEEE 754 puede ser computado como

$$(-1)^s \times 2^e \times \text{significante}$$

donde:

$$s = \begin{cases} 0 & \text{si el número es positivo} \\ 1 & \text{si el número es negativo} \end{cases}$$

- El exponente se interpreta como el valor resultante de restar el sesgo correspondiente según el formato (127 en simple precisión, 1023 en precisión doble) al valor cargado en el registro: $e = E - sesgo$.

- El significante tiene un bit no visible a la izquierda con valor 1, luego un punto tampoco visible, y luego una sucesión de bits cuyos pesos son potencias negativas decrecientes de dos:

  $$\text{significante} = 1.f$$

  donde f es cualquier secuencia de bits que representa una fracción, de manera que $0 \leq f < 1$. Por lo tanto

  $$1 \leq \text{significante} < 2.$$

  En la Sección 4.6 veremos una excepción a esta regla.

4.3. Conversión de decimal a IEEE 754 simple precisión

Ejemplo 1. Se quiere convertir el número decimal 2.625 al formato IEEE 754 simple precisión. La conversión se puede realizar a través de los siguientes pasos:

1. Convertir la parte entera a binario: $(2)_{10} = (10)_2$
2. Convertir la parte fraccional a binario:

$$\begin{array}{lcll} 0.625 \times 2 & = & \mathbf{1}.25 & \longrightarrow b_{-1} = 1 \\ 0.25 \times 2 & = & \mathbf{0}.5 & \longrightarrow b_{-2} = 0 \\ 0.5 \times 2 & = & \mathbf{1}.0 & \longrightarrow b_{-3} = 1 \end{array}$$

   Por lo tanto:

$$(0.625)_{10} = (0.101)_2$$

3. Ya tenemos el número convertido a binario: $(2.625)_{10} = (10.101)_2$. Sin embargo, este número no está normalizado según la norma IEEE 754.
4. Para normalizar, primero agregar el exponente: $(10.101)_2 = (10.101)_2 \times 2^0$
5. Luego, normalizar según lo visto previamente: $(10.101)_2 \times 2^0 = (1.0101)_2 \times 2^1$
6. Por lo tanto, el significante resulta

$$(1.01010000000000000000000)_2$$

   donde el primer 1 y el punto están implícitos, es decir no ocupan bits en el registro.

7. Corregir el exponente sumando el sesgo correspondiente (sesgo=127):

$$E = e + sesgo = 1 + 127 = 128$$

8. Convertir el exponente a binario:

$$E = (128)_{10} = (10000000)_2$$

9. El número es positivo. Por lo tanto el bit de signo es $s = (0)_2$
10. Finalmente, el número 2.625 convertido a formato IEEE 754 simple precisión resulta:

$$\underbrace{0}_{signo}\ \underbrace{10000000}_{\text{exponente}}\ \underbrace{01010000000000000000000}_{\text{significante con “1.” implícito}}$$

4.4. Conversión de IEEE 754 simple precisión a decimal

Ejemplo 2. Se quiere convertir el siguiente número expresado en IEEE 754 simple precisión a decimal:

$$N = (0100\ 0001\ 0101\ 1010\ 0000\ 0000\ 0000\ 0000)_{IEEE\ 754\ simple\ prec.}$$

La conversión se puede realizar a través de los siguientes pasos:

1. Separar el número en sus diferentes partes:

Signo	Exponente	Significante (con 1. implícito)
1 bit	8 bits	23 bits
0	10000010	10110100000000000000000

2. El bit de signo ($s = 0$) es cero por lo tanto el número es positivo.

3. Convertir el exponente a decimal:

$$E = (10000010)_2 = (130)_{10}$$

4. Restar al exponente el sesgo correspondiente (sesgo=127):

$$e = E - sesgo = 130 - 127 = 3$$

5. Convertir el significante a decimal:

$$(1.f)_2 = (1.10110100000000000000000)_2 = 1+1\times2^{-1}+1\times2^{-3}+1\times2^{-4}+1\times2^{-6} = (1.703125)_{10}$$

6. Finalmente, el número convertido a decimal resulta:

$$N = (-1)^s \times (1.f)_2 \times 2^e = 1.703125 \times 2^3 = (13.625)_{10}$$

4.5. Características principales

La Tabla 1 se muestra un resumen de las características de los números en punto flotante de la norma IEEE 754 en precisión simple, doble y cuádruple. La norma IEEE 754 va más allá de la simple definición de formatos, detallando cuestiones prácticas y procedimientos para que la aritmética en punto flotante produzca resultados uniformes y predecibles independientemente de la plataforma utilizada. Con este fin, el estándar IEEE 754 agrega a los números normalizados cuatro tipos numéricos que se describen en la Tabla 2.

4.6. Números denormalizados

El menor número normalizado en simple precisión es 1.0×2^{-126}. Antes de la definición del estándar si se presentaban problemas de "underflow" debía redondearse a cero. Los números denormalizados del estándar tienen una mejor aproximación al problema. Estos números tienen un exponente con todos los bit en cero (no permitido para los números normalizados) y una fracción de 23, 52, 64 o 112 bits, según el formato, dónde al menos un

Tabla 1: Características de los números en punto flotante (Norma IEEE 754).

	Precisión Simple	Precisión Doble	Precisión Cuádruple
Bits de signo	1	1	1
Bits de exponente	8	11	15
Bits de fracción	23	52	112
Bits totales	32	64	128
Sesgo del exponente	+127	+1023	+16383
Rango del exponente	[-126, 127]	[-1022, 1023]	[-16382, 16383]
Valor más chico (normalizado)	2^{-126}	2^{-1022}	2^{-16382}
Valor más grande (normalizado)	$\approx 2^{128}$	$\approx 2^{1024}$	$\approx 2^{16382}$
Rango decimal	$[\approx 10^{-38}, \approx 10^{38}]$	$[\approx 10^{-308}, \approx 10^{308}]$	$[\approx 10^{-4932}, \approx 10^{4932}]$
Valor más chico (desnormalizado)	$2^{-126}.2^{-23} \cong 10^{-45}$	$2^{-1022}.2^{-52} \cong 10^{-324}$	$2^{-16382}.2^{-112} \cong 10^{-4966}$

Tabla 2: Tipos numéricos del standard IEEE 754

Signo	Exponente e	Fracción f	Representa	Denominación
$\pm$	$e = e_{min} - 1$ codificado como $E = (00\ldots0)_2$	$f = 0$	± 0	Ceros
$\pm$	$e = e_{min} - 1$ codificado como $E = (00\ldots0)_2$	$f \neq 0$	$\pm 0.f \times 2^{e_{min}}$	Núm. denormalizados
$\pm$	$e_{min} \leq e \leq e_{max}$	Cualquier patrón	$\pm 1.f \times 2^{e}$	Núm. normalizados
$\pm$	$e = e_{max} + 1$ codificado como $E = (11\ldots1)_2$	$f = 0$	$\pm\infty$	Infinitos
$\pm$	$e = e_{max} + 1$ codificado como $E = (11\ldots1)_2$	$f \neq 0$	NaN	*Not a Number*

bit tiene que ser no nulo. Por otra parte, el uno implícito del significante es cero para los números denormalizados. El exponente para los números denormalizados es de -126 o -1022, según corresponda.

Los números denormalizados permiten representar números más pequeños. Por ejemplo, trabajando en simple precisión el rango que cubren los números denormalizados (que se suma al rango de los números normalizados) es $[2^{-23} \times 2^{-126}, (1 - 2^{-23}) \times 2^{-126}]$. Notar que en la representación de los números denormalizados el exponente es $e = e_{min}$ y no $e = e_{min} - 1$. Es decir, el exponente del número desnormalizado es $e = -126$ para simple precisión a pesar de que en el registro realmente se almacena el número $E = (0000\ 0000)_2$ que correspondería al exponente $e = -127$. De lo contrario quedaría un "hueco" entre 2^{-127} y 2^{-126}. En la Figura 2 se observa el desbordamiento a cero gradual mediante números de punto flotante denormalizados con simple precisión.

4.7. Ceros

El estándar IEEE 754 provee una representación para el cero. La representación consiste en todos cero en el exponente y todos ceros en la fracción. El signo puede ser cero o uno, dando lugar a ± 0. Esta representación está por fuera de la normalización previamente descrita.

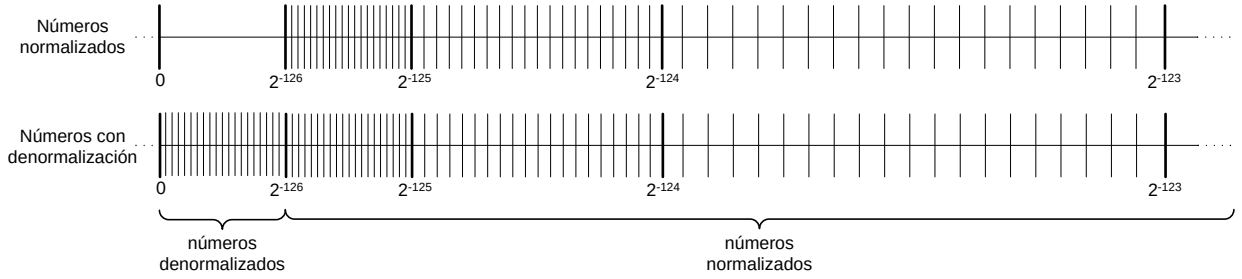


Figura 2: Números con formato de punto flotante denormalizados.

4.8. Infinitos

El estándar IEEE 754 provee una representación para $\pm\infty$. Esta representación consiste en todos unos en el exponente ($e = e_{max} + 1$) y todos ceros en la fracción. El signo puede valer cero o uno, dando lugar a $\pm\infty$. Estos números pueden ser utilizados como operador y cumplen las siguientes reglas matemáticas:

1. $N + (+\infty) = +\infty$
2. $N - (+\infty) = -\infty$
3. $N + (-\infty) = -\infty$
4. $N - (-\infty) = +\infty$
5. $(+\infty) + (+\infty) = +\infty$
6. $(-\infty) + (-\infty) = -\infty$
7. $(-\infty) - (+\infty) = -\infty$
8. $(+\infty) - (-\infty) = +\infty$

4.9. Formato NaN

El formato NaN (*Not a Number*) es una entidad simbólica utilizada en punto flotante. Sirve para representar valores de variables no inicializadas y tratamiento de tipo aritmético que no están contempladas en el estándar. La representación está compuesta por todos unos en el exponente y cualquier secuencia de bits en la fracción a excepción de la secuencia compuesta por todos ceros ($f \neq 0$). Estos números también pueden ser utilizados como operador y cumplen las siguientes reglas matemáticas:

1. $0/0 =$NaN
2. $\pm\infty/\pm\infty =$NaN
3. $0 * \pm\infty =$NaN
4. $\infty - \infty =$NaN

5. Operaciones de números en punto flotante

Para sumar o restar números en punto flotante hay que igualar los dos exponentes desplazando la mantisa. Luego, se pueden sumar o restar las mantisas. Por otra parte, para multiplicar, se multiplican las mantisas y se suman los exponentes. Análogamente, para dividir, se dividen las mantisas y se restan los exponentes. Concretamente, la secuencia de acciones que debe realizarse para cada operación es la siguiente:

5.1. Suma o resta

1. Chequear si alguno de los operandos en cero, NaN o ±INF.
2. Igualar los exponente (si es necesario), desplazando el significante.
3. Completar con ceros (si es necesario).
4. Sumar o restar los significantes.
5. Normalizar el resultado (si es necesario).

Ejemplo Restar los siguientes números en IEEE 754 simple precisión:

$$S = 123 - 4.75$$

Primero convertimos ambos números de acuerdo a la norma IEEE 754 simple precisión:

$$\begin{aligned} 123 &= 0|10000101|11101100000000000000000 \\ 4.75 &= 0|10000001|00110000000000000000000 \end{aligned}$$

Ahora tenemos que alinear los significantes de manera tal de igualar los exponentes. Se procede modificando el menor de los números para perder menos precisión. En este ejemplo modificamos el significante del número 4.75 hasta igualar el exponente del número 123, el cual es 133. Para ello debemos correr el punto 4 veces hacia la izquierda:

$$4.75 = 0|10000101|00010011000000000000000$$

Notar que este número no está normalizado dado que el significante no tiene un uno implícito sino que tiene un cero implícito. Sin embargo, ahora podemos restar los significantes:

$$\begin{array}{rr} & 1.11101100000000000000000 \\ - & 0.00010011000000000000000 \\ \hline & 1.11011001000000000000000 \end{array}$$

Este significante ya está normalizado. Por lo tanto, podemos juntar con el exponente y el resultado está expresado en IEEE 754:

$$S = 0|10000101|11011001000000000000000 = 118.25$$

5.2. Multiplicación

1. Chequear si alguno de los operandos es cero, NaN o ±INF.
2. Sumar los exponentes.
3. Multiplicar los significantes.
4. Normalizar el producto (si es necesario).

Ejemplo Multiplicar los siguientes números en IEEE 754 simple precisión:

$$P = 56 \times 12$$

Primero convertir los números a IEEE 754 simple precisión:

$$\begin{aligned} 56 &= 0|10000100|11000000000000000000000 \\ 12 &= 0|10000010|10000000000000000000000 \end{aligned}$$

Multiplicar los significantes:

$$\begin{array}{cccccccccccccccccc}
 & & & & & & & 1. & 1 & 1 & 0 & 0 & 0 & \cdots & 0 \\
 & & & & & & \times & 1. & 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\
\hline
 & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\
 & & & & & & & \vdots & & & & & & & \vdots \\
 & & 1 & 1 & 1 & 0 & 0 & 0 & 0 & \cdots & & & \cdots & 0 & 0 \\
 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots & & & \cdots & 0 & 0 \\
\hline
1 & 0. & 1 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & & & \cdots & 0 & 0
\end{array}$$

El significante resultante no está normalizado. Normalizando, resulta:

$$1.01010000000000000000000$$

Luego, sumar los exponentes y sumar 1 debido al corrimiento del punto realizado al normalizar:

$$e = e_1 + e_2 + 1 = 9$$

Por lo tanto, el exponente almacenado resulta:

$$E = e + \text{sesgo} = 136$$

Finalmente, el resultado es el siguiente:

$$P = 0|10001000|01010000000000000000000 = 672$$

5.3. División

1. Chequear si alguno de los operandos es cero, NaN o ±INF.
2. Restar los exponentes.
3. Dividir los significantes.
4. Normalizar (si es necesario).

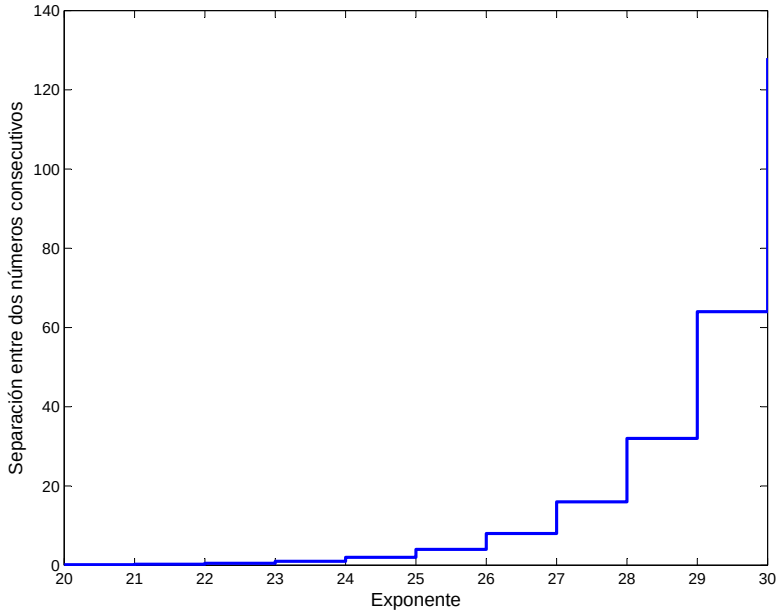


Figura 3: Evolución de la separación entre dos números consecutivos en función del exponente e en punto flotante.

6. Densidad de los números en punto flotante

Al contrario que con los números en punto fijo, en punto flotante la distribución en la recta numérica no es uniforme y a medida que nos alejamos del origen se van separando. Cómo se verá a continuación, existe un compromiso entre rango y precisión.

Supongamos que P (el número de bits del significando) sea 24. En el intervalo [1, 2) (con exponente $e = 0$) es posible representar 2^{23} números equiespaciados y separados con una distancia $1/2^{23}$. De modo análogo, en cualquier intervalo $[2^e, 2^{e+1})$ habrá 2^{23} números equiespaciados pero la densidad de este caso será $2^e/2^{23}$.

Por ejemplo, entre $2^{20} = 1048576$ y $2^{21} = 2097152$ hay $2^{23} = 8388608$ números pero el espaciado es de solo 1/8. De este modo se deriva una regla práctica que cuando es necesario comparar dos números en punto flotante relativamente grandes es preferible comparar la diferencia relativa entre esos dos números en lugar de las magnitudes absolutas de los mismos dado que la precisión es menor cuanto más grandes sean los números. En la Figura 3 se puede apreciar cómo aumenta la separación entre dos números consecutivos en función del exponente e en el rango $e = [20, 30]$.

7. Precauciones al operar con números en punto flotante

La alineación o ajuste se consigue desplazando a la derecha el número más pequeño (incrementando su exponente) o desplazando a la izquierda el más grande (decrementando su exponente). Dado que cualquiera de estas operaciones puede ocasionar que se pierdan

dígitos, conviene desplazar el número más pequeño ya que los dígitos que se pierden tienen una importancia relativa menor.
Por ejemplo, supongamos que queremos sumar $S = 123 + 2.000001$ en IEEE 754 simple precisión. Convirtiendo ambos números resulta:

$$\begin{aligned} 123 &= 0|10000101|11101100000000000000000 \qquad (e = 6) \\ 2.000001 &= 0|10000000|00000000000000000000100 \qquad (e = 1) \end{aligned}$$

Desplazando el significante del número menor hasta igualar ambos exponentes (cinco veces hacia la derecha), el significante del número 2.000001 resulta $0.000010\cdots0$. Por lo tanto, la suma de ambos significantes resulta:

$$\begin{array}{cccccccccccc} & 1. & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & \cdots & 0 \\ + & 0. & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 \\ \hline & 1. & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & \cdots & 0 \end{array}$$

Juntando con el exponente resulta:

$$S = 0|10000101|11110100000000000000000 = 125$$

El resultado obtenido es incorrecto y el error cometido es 1×10^{-6}.
Como resultado de lo anterior se pueden establecer los siguientes enunciados:

1. Se tiene más precisión si se opera con número de magnitud similares.
2. El orden de evaluación puede afectar la precisión
3. Cuando se restan dos números del mismo signo o se suman dos números con signo opuesto, la precisión del resultado puede ser menor que la precisión disponible en el formato.
4. Cuando se realiza una cadena de cálculos tratar de realizar primero los productos y cocientes: $x \otimes (y \oplus z) \neq x \otimes y \oplus x \otimes z$
5. Cuando se multiplica y divide un conjunto de números, tratar de juntarlos de manera que se multipliquen números grandes y pequeños y por otro lado se dividan números de magnitud similares.
6. Cuando se comparan dos números en punto flotante, siempre comparar con respecto a un valor de tolerancia pequeño. Por ejemplo, en lugar de comparar `x=y`, realizar la evaluación `IF ABS((x-y)/y)) <= tol`. De todas maneras, además hay que tener precaución con los casos límites: $x = y = 0$, $y = 0$.

Apéndices

A. Cálculo del epsilon de máquina

Con el siguiente código en C se puede determinar el epsilon de máquina:

```
#include <stdio.h>
int main() {
double x = 1.0;
int n = 0;
while (1.0 + (x * 0.5) > 1.0) {
++n;
x *= 0.5;
}
printf("Epsilon de la máquina en forma binaria = 2^(-%d)\n", n);
printf("Epsilon de la máquina en forma decimal = %G\n", x);
return 0;
}
```

Referencias

[1] Mano, M.M., *Computer system architecture*, Prentice-Hall, 1993.

[2] Hyde, R., *The art of assembly language*, No Starch Pr, 2003.

[3] Goldberg, D, *What every computer scientist should know about floating-point arithmetic*, ACM Computing Surveys (CSUR), **(23)**, 5-48, 1991.

[4] IEEE Computer Society, *IEEE Standard for Floating-Point Arithmetic*, 2008.

[5] Carter, P., *PC Assembly Language*, 2006.