Representación Computacional de Números Reales

Diego Feroldi
feroldi@fceia.unr.edu.ar

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

*Actualizado 26 de septiembre de 2022 (D. Feroldi, feroldi@fceia.unr.edu.ar)

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 ocupa la coma o punto 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:

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 o coma existe en realidad, pero su presencia se supone porque el número almacenado se trata como una fracción o un entero.

Ejemplo

Veamos un sistema simple de representación 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:* $(11111.111)_2 = 2^5 - 2^{-3} = (31.875)_{10}$

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 y 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 0.00000000000000976 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 1. Todo número real no nulo se puede escribir en forma única en la **notación científica normalizada** como:

$$N = \pm(0.a_1a_2a_3 \ldots a_t \ldots) \times 10^e$$

siendo el dígito $a_1 \neq 0$ y e es el exponente.

Por lo tanto, en el ejemplo anterior la forma 0.976×10^{-15} es normalizada.

Definición 2. De forma general, todo número real no nulo puede representarse en forma única respecto a la base β de la siguiente forma:

$$N = (-1)^s \times (0.a_1a_2a_3 \ldots a_t \ldots)_\beta \times \beta^e,$$

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. Finalmente, s es el **signo** con la convención $s = 0$ si N es positivo y $s = -1$ si es negativo.

Si m es la la fracción decimal correspondiente a $(0.a_1a_2a_3\ldots)_\beta$, 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

Dado el número en formato decimal $N = (3.375)_{10}$ *se puede representar en binario como* $(11.011)_2$. *Normalizando:*

$$N = 3.375 = (-1)^0 \times (0.11011)_2 \times 2^2$$

Observación

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

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

$$L \leq e \leq U,$$

con $L < 0$ y $U > 0$. Por lo tanto, esto implica que solo un rango finito puede representado.

Sea el conjunto $\mathbb{F}(\beta, t, L, U)$ de números representables en punto flotante, donde t es la cantidad de dígitos significativos de la mantisa (número de bits):

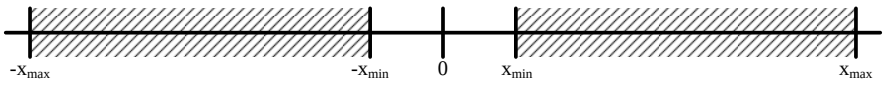
1. El número de elementos del conjunto $\mathbb{F}$ es $2(\beta - 1)\beta^{t-1}(U - L + 1)$.
2. El cero no puede ser representado como punto flotante debido a la normalización.
3. Si $x \in \mathbb{F}$, entonces su opuesto $-x \in \mathbb{F}$.

4. El conjunto está acotado tanto superior como inferiormente:

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

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 5.4).

7. Una cantidad de gran importancia es el denominado **Epsilon de máquina** $\epsilon_m = \beta^{1-t}$, el cual representa la distancia entre el número 1 y el número en punto flotante siguiente más próximo. Es decir, ϵ_m es el menor número tal que $1 + \epsilon_m > 1$.

8. Para evitar la proliferación de diversos sistemas de punto flotante se desarrolló la norma IEEE 754. Como se verá en la Sección 4.1, esta norma define tres formatos:

   - Precisión simple: $\mathbb{F}(2, 24, -127, 128)$.
   - Precisión doble: $\mathbb{F}(2, 53, -1023, 1024)$.
   - Precisión cuádruple: $\mathbb{F}(2, 113, -16383, 16384)$.

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

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

obtenemos $fl(x)$ como

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

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 que no exceda una cota de error de redondeo $\mathbf{u}$:

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

Ejemplos

Supongamos que queremos redondear con 5 dígitos significativos:

$$\begin{array}{lcll} N_1 = (0.4567689010\ldots)_{10} & \rightarrow & (0.45677)_{10} & \rightarrow \quad \mathbf{u} = 0.5 \times 10^{-5} \\ N_2 = (0.110101010\ldots)_2 & \rightarrow & (0.11011)_2 & \rightarrow \quad \mathbf{u} = 0.15625 \times 10^{-1} \end{array}$$

4. Estándar 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) para números en punto flotante. Un número en formato IEEE 754 puede ser expresado como

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

donde:

- s indica el signo del número con la siguiente convención:

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

- e es el exponente del número. Este es el valor resultante de restar el sesgo correspondiente al valor del exponente que está efectivamente codificado (exponente sesgado). Es decir, $e = E - sesgo$. El valor E es siempre positivo y de esta manera el exponente e puede adoptar tanto valores positivos como negativos.

- 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:

$$\text{significante} = (1.b_{n-1} \cdots b_0)_2 = 1 + b_{n-1} \times 2^{-1} + b_0 \times 2^{-n}$$

  Por lo tanto, el significante es mayor o igual que 1.0 y menor que 2. En la Sección 4.7 veremos una excepción a esta regla.

4.1. Formatos

La representación a nivel bits de un número de punto flotante formato IEEE 754 se divide en tres campos de bits:

- El bit más a la izquierda codifica directamente el signo s con la convención expresada en (1).

- Los siguientes k bits codifican el exponente sesgado E.

- Los últimos n bits codifican la mantisa m (parte fraccionaria del significante).

La norma IEEE 754 define 3 formatos estándar para números en punto flotante:

1. **Precisión simple**

   La precisión simple tiene 32 bits con la siguiente distribución de bits y sesgo 127:

2. **Precisión doble**

   La precisión simple tiene 64 bits con la siguiente distribución de bits y sesgo 1023:

3. **Precisión cuádruple**

   La precisión simple tiene 128 bits con la siguiente distribución de bits y sesgo 16383:

4.2. Exponente

Los valores mínimos (0) y máximo del exponente (255, 2047 y 32767, según el formato) no se utilizan para números normalizados sino que tienen usos especiales como se verá más adelante. 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 cuádruple (sesgo 16383):
  - $e_{min} = 1 - 16383 = -16382$
  - $e_{max} = 32766 - 16383 = 16383$

4.3. 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 23, 52 o 112 bits (Ver Figura 1). Si todos los bits son cero el significante será 1.0, mientras que si todos son uno resultará un valor ligeramente inferior a 2. Notar que esto en principio deja sin representación al cero. Sin embargo, en la Sección 4.7 veremos que el número cero se puede representar como caso especial.

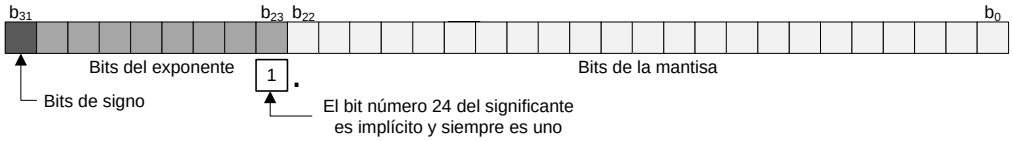


Figura 1: Formato IEEE 754 para números en punto flotante en precisión simple.

4.4. Conversión de decimal a IEEE 754

A continuación mostramos el procedimiento de conversión de decimal a IEEE 754 mediante un ejemplo en simple precisión. Para otras precisiones el procedimiento es análogo con los cambios necesarios en el sesgo y el número de bits.

Ejemplo

Se quiere convertir el número decimal $N = (2.625)_{10}$ *a IEEE 754 simple precisión. El procedimiento consiste en los siguientes pasos:*

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

2. *Convertir la parte fraccional a binario:*

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

   Por lo tanto:

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

3. *Ya tenemos el número convertido a binario:*

$$N = (2.625)_{10} = (10.101)_2$$

4. *Agregar exponente:* $N = (10.101)_2 = (10.101)_2 \times 2^0$

5. *Normalizar según lo visto previamente:*

$$N = (10.101)_2 \times 2^0 = (1.0101)_2 \times 2^1$$

6. *Por lo tanto, el significante resulta* $(1.01010000000000000000000)_2$ *donde el primer* 1 *y el punto están implícitos, es decir no se almacenan.*

7. *Corregir el exponente sumando el sesgo correspondiente (sesgo* $= 127$*):* $E = 1 + 127 = 128$

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* $N = (2.625)_{10}$ *convertido a IEEE 754 simple precisión resulta:*

$$\underbrace{0}_{S}\ \underbrace{10000000}_{exponente}\ \underbrace{01010000000000000000000}_{significante}$$

A esta secuencia de bits la podemos expresar como $(0|10000000|01010000000000000000000)_2$*, utilizando el símbolo* | *como separador para los tres campos del números.*

4.5. Conversión de IEEE 754 a decimal

A continuación mostramos el procedimiento de conversión de IEEE 754 a decimal mediante un ejemplo en simple precisión. Para otras precisiones el procedimiento es análogo con los cambios necesarios en el sesgo y el número de bits.

Ejemplo

Se quiere convertir el número expresado en IEEE 754 simple precisión $N = (0|10000010|10110100000000000000000)_2$ *a decimal. El procedimiento consiste en los siguientes pasos:*

1. *Separar el número en sus diferentes partes:*

Signo	*Exponente*	*Significante*
1 bit	*8 bits*	*23 bits*
0	10000010	10110100000000000000000

2. *El bit de signo 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* $= 127$*):*

$$e = 130 - 127 = 3$$

5. *Convertir el significante a decimal:*

$$(1.10110100000000000000000)_2 = 1 + 1 \times 2^{-1} + 1 \times 2^{-3} + 1 \times 2^{-4} + 1 \times 2^{-6} = (1.703125)_{10}$$

6. *Finalmente, el número convertido a decimal resulta:*

$$N = (-1)^0 \times 1.703125 \times 2^3 = (13.625)_{10}$$

4.6. Características principales

En 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 extendida. 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.

4.7. 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

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

redondearse a cero. Los números denormalizados del estándar tienen una mejor aproximación al problema. Estos números tienen un exponente E igual a cero (no permitido para los números normalizados) y una fracción de 23, 52 o 112 bits, según la precisión, distinta de cero. El "1" implícito del significante es cero para los números denormalizados. Por lo tanto, el significante de los números denormalizados es mayor a cero y menor a uno. De esta manera se pueden representar números más pequeños que los números normalizados. Por ejemplo, trabajando en simple precisión el rango que cubren los números desnormalizados (que se suma al rango de los números normalizados) es $[2^{-23} \times 2^{-126}, (1 - 2^{-23}) \times 2^{-126}]$. En la Figura 2 se observa el desbordamiento a cero gradual ("*gradual underflow*") introduciendo los números de punto flotante desnormalizados. Un caso especial dentro del conjunto de los números denormalizados es el número cero, el cual tiene un patrón de bits de todo ceros: el bit de signo es 0, el campo de exponente es todo ceros (lo que indica un valor desnormalizado) y el campo de fracción es todo ceros. Curiosamente, cuando el bit de signo es 1, pero los demás campos son todos ceros, obtenemos el valor -0.0. Con el formato de punto flotante IEEE, los valores -0.0 y $+0.0$ se consideran diferentes en algunos aspectos e iguales en otros.

Observación

Notar que en la representación de los números desnormalizados el exponente es $e = e_{min}$ y no $e = e_{min} - 1$. Es decir, el exponente del número representado es $e = -126$ para simple precisión a pesar de que en el registro realmente se almacena el exponente sesgado $E = (0000\ 0000)_2$ que corres-

pondería al exponente $e = -127$. *De lo contrario quedaría un "hueco" entre* 2^{-127} *y* 2^{-126}.

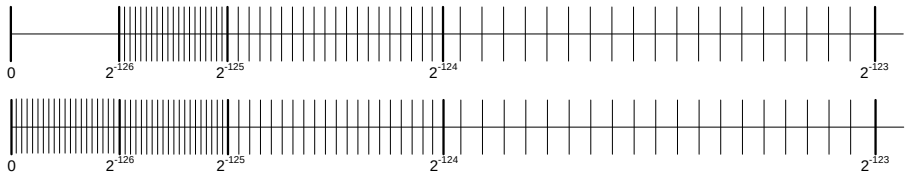


Figura 2: Desbordamiento a cero gradual mediante números de punto flotante denormalizados (Ejemplo con simple precisión). El rango entre 0 y 2^{-126} se divide en 2^{23} valores.

4.8. Infinitos

El estándar IEEE 754 provee una representación para los números infinitos ($\pm\infty$). Estos números pueden ser utilizados como operador.

Ejemplos

A continuación se enumeran algunas operaciones que dan como resultado un valor de tipo infinito:

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.

Ejemplos

A continuación se enumeran algunas operaciones que dan como resultado un valor de tipo NaN:

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

5. Operaciones con números en punto flotante

5.1. Suma/resta

Para sumar o restar números en punto flotante hay que igualar los dos exponentes desplazando el significante. Luego, se pueden sumar o restar los significantes. Concretamente, la secuencia de acciones que debe realizarse para sumar o restar es la siguiente:

1. Verificar `NaN`: Si alguno de los operandos es `NaN`, el resultados es `NaN`.
2. Verificar `INF-INF`: En este caso el resultado es `NaN`.
3. Verificar `+INF`: Si alguno de los operandos es `+INF`, el resultado es `+INF`.
4. Verificar `-INF`: Si alguno de los operandos es `-INF`, el resultado es `-INF`.
5. Chequear por ceros: Si alguno de los operandos es 0, el resultado es el otro operando ($X + 0 = X$).
6. Verificar diferencia de exponentes: Si la diferencia entre los exponentes de los argumentos es mayor a los bits de la mantisa, el resultado es

el mayor argumento ya que el argumento menor no logra afectar el resultado.

7. Igualar exponentes (si es necesario): Para poder realizar la suma/resta los exponentes deben ser iguales. Se debe desplazar hacia la derecha el punto del significante con menor exponente la cantidad de bits necesarios hasta igualar los exponentes.

8. Sumar o restar los significantes

9. Normalizar el resultado (si es necesario): El resultado se debe volver a normalizar teniendo en cuenta que que el significante debe tener la forma $1.b_{-1}\cdots$. En caso de tener que ajustar el punto, hay que ajustar el exponente del resultado.

Ejemplo

Supongamos que queremos hacer la suma $S = 123 + 456 = 579$. *El número* 123 *se puede expresar como* $(-1)^0 \times 2^6 \times 1.921875$ *que a su vez equivale a la secuencia de bits en formato IEEE 754* $(0|10000101|11101100000000000000000)_2$. *Por otra parte, el número 456 se puede expresar como* $(-1)^0 \times 2^8 \times 1.78125$ *que equivale a la secuencia de bits* $(0|10000111|11001000000000000000000)_2$.

Como ambos números tienen diferentes exponentes lo primero que hay que hacer es igualar los exponentes (el menor hacia el mayor) ajustando al mismo tiempo los significantes. En este ejemplo tenemos que correr la coma dos lugares hacia a izquierda. Luego podemos hacer la suma de los significantes:

$$\begin{array}{rr} & 1.11001000000000000000000 \\ + & 0.01111011000000000000000 \\ \hline & 10.01000011000000000000000 \end{array}$$

Una vez realizada la suma debemos ajustar la coma y el exponente para que el significante quede normalizado:

$$Significante = (1.00100001100000000000000)_2$$

y el exponente de la suma resulta $e = 9$. *Por lo tanto, el resultado es:*

$$S = (-1)^0 \times 2^9 \times 1.130859375 = 579$$

que equivale a la secuencia de bits $(0|10001000|00100001100000000000000)_2$. *Este resultado también lo podemos expresar usando notación hexadecimal para que quede más compacto:* `0x4410c000`.

5.2. Multiplicación

Para multiplicar, se multiplican los significantes y se suman los exponentes. Concretamente, la secuencia de acciones que debe realizarse para multiplicar es la siguiente:

1. Verificar `NaN`: si algunos de los operandos en `NaN`, el resultado es `NaN`.
2. Verificar `0×INF`: si un operando es 0 y el otro es `INF`, el resultado es `NaN`.
3. Verificar por ceros: si alguno de los operando es cero, el resultado es cero.
4. Sumar los exponentes.
5. Multiplicar los significantes.
6. Normalizar el producto (si es necesario).

Ejemplo

Supongamos que queremos hacer la multiplicación $M = 120 \times 12 = 1440$ *en norma IEEE 754. Primero convertimos ambos números:*

$$\begin{aligned} 120 = \quad 2^6 \times 1.875 \quad &= (0|10000101|11100000000000000000000)_2 \\ 12 = \quad 2^3 \times 1.5 \quad &= (0|10000010|10000000000000000000000)_2 \end{aligned}$$

Luego podemos multiplicar los significantes y sumar los exponentes:

$$M = 2^9 \times 2.8125 \tag{2}$$

Como el significante en (2) no está normalizado, es necesario normalizarlo aumentando en uno el exponente y corriendo un lugar el punto del significante:

$$(2.8125)_{10} = (10.11010000000000000000000)_2$$

Notar que correr el punto binario un lugar hacia la izquierda equivale a dividir por dos su equivalente en decimal. Por lo tanto, el resultado de la multiplicación es:

$$M = 2^{10} \times 1.40625 = 1440$$

que equivale a la secuencia de bits $(0|10001001|01101000000000000000000)_2$ *o a* `0x44b40000` *en notación hexadecimal.*

5.3. División

Para dividir, se dividen las mantisas y se restan los exponentes. Concretamente, la secuencia de acciones que debe realizarse para dividir es la siguiente:

1. Verificar `NaN`: Si algunos de los operandos en `NaN`, el resultado es `NaN`.
2. Verificar `0/0`: Si ambos operandos son ceros, el resultado es `NaN`.
3. Verificar `INF/INF`: Si ambos operandos son `INF`, el resultado es `NaN`.
4. Verificar `INF/X`: Si el operando dividendo es `INF`, el resultado es `INF`.
5. Verificar `X/INF`: Si el operando divisor es `INF`, el resultado es 0.
6. Chequear por ceros: Si el argumento divisor es `0`, el resultados es `INF` con el signo del dividendo. Si argumento dividendo es 0, el resultado es 0 ($0/X = 0$, con $X \neq 0$).
7. Restar los exponentes.
8. Dividir los significantes.
9. Normalizar (si es necesario).

Ejemplo

Supongamos que queremos hacer la división $D = N_1/N_2 = 96/15 = 6.4$. *Primero convertimos ambos números:*

$$\begin{aligned} 96 = \quad & 2^6 \times 1.5 \quad && = (0|10000101|10000000000000000000000)_2 \\ 15 = \quad & 2^3 \times 1.875 \quad && = (0|10000010|11100000000000000000000)_2 \end{aligned}$$

Luego podemos dividir los significantes y restar los exponentes:

$$D = 2^3 \times 0.8$$

Como el significante no está normalizado, es necesario normalizarlo disminuyendo en uno el exponente y corriendo un lugar el punto del significante. Notar que correr el punto binario un lugar hacia la derecha equivale a multiplicar por dos su equivalente en decimal. Por lo tanto, el resultado de la división es:

$$M = 2^2 \times 1.6 = 6.4$$

En realidad, el número 6.4 *no tiene representación exacta y el número más cercano es* 6.400000095367431640625 *que equivale a la secuencia de bits* $(0|10000001|10011001100110011001101)_2$ *o a* `0x40ccccd` *en notación hexadecimal.*

5.4. 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 de la mantisa) sea 23. 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 0.125. 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 $e = [20, 30]$.

5.5. 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.

Ejemplo

Queremos sumar $S = N_1 + N_2$ *con* $N_1 = 1230.015625$ *y* $N_2 = 4.56$ *en formato IEEE 754 simple precisión. Convirtiendo los números* N_1 *y* N_2

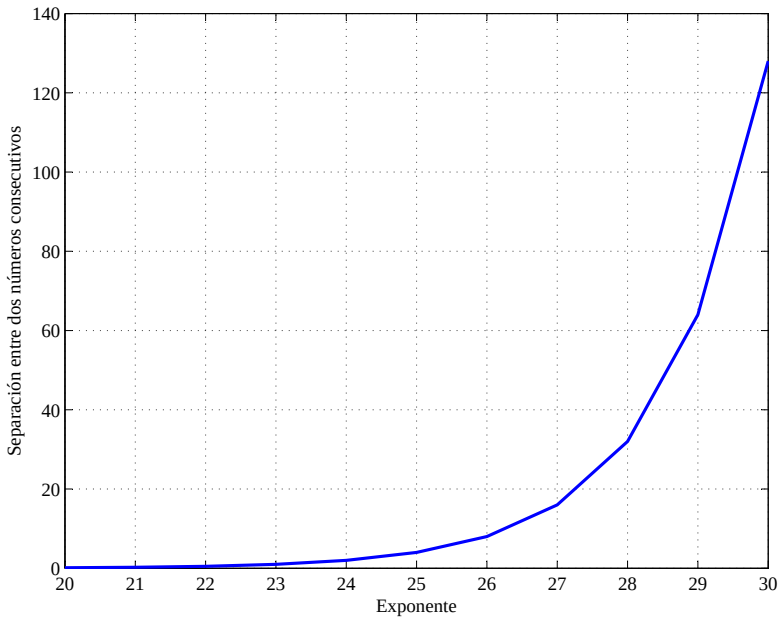


Figura 3: Evolución de la separación entre dos números consecutivos en función del exponente e en punto flotante.

obtenemos:

$$\begin{aligned} N_1 = &\quad (0|10001001|00110011100000010000000)_2 \\ N_2 = &\quad (0|10000001|00100011110101110000101)_2 \end{aligned}$$

El exponente del primer número es $e_1 = 10$ *mientras que el exponente del segundo número es* $e_2 = 2$*. Por lo tanto, tenemos que ajustar los exponentes corriendo el punto del segundo número 8 posiciones. El problema es que en este caso perdemos los 8 bits menos significativos del significante de* N_2*. Como conclusión, el error cometido es* $2^2 \times (2^{-16} + 2^{-21} + 2^{-23}) = 0.6342 \times 10^{-4}$*. Por el contrario, si hubiéramos ajustado el exponente de* N_1 *hacia* N_2 *el error cometido hubiera sido* $2^{10} \times 2^{-16} = 0.1563 \times 10^{-1}$*.*

Como resultado de lo anterior se pueden establecer los siguientes enunciados:

1. Se tiene más precisión si se suman números de magnitud similares.

2. Al restar dos números muy próximos el resultado que se obtendrá puede ser todo error de redondeo.

3. El orden de evaluación puede afectar la precisión: $A + (B + C)$ puede ser distinto a $(A + B) + C$.

4. 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.

5. Cuando se realiza una cadena de cálculos tratar de realizar primero los productos y cocientes: $x \times (y + z) \neq x \times y + x \times z$

6. 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.

7. 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

Código

```
#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 máquina en binario = 2^(-\%d)\n", n);
```

```
    printf("Epsilon de máquina en 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.