NÚMEROS REALES (PUNTO FLOTANTE)

Son la representación de cualquier número en la recta numérica o también conocidos como números en punto flotante.

Web ONLINE
http://www.zator.com/Cpp/E2_2_4a1.htm
http://www.h-schmidt.net/FloatConverter/IEEE754.html
http://www.ajdesigner.com/fl_ieee_754_word/ieee_32_bit_word.php
http://weitz.de/ieee/

Para ANDROID
https://apkpure.com/es/ieee-754-decimal-calculator/de.FrankM.Tools.ieee754Dez

Para PC
https://sourceforge.net/projects/ieeecalc/

Se basan en la representación en notación científica comúnmente utilizada en matemáticas en la que una cantidad se representa de la forma:

$$N = m * b \text{ ^}c$$

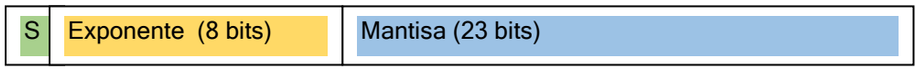
N = número expresado en punto flotante.
m = mantisa (es la fracción del signo).
b = base del exponte o raíz (Que es **2** ^ Exponente).
c = exponente (con característica de ser + ó -)

REPRESENTACIÓN EN LA MEMORIA DE UN PC

Estándar internacional de los ingenieros Eléctricos y Electrónicos (IEEE) Norma 754, define la representación en punto flotante de un número se dos formas:

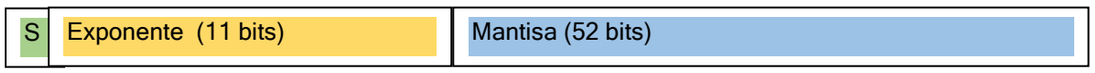
Precisión Sencilla (4 Bytes o 32 bits)

bit 31 30 ……..…**23 22** ……**0**



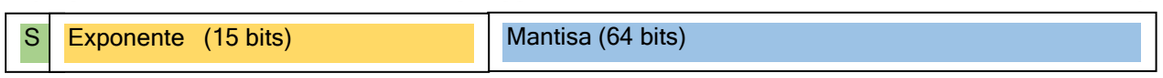
Doble Precisión (8 Bytes ó 64 bits)

bit 63 62 ……**52 51** ……..…**0**



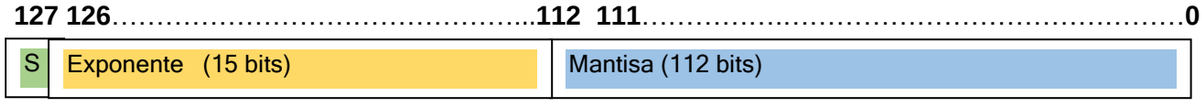
Extra Precisión (10 Bytes ú 80 bits) temporal y algunos procesadores particulares

bit 79 78 ……..**64 63** ……**0**



Quadruple Precisión (16 Bytes ú 128 bits)

bit 127 126……...**112 111**……**0**



https://es.wikipedia.org/wiki/Coma_flotante

Representación normalizada de un número

Consiste en que la mantisa no tiene parte entera y la primera cifra o digito a la derecha de la coma es distinto de cero, salvo en la representación del número cero.

Ejemplo:

Base 10 734.5 * 10^0 = 73.45 * 10^1 = 7.345*** 10^2** = 0.7345 * 10^3

Base 2 123.4 * 2^0
61.7 * 2^1
30.85 * 2^2
15.425 * 2^3
7.7125 * 2^4
3.85625 * 2^5
1.928125 * 2^6 Normalizado PC
0.9640625 * 2^7 Normalizado Matemáticas

REPRESENTACIÓN DE UN NÚMERO EN EL ESTÁNDAR IEE 754

Para la representación de un número en punto flotante siguiendo el estándar, se debe realizar los siguientes pasos:

1. **Normalizar el número entero.**
2. **Polarizar el exponente.**
3. **Convertir la mantisa a binario.**
4. **Representarlo según la precisión (Sencilla o Doble).**

PASO 1 Normalización: Es ajustar un número a 1 pero menor que 2 y mayor que cero.

16.5 * 2^0 = 8,25 * 2^1 = 4,125 * 2^2 = 2,0625 * 2^3 = **1,03125 * 2^4**
21.6 * 2^0 = 10,8 * 2^1 = 5,4* 2^2 = 2,7 * 2^3 = **1,35 * 2^4**

PASO 2: Consiste en definir la forma de representar el número, si es con precisión sencilla se trabaja con 127 y si es con doble precisión se trabaja con 1023. A este valor definido por la precisión se le suma el exponente logrado en la normalización.

Siguiendo en ejemplo anterior tenemos que:

			Nuevo exponente
16.5 = 1.03125 * 2^4	Exponente es 4	Con precisión sencilla seria	127 + 4 = 131
		Con doble precisión sería	1023 + 4 = 1027
		Con extra precisión sería	16383 + 4 = 16387

PASO 3: Convertir el número decimal a su respectivo valor binario.

Siguiendo en ejemplo anterior tenemos que: 1.03125

0.03125 * 2 = 0,0625
0,0625 * 2 = 0,125
0,125 * 2 = 0.25
0,25 * 2 = 0,5
0.5 * 2 = 1

Por lo tanto 0.03125 es igual a 0.00001

PASO 4: Representación del numero con los valores calculados en los pasos 1 y 2

Siguiendo el ejemplo anterior y trabajando con precisión sencilla tenemos que:

Paso 1 16.5 = 1.03125 * 2^4 Exponente es 4
Paso 2 Con precisión sencilla seria 127 + 4 = 131 = 10000011
Paso 3 Representación del punto decimal 0.3125 = 0.00001
Paso 4

Signo	***Exponente***	***Mantisa***
0	1 0 0 0 0 0 1 1	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 100 0001 **1000** 0100 **0000** 0000 **0000** 0000
4 1 8 4 0 0 0 0
Número: 41840000 → En secuencia inversa en memoria sería 00008441

NOTA DE LA MEMORIA

Little endian y Big endian

Little endian y big endian se refieren al orden que las máquinas asignan a los bytes que representan números o valores numéricos. Una máquina little endian asigna los bytes menos significativos en el extremo más bajo de la memoria, mientras que una máquina big endian asigna los bytes menos significativos en el extremo más alto. En los computadores, cada byte se identifica con su posición en la memoria (dirección). Cuando se manejan números de más de un byte, estos bytes también deben estar ordenados de menor a mayor, indicando la posición del byte menos significativo y del byte más significativo. De este modo, un byte con el número decimal 27 se almacenaría en una máquina little endian igual que en una máquina big endian, ya que sólo ocupa un byte. Sin embargo, para números más grandes los bytes que los representan se almacenarían en distinto orden en cada arquitectura. Este aspecto es particularmente importante en la programación en lenguaje ensamblador o en código máquina, ya que algunas máquinas consideran el byte situado en la dirección más baja de la memoria el menos significativo (arquitectura little endian, como los procesadores Intel) mientras que otras consideran que ése es el byte más significativo (arquitectura big endian, como los procesadores Motorola).

Por ejemplo, consideremos el número hexadecimal entero AABBCCDD, de 32 bits (4 bytes), localizado en la dirección 100 de la memoria. El número ocuparía las posiciones desde la 100 a la 103, pero dependiendo de si la máquina es little o big endian, los bytes se almacenarían de diferente manera:

Little-endian (como Intel)

	100	101	102	103	
...	DD	CC	BB	AA	...

Big-endian (como Motorola)

	100	101	102	103	
...	AA	BB	CC	DD	...

En las imágenes de arriba, en donde se representan las posiciones de memoria 100, 101, 102 y 103 creciendo de izquierda a derecha, «parece» que la representación big endian es más natural, ya que el número AABBCCDD lo podemos leer correctamente (ver figura), mientras que en la representación little endian parece que el número está al revés, o «patas arriba». Sin embargo, no hay nada que nos impida imaginar que las direcciones de memoria «crecen» de derecha a izquierda, y al observar la memoria de esta manera, la representación little endian «se ve natural» y es la big endian la que «parece» al revés, como se muestra en las figuras de abajo.

Little-endian (como Intel)

	103	102	101	100	
...	AA	BB	CC	DD	...

Big-endian (como Motorola)

	103	102	101	100	
...	DD	CC	BB	AA	...

Independiente de si la máquina es de arquitectura little endian o big endian, los bits dentro de cada byte siempre están en el mismo orden, con el bit más significativo a la izquierda y el menos significativo a la derecha. Los registros del procesador, que pueden ser de 4 a 64 bits, y más, también tienen sus bits en el mismo orden en ambos tipos de máquina. La diferencia entre little y big endian solo existe externamente, en el orden en que los bytes se representan en memoria.

- *Secuencia inversa de la memoria por PALABRA s**e maneja en los microprocesadores INTEL y AMD y se conoce como little endian.***
- ***En los procesadores Motorota NO se invierte la memoria y se conoce como big endian.***

EJERCICIOS ()

14.5 = 4168 0000 - 0.00015 = B91D 4951

-0.25 = BE8 00000 - 0.01 = BC23 D70A
21.6 = 41AC CCCD - 480 = C3F0 0000
-12 = C140 0000 6.5 = 40D0 0000

CASOS PARTICULARES

1. NÚMEROS NEGATIVOS

PASO 1 Normalización **-5.25** * 2^0 = 2.625 * 2^1 = **1,3125 * 2^2**

PASO 2: Polarización. **-5.25** = 1.3125 * 2^2 Precisión sencilla 127 + 2 = 129

PASO 3: Convertir decimal.

0.3125 * 2 = 0,625
0,625 * 2 = 1,25
0,25 * 2 = 0,5
0.5 * 2 = 1
Por lo tanto 0.3125 es igual a 0.0101

PASO 4: Representación.

Signo	***Exponente***	***Mantisa***
1	1 0 0 0 0 0 0 1	0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

1 1 0 0 0 0 0 0 **1 0 1 0** 1 0 0 0 **0 0 0 0** 0 0 0 0 **0 0 0 0** 0 0 0 0
C 0 A 8 0 0 0 0
Número: C0A80000 → En secuencia inversa en memoria sería 0000A8C0

2. NÚMERO YA EXPRESADO EN LA COMO 1

Se ahorra el primer paso, los demás pasos se realizan de forma normal.

3. NÚMERO MENOR QUE 0

PASO 1 Normalización: Caso especial, toca llevarlo a uno (al contrario) y no bajarlo a 1.
0.25 * 2^0 = 0.5 * 2^ (- 1) = **1,0 * 2^(- 2)**

PASO 2: Polarización. **-0.25** = 1.0 * 2^ (- 2) Precisión sencilla 127 + (-2) = 125

PASO 3: Convertir decimal. 0.0 * 2 = 0

PASO 4: Representación.

Signo	***Exponente***	***Mantisa***
0	0 1 1 1 1 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 1 1 1 1 1 0 **1 0 0 0** 0 0 0 0 **0 0 0 0** 0 0 0 0 **0 0 0 0** 0 0 0 0
3 E 8 0 0 0 0 0
Número: 3E800000 → En secuencia inversa en memoria sería 0000803D

4. PROCESO INVERSO DADO EN MEMORIA CONVERTIRLO A DECIMAL

Dado el número C1950000

PASO 1 Expresarlo en binario para poderlo representar en la tabla IEEE 754

C 1 9 5 0 0 0 0
1100 0001 1001 0101 0000 0000 0000 0000

PASO 2 Identificar cada uno de los bits en cada grupo del estándar

1 10000011 001 0101 0000 0000 0000 0000
Signo Exponente Mantisa

Signo 1 = Negativo
Exponente = 10000011 = 131 = 131-127 = 4
Mantisa = 0010101 00.. = 1 * 2^ (-3) + 1 * 2 ^ (-5) + 1 * 2 ^ (-7) = 0,125 + 0,03125 + 0,0078125
= 0,1640625

Número sería = - 1. 1640625 * 2 ^ 4 = - 1. 1640625 * 16 = **-18.625**

5. REPRESENTACIONES IMPORTANTES

0 = 0 00000000 00000000000000000000
-infinito = 1 11111111 00000000000000000000
+ infinito = 0 11111111 00000000000000000000
No es numero = 0 11111111 00010000000000000000
No es numero = 1 11111111 00100000010100000000 01

PROGRAMA SENCILLO

```
.model small
.stack 64
.data
	v1 dd 16.5	;41840000
	v2 dd 21.6	;41ACCCCD
	v3 dd -5.25	;C0A80000
	v4 dd -18.625 ;C1950000
.code
	begin proc near
	  mov ax,@data
	  mov ds,ax
			mov ax,v1
			mov bx,v2
			mov ax,v3
			mov bx,v4
	    mov ax,4c00h
	    int 21h
	end begin
end
```

CONVERSIONES DE PUNTOS DECIMALES

Decimal a Binario
0.125 (10) a (2) = 0.125 * 2 = 0.25 0.125 (10) es 0.001 (2)
0.25 * 2 = 0.5
0.5 * 2 = 1.0

Decimal a Octal
0.125 (10) a (8) = 0.125 * 8 = 1.0 0.125 (10) es 0.1 (8)

Decimal a Hexadecimal
0.125 (10) a (16) = 0.125 * 16 = 0.2 0.125 (10) es 0.2 (16)

REPRESENTACIÓN DE CARACTERES

Es una abstracción de datos:

- El CPU solamente sabe procesar números
- Otros datos deben representarse en términos numéricos

Código de carácter: define una tabla de correspondencias entre caracteres y números asignados

- EBCDIC: Antiguo, usado en Mainframes.
- ASCII: es de 7 bits (0 a 127), en PC's se extiende a 256
- ISO 8859-1
- Unicode Nuevo estándar 8, 16 y 32 bits.

Unicode define tres formas de codificación bajo el nombre UTF o Formato de Transformación Unicode (Unicode Transformation Format):10

UTF-8 — codificación orientada a byte con símbolos de longitud variable.
UTF-16 — codificación de 16 bits de longitud variable optimizada para la representación del plano básico multilingüe (BMP).
UTF-32 — codificación de 32 bits de longitud fija, y la más sencilla de las tres.

Esquema de codificación	***Endianness***	**Admite BOM (*Byte Order Mark*)**
UTF-8	No aplicable	Sí
UTF-16	*Big-endian* o *Little-endian*	Sí
UTF-16BE	*Big-endian*	No
UTF-16LE	*Little-endian*	No
UTF-32	*Big-endian* o *Little-endian*	Sí
UTF-32BE	*Big-endian*	No
UTF-32LE	*Little-endian*	No

BCD: Decimal codificado Binario ---------- Se almacena en forma empaquetada o desempaquetada.
BCH: Hexadecimal codificado Binario --------- No. Hex donde cada digito se representa en 4 bits.
BCO: Octal codificado Binario ---------- No. Oct donde cada digito se representa en 3 bits.

BCD *DECIMAL CODIFICADO BINARIO*

Se utiliza para la representación de los números decimales bajo dos normas:

Empaquetado: Agrupa por Byte dos digitos.
Desempaquetado: Representa cada digito por Byte.

Digito	***Empaquetado***	***Desempaquetado***
15	0001 0101	0000 0001 0000 0101
243	0000 0010 0100 0011	0000 0010 0000 0100 0000 0011