SISTEMAS NUMERICOS Y ERRORES

1. Introducción a la Computación Numérica

El primer computador electrónico en base a la tecnología de tubos al vacío fue el ENIAC de la Universidad de Pensilvania, en la década del 40. Durante la década del 50, el primer uso de los computadores fue para las aplicaciones científicas.

En la década del 60, el uso de los computadores se amplió a los negocios, y el propósito más extendido fue el tratamiento de todo tipo de información.
En las tres últimas décadas (70 a 90) continuó extendiéndose hacia las medianas empresas en los 70; y hacia varios millones de pequeñas empresas y personas en la llamada revolución de las PC, en los 80 y 90.

La mayor parte de esos usuarios del computador no consideran de primer interés a la computación como medio de *cálculo con números*. En realidad lo que más se utiliza es el procesamiento de la información en otros campos como los negocios y la administración. Sin embargo, en muchas disciplinas científicas, el cálculo con números permanece como el uso más importante de los computadores.

Ejemplos:

Físicos: resolución de complicadas ecuaciones en modelos tales como la estructura del universo o del átomo.
Médicos: que usan los computadores para diseñar mejores técnicas.
Meteorólogos: usan la computación numérica para resolver ecuaciones en modelos que pronostican el clima.
Ingenieros Aeronáuticos: Diseño de cohetes espaciales.

En la Ciencia de la Computación, la computación numérica tiene mayor importancia por los requerimientos de algoritmos confiables y rápidos para computación gráfica, robótica, etc.

2. Números Reales

Una clasificación de los números reales es: $R = Q \cup I$; y a su vez $Q = Z \cup F$, donde: R reales, Q racionales, I irracionales, Z enteros, F fraccionarios.
Los números reales que no pueden representarse como enteros o fracciones, se llaman irracionales.
Ejemplo:
- **Π** se define como la razón entre la longitud de una circunferencia y su diámetro.
- **e** se define como el límite de $(1+1/n)$ cuando $n \to \infty$, un límite de una sucesión de números racionales {2;9/4;64/27...}

Propiedad:
El conjunto de números irracionales no es numerable.
Una ordenación de los elementos de un conjunto en una sucesión análoga a la de los naturales se llama una enumeración del conjunto, que así resulta con la propiedad de ser numerable. El conjunto Q es numerable.

Consecuencia: No hay manera de listar a todos los números irracionales.

2.1 Sistemas de representación de números reales

Históricamente los Romanos usaban distintos símbolos para representar las potencias de 10: X, C, M, etc., lo que es engorroso para grandes números.
El uso del cero como símbolo fue usado en la India y luego introducido en Europa por medio de los Arabes, hace aproximadamente 1000 años.
El único sistema que usaba el cero (sin influencia de los Indios) fue el de los Mayas. Este sistema posicional tenía como base 20.
Nuestro sistema actual se llama decimal o de base 10, pues requiere 10 símbolos {0,1,2,3,4,5,6,7,8,9}. El sistema se llama posicional, pues el significado del número depende de la posición de los símbolos.

Los Babilonios usaban el sistema de base 60, cuyas influencias llegan a nuestro tiempo con el sistema de medición del tiempo (1 hora = 60 min.; 1 min.= 60 seg.).
El sistema de base igual a 2, que no es tan natural para los humanos, es el más conveniente para los computadores. Todo número n esta formado por una sucesión (cadena o string) de ceros y unos.

Todo número real posee una representación *decimal* y otra *binaria*; y por lo tanto, una representación en toda base $B_{(n}$, tal que n >1.

Propiedad:
La representación de la base B, en base B, es siempre "10" (uno, cero).

Ej. : $2_{(2} = 10_{(2}$; $10_{(10} = 10_{(10}$

2.2 Conversiones entre representaciones de sistemas más usuales

- Caso de números enteros:

$x_{(10} = 61_{(10} = 6*10^1 + 1*10^0$

Nota:
La mayor potencia de 10 en el segundo miembro es igual al número de cifras del número $x_{(10}$, menos 1.

- Caso de números fraccionarios:

Ejemplos:

$11/2 = 5.5_{(10} = 5\times10^0 + 5\times10^{-1} = 5\times1 + 5\times1/10$

$$\frac{11_{(10}}{2_{(10}} = \frac{1011_{(2}}{10_{(2}} = 101.1_{(2} = 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} = 1\times4 + 0\times2 + 1\times1 + 1\times\frac{1}{2}$$

Un ejemplo de representación binaria infinita es 1/3, "periódica repetitiva", en ambos sistemas:

$1/3 = (0.333\ldots)_{(10} = (0.010101\ldots)_{(2}$

- Si la representación es infinita, debe ser periódica repetitiva.
- Los números irracionales siempre tienen una representación infinita, no periódica.

$$\sqrt{2} = (1.414213\ldots)_{(10}$$

Reglas Prácticas:

1) Para convertir un número **x** escrito en base **B = 2,** a base **B' = 10**, se aplica el algoritmo de descomposición del número, según las potencias de 2.

Ej.:

$x = 1001.11_{(2} = 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2} = 8+1+1/2+1/4 = 9.75$

2) Para convertir un número **x,** de base **B = 10,** a base **B' = 2**, se determinan los coeficientes $a_0,a_1,...,a_n$, de la base B', **ceros (0)** o **unos (1)** , de modo tal que:

$x_{(10} = a_n\times2^n + a_{n-1}\times2^{n-1} + ... + a_0\times2^0$; y que $(a_n\, a_{n-1}...a_0)_{(2} = x_{(10}$

Ej. $7_{(10} = a_2\times2^2 + a_1\times2^1 + a_0\times2^0$

Se divide **x** por **2**, lo que da $\mathbf{a_0}$ como resto y un cociente, que dividido por 2, da $\mathbf{a_1}$ como resto; y así siguiendo hasta que el último cociente (es menor que 2) da como resto $\mathbf{a_n}$.

3) Conversión de Binario a Octal

Se comienza agrupando las cifras binarias, de tres en tres, de derecha a izquierda; luego se escribe el equivalente en base 8, en cada grupo.

Si se aplica el desarrollo polinómico a partir del coeficiente de 8^{n-1} (n = número de grupos), trabajando en base 10, se obtiene la expresión decimal del número binario dado.

$$110011101_{(2} = \underbrace{110}_{6}\underbrace{011}_{3}\underbrace{101}_{5} = 6\times8^2 + 3\times8^1 + 5\times8^0 = 635_{(8} = 413_{(10}$$

4) Conversión de Binario a Hexadecimal

Para pasar un número escrito en base 2, a base 16, se agrupan las cifras binarias en grupos de 4, desde la derecha a izquierda, y luego se sustituye en cada grupo su equivalente por la cifra hexadecimal correspondiente.

$$000110011101_{(2} = \underbrace{0001}_{1}\underbrace{1001}_{9}\underbrace{1101}_{D}{}_{(2} = 1\times16^2 + 9\times16^1 + D\times16^0 = 19D_{(16}$$

Las relaciones entre grupos de cifras binarias y los sistemas de bases 2,8,10 y 16, siendo sus cifras, $B_{(2} = \{0,1\}$; $B_{(8} = \{0,1,...,7\}$; $B_{(16} = \{0,1,...,9,A,B,C,D,E,F\}$; $B_{10} = \{0,1,...,9\}$, se muestran en el Cuadro N°1.

Cuadro N°1

Grupo 3 bits	Grupo 4 bits	Base 2	Base 8	Base 16	Base 10
000	0000	0	0	0	0
001	0001	1	1	1	1
010	0010	10	2	2	2
011	0011	11	3	3	3
100	0100	100	4	4	4
101	0101	101	5	5	5
110	0110	110	6	6	6
111	0111	111	7	7	7
	1000	1000	10	8	8
	1001	1001	11	9	9
	1010	1010	12	A	10
	1011	1011	13	B	11
	1100	1100	14	C	12
	1101	1101	15	D	13
	1110	1110	16	E	14
	1111	1111	17	F	15

2.3 Representación de números en el computador

Sea un computador con una palabra de memoria de 32 bits.

Nos interesa analizar cómo pueden representarse los dos números siguientes 2^{32} y 2^{31} (en lugar del punto reservado para la "coma decimal" se usan apóstrofes):

$$2^{32} = 4'294'967'296$$
$$2^{31} = 2'147'483'648$$

Propiedad:

El máximo número que almacena una palabra de 32 bits es $2^{32}-1 = 4'294'967'295$

Cuadro N°2

Bits en cada palabra	Nro. Máximo	Equivalente en Base 10	Expresión Gral.
2	11	3	2^2-1
3	111	7	2^3-1
4	1111	15	2^4-1
....	....	....	...
n	111....1		2^n-1

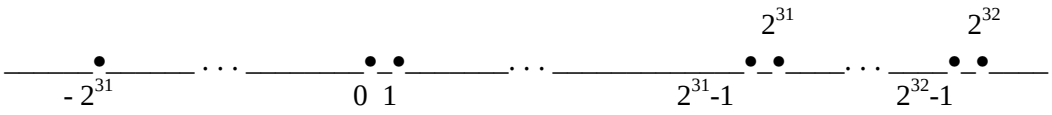
El número máximo en ese computador será entonces 2^{32}-1, por lo tanto el número $2^{32} = 2 \times 2^{31}$ no es posible representar, en cambio sí es representable 2^{31}-1 =2'147'483'648

2.3.1 Representación de números enteros negativos

Una forma simple de representar el signo y el módulo de un número entero consiste en destinar un bit, de los 32 bits de una palabra-memoria, para el signo; y el resto de los 31 bits para el valor absoluto del número.

Si reducimos en **1** el número de bits de una palabra de n =32 bits, entonces el máximo número que puede almacenar la palabra es 2^{n-1}-1= 2^{32-1}-1 = 2^{31}-1 = 2'147'483'647.
Por lo tanto, al representar números negativos se reduce la magnitud, de 2^{32} - 1 a 2^{31}- 1.

- La técnica del complemento a 2.



Los números enteros positivos entre 0 y 2^{31}-1 se almacenan mediante "0" y "1" empleando 31 bits.
Los números negativos del tipo: **- x**, con $|-\mathbf{x}|$ entre 1 y 2^{31} son almacenados como la diferencia ($\mathbf{2^{32} - x}$), la cual debe tener un valor entre 2^{31} y 2^{32}-1.

Ej. x = -14 $_{(10}$ = -1110$_{(2}$
es almacenado como 2^{32}-14 , el cual está comprendido entre 2^{31} y 2^{32}-1.

Es decir: 2'147'283'648 < 4'294'967'282 < 4'294'967'295

En términos de bits, vemos que si cambiamos 0 por 1 y 1 por 0 en la representación binaria del número dado 14 $_{(10}$ = (0 . . . 01110) $_{(2}$, y a ese resultado le sumamos 1, se tiene:

$$-14_{(10} = 1. . .1\ 0001_{(2} + 1 = \mathbf{1. . . 0010}_{(2}$$

Justificación: Como $\mathbf{2^{32} - x} = (2^{32} - 1 - x) + 1$ se puede ver en el Cuadro N°3, que si restamos de la palabra formada por todos los bits igual a 1 (o sea 2^{32} - 1), el valor x $_{(2}$, y luego le sumamos 1, entonces obtenemos el opuesto de x $_{(2}$.

Cuadro N°3

$2^{32}-1$	1	1	1	.	.	.	1	1	1	1	1
|-x| =14	0	0	0	.	.	.	0	1	1	1	0
$(2^{32}-1)-14$	1	1	1	.	.	.	1	0	0	0	1
-x = $(2^{32}-1)-14+1$	**1**	**1**	**1**	**.**	**.**	**.**	**1**	**0**	**0**	**1**	**0**
(x)+(-x), overflow **1**	0	0	0	.	.	.	0	0	0	0	0

Conclusión:
La representación del número negativo, **(-x),** (4ta.fila) , se obtiene:
Primero, cambiando en la representación de (+x)(2da. fila), los "1" por "0", y viceversa, lo que da la 3ra. fila; finalmente, añadiendo un "1" a ese resultado se obtiene (-x)(4ta.. fila).
La comprobación de la 5ta. fila (x)+(-x), da un string de "0" con un "1", en la parte más izquierda que no puede representarse (overflow bit)

- La técnica de un bit para el signo

La técnica de 1 bit para el signo utiliza el mismo procedimiento para almacenar el valor del número y 1 bit extra para el signo.
Nota:
No es conveniente esta representación.

Ejercicios: Usando palabras-memoria de 4 bits, dar ejemplos de suma y resta de números enteros (positivos y negativos).

2.3.2 Representación de Números Racionales

Para la representación de los números Racionales existen dos métodos muy conocidos como el del **punto fijo,** y la representación en **punto flotante.**

1) Punto Fijo

El sistema usa palabras divididas en 3 campos:

Signo	Parte del número precedente al punto binario	Parte posterior al pto. binario

Desventajas: sólo se puede representar una pequeña cantidad de números. En nuestro caso, la palabra de 32 se divide en campos de 1, 15 y 16 bits, respectivamente, y los números están en el rango: $2^{-16} \leq x < 2^{15}$
Nota: raramente usada hoy en aplicaciones científicas.

2) Punto Flotante

Con el sistema de Punto Flotante, se resuelve el problema del punto fijo, ya que amplía el rango. Los números que conocemos como escritos en notación científica o notación exponencial se asemejan a la notación de punto flotante.

Un sistema de números en punto flotante es un subconjunto $F \subset \Re$ de números reales, cuyos elementos x, $x \in R-\{0\}$ se pueden escribir como**:**

$$\mathbf{x = \pm m \times 10^{E}}\ , \text{ con } \ 1 \leq m < 10$$

Otra forma equivalente:

$$\mathbf{x = \pm 0.\ d_1 d_2 \ldots d_t \times 10^{E}}$$

Los parámetros que caracterizan el sistema de números flotantes de base diez, son:

- La base **B**
- La mantisa **m**, que representa a la parte fraccionaria del número
- El exponente **E**, que varía entre dos cotas: $E_{min} \leq E \leq E_{max}$
- La precisión **t** referida a la cantidad de dígitos d_i donde: $0 \leq d_i \leq B-1$

En los computadores se prefiere la notación de base 2:

$$\mathbf{x = \pm m \times 2^{E}} \ , \text{ con } \ 1 \leq m < 2$$

Para una palabra de memoria de 32 bits la representación se hace en 3 campos: 1 bit para el signo; 8 bits para el exponente E; y 23 bits para la mantisa.

El campo de la **mantisa m** puede representar los 23 bits que denominaremos:

$$b_0\ b_1\ b_2 \ldots b_{22}.$$

Si las cifras que siguen a b_{22} , o sea, b_{23}, b_{24}, ... no son todas 0, esta representación del **número flotante x** , designado **fl (x)**, no es exacta, sino aproximada.
Sin embargo **fl (x)** puede ser expresado en forma aproximada mediante dos técnicas: **truncamiento** y **redondeo** (cfr.7)

2.3.3 Definición de Número en Punto Flotante

Si un número **x** puede ser almacenado exactamente en el computador usando la representación de punto flotante con $b_{23} = b_{24} = ... = 0$, se llama **número en punto flotante**, y se lo designa **fl (x)**

Ejemplo 1:

$x = \dfrac{11_{(10}}{2_{(10}} = 101.1_{(2}$ Este cociente puede multiplicarse y dividirse por 2^2, de modo que aparezca el primer dígito distinto de cero a la izquierda, y después la coma decimal (flotante). En general esto se obtiene cambiando el exponente de la base convenientemente.
Luego: $f\,l\,(x) = 1.011 \times 2^2$

En el caso general, la expresión de **m** en binario es: $\mathbf{m = (b_0.b_1\ b_2\ b_3...)_{(2}}$, con $b_0 = 1$

0	E=2	**1. 0 1 1 ... 0**
$\underbrace{1\,bit}$	$\underbrace{8\,bits}$	$\underbrace{23\,bits}$

Ejemplo 2:

$x = 71_{(10} = 1000111\ _{(2}$
Expresando en forma de número flotante en base B=2, con exponente E=6 (al desplazar el punto flotante desde la última cifra a la primera), se tiene:
$fl(x) = (1.000111)_{(2} \times 2^6$, siendo su representación en bits:

0	E = 6	**1. 000 11 10 ... 0**

Notas:

1) El punto decimal que aparece entre b_0 y b_1 en el tercer campo de 23 bits es mostrado a los fines didácticos pero no suele ser almacenado realmente. Por consiguiente, los 23 bits se enumeran desde $\mathbf{b_1}$ hasta $\mathbf{b_{23}}$.
2) El exponente se coloca en base 10, pero en rigor tiene expresión binaria.
3) El bit del signo: 0 indica un número positivo; 1 es negativo.

2.3.4 Número en Punto Flotante Normalizado

Definición: Se llama número normalizado a aquel número que tiene en la mantisa $b_0 = 1$

Ej.:
$(1/10)_{(10} = (1.100110011...)_{(2} \times 2^{-4}$

0	E = - 4	1. 100110011 ... 110

Caso particular:
El cero es un caso especial. No se puede normalizar pues todos los bits son ceros.

Una regla:

Para normalizar un número se desplaza su mantisa hacia la izquierda hasta que el primer dígito sea distinto de cero, y se reduce el exponente en una cantidad igual al número de desplazamientos que se efectuaron.
Como consecuencia puede haber desbordamiento (overflow o underflow), lo que implica que el exponente es muy grande o muy chico, respectivamente.

3. La representación de Punto Flotante IEEE

IEEE: Institut for Electrical an Electronics Engineers
Antecedentes: Entre 1960 y 1970 cada fabricante desarrollaba para sus máquinas el propio sistema de punto flotante. Mientras algunas máquinas usaban el sistema binario, la IBM 360/370 usaba hexadecimal: $\pm m \times 16^E$.
A comienzos de 1980, en base al esfuerzo de muchos científicos de la computación, como W. Kahan, un sistema de punto flotante fue desarrollado, el que mereció el seguimiento de los primeros fabricantes de chips para la construcción de las PC, como Intel y Motorola.
Ese sistema se transformó en el sistema de punto flotante IEEE que existe para números en binario y decimal.

El estándar IEEE tiene 3 requerimientos:

1) La representación de los números en punto flotante debe ser consistente en todas las máquinas que lo adopten.
2) La aritmética de redondeo debe ser correcta.
3) El tratamiento de casos excepcionales como la división por cero debe ser consistente.

Tipos de Punto Flotante en IEEE

Simple Precisión IEEE, Doble Precisión IEEE, y Precisión Extendida IEEE.
En las tablas siguientes mostramos algunos ejemplos de los dos primeros. Se observa en todos los ejemplos que el bit $\mathbf{b_0}$ no está consignado.

- **Simple Precisión**

±	$a_1\ a_2\ a_3 \ldots a_8$	$b_1\ b_2\ b_3 \quad \ldots \quad b_{23}$

Si el string del Exponente $a_1\ a_2 \ldots a_8$ es:	El valor numérico representado es:
$00000000_{(2} = 0_{(10}$	$\pm(0.b_1b_2...b_{23})_{(2} * 2^{-126}$
$00000001_{(2} = 1_{(10}$	$\pm(1.b_1b_2...b_{23})_{(2} * 2^{-126}$
. . .	. . .
$01111111_{(2} = 127_{(10}$	$\pm(1.b_1b_2...b_{23})_{(2} * 2^0$
$10000000_{(2} = 128_{(10}$	$\pm(1.b_1b_2...b_{23})_{(2} * 2^1$
$11111110_{(2} = 254_{(10}$	$\pm(1.b_1b_2...b_{23})_{(2} * 2^{127}$
$11111111_{(2} = 255_{(10}$	$\pm\infty$ si $b_1=b_2=...=b_{23}=0$, sino, NaN (*)

(*) Casos particulares:
Las 2 representaciones de cero +0 y -0 son dos representaciones del mismo valor "cero".
Las dos representaciones $+\infty$ y $-\infty$ son representaciones de distintos números.
Un número especial es NaN, que en inglés deriva de Not a Number, que no es en rigor un número, sino que indica un error en el patrón de bits.
Estos casos se anotan mediante el uso de un patrón especial de bits en el campo correspondiente al exponente.

- **Doble Precisión IEEE**

± (1bit)	$a_1a_2a_3...a_{11}$ (11 bits)	$b_1b_2...b_{23}...b_{52}$ (52 bits)

Si el string del exponente es $a_1a_2...a_{11}$:	El valor numérico representado es:
$00...00_{(2} = 0_{(10}$	$\pm 0.b_1b_2...b_{52(2} * 2^{-1022}$
$00...01_{(2} = 1_{(10}$	$\pm 1.b_1b_2...b_{52(2} * 2^{-1022}$
$00...10_{(2} = 2_{(10}$	$\pm 1.b_1b_2...b_{52(2} * 2^{-1021}$
. . .	. . .
$100...00_{(2} = 10^{11}{}_{(10} = 1024_{(10}$	$\pm 1.b_1b_2...b_{52(2} * 2^{1}$
...	...
$111...10_{(2} = 2046_{(10}$	$\pm 1.b_1b_2...b_{52(2} * 2^{1023}$
$111...11_{(2} = 2047_{(10}$	$\pm \infty$ si $b_1 = b_2 = ... = b_{52} = 0$, sino NaN

- **Cuadro comparativo de Simple y Doble Precisión IEEE**

Tipo	Tamaño	Signo	Mantisa(Frac.)	Exponente	Rango (≈)
Simple	32 bits	1 bit	23+1 bits	8 bits	$1.2*10^{-38}$ a $3.4*10^{+38}$
Doble	64 bits	1 bit	52+1 bits	11 bits	$2.2*10^{-308}$ a $1.8*10^{+308}$

Nota: En este sistema de punto flotante los números deben estar normalizados, es decir que el bit más significativo (b_0) debe ser siempre igual a 1 (uno) y no se lo almacena. Se llama por ello "hidden bit" (bit oculto).

- **Definición de Sistema Flotante IEEE**

Es un subconjunto de números reales que está formado por todos los números aceptables en el sistema de aritmética de punto flotante IEEE, es decir consiste de ± 0, los números normalizados con b_0=1, los números subnormalizados con b_0=0 y E=-126, ±∞ (sin incluir los valores NaN).

4. Precisión del sistema de Punto Flotante y Precisión de la máquina o Epsilon Máquina

Se llama precisión del sistema de punto flotante al número **t** de bits de la mantisa. En el sistema descrito t = 24, corresponde a 7 dígitos significativos puesto que $2^{-24} \approx 10^{-7}$. En doble precisión, es t =53, que tiene 16 cifras decimales significativas.

En cualquier sistema, el número flotante más próximo, mayor que 1, es el que se obtiene al sumarle el bit 1.

Se llama epsilon (ε) de máquina al número (gap) que existe como diferencia entre 1 y el número próximo más grande.

Un número flotante (normalizado, en base 2) con precisión **t** se expresa así:

$$f\,l\,(x) = \pm\,(\,1.\,b_1b_2...b_{\,t-1})_{(2}\times 2^E$$

En el sistema de simple precisión (t=24), se tiene: $f\,l\,(x) = \pm\,(\,1.\,b_1b_2...b_{\,23})_{(2}\times 2^E$
Si el número **1** se expresa $(1.00...0)_{(2}$, entonces el primer número mayor es el que se obtiene sumándole 1 (un) bit, es decir: $(\,1.00...1)_{(2}$.
Expresando este valor mediante las potencias (negativas) de 2, se obtiene:

$$1\times 2^0 + 0\times 2^{-1} + 0\times 2^{-2} + \ldots + 1\times 2^{-(t-1)} = 1 + 2^{-(t-1)}$$

Al reemplazar t =24 y luego efectuar la diferencia con **1** se obtiene el "épsilon de máquina" para este sistema de "simple precisión":

$$\varepsilon = (1+2^{-23})-1 = 0.\,000\,000\,119_{(10} \approx 1.2\times 10^{-7}$$

Con el fin de obtener una aproximación de ε **,** se pueden usar varios algoritmos, uno de los cuales se expresa mediante pseudocódigo se.iguidamente:

E1) Definir las variables N (entero), EPS , T (reales)
E2) N ← 0 ; EPS ← 1
E3) Repetir hasta que T=1: N ← N +1 ; EPS ← EPS/2 ; T ← 1+EPS
E4) Escribir: ' Epsilon-maquina = ' , 2**(1-N) . Alto

A modo de conclusión, en el cuadro siguiente se comparan los sistemas de simple y doble precisión IEEE:

	Precisión (t)	Epsilon máquina
Simple IEEE	24	$\varepsilon = 2^{-23} \approx 1.2*10^{-7}$
Doble IEEE	53	$\varepsilon = 2^{-52} \approx 2.2*10^{-16}$

5. Un ejemplo didáctico de un sistema de punto flotante

±	E	b_0 b_1 b_2
Signo	Exponente	Mantisa

E: 0,1,-1 (los únicos valores)
$Valor_{max}$: $1.11_{(2} * 2^1 = (1*2^0+1*2^{-1}+1*2^{-2})*2 = 1.75*2 = 3.5_{(10}$
$Valor_{min}$: $1.00_{(2} * 2^{-1} = 0.5_{(10}$

Precisión del sistema:

El gap entre 1 y el siguiente es la diferencia ε entre:

$1.00_{(2}$ y $1.01_{(2}$; $\varepsilon = 1_{(10} - 1.25_{(10} = 0.25$; $\varepsilon = 0.25$

Propiedad:
El gap entre los números de punto flotante es proporcional al exponente de la base.

Ejemplo:

Si E = 0 los números a representar son:
$1.00_{(2}*2^0$; $1.01_{(2}*2^0$; $1.10_{(2}*2^0$ y $1.11_{(2}*2^0$

es decir : { $1_{(10}$; $1.25_{(10}$; $1.5_{(10}$; $1.75_{(10}$}
gap = ε = 0.25

Si E=1 tendremos:
{ $2_{(10}$; $2.5_{(10}$; $3_{(10}$; $3.5_{(10}$ }
gap = ε = 0.5

Si E=1/2 tendremos:

{ $0.5_{(10}$; $0.625_{(10}$; $0.75_{(10}$; $0.875_{(10}$ }

gap = ε = 0.125

Dado un número en punto flotante $b_0.b_1b_2{}_{(2} * 2^E$, el próximo número punto flotante mayor que el dado se obtiene multiplicando: $\varepsilon*2^E$.

Propiedad:
Los números normalizados (para todos los valores de E) en un sistema de base B (2,10,etc.) no están igualmente espaciados.
Observación: Hay un gap mayor entre 0 y el primer número positivo, que entre éste y el siguiente.

Definición:
Se llaman números subnormalizados a aquellos números que no son normalizados (b_0=0) y tienen el mínimo exponente. Se utilizan para representar números (no normalizados) mediante una combinación especial de un string de bits con ceros en el campo del exponente y una parte fraccionaria (distinta de cero) en la mantisa.

6. Operaciones elementales con números en Punto Flotante

Supongamos que los números reales **x** e **y** tienen como representaciones en punto flotante a las cantidades ***f l (x) y f l (y).*** .

Sean los símbolos **(+), (-), (*), (÷)** elegidos para indicar las operaciones aritméticas básicas en el computador, que corresponden a las operaciones aritméticas básicas elementales de suma, resta, multiplicación y división.
Se tiene entonces:

$$x\ (+)\ y = fl\ [\ fl\ (x) + fl(y)]$$
$$x\ (-)\ y = fl\ [\ fl\ (x) - fl(y)]$$
$$x\ (*)\ y = fl\ [\ fl\ (x) * fl(y)]$$
$$x\ (\div)\ y = fl\ [\ fl\ (x) \div fl(y)]$$

Ejemplo:
(x + y) puede no ser un número en F, pero **x (+) y** es el número en punto flotante que el computador calcula como una aproximación de (x+y).

7. Redondeo y truncamiento

Esta aritmética idealizada consiste en efectuar la aritmética exacta en las representaciones del punto flotante de **x** e **y**, y luego convertir el resultado exacto en su representación de punto flotante.

Ejemplo:

Supongamos que **x =1/3 ; y = 5/7** y se calcula la suma **x + y = 22/21= 1. 047 619 047 619... ,** empleando sus representaciones flotantes, trabajando con 5 decimales.

Obtenemos:
$fl(x) = 0.33333 \times 10^0$; $fl(y) = 0.71428 \times 10^0$

Entonces: $[\ fl(x) + fl(y)] = 0.10476 \times 10^1$
Para x (+) y se tomará el número flotante del segundo miembro que corresponda según el número de dígitos de la mantisa.
Considerando una de las expresiones de un número flotante en base diez:

$$\pm 0.d_1 d_2 ... d_k \times 10^n\ ;\ d_1 = 1,2,...,9 \quad ;$$
$$con \quad i = 2,3,...,k;\ \ 0 \le d_i \le 9$$

Esta forma de número flotante se puede obtener incluyendo **k cifras** en la mantisa. Esto implica **cortar o truncar** el número y el método es llamado, por esa razón, **truncamiento.**

El otro método se conoce como **redondeo**:

Si $d_{k+1} \ge 5$, entonces $d_k + 1$ provee el valor redondeado de $f\,l\,(x)$

En caso contrario se aplica el método de **cortar o truncar** los dígitos: $d_{k+1}, d_{k+2}....$
lo cual equivale a un redondeo “hacia abajo”.

8. Errores

8. 1 Error de redondeo

El error que resulta de reemplazar un número por su representación en punto flotante, se denomina error de redondeo, sin especificar el método de redondeo aplicado.
Se expresa mediante la diferencia entre **x** y **fl(x).**

8. 2 Definiciones de errores absoluto y relativo

Si la cantidad $\mathbf{p}^*$ es una aproximación de la cantidad (exacta) **p,** se llama **error absoluto** al valor absoluto $\left|p^* - p\right|$. El error **relativo** está dado por: $\dfrac{\left|p^* - p\right|}{\left|p\right|}$; $p \neq 0$

Ejemplo:

a) Si p = 0.3000 x 10^1 , p^* = 0.3100 x 10^1, el error absoluto es 0.1 y el error relativo es 0.3333 x 10^{-1}

b) Si p = 0.3000 x 10^{-3} y p^* = 0.3100 x 10^{-3}, el error absoluto es 0.1 x 10^{-4} y el error relativo es 0.3333 x 10^{-1} .

Este ejemplo muestra que el error relativo permanece constante para valores muy diferentes del error absoluto, por lo que es una medida que da más información significativa y menos engañosa que la del error absoluto.

8. 3 Dígitos significativos

El uso frecuente de aritmética de redondeo en computadoras lleva a la siguiente definición.
Se dice que el número ***p**** *aproxima a* ***p*** con **t** dígitos significativos (o cifras) si **t** es el entero más grande no negativo para el cual, el error relativo verifica:

$$\frac{\left|p^* - p\right|}{\left|p\right|} < 5 \times 10^{-t}$$

Ejemplo 1)
El número $\mathbf{p}^*$ = 3.14 aproxima a **p** = 3.141592 con t =3 cifras significativas.
En efecto: 0.000 506749 < 5×10^{-3}, donde t =3 es el menor entero positivo que verifica la desigualdad.

Ejemplo 2)
Dado p = 1000 y como consecuencia de la definición, podemos despejar p^* para establecer su rango de variación, para t = 4 cifras significativas. Entonces p y p^* concuerdan en 4 cifras significativas. Luego, $\dfrac{\left|p^* - 1000\right|}{\left|1000\right|} < 5 \times 10^{-4}$ implica: 999.5 < p^* < 1000.5

Nota:
Las cifras 0,1,...,9 que figuran en un número, se llaman *significativas*, excepto la cifra 0 (cero) cuando es usada a la izquierda para determinar el lugar de la coma (punto decimal).

8.4 Propagación de errores

Llamando ε a la "magnitud" del error absoluto, al aproximar el "valor exacto" **a** con el valor aproximado $\mathbf{a^*}$, se pueden dar la siguiente expresión del error :

$$a = a^* \pm \varepsilon$$

- Propiedad

El error absoluto de una suma (o resta) es igual a la **suma** de los errores de los términos.

Si $a = a^* + \varepsilon_a$; $b = b^* + \varepsilon_b$, entonces:

$$\varepsilon_{a+b} = \varepsilon_a + \varepsilon_b \; ; \; \varepsilon_{a-b} = \varepsilon_a + \varepsilon_b$$

Una expresión que vincula el **error relativo r,** de un número exacto **a** (positivo), y su aproximación $\mathbf{a^*}$ es:

$$\mathbf{a^* = a + a\,r = a\,(1 + r)}$$

- Propiedad

Si $\mathbf{a^*}$ y $\mathbf{b^*}$ son positivos y tienen como errores relativos r_a, r_b entonces el error relativo del producto es aproximadamente igual a la **suma** de los errores relativos de los factores.
Es decir:

$$r_{ab} \approx r_a = r_a + r_b$$

- Ejemplos(cfr. Dahlquist y Björck , pg. 26):

  $a = 2.31 \pm 0.02$; $b = 1.42 \pm 0.03$
  $\varepsilon_{a+b} = \varepsilon_a + \varepsilon_b = 0.05$
  $r_a = 0.02$; $r_b = -0.01$, $r_{ab} = (1 + r_a)(1 + r_b) - 1 = 0.0098 \approx r_a + r_b = 0.01$

- Nota: Una propiedad deseable en todo proceso de cálculo es que un error pequeño inicial, produzca errores pequeños en el resultado final. Un algoritmo que posee esta propiedad se denomina *estable*, en caso contrario *inestable.*

- Definición: Sea ε el error inicial y $\varepsilon(n)$ el crecimiento de dicho error propagado a lo largo de **n** operaciones. Si $|\varepsilon(n)| \approx n\varepsilon$, entonces, el crecimiento se dice *lineal*. Si $|\varepsilon(n)| \approx K^n \varepsilon$, entonces el crecimiento es *exponencial.*
- Ejemplos: Relativos al punto anterior (cfr. R. Burden y J.D. Faires, pg. 34)