Curso de Métodos Numéricos

DEMAT, Universidad de Guanajuato

Clase 1: Representación de los números en la computadora

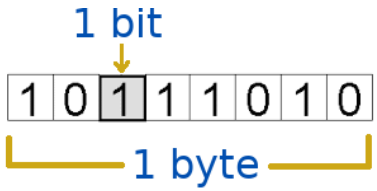
- Representación de punto fijo y punto flotante
- Formato de la IEEE

MAT–251

Dr. Joaquín Peña Acevedo
CIMAT A.C.
e-mail: joaquin@cimat.mx

Representación de los números (I)

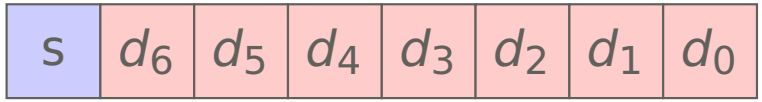
- Los números en la computadora se representan mediante una secuencia de bits (0's o 1's).



- Se representan los números en base 2.
- Para interpretar una secuencia de bits como un número, primero hay que establecer la cantidad de bits que se usarán para almacenar un número y luego definir las reglas para representarlos.
- Hay dos tipos de aritmética que se realiza en las computadoras: aritmética con enteros y aritmética con números reales.
- La representación de enteros es simple. Hay que convertir el número a base 2 y tomar en cuenta el rango de números que se pueden representar.

  Por ejemplo, si se tienen 8-bits para representar un entero, y se usa un bit para indicar el signo del número, entonces

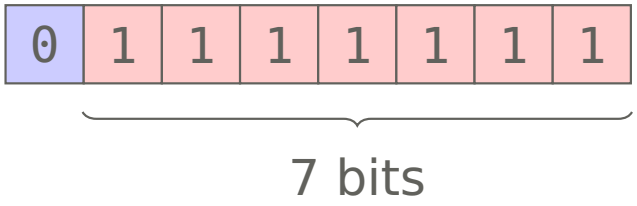
Representación de los números (II)



$s, d_i \in \{0, 1\}$

$$(-1)^s (d_6 d_5 d_4 d_3 d_2 d_1 d_0)_2 \quad \Longrightarrow \quad (-1)^s \sum_{i=0}^{6} d_i 2^i$$

El entero positivo más grande que se puede representar es

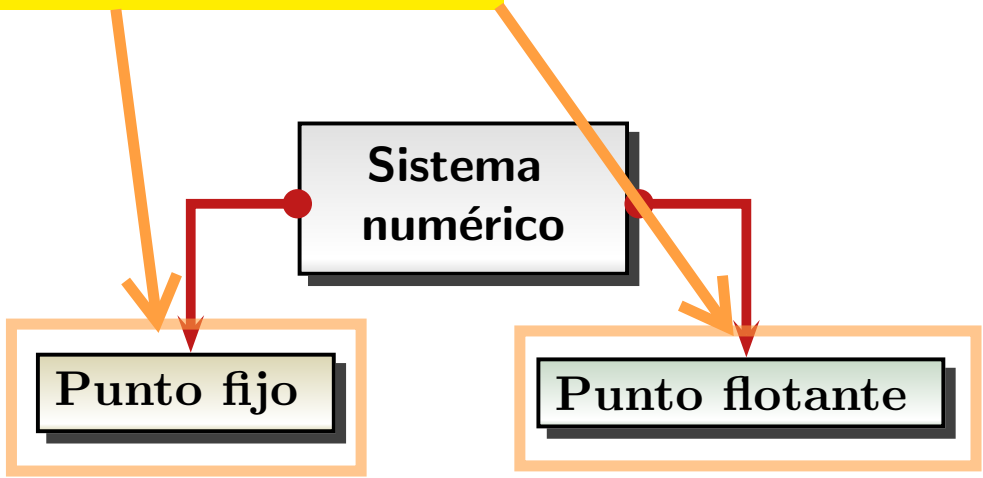


$$2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2 + 1 = 127$$

Para 32 bits, usan un bit para indicar el signo, el número positivo más grande que se puede representar es 2147483647.

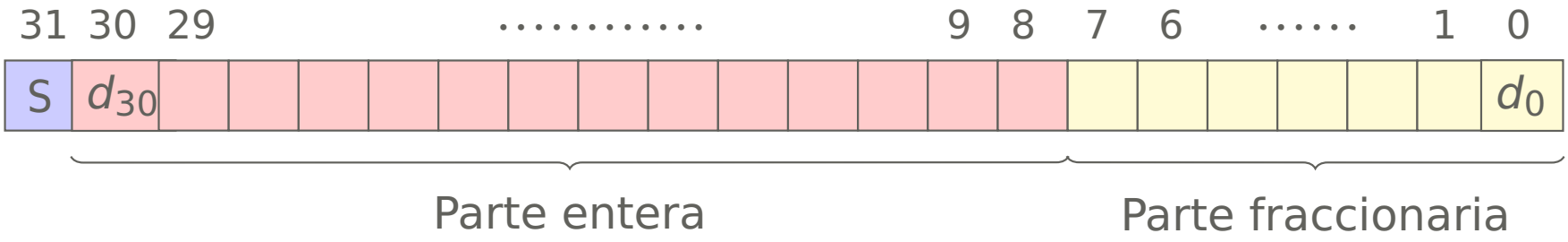
Representación de los números (III)

- Para representar los números reales en la computadora tenemos dos sistemas:



Números de punto fijo (I)

- Para los sistemas basados en punto fijo, se fija la cantidad de bits que se usan para representar la parte fraccionaria y la parte entera. Por ejemplo si se designan 32 bits de la siguiente manera:



$$(-1)^s \sum_{i=0}^{30} d_i \, 2^{i-8}.$$

Números de punto fijo (II)

Para esta representación el número positivo más grande corresponde a

$$+\underbrace{(11\cdots11)}_{23\ bits}.\underbrace{(11111111)}_{8\ bits}=8388607.99609375$$

Y el número positivo más pequeño es

$$+\underbrace{(00\cdots00)}_{23\ bits}.\underbrace{(00000001)}_{8\ bits}=0.00390625$$

Números de punto flotante (I)

Hay que representar los números que son de la forma:

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

Importante: Número positivo que no excede a la base y representa a la parte siginificativa del número.

donde la mantisa es un número positivo que no excede a la base.

$$\text{mantisa} = (d_0.d_1\, d_2 \cdots d_{p-1})_2 = d_0 + \frac{d_1}{2} + \frac{d_2}{2^2} + \cdots + \frac{d_{p-1}}{2^{p-1}}$$

Fija una cantidad de bits, y usando un bit para representar el signo de la cantidad, lo que resta es:

- Determinar cuantos bits se usan para representar a la mantisa
- Determinar cuantos bits se usan para representar al exponente e
- Decidir cómo representar el signo del exponente e.

Ejemplo: Usando 32 bits tomamos 1 bit para el signo, 23 bits para la mantisa y los 8 bits restantes para el exponente y su signo:

Números de punto flotante (II)

Respecto al signo del exponente

- Si usamos 1 bit para representar el signo del exponente y 7 bits para el valor del exponente, el rango de valores de e es de -127 a $+127$.

  Esta sería la forma más directa, pero tiene un inconveniente.

- Otra opción es aplicar un *formato con bias*, en el que $e = c - 127$ donde c usa los 8 bits y su rango de valores es de 0 a 255, por lo que el rango de valores e es de -127 a $+128$.

Bajo este esquema, el número positivo más grande que se puede representar es

$$+(\underbrace{1.111\cdots11}_{23\ bits})_2 \times 2^{128} \approx 6.805646 \times 10^{38}$$

Y el número positivo más pequeño es

$$+(\underbrace{0.000\cdots01}_{23\ bits})_2 \times 2^{-127} \approx 1.401298 \times 10^{-45}$$

IEEE Floating Point Standard 754-1985 (I)

El estándar de 1985 cubre los números de punto flotante de 32 y 64 bits, da uniformidad en el redondeo de números, el tratamiento de excepciones como la división por cero, la representación de 0 y de infinito, etc.

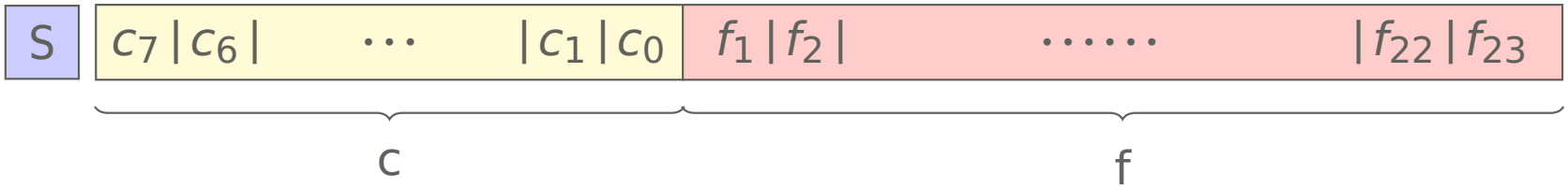
Para representar números reales binarios usando 32 bits (simple precisión) el estándar indica lo siguiente:

1. El signo es representado por un bit.
2. El exponente es almacenado en 8 bits, usando un *bias* de 127, de modo que $e = c - 127$, con c un entero no negativo.
3. La mantisa es almacenada en 23 bits.
4. Para incrementar la precisión de la mantisa, el estándar usa una *representación normalizada*, lo cual implica que su bit más significativo es siempre 1 si c es positivo.

IEEE Floating Point Standard 754-1985 (II)

$$(-1)^s\,(1.f)_2 \times 2^{c-127} \quad \text{si } c > 0,$$

donde f representa la parte fraccionaria de la mantisa $1 \leq 1.f < 2$.



$$c = \sum_{i=0}^{7} c_i 2^i, \qquad f = \sum_{1}^{23} \frac{f_i}{2^i}, \qquad c_i, f_i \in \{0, 1\}.$$

En general, para números normalizados, si tenemos q bits para el exponente, entonces

IEEE Floating Point Standard 754-1985 (III)

$$bias = 2^{q-1} - 1.$$

Ejemplo: El número 52.21875 tiene la siguiente representación usando 32 bits:

$$\begin{aligned} 52.21875 &= (110100.00111)_2 = (1.1010000111)_2 \times 2^5 \\ f &= 10100001110000000000000 \\ c - 127 &= 5 \\ c &= 132 = (10000100)_2 \end{aligned}$$

IEEE Floating Point Standard 754-1985 (IV)

Entonces la secuencia de bits para representar 52.21875 usando 32 bits en el formato de la IEEE es:

0	10000100	10100001110000000000000
s 1 bit	c 8 bits	f 23 bits

Representación de 0 (32 bits)

Dado que se supone que la mantisa tiene un 1 en la parte entera cuando $c > 0$, para representar al cero debemos tener que $c = 0$ y se hace el convenio de que en ese caso la parte entera de la mantisa es 0, por lo que f también debe ser cero:

$$(-1)^s\,(0.f)_2 \times 2^{c-127} \quad \text{si } c == 0.$$

$+0$

0	00000000	00000000000000000000000
s 1 bit	c 8 bits	f 23 bits

-0

1	00000000	00000000000000000000000
s 1 bit	c 8 bits	f 23 bits

Representación de infinito (32 bits)

Para representar infinito, todos los bits de c deben ser 1 y $f = 0$

$+\infty$

0	11111111	00000000000000000000000
s 1 bit	c 8 bits	f 23 bits

$-\infty$

1	11111111	00000000000000000000000
s 1 bit	c 8 bits	f 23 bits

Hay que tomar en cuenta que con un número $n > 0$ se tiene que

$$n/(\pm\infty) = 0, \quad \pm n/0 = \pm\infty, \quad \infty + \infty = \infty.$$

Representación de "Not a Number" (32 bits)

Cuando se realiza una operación entre números cuyo resultado no está definido o es indeterminado, hay que indicar que esto ocurrió usando que todos los dígitos de c sean 1 y $f \neq 0$. Por ejemplo

NaN

0	11111111	00001000000000000000000
s 1 bit	c 8 bits	f 23 bits

NaN

1	11111111	10000000010000000000000
s 1 bit	c 8 bits	f 23 bits

Operaciones que producen NaN:

$$\pm 0/\pm 0, \quad \infty - \infty, \quad \pm\infty/\pm\infty, \quad \pm\infty \times 0$$

Rango de números normalizados (32 bits) (I)

El número real positivo normal más grande es:

0	11111110	11111111111111111111111
s 1 bit	c 8 bits	f 23 bits

$$\begin{aligned}(1.f)_2 &= 2-2^{-23}\\ e &= 254-127=127\end{aligned}$$

Así, el número es

$$(2-2^{-23})\times 2^{127}\approx 3.403\times 10^{38}$$

Rango de números normalizados (32 bits) (II)

El número real positivo normal más pequeño es:

0	00000001	00000000000000000000000
s 1 bit	c 8 bits	f 23 bits

$$\begin{aligned}(1.f)_2 &= 1\\ e &= 1-127=-126\end{aligned}$$

Así, el número es

$$1.0\times 2^{-126}\approx 1.1755\times 10^{-38}$$

Números subnormales (32 bits)

Para representar un número que no está normalizado, es decir, que la parte entera de la mantisa es 0, el convenio es que todos los bits del exponente son 0 y $f \neq 0$. A estos números se llaman *subnormales*:

$$(-1)^s \, (0.f)_2 \times 2^{-126}$$

El número subnormal positivo más grande es $0.999999988 \times 2^{-126}$ y el más pequeño es:

0	00000000	00000000000000000000001
s 1 bit	c 8 bits	f 23 bits

Esto da

$$2^{-23} \times 2^{-126} = 2^{-149} \approx 1.401 \times 2^{-45}$$

Resumen del formato de la IEEE (I)

Para flotantes de simple precisión se usan 32 bits:

- 1 bit para el signo s
- 8 bits para calcular el término c del exponente
- 23 bits para la fracción f

Número	Condición	Expresión
Normal	$0 < c < 255$	$(-1)^s \times 1.f \times 2^{c-127}$
Subnormal	$c = 0,\ f \neq 0$	$(-1)^s \times 0.f \times 2^{-126}$
Cero	$c = 0,\ f = 0$	$(-1)^s \times 0.0$
Infinito	$c = 255,\ f = 0$	+INF, -INF
No es número	$c = 255,\ f \neq 0$	NaN

Resumen del formato de la IEEE (II)

Para flotantes de doble precisión se usan 64 bits:

- 1 bit para el signo s
- 11 bits para calcular el término c del exponente
- 52 bits para la fracción f

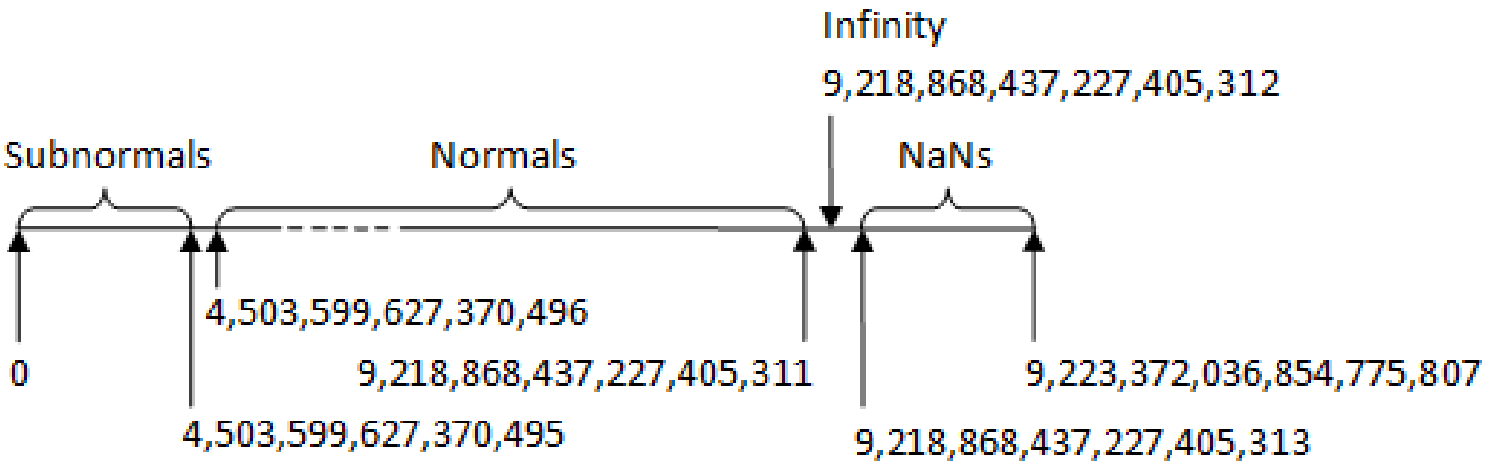
Número	Condición	Expresión
Normal	$0 < c < 2047$	$(-1)^s \times 1.f \times 2^{c-1023}$
Subnormal	$c = 0,\ \ f \neq 0$	$(-1)^s \times 0.f \times 2^{-1022}$
Cero	$c = 0,\ \ f = 0$	$(-1)^s \times 0.0$
Infinito	$c = 2047,\ f = 0$	`+INF`, `-INF`
No es número	$c = 2047,\ \ f \neq 0$	`NaN`

Ejemplo: Si usamos 64 bits para representar a los números en la computadora, tendremos

$$2^{64} = 18,446,744,073,709,551,616 \quad \text{"números"}$$

Resumen del formato de la IEEE (III)

Los números normalizados más chicos son del orden de 10^{-308} y los más grandes del orden 10^{308}. Los subnormales pueden más chicos que 10^{-323}.

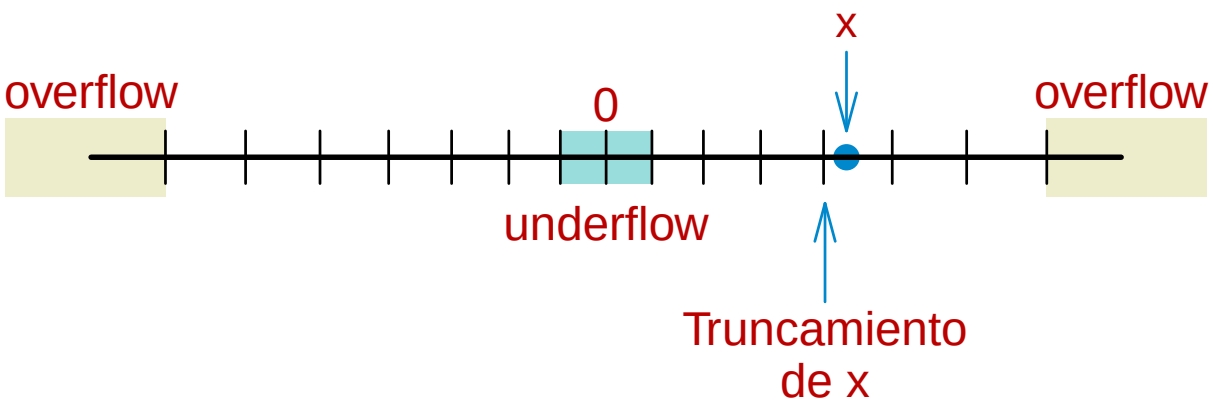


Para hacer pruebas, puede consultar:
`http://www.h-schmidt.net/FloatConverter/IEEE754.html`

Comentarios (I)

Sea x un número real. Dado que

- El exponente sólo tomar valores dentro de cierto conjunto finito.
- Como sólo hay un número finitos de valores, tanto para la fracción como para el exponente, sólo se pueden representar de forma precisa en la computadora una cantidad finita de números.
- Luego, con esos números se tienen que aproximar al resto.



- Números más pequeños que se pueden representar producen un subdesbordamiento (underflow) y se tratan como un cero a nivel máquina.

Comentarios (II)

- Números más grandes que los que se pueden representar producen un desbordamiento (overflow) y se genera un error que detiene los cálculos, a menos que se maneje la excepción.

Denotamos por $fl(x)$ a la representación en la computadora del número real x, la cual se obtiene por truncamiento o redondeo hacia arriba de x.

$$fl(x) = x + \epsilon$$