FACULTAD DE INGENIERIA
Universidad de Buenos Aires

86.01 Técnica Digital

Sistemas Numéricos

Ing. Jorge H. Fuchs

Introducción

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Objetivos de la clase:

Comprender la representación de los números en distintos sistemas de numeración y las conversiones entre los mismos.

Introducir los elementos que definen un sistema numérico posicional.

Realizar operaciones aritméticas básicas en binario.

Plantear las diferentes formas de representación de los números negativos. Emplear las mismas para realizar operaciones aritméticas.

Interpretar las funciones de los indicadores (flags) de las unidades aritméticas: transporte, rebalse, signo, paridad, cero, etc.

Definiciones

Definiremos un **sistema de numeración** como un conjunto de símbolos y reglas de generación que permitan construir todos los números válidos del sistema. Un sistema de numeración puede representarse como:

$$\mathbf{N = S + R}$$

- **N** es el sistema de numeración considerado.
- **S** son los símbolos permitidos del sistema.
- **R** son las reglas de generación que nos indican que números son válidos y cuales no en el sistema.

Dentro de los sistemas numéricos podemos distinguir:

No posicionales

Posicionales, pesados o ponderados

Sistemas numéricos No Posicionales

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

No Posicionales o No Ponderados son aquellos en los cual el valor de los símbolos que componen el sistema es fijo, o sea no depende de la posición que ocupe este dentro del número, ejemplo de este tipo es el sistema romano.

Sistemas numéricos Posicionales

Posicionales, Pesados o Ponderados son aquellos que el valor de los símbolos depende del valor que se les ha asignado y también de la posición que ocupa el símbolo dentro del número.

Podemos decir que cada símbolo tiene dos valores, uno **absoluto** que depende del valor del signo y otro **relativo** que depende de la posición que ocupa dentro del mismo.

Dígito es cada uno de los símbolos diferentes que constituyen el sistema.

Un dígito binario se denomina **bit**, que proviene del inglés "**Bi**nary Dig**it**".

Base es la cantidad de dígitos que lo forman. Las cantidades mayores a la base se obtienen combinando en forma adecuada los diferentes dígitos del sistema.

$$3435{,}62_{|10} = 3000 + 400 + 30 + 5 + 0{,}6 + 0{,}02$$

$$3435{,}62_{|10} = 3 \,.\, 10^{3} + 4 \,.\, 10^{2} + 3 \,.\, 10^{1} + 5 \,.\, 10^{0} + 6 \,.\, 10^{-1} + 2 \,.\, 10^{-2}$$

Sistemas numéricos Posicionales

Generalizando, un número **N** de **n** dígitos enteros y **k** dígitos fraccionarios, expresado en la base **B**, será:

$$N_B = a_{-k} \,.\, B^{-k} + a_{-k+1} \,.\, B^{-k+1} + \ldots\ldots\ldots + a_0 \,.\, B^0 + \ldots\ldots\ldots + a_{n-2} \,.\, B^{n-2} + a_{n-1} \,.\, B^{n-1}$$

$$N_B = \sum_{i=-k}^{n-1} a_i \,.\, B^i$$

Representación en otras bases

En toda base **B** tengo **B símbolos**, los números se van formando tal como en decimal.

La base **siempre** se escribe 10.

En **hexadecimal** (16) necesito 6 símbolos más, uso las **6 primeras letras**.

A mayor base, menor cantidad de dígitos.

Bases						
10	**2**	**3**	**4**	**7**	**8**	**16**
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	**10**	2	2	2	2	2
3	11	**10**	3	3	3	3
4	100	11	**10**	4	4	4
5	101	12	11	5	5	5
6	110	20	12	6	6	6
7	111	21	13	**10**	7	7
8	1000	22	20	11	**10**	8
9	1001	100	21	12	11	9
10	1010	101	22	13	12	A
11	1011	102	23	14	13	B
12	1100	110	30	15	14	C
13	1101	111	31	16	15	D
14	1110	112	32	20	16	E
15	1111	120	33	21	17	F
16	10000	121	100	22	20	**10**
17	10001	122	101	23	21	11
18	10010	200	102	24	22	12
19	10011	201	103	25	23	13

Dígitos binarios y conjuntos de bits

Los circuitos lógicos binarios trabajan con señales eléctricas que solo pueden tener dos estados posibles. En **lógica positiva** los estados se representan con dos niveles de tensión eléctrica: **Bajo** (0) y **Alto** (1).

Un dígito binario se denomina **bit**, que proviene del inglés “**Bi**nary Dig**it**”.

Los conjuntos de 4, 8, 16, 32 bits reciben denominaciones particulares según veremos a continuación o también “**palabra** de n bits”.

Los conjuntos de bits son utilizados para **representar** números, letras, símbolos, etc.

Dígitos binarios y conjuntos de bits

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Tamaño	Valor mínimo y máximo	Denominación
1 bit	0 – 1	Bit
4 bits	0000 – 1111	Nibble
8 bits	0000 0000 – 1111 1111	Byte
16 bits	0000 0000 0000 0000 – 1111 1111 1111 1111	Word
32 bits	0000 0000 0000 0000 0000 0000 0000 0000 – 1111 1111 1111 1111 1111 1111 1111 1111	Double Word

Cambio de otra base a decimal

Utilizando esta **representación polinómica** podemos convertir un número expresado en una base cualquiera al sistema de numeración decimal.

Por ejemplo para el número binario $1010{,}01_{|2}$, tenemos k=2, n=4 y B=2:

$$
\begin{aligned}
1010{,}01_{|2} &= 1 \times 2^{3} + 0 \times 2^{2} + 1 \times 2^{1} + 0 \times 2^{0} + 0 \times 2^{-1} + 1 \times 2^{-2} = \\
&= (8) + (0) + (2) + (0) + (0/2) + (1/4) = 10{,}25
\end{aligned}
$$

Para el número expresado en base 3 $201{,}02_{|3}$, tenemos k=2, n=3 y B=3:

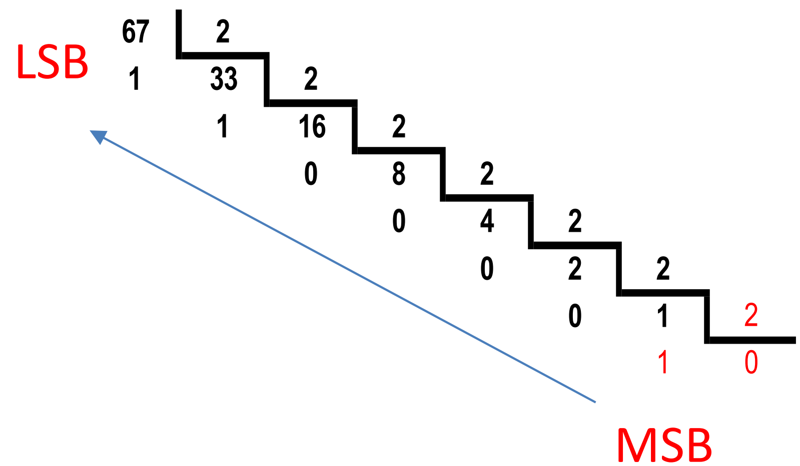
$$
\begin{aligned}
201{,}02_{|3} &= 2 \times 3^{2} + 0 \times 3^{1} + 1 \times 3^{0} + 0 \times 3^{-1} + 2 \times 3^{-2} = \\
&= (18) + (0) + (1) + (0/3) + (2/9) = 19{,}22222222
\end{aligned}
$$

Definición de MSB y LSB, MSD y LSD.

Cambio de decimal a otra base

Podemos convertir cualquier número decimal a otra base mediante el siguiente método, lo veremos con un ejemplo $67{,}74_{|10}$ a base 2.

Tomamos la **parte entera** y calculamos los residuos de sucesivas divisiones enteras por la base de llegada:



Teniendo en cuenta el sentido (der. a izq.) tenemos: $67 = 1000011_{|2}$

Podemos verificarlo:

$$1000011_{|2} = 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 67$$

Cambio de decimal a otra base

Para la **parte fraccionaria** multiplicamos por la base de llegada y la parte entera del producto la agregamos al número, luego seguimos multiplicando la parte fraccionaria:

0,74 x 2 = 1,48	>>>>>	1
0,48 x 2 = 0,96	>>>>>	0
0,96 x 2 = 1,92	>>>>>	1
0,92 x 2 = 1,84	>>>>>	1

Entonces obtenemos:

$$67{,}74 = 1000011{,}1011_{|2}$$

Pero hasta qué cifra seguimos si los restos no se hacen cero? Depende del error de truncamiento que queramos, será menor que el peso del último dígito a la derecha: $\epsilon < B^{-k}$

En este caso tenemos B = 2 y k = 4, por lo tanto $\epsilon < 2^{-4}$

Cambio de decimal a otra base

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Ahora pasemos el mismo número $67,74_{|10}$ a base 5, pero como requisito adicional **ϵ < 1/100**:

	67	5		
LSD	2	13	5	
		3	2	5
			2	0
			MSD	

Tenemos: $67 = 232_{|5}$

Podemos verificarlo:

$$232_{|5} = 2 \times 5^2 + 3 \times 5^1 + 2 \times 5^0 = 67$$

Para la parte fraccionaria:

0,74 x 5 = 3,7	>>>>>	3
0,7 x 5 = 3,5	>>>>>	3
0,5 x 5 = 2,5	>>>>>	2

Resultando: $67,74 = 232,332_{|5}$ con $\epsilon < 5^{-3} = 1/125 < 1/100$

Base potencia de otra

Para pasar por ejemplo de base 7 a base 5, puedo hacerlo de forma directa pero operando en la base de llegada, por lo tanto lo más recomendable es pasar de base 7 a base 10 y luego de base 10 a base 5.

En el caso particular de una base que es potencia de la otra, $\mathbf{B_2 = B_1^k}$, puedo usar un método más sencillo: utilizando **k** bits para representar en binario un dígito del sistema de numeración de base $\mathbf{2^k}$, siendo **k** un número entero, por lo que para base $\mathbf{8 = 2^3}$ se requieren tres bits por dígito, en tanto que para base $\mathbf{16 = 2^4}$ se requieren cuatro bits por dígito.

Ejemplo: pasar $346{,}D_{|H}$ a binario y a octal.

3	4	6	,	D $_{\|H}$	
0 0 1 1	0 1 0 0	0 1 1 0	,	1 1 0 1	$_{\|2}$

0 0 1	1 0 1	0 0 0	1 1 0	,	1 1 0	1 0 0	$_{\|2}$
1	5	0	6	,	6	4	$_{\|8}$

Aritmética en otras bases

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

x	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	10	12	14	16	18
3	0	3	6	9	12	15	18	21	24	27
4	0	4	8	12	16	20	24	28	32	36
5	0	5	10	15	20	25	30	35	40	45
6	0	6	12	18	24	30	36	42	48	54
7	0	7	14	21	28	35	42	49	56	63
8	0	8	16	24	32	40	48	56	64	72
9	0	9	18	27	36	45	54	63	72	81

+	0	1
0	0	1
1	1	10

x	0	1
0	0	0
1	0	1

+	0	1	2
0	0	1	2
1	1	2	10
2	2	10	11

x	0	1	2
0	0	0	0
1	0	1	2
2	0	2	11

Tablas de suma y producto en bases 10, 2 y 3

Aritmética binaria

Este ejemplo que veremos es muy simple, por ejemplo queremos **sumar** y **multiplicar** dos números binarios, $6 = 110_{|2}$ y $5 = 101_{|2}$.

$$
\begin{array}{cccc}
 & 1 & 1 & 0 \\
+ & 1 & 0 & 1 \\
\hline
1 & 0 & 1 & 1
\end{array}
\qquad\qquad
\begin{array}{cccccc}
 & & & 1 & 1 & 0 \\
 & & \times & 1 & 0 & 1 \\
\cline{4-6}
 & & & 1 & 1 & 0 \\
+ & 1 & 1 & 0 & 0 & \\
\cline{2-6}
 & 1 & 1 & 1 & 1 & 0
\end{array}
$$

Vemos que obviamente el resultado de la suma y la multiplicación dan 11 y 30 respectivamente.

Los sistemas digitales normalmente utilizan un algoritmo para realizar el producto, **no necesitan la tabla del producto binario**, solo la de suma.

Suma y resta binaria

Suma: ya vimos la tabla de suma binaria, aparece el "me llevo" a la columna inmediata a la izquierda (carry).

+	0	1
0	0	1
1	1	10

A	B	Suma	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Resta: de la misma forma en la resta aparece el "pido prestado" de la columna inmediata a la izquierda (borrow).

A	B	Resta	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Complementos

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

<u>Complemento a la base (o al módulo)</u>: (o complemento a **10**) es lo que le falta a un número para alcanzar a una potencia de la base.

$$C(A) = B^n - A$$

Ejemplo: $C_{10}(345) = 10^3 - 345 = 655$

$$\begin{array}{rr} & 862 \\ - & \underline{345} \\ & 517 \end{array} \qquad \begin{array}{r} 862 \\ +\underline{655} \\ 1517 \end{array}$$

<u>Complemento a la base (o al módulo) menos 1</u>: (o complemento a **9**) es lo que le falta a un número para alcanzar a una potencia de la base menos 1.

$$C(A) = (B^n - 1) - A$$

$C_9(345) = (10^3 - 1) - 345 = 999 - 345 = 654$

$$\begin{array}{rr} & 862 \\ - & \underline{345} \\ & 517 \end{array} \qquad \begin{array}{r} 862 \\ +\underline{654} \\ 1516 \\ \underline{+1} \\ 517 \end{array}$$

Reglas: Complemento dígito a dígito.

Si hay acarreo lo debo sumar.

Complementos en binario

Complemento a la base (o al módulo): (o complemento a **2**) es lo que le falta a un número para alcanzar a una potencia de la base.

$$C(A) = B^n - A$$

Ejemplo: $C_2(0101) = 2^4 - 0101_{|2} = 10000_{|2} - 0101_{|2} = 1011_{|2}$

Regla: recorro de derecha a izquierda, el primer uno que encuentro queda como está y después invierto todos los bits.

Complemento a la base (o al módulo) menos 1: (o complemento a **1**) es lo que le falta a un número para alcanzar a una potencia de la base menos 1.

$$C(A) = (B^n - 1) - A$$

Ejemplo: $C_1(0101) = (2^4 - 1) - 0101_{|2} = 1111_{|2} - 0101_{|2} = 1010_{|2}$

Regla: complemento bit a bit.

Otra forma: $\mathbf{C_{B-1} = C_B - 1}$ >>>>> $\mathbf{C_B = C_{B-1} + 1}$

$C_2(0101) = C_1(0101) + 1 = 1010_{|2} + 1 = 1011_{|2}$

Representación de números negativos

Convención: para representar números negativos los sistemas digitales utilizan el bit más significativo para indicar el signo, si toma el valor 0 interpreta que es un número positivo, y si es 1 que corresponde a uno negativo.

Convenciones de representación: complementos permiten realizar restas mediante sumas.

Ejemplo: 1011 representa 11, - 3, - 5, o - 4 según cada uno de los casos.

Número binario	Convención			
	Valor Abs.	BS y VA	C_2	C_1
0000	0	+0	0	+0
0001	1	1	1	1
0010	2	2	2	2
0011	3	3	3	3
0100	4	4	4	4
0101	5	5	5	5
0110	6	6	6	6
0111	7	7	7	7
1000	8	-0	-8	-7
1001	9	-1	-7	-6
1010	10	-2	-6	-5
1011	11	-3	-5	-4
1100	12	-4	-4	-3
1101	13	-5	-3	-2
1110	14	-6	-2	-1
1111	15	-7	-1	-0
Rango de representación	$\{0 ; + 2^n - 1\}$	$\{- 2^{n-1} - 1 ; + 2^{n-1} - 1\}$ $\exists +0 \wedge -0$	$\{- 2^{n-1} ; + 2^{n-1} - 1\}$	$\{- 2^{n-1} - 1 ; + 2^{n-1} - 1\}$ $\exists +0 \wedge -0$

Flags o banderas de la UAL

Indican características del resultado de una operación. Los circuitos directamente realizan la operación, la interpretación del resultado la hace el usuario o el programador del sistema digital. Dentro de un sistema microprocesado, estos indicadores generalmente forman parte de un registro llamado de estado o de códigos de condición.

C: Carry (acarreo): Cuando sumo dos números de **n bits** puede ocurrir que necesite **n + 1 bits,** este bit adicional es el **Carry.** Es importante tener el concepto que es **independiente** de que los operandos tengan signo o no. O sea que se puede producir sumando números con signo o sin signo.

V: Overflow (desbordamiento o rebalse): Este fenómeno ocurre solo cuando trabajo con números **con signo.** Sucede solo cuando sumo dos positivos o dos negativos y el resultado tiene el signo contrario, es decir que se cambia el **bit de signo,** lo cual es una incoherencia. Nunca existirá Overflow en la suma de un número negativo y un número positivo.
Si bien la UAL siempre lo informa, cuando sumo números sin signo puedo ignorarlo.

Flags o banderas de la UAL

Z: Zero (cero): Esta bandera indica que el resultado es cero.

N: Negative (negativo): También llamado S (sign), indica el signo del resultado.

P: Parity (paridad): es una convención, por ejemplo usando paridad **par** este bit se pone en 1 si la cantidad de unos del resultado es impar**,** y se pone en 0, si la cantidad de unos del resultado es par**.** Esto es así para que toda la palabra completa (resultado más bit de paridad) **tenga una cantidad par de unos.** Lo contrario ocurre usando paridad **impar.**

H: Half-Carry (acarreo de primer nibble): En el caso de realizar operaciones de suma con números binarios de 8 bits, esta bandera indica que se produjo un acarreo en la suma de los 4 bits menos significativos.

Operaciones con complementos

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

$$\begin{array}{r} 2 \\ +\underline{3} \\ 5 \end{array} \qquad \begin{array}{r} 0010 \\ +\underline{0011} \\ 0101 \end{array} \qquad \begin{array}{l} C=0 \\ V=0 \\ N=0 \\ Z=0 \end{array}$$

$$\begin{array}{r} 5 \\ +\underline{6} \\ 11 \end{array} \qquad \begin{array}{r} 0101 \\ +\underline{0110} \\ 1011 \end{array} \qquad \begin{array}{l} C=0 \\ V=1 \\ N=1 \\ Z=0 \end{array}$$

Sin signo **ignoro** el Overflow!!!

$\underline{C_1}$:

$$\begin{array}{r} 4 \\ -\underline{1} \\ 3 \\ \\ \\ \end{array} \qquad \begin{array}{r} 0100 \\ +\underline{1110} \\ 10010 \\ \underline{+1} \\ 0011 \end{array} \qquad \begin{array}{l} C=1 \\ V=0 \\ N=0 \\ Z=0 \\ \\ \end{array}$$

En C_1 debo **sumar** el Carry!!!
(desventaja pero más fácil complementar)

$\underline{C_2}$:

$$\begin{array}{r} 4 \\ -\underline{1} \\ 3 \end{array} \qquad \begin{array}{r} 0100 \\ +\underline{1111} \\ 10011 \end{array} \qquad \begin{array}{l} C=1 \\ V=0 \\ N=0 \\ Z=0 \end{array}$$

En C_2 **no sumo** el Carry!!!
(ventaja pero más difícil complementar)

Operaciones con complementos

C_1:

$$\begin{array}{r} 3 \\ -\underline{5} \\ -2 \end{array} \qquad \begin{array}{r} 0011 \\ +\underline{1010} \\ 1101 \end{array} \qquad \begin{array}{l} C=0 \\ V=0 \\ N=1 \\ Z=0 \end{array}$$

C_2:

$$\begin{array}{r} 3 \\ -\underline{5} \\ -2 \end{array} \qquad \begin{array}{r} 0011 \\ +\underline{1011} \\ 1110 \end{array} \qquad \begin{array}{l} C=0 \\ V=0 \\ N=1 \\ Z=0 \end{array}$$

C_1:

$$\begin{array}{r} -4 \\ -\underline{2} \\ -6 \end{array} \qquad \begin{array}{r} 1011 \\ +\underline{1101} \\ 11000 \\ \underline{+1} \\ 1001 \end{array} \qquad \begin{array}{l} C=1 \\ V=0 \\ N=1 \\ Z=0 \end{array}$$

C_2:

$$\begin{array}{r} -4 \\ -\underline{2} \\ -6 \end{array} \qquad \begin{array}{r} 1100 \\ +\underline{1110} \\ 11010 \end{array} \qquad \begin{array}{l} C=1 \\ V=0 \\ N=1 \\ Z=0 \end{array}$$

En C_1 debo **sumar** el Carry!!!

En C_2 **no sumo** el Carry!!!

Debo **complementar** siempre los resultados **negativos**!!!

$C_1(1001) = 0110$

$C_2(1010) = 0110$

En ambos casos el resultado es: – 6

Operaciones con complementos

C_1:

$$\begin{array}{r} 3 \\ -\underline{3} \\ 0 \end{array} \qquad \begin{array}{r} 0011 \\ +\underline{1100} \\ 1111 \end{array} \qquad \begin{array}{l} C = 0 \\ V = 0 \\ N = 1 \\ Z = 0 \end{array}$$

C_2:

$$\begin{array}{r} 3 \\ -\underline{3} \\ 0 \end{array} \qquad \begin{array}{r} 0011 \\ +\underline{1101} \\ 10000 \end{array} \qquad \begin{array}{l} C = 1 \\ V = 0 \\ N = 0 \\ Z = 1 \end{array}$$

Vemos que la **misma cuenta** en complementos distintos arroja valores de flags distintos.

La UAL no interpreta resultados más allá de los Flags, para un mismo par de sumandos, los **resultados** deben interpretarse según la **convención** que se esté utilizando. Por ejemplo:

$$\begin{array}{r} 0101 \\ +\underline{1110} \\ 10011 \end{array} \qquad \begin{array}{l} C = 1 \\ V = 0 \\ N = 0 \\ Z = 0 \end{array}$$

Si es una suma de números **sin signo** (5+14), considerando el Carry el resultado es 19, si se trata de **C_1** (5-1), debo sumar el Carry obteniendo como resultado 4, y si se trata de **C_2** (5-2), el resultado es 3.

Punto fijo

Ejemplo: utilizo 1 byte para representar un número positivo fraccionario

$$\begin{array}{ccccccccc} 1 & 0 & 1 & 0 & . & 1 & 0 & 1 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & \downarrow \\ 2^{3} & 2^{2} & 2^{1} & 2^{0} & & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} \end{array}$$

$$8+2 = 10 \; + \; 0{,}5+0{,}125+0{,}0625 = 0{,}6875$$

$$10{,}6875$$

Puedo representar los siguientes rangos:

$$\{0\} \; ; \; \{ + 0{,}0001_{|2} \; ; \; + 1111{,}1111_{|2}\}$$

Punto flotante

En muchos cálculos el intervalo de números que se usan es muy grande.

La masa del electrón, 9×10^{-28} gramos

0000000000000000000000000000000000.0000000000000000000000000009

La masa del Sol, 2×10^{33} gramos

2000000000000000000000000000000000.0000000000000000000000000000

$$\mathbf{n = f \times 10^{e}}$$

$$\mathbf{\pm\ 1{,}Mantisa \times 2^{E}}$$

Punto flotante

Norma IEEE 754:

Simple precisión: 32 bits (1 + 8 + 23)

signo	exponente	mantisa
1 bit	8 bits	23 bits

Doble precisión: 64 bits (1 + 11 + 52)

signo	exponente	mantisa
1 bit	11 bits	52 bits

Sumas y más sumas

Cuáles de las siguientes operaciones son correctas?

$1 + 1 = 2$	Suma aritmética $B > 2$
$1 + 1 = 10$	Suma aritmética $B = 2$
$1 + 1 = 1$	Suma lógica

FACULTAD DE INGENIERIA
Universidad de Buenos Aires