Arquitectura de ordenadores

4-2. Aritmética III

Ignacio Calles González ignacio.gonzalez@ext.live.u-tad.com

Tiago Manuel Louro Machado de Simas tiago.louro@u-tad.com

Francisco Javier García Algarra javier.algarra@u-tad.com

Carlos M. Vallez Fernández carlos.vallez@u-tad.com

2023-2024

Índice

1. Multiplicación con signo. Algoritmo de Booth

- El algoritmo de multiplicación de Booth para números con signo fue creado por Andrew Booth en 1950
- Tras una observación sobre cómo trabajaban los calculadores humanos para hacer más rápida la multiplicación con sus equipos mecánicos.
- Para ello empleaban el complemento a 10 de las cifras del multiplicando.
- El algoritmo usa dos pasos. suma y desplazamiento, guiándose por cuatro posibles condiciones.
- Veamos primeramente el algoritmo y luego haremos un ejemplo.

1. Multiplicación con signo. Algoritmo de Booth

El algoritmo trabaja con los números en **complemento a dos**.

Se inicializan tres registros M, S y Q con el multiplicando, su **complemento a dos** y el multiplicador.

Se inicializa un registro A a 0 y un registro de un único bit Q_{-1} a 0.

Si Q_0 y Q_{-1} no son iguales (0-1, 1-0):

Si 0,1: Se suma A + M y se almacena en A. El acarreo se pierde.

Si 1,0: Se suma A + S y se almacena en A. El acarreo se pierde.

Se desplaza un bit a la derecha A, Q, y Q_{-1}. (Se rellena A extendiendo el signo)

$A_0 \rightarrow Q_{n-1}$

$Q_0 \rightarrow Q_{-1}$.

Q_{-1} se pierde.

Si Q_0 y Q_{-1} son iguales (1-1, 0-0): sólo se realiza desplazamiento de A, Q, y Q_{-1}.

Se vuelve a 3 hasta que se agoten los bits del multiplicador original.

El resultado del producto es la concatenación de A y Q.

1. Multiplicación con signo. Algoritmo de Booth

La mejor forma de entender el algoritmo y dominarlo es practicarlo con ejemplos.

A continuación vamos a multiplicar 7 x 3 con n=4. Es decir, multiplicar 0111(7) x 0011 (3):

MULTIPLICACIÓN CON SIGNO: ALGORITMO DE BOOTH

- Multiplicar 0111(7) x 0011 (3)   n = 4 bits

1. Se inicializan tres registros M, S y Q con el multiplicando, su **complemento a dos** y el multiplicador.
2. Se inicializa un registro A a 0 y un registro de un único bit Q_{-1} a 0.

A	Q	Q_{-1}	M	S
0000	0011	0	0111	1001

MULTIPLICACIÓN CON SIGNO: ALGORITMO DE BOOTH

De forma resumida se puede decir que las reglas son:

Q_0Q_1 =10 => sumar S a A y desplazar

Q_0Q_1 =00 ó 11 => Sólo desplazar

Q_0Q_1 =01 => sumar M a A y desplazar

- Multiplicar 0111(7) x 0011 (3) n = 4 bits

  3.2. Si 1,0: Se suma A + S y se almacena en A. El acarreo se pierde.

A	Q	Q_{-1}	M	S
0000	0011	0	0111	1001
1001	0011	0	0111	1001

MULTIPLICACIÓN CON SIGNO: ALGORITMO DE BOOTH

- Multiplicar 0111(7) x 0011 (3) n = 4 bits

    3.3. Se desplaza un bit a la derecha A, Q, y Q_{-1}.

se pierde

- Si A_{n-1} es 1, se rellena con 1.
- Si A_{n-1} es 0, se rellena con 0.

A	Q	Q_{-1}	M	S
0000	0011	0	0111	1001
1001	0011	0	0111	1001
1100	1001	1	0111	1001

MULTIPLICACIÓN CON SIGNO: ALGORITMO DE BOOTH

- Multiplicar 0111(7) x 0011 (3)   n = 4 bits

4. Si Q_0 y Q_{-1} son iguales (1-1, 0-0): sólo se realiza desplazamiento de A, Q, y Q_{-1}.

se pierde

A	Q	Q_{-1}	M	S
0000	0011	0	0111	1001
1001	0011	0	0111	1001
1100	1001	1	0111	1001
1110	0100	1	0111	1001

- Si A_{n-1} es 1, se rellena con 1.
- Si A_{n-1} es 0, se rellena con 0.

MULTIPLICACIÓN CON SIGNO: ALGORITMO DE BOOTH

- Multiplicar 0111(7) x 0011 (3)   n = 4 bits

    3.1. Si 0,1: Se suma A + M y se almacena en A. El acarreo se pierde.

el
acarreo
se pierde

A	Q	Q_{-1}	M	S
0000	0011	0	0111	1001
1001	0011	0	0111	1001
1100	1001	1	0111	1001
1110	0100	1	0111	1001
0101	0100	1	0111	1001

MULTIPLICACIÓN CON SIGNO: ALGORITMO DE BOOTH

- Multiplicar 0111(7) x 0011 (3) n = 4 bits

3.3. Se desplaza un bit a la derecha A, Q, y Q_{-1}.

se pierde

A	Q	Q_{-1}	M	S
0000	0011	0	0111	1001
1001	0011	0	0111	1001
1100	1001	1	0111	1001
1110	0100	1	0111	1001
0101	0100	1	0111	1001
0010	1010	0	0111	1001

- Si A_{n-1} es 1, se rellena con 1.
- Si A_{n-1} es 0, se rellena con 0.

MULTIPLICACIÓN CON SIGNO: ALGORITMO DE BOOTH

- Multiplicar 0111(7) x 0011 (3) n = 4 bits

4. Si Q_0 y Q_{-1} son iguales (1-1, 0-0): sólo se realiza desplazamiento de A, Q, y Q_{-1}.

se pierde

A	Q	Q_{-1}	M	S
0000	0011	0	0111	1001
1001	0011	0	0111	1001
1100	1001	1	0111	1001
1110	0100	1	0111	1001
0101	0100	1	0111	1001
0010	1010	0	0111	1001
0001	0101	0	0111	1001

- Si A_{n-1} es 1, se rellena con 1.
- Si A_{n-1} es 0, se rellena con 0.

MULTIPLICACIÓN CON SIGNO: ALGORITMO DE BOOTH

- Multiplicar 0111(7) x 0011 (3) n = 4 bits

6. El resultado del producto es la concatenación de A y Q.

$\boxed{0001\ 0101 = 21}$

A	Q	Q_{-1}	M	S
0000	0011	0	0111	1001
1001	0011	0	0111	1001
1100	1001	1	0111	1001
1110	0100	1	0111	1001
0101	0100	1	0111	1001
0010	1010	0	0111	1001
0001	0101	0	0111	1001

1. Multiplicación con signo. Algoritmo de Booth

Ejercicio: Ahora es vuestro turno de realizarlo con un número positivo y otro negativo

A continuación vamos a multiplicar 6 x -73 con n=4. Es decir, multiplicar 0110(6) x 1001 (-7):

Nota: 7 es: 0111 .Para calcular -7 creo su complemento a dos: 1001

2. División con signo o con restauración.

Es un algoritmo empleado para realizar la división entre números enteros con signo. Trabaja con los números en **complemento a dos**.

Mostramos el algoritmo en la siguiente diapositiva

Índice

2. División con signo o con restauración.

1. Se inicializan dos registros M y S con el divisor y su **complemento a dos**.
2. Se inicializan dos registros A y Q con el dividendo en complemento a dos extendido al tamaño 2n.
3. Desplazar A y Q un bit a la izquierda.
   a) Si M y A tienen el mismo signo: Se suma A + S y se almacena en A. El acarreo se pierde.
   b) Si M y A tienen distinto signo: Se suma A + M y se almacena en A. El acarreo se pierde.
4. La operación tiene éxito si el signo de A no cambia.
   a) Si tiene éxito o (A = 0 y Q = 0), hacer Q_0 = 1.
   b) Si no tiene éxito y (A != 0 ó Q != 0), entonces hacer Q_0 = 0 y restablecer el valor anterior de A.
5. Se vuelve a 3 hasta que se agoten los bits del dividendo original.
6. El resto está en A.
   a) Si los signos del divisor y el dividendo eran iguales, el cociente está en Q.
   b) Si no, el cociente es el complemento a dos de Q

2. División con signo o con restauración.

Conviene reforzar antes de ver un ejemplo el concepto de “éxito” empleado dentro del algoritmo. **Se dice que una división parcial tiene éxito, si el signo del registro de acumulación no cambia.**

2. División con signo o con restauración.

La mejor forma de entender el algoritmo y dominarlo es practicarlo con ejemplos.

A continuación vamos a dividir 7 / 3 con n=4. Es decir, dividir 0111(7) entre 0011 (3):

DIVISÓN CON SIGNO

- Desplazar A y Q 1 bit a la izquierda
  - Si M y A tienen el mismo signo, A= A+S y se ignora el acarreo
  - Si signo(M) no es igual a signo(A), A= A+m y se ignora también el acarreo.
- La operación tiene éxito si el signo de A no cambia (actuar según proceda)

- Dividir 0111(7) / 0011 (3) n = 4 bits

1. Se inicializan dos registros M y S con el divisor y su **complemento a dos**.
2. Se inicializan dos registros A y Q con el dividendo extendido al tamaño 2n.

A	Q	M	S
0000	0111	0011	1101

DIVISÓN CON SIGNO

- Dividir 0111(7) / 0011 (3)   n = 4 bits

3. Desplazar A y Q un bit a la izquierda.

se pierde

A	Q	M	S
0000	0111	0011	1101
0000	1110	0011	1101

DIVISÓN CON SIGNO

- Dividir 0111(7) / 0011 (3) n = 4 bits

3.1. Si M y A tienen el mismo signo: Se suma A + S y se almacena en A. El acarreo se pierde.

A	Q	M	S
0000	0111	0011	1101
0000	1110	0011	1101
1101	1110	0011	1101

DIVISÓN CON SIGNO

- Dividir 0111(7) / 0011 (3) n = 4 bits

4.2. Si no tiene éxito y A $\neq$ 0, entonces hacer Q_0 = 0 y restablecer el valor anterior de A.

A	Q	M	S
0000	0111	0011	1101
0000	1110	0011	1101
1101	1110	0011	1101
0000	1110	0011	1101

DIVISÓN CON SIGNO

- Dividir 0111(7) / 0011 (3)   n = 4 bits

3. Desplazar A y Q un bit a la izquierda.

se pierde

A	Q	M	S
0000	0111	0011	1101
0000	1110	0011	1101
1101	1110	0011	1101
0000	1110	0011	1101
0001	1100	0011	1101

DIVISÓN CON SIGNO

- Dividir 0111(7) / 0011 (3) n = 4 bits

3.1. Si M y A tienen el mismo signo: Se suma A + S y se almacena en A. El acarreo se pierde.

A	Q	M	S
0000	0111	0011	1101
0000	1110	0011	1101
1101	1110	0011	1101
0000	1110	0011	1101
0001	1100	0011	1101
1110	1100	0011	1101

- Dividir 0111(7) / 0011 (3)   n = 4 bits

4.2. Si no tiene éxito y $A \neq 0$, entonces hacer $Q_0 = 0$ y restablecer el valor anterior de A.

A	Q	M	S
0000	0111	0011	1101
0000	1110	0011	1101
1101	1110	0011	1101
0000	1110	0011	1101
0001	1100	0011	1101
1110	1100	0011	1101
0001	1100	0011	1101

DIVISÓN CON SIGNO

- Dividir 0111(7) / 0011 (3) n = 4 bits

3. Desplazar A y Q un bit a la izquierda.

A	Q	M	S
0000	0111	0011	1101
0000	1110	0011	1101
1101	1110	0011	1101
0000	1110	0011	1101
0001	1100	0011	1101
1110	1100	0011	1101
0001	1100	0011	1101
0011	1000	0011	1101

se pierde

DIVISÓN CON SIGNO

- Dividir 0111(7) / 0011 (3) n = 4 bits

3.1. Si M y A tienen el mismo signo: Se suma A + S y se almacena en A. El acarreo se pierde.

A	Q	M	S
0000	0111	0011	1101
0000	1110	0011	1101
1101	1110	0011	1101
0000	1110	0011	1101
0001	1100	0011	1101
1110	1100	0011	1101
0001	1100	0011	1101
0011	1000	0011	1101
0000	1000	0011	1101

DIVISÓN CON SIGNO

- Dividir 0111(7) / 0011 (3) n = 4 bits

4.1 Si tiene éxito o A = 0, hacer $Q_0 = 1$.

A	Q	M	S
0000	0111	0011	1101
0000	1110	0011	1101
1101	1110	0011	1101
0000	1110	0011	1101
0001	1100	0011	1101
1110	1100	0011	1101
0001	1100	0011	1101
0011	1000	0011	1101
0000	1000	0011	1101
0000	1001	0011	1101

DIVISÓN CON SIGNO

- Dividir 0111(7) / 0011 (3) n = 4 bits

3. Desplazar A y Q un bit a la izquierda.

A	Q	M	S
0000	0111	0011	1101
0000	1110	0011	1101
1101	1110	0011	1101
0000	1110	0011	1101
0001	1100	0011	1101
1110	1100	0011	1101
0001	1100	0011	1101
0011	1000	0011	1101
0000	1000	0011	1101
0000	1001	0011	1101
0001	0010	0011	1101

se pierde

DIVISÓN CON SIGNO

- Dividir 0111(7) / 0011 (3) n = 4 bits

3.1. Si M y A tienen el mismo signo: Se suma A + S y se almacena en A. El acarreo se pierde.

A	Q	M	S
0000	0111	0011	1101
0000	1110	0011	1101
1101	1110	0011	1101
0000	1110	0011	1101
0001	1100	0011	1101
1110	1100	0011	1101
0001	1100	0011	1101
0011	1000	0011	1101
0000	1000	0011	1101
0000	1001	0011	1101
0001	0010	0011	1101
1110	0010	0011	1101

DIVISÓN CON SIGNO

- Dividir 0111(7) / 0011 (3) n = 4 bits

4.2 Si no tiene éxito y $A \neq 0$, entonces hacer $Q_0 = 0$ y restablecer el valor anterior de A.

A	Q	M	S
0000	0111	0011	1101
0000	1110	0011	1101
1101	1110	0011	1101
0000	1110	0011	1101
0001	1100	0011	1101
1110	1100	0011	1101
0001	1100	0011	1101
0011	1000	0011	1101
0000	1000	0011	1101
0000	1001	0011	1101
0001	0010	0011	1101
1110	0010	0011	1101
0001	0010	0011	1101

DIVISÓN CON SIGNO

- Dividir 0111(7) / 0011 (3) n = 4 bits
  1. El resto está en A.
     1. Si los signos del divisor y el dividendo eran iguales, el cociente está en Q.
     2. Si no, el cociente es el complemento a dos de Q.

A	Q	M	S
0000	0111	0011	1101
0000	1110	0011	1101
1101	1110	0011	1101
0000	1110	0011	1101
0001	1100	0011	1101
1110	1100	0011	1101
0001	1100	0011	1101
0011	1000	0011	1101
0000	1000	0011	1101
0000	1001	0011	1101
0001	0010	0011	1101
1110	0010	0011	1101
0001	0010	0011	1101

Cociente = 0010 (2)

Resto = 0001 (1)

2. División con signo o con restauración.

La mejor forma de entender el algoritmo y dominarlo es practicarlo con ejemplos.

A continuación vamos a dividir -7 / 3 con n=4. Es decir, dividir 1001(-7) entre 0011 (3):

2. División con signo o con restauración.

- Dividir 1001(-7) / 0011 (3) n = 4 bits

1. Se inicializan dos registros M y S con el divisor y su **complemento a dos**.
2. Se inicializan dos registros A y Q con el dividendo extendido al tamaño 2n (en este caso con 1 por ser de distinto signo).

- Ca2 (7) = 1001
- Ca2 (3) = 1101

A	Q	M	S
1111	1001	0011	1101

2. División con signo o con restauración.

- Dividir 1001(-7) / 0011 (3) n = 4 bits

	A	Q	M	S
Si los operandos son distinto signo	1111	1001	0011	1101
Desplazar A y Q un bit a la izquierda	1111	0010		
Signo A != signo M => A= A+M	~~1~~0010	0010		
1 NO EXITO	1111	0010		
Desplazar A y Q un bit a la izquierda	1110	0100		
Signo A != signo M => A= A+M	~~1~~0011	0100		
2 NO EXITO	1110	0100		
Desplazar A y Q un bit a la izquierda	1100	1000		
Signo A != signo M => A= A+M	1111	1000		
3 EXITO	1111	1001		
Desplazar A y Q un bit a la izquierda	1111	0010		
Signo A != signo M => A= A+M	0010	0010		
4 NO EXITO	1111	0010		

2. División con signo o con restauración.

- Dividir 1001(-7) / 0011 (3) n = 4 bits

Al dividir -7 entre 3 debería dar : -2 de cociente y -1 de resto. Sin embargo obtenemos:

Cociente = 0010 (2)

Al ser diferente signo será - Cociente

Resto = 1111 (-1)

Nota: 1111 es 0001 complemento a 2, es decir 1 en negativo

A	Q	M	S
1111	1001	0011	1101
1111	0010		
~~1~~0010	0010		
1111	0010		
1110	0100		
~~1~~0011	0100		
1110	0100		
1100	1000		
1111	1000		
1111	1001		
1111	0010		
0010	0010		
1111	0010		

La clase de división que produce un módulo no negativo es la división Euclídea:
-7/3 = -3 +2/3 = -3 + 0,6666 = -2,334
Si dividimos con calculadora -7/3= -2,33333

Para pasar de la forma "resto negativo" a la Euclídea:

q' = q-1 y r' = r+d

q'= -2 -1 = -3
R' = -1 +3 =2

2. División con signo o con restauración.

Las reglas se podrían resumir de la siguiente manera:

- Desplazar A y Q 1 bit a la izquierda
  - Si M y A tienen el mismo signo, A= A+S y se ignora el acarreo
  - Si signo(M) no es igual a signo(A), A= A+M y se ignora también el acarreo.
- La operación tiene éxito si el signo de A no cambia (actuar según proceda)
- Si signo dividendo =signo divisor al final => cociente = Q, en caso contrario cociente =Ca2(Q).

Índice

3. Representación en coma flotante

- Tanto la representación en S-M como el complemento a uno y complemento a dos sirven para representar rangos de números enteros negativos y positivos centrados en el cero.
- Las representaciones vistas hasta ahora se denominan “en punto fijo”.
- Tienen un inconveniente grave, no sirven para representar cantidades muy grandes o muy pequeñas.

3. Representación en coma flotante

- Asumiendo una coma fija en una determinada posición de la cadena de bits, se pueden representar números con parte fraccionaria: 10001,01; 11000,11; 11100,10; 10101,01
  - Como hemos comentado, esta representación tiene limitaciones:
    - No puede representar números muy grandes.
    - No puede representar fracciones muy pequeñas.

3. Representación en coma flotante

- Asumiendo una coma fija en una determinada posición de la cadena de bits, se pueden representar números con parte fraccionaria: 10001,01; 11000,11; 11100,10; 10101,01
    - Como hemos comentado, esta representación tiene limitaciones:
        - No puede representar números muy grandes.
        - No puede representar fracciones muy pequeñas.

- Para solucionarlo se recurre a la anotación en punto flotante, que no es otra cosa que la notación científica en base dos. Las representaciones en **coma flotante** permiten representar números positivos y negativos con parte fraccionaria sin estas limitaciones.

3. Representación en coma flotante

Vamos a recordar qué es eso de notación científica con dos ejemplos:

- Nº de Avogadro: 6.02 x 10 23
  - Donde 6.02 es la mantisa, 10 es la base y 23 es el exponente.
- Constante de Planck: 6,62 x 10 -34

Estas dos constantes aparecen normalizadas, es decir, hay una y solo una cifra significativa antes del punto decimal.

En punto flotante se usa la base 2. tiene la propiedad de que si la cantidad está normalizada. la cifra que hay antes del punto es siempre 1.

Ejemplo: 0.28125 en base 10 equivale a 0,01001 en base 2. Y para expresarlo en coma flotante pondríamos 1,001 x 2-2.

3. Representación en coma flotante

La representación en **coma flotante** más usual tiene el siguiente formato:

Signo	Exponente	Magnitud (mantisa)

El formato sigue la misma técnica que la **notación científica**:

- en decimal: $\pm M \times B^{Exp} = +3 \cdot 10^{-3} = 0{,}003$
- en binario: $\pm M \times B^{Exp} = +1 \cdot 2^{-3} = 0{,}125$

Donde +/- es el signo, M la magnitud, Exp el exponente y B la base (también llamada radix).

La **base** siempre es fija (en decimal B = 10, en binario B = 2), por lo que **no se representa**.

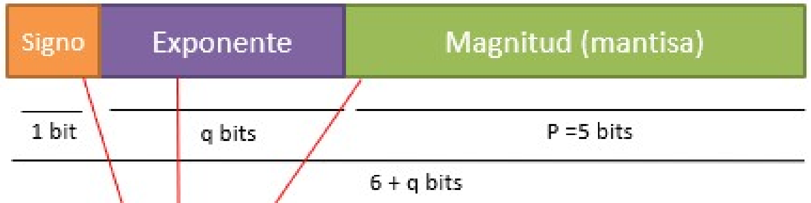
El resto de partes se pueden codificar mediante diferentes sistemas de representación.

3.1 Repres. coma flotante S-M

En este apartado vamos a ver cómo se representan cada uno de los campos de la coma flotante (signo, exponente y mantisa) en formato S-M.

3.1 Repres. coma flotante S-M

<u>Signo:</u> se representa con un bit. 1 es negativo y 0 es positivo.



$$0 \text{ EXP } 10101 = 2^4 + 2^2 + 2^0 = 21 \cdot 2^{EXP}$$

$$1 \text{ EXP } 10101 = -\left(2^4 + 2^2 + 2^0\right) = -21 \cdot 2^{EXP}$$

En este ejemplo se emplean q bits para el exponente y P =5 para la mantisa.

3.1 Repres. coma flotante S-M

Magnitud:

Las **magnitudes** se deben presentar **normalizadas**.

La coma se puede situar entre cualquier par de bits o en alguno de los extremos. En una magnitud normalizada, se sobreentiende que **la coma se sitúa a la izquierda del bit más significativo** (el más a la izquierda).

En S-M, **una magnitud está normalizada si el bit que acompaña al bit de signo es igual a 1.**

S-M	Valor	Normalizado
1 01010	- 0,01010	NO
0 01010	+ 0,01010	NO
1 10101	- 0,10101	SÍ
0 10101	+ 0,10101	SÍ

3.1 Repres. coma flotante S-M

Magnitud:

Como el **bit más significativo** de la magnitud siempre debe ser 1, nunca se representa, sino que se entiende que está **implícito**.

•

S-M codificado	S-M representado
0 01011	0 101011
1 01011	1 101011

1 + 5 bits

1 + 6 bits

Se gana un bit

3.1 Repres. coma flotante S-M

Magnitud:

Dicho de otra forma, el primer bit de la mantisa se suprime, se dice que va "implícito" y se gana un bit de precisión. En nuestro ejemplo:

0,28125 en decimal se representaría como 0 001 0010. Donde el primer 0 indica que es positivo, el 001 es el exponente que explicaremos seguidamente y el 0010 corresponde al 1001 al cual se ha quitado el primer 1 por considerarlo implícito (por tanto no se almacena) y a su vez hemos ganado más precisión con un cero más al final

3.1 Repres. coma flotante S-M

Magnitud:

Para un valor de p = 5 bits, el rango de representación con normalización y bit implícito es el siguiente:

		S-M	Valor
Negativos	Mayor	1 111111	$-\left(1-2^{-6}\right)$
	Menor	1 100000	-2^{-1}
Positivos	Menor	0 100000	2^{-1}
	Mayor	0 111111	$\left(1-2^{-6}\right)$

Bits implícitos

3.1 Repres. coma flotante S-M

Exponente:

Tiene q bits para el exponente. Ejemplo: p = 5 y q = 4

- Está codificado en **representación sesgada**, también llamada **exceso M**:
- El exponente puede ser negativo, pero el signo no se codifica en ningún bit.
- El valor codificado en el exponente es igual al valor en binario más un valor fijo llamado sesgo. El valor típico del sesgo es $2^{q-1}-1$. En el ejemplo, el sesgo de 4 es:
- El sesgo permite que se codifiquen exponentes negativos. Con q = 4 se puede codificar desde 0000 hasta 1111 en binario puro

3.1 Repres. coma flotante S-M

Exponente:

Codificado	Codificado	Exponente
0000	0	-7
1111	15	8

SESGO= PESO BIT MAS SIGNIFICATIVO – 1
COD=EXP+SESGO
EXP=COD-SESGO

- **Representación sesgada** también es llamada **exceso M.**
- Para un valor genérico de q bits, se tiene que el rango del exponente en representación sesgada es: $[\,-(2^{q-1} - 1), 2^{q-1}\,]$:

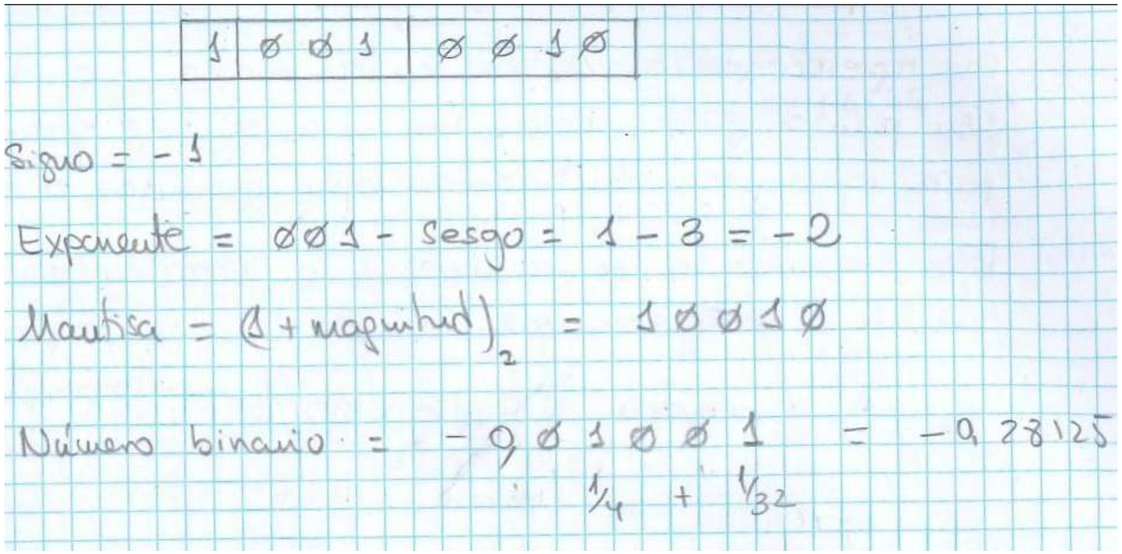
$$\left[-2^{q-1}-1, 2^{q-1}\right]$$

q	Rango en binario puro	Rango con sesgo
4	[0, 15]	[-7, 8]
8	[0, 255]	[-127,128]
q	$\left[0, 2^q - 1\right]$	$\left[-2^{q-1}+1, 2^{q-1}\right]$

3.1 Repres. coma flotante S-M

Veamos un primer ejemplo completo negativo:

representar -0,28125 en base 10 cómo sería en coma flotante y aprovechamos para recordar cómo pasar de coma flotante a decimal de nuevo:



| 1 | 0 0 1 | 0 0 1 0 |

Signo = -1

Exponente = 001 - sesgo = 1 - 3 = -2

Mantisa = $(1 + \text{magnitud})_2$ = 10010

Número binario = -0,01001 = -0,28125

1/4 + 1/32

0,28125 = 1/4 + 1/32 = 0,25 + 0,03125

EXP = COD – SESGO => COD= EXP – SESGO=-2 +3 =1 => 001

3.1 Repres. coma flotante S-M

- Ejemplo: p = 5, q = 4 y 1 bit de signo. Magnitud normalizada y con bit implícito.

Signo	Exponente	Magnitud (mantisa)
1 bit	4 bits	5 bits

10 bits

SESGO= PESO BIT MAS SIGNIFICATIVO – 1
COD=EXP+SESGO
EXP=COD-SESGO

- Rango de la representación en coma flotante:

		Coma flotante			Valor
		Signo	Exp sesgo 7	Magnitud normalizada con bit implícito	
Negativos	Mayor	1	1111	**1**11111	$-\left(1-2^{-6}\right)\cdot 2^{8}$
	Menor	1	0000	**1**00000	$-2^{-1}\cdot 2^{-7}$
Positivos	Menor	0	0000	**1**00000	$2^{-1}\cdot 2^{-7}$
	Mayor	0	1111	**1**11111	$\left(1-2^{-6}\right)\cdot 2^{8}$

3.1 Repres. coma flotante S-M

- Rango del exponente (otra forma de calcularlo como vimos): 15 -7= 8 y 0-7= -7 o simplemente aplicando: $[-2^{q-1} - 1, 2^{q-1}] = [-2^{4-1} - 1, 2^{4-1}] = [-2^{3} - 1, 2^{3}]$:[-7,8]
- Para hallar el número mayor: tomamos el mayor exponente posible 2^8 por el número más grande que es 0,111111 . Una forma más cómoda de expresar ese número es poner ($1-2^{-6}$) el cual multiplicamos por e exponente y no da $(1-2^{-6}) * 2^8$. Aplicaremos
  signo en función de que queramos hallar el mayor positivo o negativo.
- Para hallar el menor valor conseguimos el menor exponente 2^{-7}. La mantisa más pequeña es 0,100000 que equivale a $2^{-1.}$ Lo multiplicamos por el exponente y nos queda $2^{-1} * 2^{-7}$ con el signo que corresponda en función de si queremos el menor negativo o le menor positivo.

3.1 Repres. coma flotante S-M

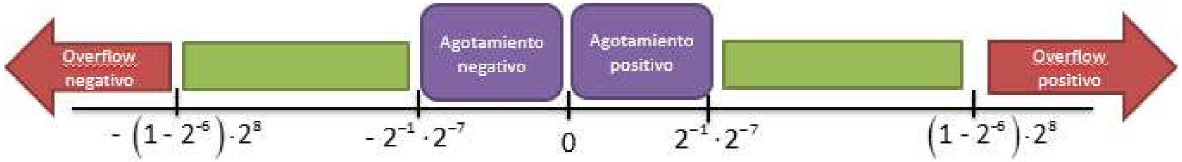
Como características de la representación en como flotante hay que comentar:

- No se representa el 0.
- No se representa un intervalo abierto alrededor del 0

- Rango de la representación en coma flotante:

		Coma flotante			Valor
		Signo	Exp sesgo 7	Magnitud normalizada con bit implícito	
Negativos	Mayor	1	1111	111111	$-\left(1-2^{-6}\right)\cdot 2^{8}$
	Menor	1	0000	100000	$-2^{-1}\cdot 2^{-7}$
Positivos	Menor	0	0000	100000	$2^{-1}\cdot 2^{-7}$
	Mayor	0	1111	111111	$\left(1-2^{-6}\right)\cdot 2^{8}$

- No se representa el 0.
- No se representa un intervalo abierto alrededor del 0: $\left(-2^{-1}\cdot 2^{-7},\ 2^{-1}\cdot 2^{-7}\right)$



Como hemos podido comprobar el 0 no puede representarse en pura lógica. Para evitarlo vamos a ver cómo se resuelve con los estándares del IEEE.

3.2 IEEE 754

IEE 754 es un estándar de representación de punto flotante de precisión simple con 32 bits. se utiliza 1 bit para el signo, 8 bits para el exponente y 23 para la magnitud.

El sesgo vale 2^7-1 =127

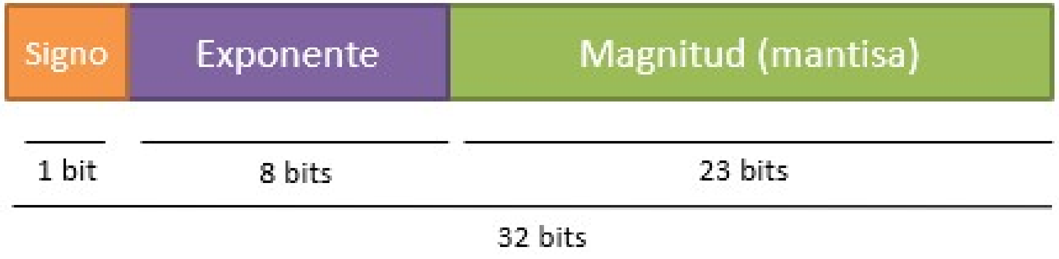
3.2 IEEE 754

La siguiente imagen resume sus características. Conviene prestar especial atención a los valores especiales que resumidos son:

- Exponente 0 y Mantisa 0 = Número 0 (ahora sí que lo podemos representar +0y -0)
- Exponente 255 y Mantisa 0 = + o – menos infinito
- Exponente 255 y Mantisa distinto 0 = NaN not a number)

3.2 IEEE 754

- Precisión simple (Float)

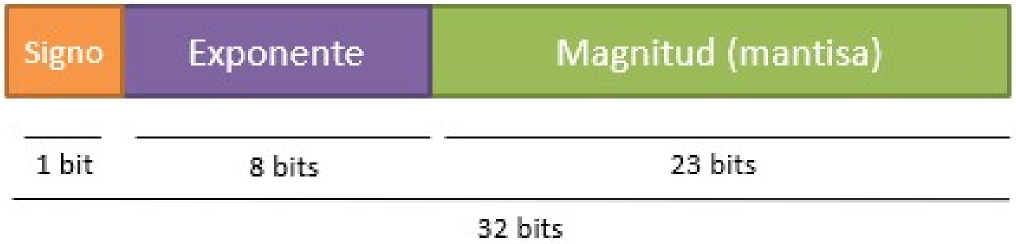


- Exponente en exceso 127 Exp = (valor - 127)
    - Exp 0 y Mantisa 0. Numero 0
    - Expo 0 y Mantisa distinta de 0. Número desnormalizados 0, pero con exponente -126 no -127
    - Exp 255 y Mantisa 0. + Infinito y –Infinito
    - Exp 255 y Mantisa distinto de 0. Not a number (NaNs)
    - Exp 1 a 254 Números normalizados con Exponente [-126,127]
- Mantisa normalizada a 1, y bit implícito

3.2 IEEE 754

En lo referente a rango de representación podemos observarlo en la siguiente imagen:

- Precisión simple (Float)



- Máximo positivo representable $(2\text{-}2^{-23})x\ 2^{127} = 3.402823 \times 10^{38}$
- Mínimo positivo normalizado $(1,)x\ 2^{-126} = 1.18 \times 10^{-38}$
- Máximo positivo no normalizado $(1 - 2^{-23})x\ 2^{-126} = 1.175 \times 10^{-38}$
- Mínimo positivo no normalizado $(2^{-23})x\ 2^{-126} = 1.4 \times 10^{-45}$
- Dos representaciones para el 0 (-0 y +0)

3.2 IEEE 754

En lo referente a rango de representación podemos observarlo en la siguiente imagen:

- Precisión simple (Float)

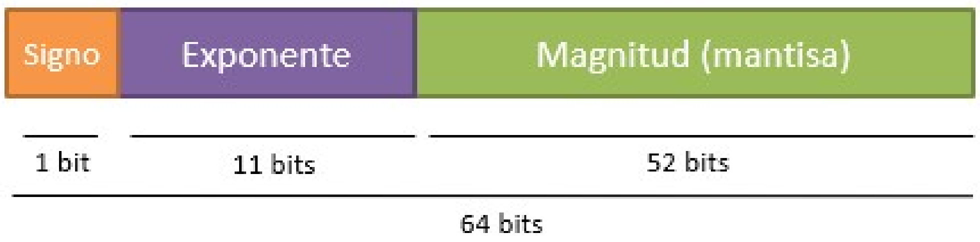
- Máximo positivo representable $(2\text{-}2^{-23})x\ 2^{127}$ = 3.402823 x 10^{38}
- Mínimo positivo normalizado $(1,)x\ 2^{-126}$ = 1.18 x 10^{-38}
- Máximo positivo no normalizado $(1 - 2^{-23})x\ 2^{-126}$ = 1.175 x 10^{-38}
- Mínimo positivo no normalizado $(2^{-23})x\ 2^{-126}$ = 1.4 x 10^{-45}
- Dos representaciones para el 0 (-0 y +0)

EXPONENTE	MANTISA	Numero
0	0	+0 [signo (0)] - 0 [signo (1)]
255	0	+ infinito [signo (0)] - infinito [signo (1)]
255	≠0	NaN (Not a Number) (ej. 0/0, √ - 1)
1 a 254 *(Sin codificar -126 a 127)*	Cualquier valor	Número normalizados 1, ------- (bit implícito) (Número normales)
0 (Sin codificar valor fijo de exponente -126)	Cualquier valor ≠0	Número (des)normalizados 0, ------- (no hay bit implícito) (Número muy pequeños) < 1 x 2^{-126}

3.2 IEEE 754

Adicionalmente al estándar visto el IEEE ofrece otro con precisión doble y 64 bits, el IEEE 754 (64 bits). En la siguiente imagen se pueden ver sus características:

- Precisión doble (Double)



- Máximo positivo representable $(1+(1-2^{-52}))x\ 2^{1023}$ = 1.79769 x 10^{308}
- Mínimo positivo normalizado $(1,)x\ 2^{-1022}$ = 2.22507 x 10^{-308}
- Mínimo positivo no normalizado $(2^{-52})x\ 2^{-1022}$ = 4.94065 x 10^{-324}
- Dos representaciones para el 0 (-0 y +0)