ARITMÉTICA EN PUNTO FLOTANTE

AMPLIACIÓN DE ESTRUCTURA DE COMPUTADORES

Daniel Mozos Muñoz
Facultad de Informática

Aritmética en punto flotante

1. Representación de números en punto flotante
2. IEEE 754
3. Multiplicación y división en punto flotante
4. Suma y resta en punto flotante

Bibliografía

1.-"Computer arithmetic algorithms". I. Koren, Prentice Hall, 1993

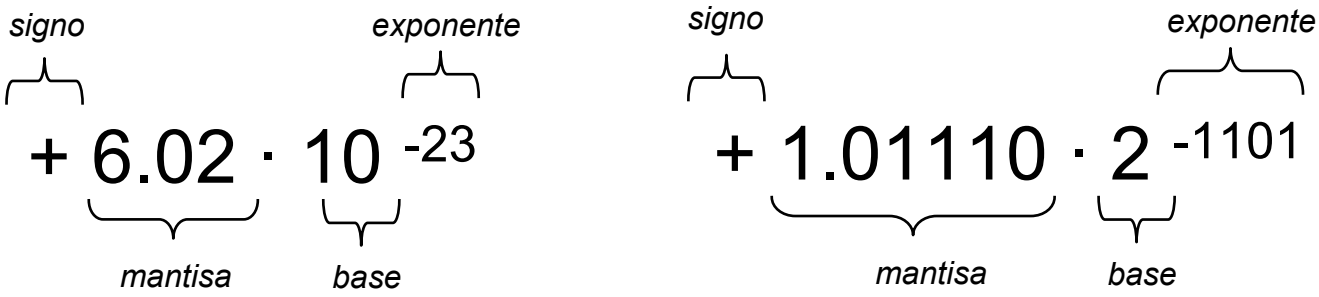
2.-"Computer architecture. A quantitative approach". Hennessy & Patterson, Morgan Kaufmann, 1995. Apéndice A.

3.-"What every computer scientist should know about floating-point arithmetic", ACM Computing Surveys. V.23, n.1, pg.5-18.

Representación de nºs. en punto flotante

La representación en punto flotante está basada en la **notación científica**:

- El punto decimal no se halla en una posición fija dentro de la secuencia de bits, sino que su posición se indica como una potencia de la base:



En todo número en punto se flotante distinguen tres componentes:

- **Signo**: indica el signo del número (0= positivo, 1=negativo)
- **Mantisa**: contiene la magnitud del número (en binario puro)
- **Exponente**: contiene el valor de la potencia de la base (sesgado)
- La base queda implícita y es común a todos los números, la más usada es 2.

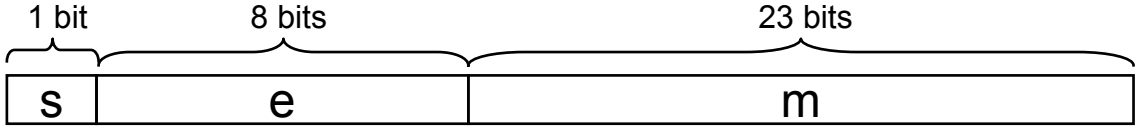
El **valor** de la secuencia de bits (s, e_{p-1}, ..., e_0, m_{q-1}, ..., m_0) es: $(-1)^s \cdot V(m) \cdot 2^{V(e)}$

Dado que un mismo número puede tener varias representaciones ($0.110 \cdot 2^5 = 110 \cdot 2^2 = 0.0110 \cdot 2^6$) los número suelen estar normalizados:

- un número está normalizado si tiene la forma $1.xx... \cdot 2^{xx...}$ (ó $0.1xx ... \cdot 2^{xx...}$)
- dado que los números normalizados en base 2 tienen siempre un 1 a la izquierda, éste suele quedar implícito (pero debe ser tenido en cuenta al calcular el valor de la secuencia)

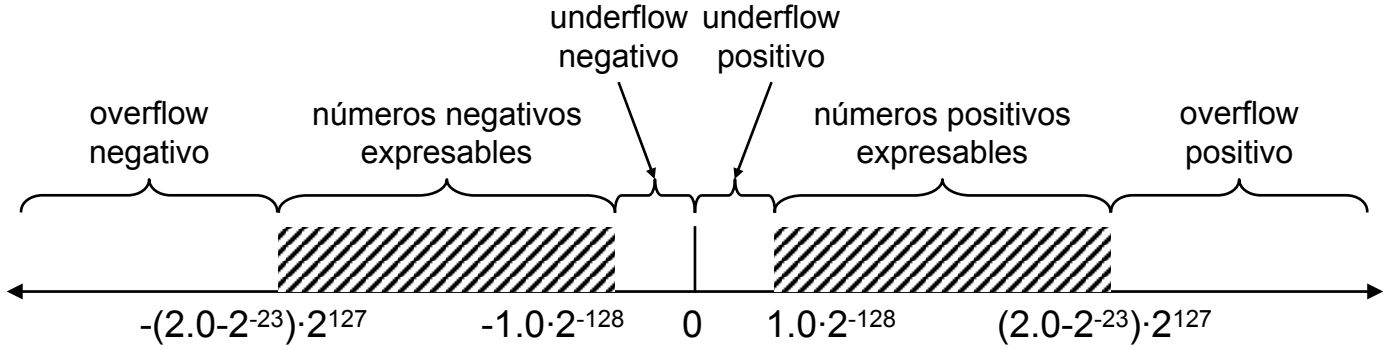
Representación de nºs. en punto flotante

Sea el siguiente formato de punto flotante de 32 bits (base 2, normalizado)

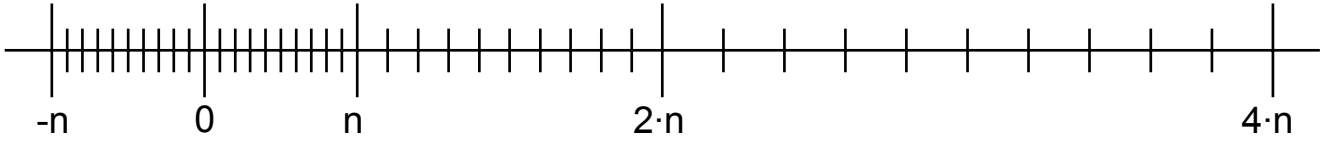


✓ El rango de valores representable por cada uno de los campos es:

- **Exponente** (8 bits con sesgo de 128) : -128 ... +127
- **Mantisa** (23 bits normalizados) : los valores binarios representables oscilan entre 1.00... y 1.11..., es decir entre 1 y $2-2^{-23}$ (2-ulp) (1.11...1 = 10.00...0 - 0.0...1)



✓ Obsérvese que la cantidad de números representables es 2^{32} (igual que en punto fijo). Lo que permite la representación punto flotante es ampliar el rango representable a costa de aumentar el espacio entre números representable (un espacio que no es uniforme).



IEEE 754

2 formatos con signo explícito, representación sesgada del exponente (sesgo igual a (2^{n-1}-1=127), mantisa normalizada con un 1 implícito (1.M) y base 2.

- ***precisión simple*** (32 bits): 1 bit de signo, 8 de exponente, 23 de mantisa
  - $-1.0 \cdot 2^{-126}$ ... $(2-2^{-23}) \cdot 2^{127} = 1.2 \cdot 10^{-38}$ .. $3.4 \cdot 10^{38}$
- ***precisión doble*** (64 bits): 1 bit de signo, 11 de exponente, 52 de mantisa
  - $-1.0 \cdot 2^{-1022}$ ... $(2-2^{-52}) \cdot 2^{1023} = 1.2 \cdot 10^{-308}$ .. $3.4 \cdot 10^{308}$

2 formatos ampliados para cálculos intermedios (43 y 79 bits).

Codificaciones con significado especial

- ***Infinito*** (e=255, m=0): representan cualquier valor de la región de overflow
- ***NaN*** (*Not-a-Number*) (e=255, m>0): se obtienen como resultado de operaciones inválidas
- ***Número denormalizado*** (e=0, m>0): es un número sin normalizar cuyo bit implícito se supone que es 0. Al ser el exponente 0, permiten representar números en las regiones de underflow
- ***Cero*** (e=0, m=0): número no normalizado que representa al cero (en lugar de al 1)

4 modos de redondeo:

- Redondeo al **más cercano** (al par en caso de empate)
- Redondeo a **más infinito** (por exceso)
- Redondeo a **menos infinito** (por defecto)
- Redondeo a **cero** (truncamiento)

Excepciones:

- **Operación inválida**: $\infty \pm \infty$, $0 \times \infty$, $0 \div 0$, $\infty \div \infty$, x mod 0, $\sqrt{x}$ cuando $x<0$, $x=\infty$
- **Inexacto**: el resultado redondeado no coincide con el real
- **Overflow** y **underflow**
- **División por cero**

IEEE 754

4 modos de redondeo:

- Redondeo al **más cercano** (al par en caso de empate)
- Redondeo a **más infinito** (por exceso)
- Redondeo a **menos infinito** (por defecto)
- Redondeo a **cero** (truncamiento)

El estándar exige que el resultado de las operaciones sea el mismo que se obtendría si se realizasen con precisión absoluta y después se redondease.
Hacer la operación con precisión absoluta no tiene sentido pues se podrían necesitar operandos de mucha anchura.

Al realizar una operación ¿cuántos bits adicionales se necesitan para tener la precisión requerida?

•Un bit r para el redondeo

•Un bit s (sticky) para determinar cuando r=1 si el número está por encima de 0,5

Tipo de redondeo	Signo del resultado≥0	Signo del resultado≤0
$-\infty$		+1 si (r or s)
$+\infty$	+1 si (r or s)	
0		
Más próximo	+1 si (r and p_0) or (r and s)	+1 si (r and p_0) or (r and s)

Multiplicación en punto flotante

Sean x e y dos números representados en punto flotante con valor:

- $X= (-1)^{s1}*1.mant1*2^{e1}$
- $Y= (-1)^{s2}*1.mant2*2^{e2}$

El producto de estos dos números será otro número z con valor:

- $Z=(-1)^{(s1 \oplus s2)}*(1.mant1*1.mant2)*2^{(e1+e2)}$

El proceso de multiplicación tiene varios pasos:

1.- El signo del resultado es igual a la o-exclusiva de los signos de los operandos.

2.- La mantisa del resultado es igual al producto de las mantisas.

Este producto es sin signo.

Dado que los dos operandos están comprendidos entre 1 y 2 el resultado r será:

$1 \leq r < 4$

Este paso puede requerir una normalización, mediante desplazamiento a la derecha y ajuste del exponente resultado.

3.- Si la mantisa resultado es del mismo tamaño que la de los operandos habrá que redondear. El redondeo puede implicar la necesidad de normalización posterior.

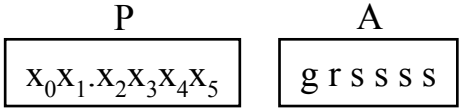
4.- El exponente del resultado es igual a la suma de los exponentes de los operandos.

Considerando que usamos una representación sesgada del exponente, al hacer esta suma estamos sumando dos veces el sesgo, y por tanto habrá que restar este sesgo una vez para obtener el resultado correcto.

Multiplicación en punto flotante

Pasos 2 y 3: Producto y redondeo

Supongamos que usamos un multiplicador secuencial que es capaz de multiplicar dos números almacenados en dos registros de p bits A y B, y almacenar el resultado en otro registro P de p bits (parte más significativa del producto) y en A (parte menos significativa).



Como el resultado final sólo se va a almacenar sobre el registro P, los bits contenidos en A nos servirán para redondear el resultado.

Como el número está comprendido entre 1 y 3, x_0 puede ser tanto 1 como 0.

En cada uno de estos casos el redondeo se realiza de modo distinto:

- $X_o=0$ => Desplazar P una posición a la izquierda, introduciendo el bit g de A como bit menos significativo de P. Los bit r y s (or de todos los s de A) nos sirven para redondear.
- $X_0=1$ => El punto decimal se desplaza una posición a la izquierda, y se ajusta el exponente sumándole 1. Poner s= (r or s) y r = g.

El redondeo siempre se hace de acuerdo a la tabla vista anteriormente.

Multiplicación en punto flotante

Overflow y underflow
El resultado redondeado es demasiado grande o pequeño para ser representado.

Números denormales:
El uso de este tipo de números complica el proceso de multiplicación.
Si el resultado del cálculo del exponente en una multiplicación es menor que el exponente mínimo representable, puede ocurrir que el resultado sea un número denormalizado.
En general si el exponente es menor que EXPmin, la mantisa se desplazará a la derecha hasta que el exponente alcance EXPmin. Si toda la mantisa se ha desplazado fuera, se ha producido un underflow.

Al usar como factores de una multiplicación números denormales, se opera como si fuesen números normales, considerando que el exponente no se calcula de igual modo.

División en punto flotante

Sean x e y dos números representados en punto flotante con valor:

- $X= (-1)^{s1}*1.mant1*2^{e1}$
- $Y= (-1)^{s2}*1.mant2*2^{e2}$

La división de estos dos números será otro número z con valor:

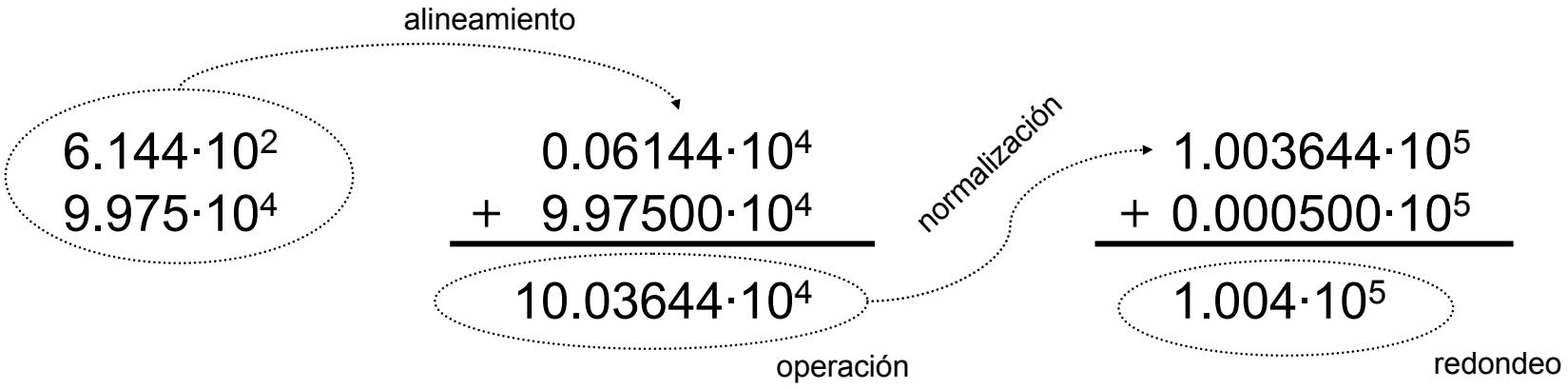
- $Z=(-1)^{(s1 \oplus s2)}*(1.mant1/1.mant2)*2^{(e1-e2)}$

El proceso de división es similar al de multiplicación.
Al hacer la resta de los exponentes hay que considerar que los sesgos se anularán y por tanto al resultado hay que sumarle el sesgo.
Al operar con números normalizados, la mantisa del resultado será:

$0,5 < r < 2$

Lo que implicará que la única normalización posible será mediante un desplazamiento a la izquierda.

Suma/resta en punto flotante



Método de suma/resta :

- Extraer signos, exponentes y magnitudes.
- Tratar operandos especiales (por ejemplo, alguno de ellos a cero)
- Desplazar la mantisa del número con exponente más pequeño a la derecha $|e_1-e_2|$ bits
- Fijar el exponente del resultado al máximo de los exponentes
- Si la operación es suma y los signos son iguales, o si la operación es resta y los signos son diferentes, sumar las mantisas. En otro caso restarlas
- Detectar overflow de la mantisa
- Normalizar la mantisa, desplazándola a la derecha o a la izquierda hasta que el dígito más significativo esté delante del punto.
- Redondear el resultado y renormalizar la mantisa si es necesario.
- Corregir el exponente en función de los desplazamientos realizados sobre la mantisa.
- Detectar overflow o underflow del exponente

Suma/resta en punto flotante

Redondeo

El estándar exige que el resultado de las operaciones sea el mismo que se obtendría si se realizasen con precisión absoluta y después se redondease.
Hacer la operación con precisión absoluta no tiene sentido pues se podrían necesitar operandos de mucha anchura.

Al realizar una operación ¿cuántos bits adicionales se necesitan para tener la precisión requerida?

Suma

- $1 \leq s1 < 2$
- $ulp \leq s2 < 2$

Por tanto $s=s1+s2$ cumplirá:

- $1 < s3 < 4$

Si $s3>2$ se deberá normalizar desplazando a la derecha una posición, y ajustando el exponente.

Redondeo:
Caso 1: e1=e2 y s3>2

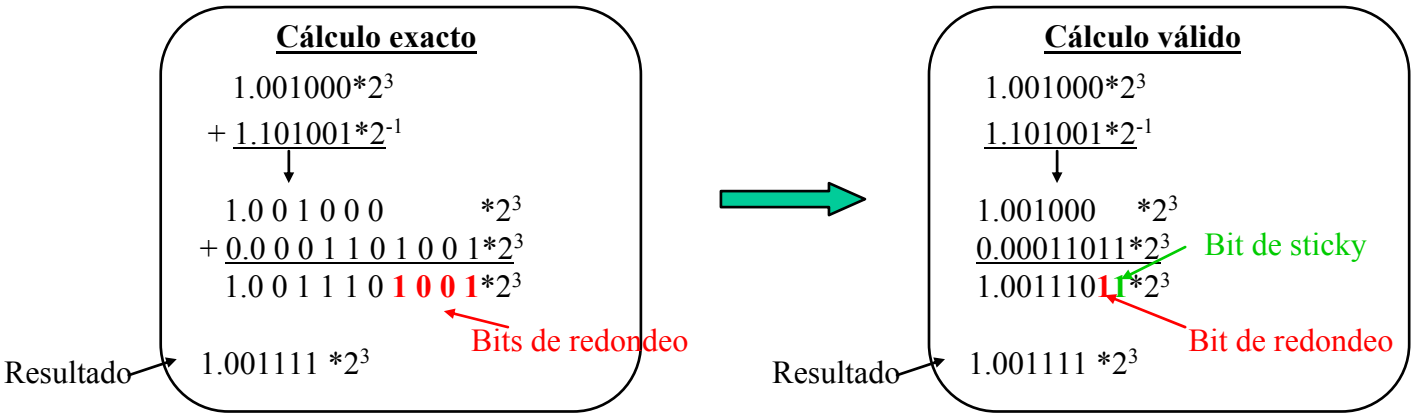
$1.001000 * 2^3$
$1.110001 * 2^3$
$10.111001 * 2^3$
$1.0111001 * 2^4$
Bit de redondeo

Suma/resta en punto flotante

Suma

Redondeo:

Caso 2: e1-e2>0



Suma/resta en punto flotante

•Resta:

•Caso 1: e1=e2

$1 \le s_i < 2 \Rightarrow s \subset [0,1) \Rightarrow$ Normalización

No se necesitan bits adicionales.

$1.001111 * 2^3$

$- \underline{1.001001 * 2^3}$

$0.000110 * 2^3$

$1.100000 * 2^{-1}$

Normalización

•Caso 2: e1-e2=1 y s>0,5

$1 \le s1 < 2$

$0.5 \le s2 < 1$ } ulp $\le s < 1.5$

Cálculo exacto

$1.0\,0\,1\,1\,1\,1 * 2^3$

$- \underline{1.0\,0\,1\,0\,0\,1 * 2^2}$

$1.\,0\,0\,1\,1\,1\,1 \quad * 2^3$

$-\underline{0.\,1\,0\,0\,1\,0\,0\,1 \quad * 2^3}$

$0\,.1\,0\,1\,0\,1\,0\,\mathbf{1} \quad * 2^3$

Bit de guarda

Resultado $1.010101 * 2^2$

Cálculo exacto

$1.1\,1\,1\,0\,0\,0 * 2^3$

$- \underline{1.1\,0\,0\,0\,0\,1 * 2^2}$

$1.\,0\,0\,1\,1\,1\,1 \quad * 2^3$

$-\underline{0.\,1\,1\,0\,0\,0\,0\,1 \quad * 2^3}$

$1\,.0\,0\,0\,1\,1\,1\,\mathbf{1} \quad * 2^3$

Bit de redondeo

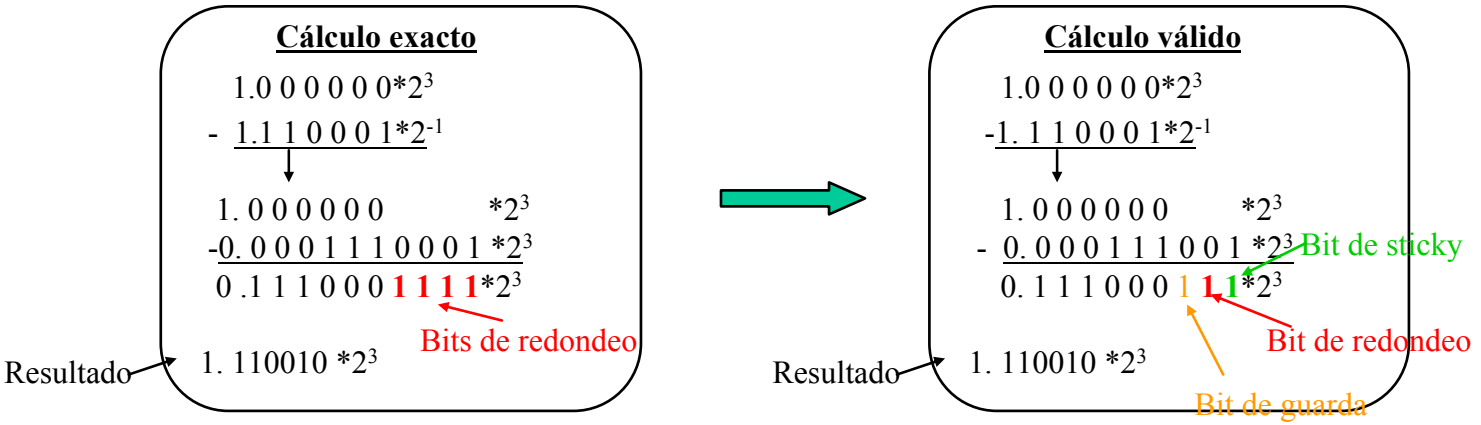
Resultado $1.001000 * 2^2$

Suma/resta en punto flotante

•Resta:

Caso3: e1-e2>1

$1 \le s1 < 2$
$ulp \le s2 < 0.5$ } $0{,}5 < s \le 2$



Suma/resta en punto flotante

Algoritmo:
Sean e1 y e2 los exponentes de los dos operandos y m1, m2 las mantisas con el bit oculto.

1.- Si e1<e2 intercambiar los operandos. Esto garantiza que la diferencia de exponentes d=e1-e2 es siempre positiva.
Poner como exponente tentativo del resultado e=e1.

2.- Si los signos de los operandos difieren, reemplazar m2 por C2(m2).

3.- Colocar m2 en un registro de p bits y desplazarlo a la derecha d posiciones (introduciendo unos si se ha complementado m2) para igualar los exponentes.
De los bits desplazados fuera de p, poner el más significativo en un biestable g, el siguiente más significativo en un biestable r, y la or del resto de bits en un biestable s (sticky).

4.- Calcular m como la suma de m1 y el contenido de p (m2 modificado).
Si (signo de a1 ≠ signo de a2) and (bit_mas_significativo(m)=1) and (no carry-out) entonces m es negativo.
Reemplazar m con C2(m).
Esto sólo puede ocurrir si d=0.

Suma/resta en punto flotante

5.- Normalización y ajuste de r y s :

Caso 1. Si (signo de a1 = signo de a2) and (carry-out) desplazar m una posición a la derecha, introduciendo un 1.

- S=g or r or s. R=bit que sale de m al desplazar.

Caso 2. Si no estamos en el caso 1, desplazar a la izquierda hasta que el número esté normalizado. (En el primer desplazamiento a la izquierda se introduce el contenido de g, en los restantes se introduce 0).

- Si no se ha necesitado desplazar => s=r or s r=g
- Si se ha desplazado una posición => r=r s= s
- Si se ha desplazado más de una posición => r=s=0 (esto sólo ocurre a1 y a2 tienen signos opuestos y el mismo exponente, por lo que la suma ha sido exacta).

En ambos casos, ajustar el exponente adecuadamente.

6.- Redondear según la tabla que vimos en la multiplicación. Si se produce carry-out desplazar a la derecha y ajustar el exponente.

7.- Signo del resultado:

Si (signo de a1 = signo de a2) este es el signo del resultado.

Si (signo de a1 ≠ signo de a2) el signo depende de cuál de los dos operandos fuera negativo, de si en el paso 1 se intercambiaron y de si en el paso 4 s se reemplazó por su c2, como se muestra en la tabla.

Intercambio	*C2*	*signo(a1)*	*signo(a2)*	*signo(resultado)*
si		+	-	-
si		-	+	+
no	no	+	-	+
no	no	-	+	-
no	si	+	-	-
no	si	-	+	+