Norma IEEE-754

Teoría de redondeo

Precisión de la mantisa

↳ En ocasiones el resultado de una operación FLP puede exceder los p dígitos de precisión reservados para la mantisa.

Ej -> multiplicar 2 mantisas (pq se suman los exponentes de c/u)

Truncado

↳ el ov virtual se puede compensar corriendo la mantisa a la derecha una posición

-> efecto colateral : eliminar el últ bit del resultado (lsb)

Redondeo

↳ Multiplicación -> una forma razonable de descartar los p dígitos menos significativos de los 2p dígitos del resultado consiste en *analizar el bit más significativo* pero *entre* los que serán *descartados*:

↦ Si ese bit es 1, ent. sumamos 1 LSB al resultado preliminar formado con los p dígitos del resultado que han de ser conservados.

↦ Si ese bit es 0 ent. el resultado preliminar es el definitivo.

↳ Ej. para X = 0.12345678 con las mismas condiciones que antes, el redondeo a 2 dígitos de precisión resulta 0.12 pero en cambio el redondeo a 5 dígitos resulta 0.1234**6**

↳ De forma análoga para Y = 0.110101 en el marco ahora de una base b = 2, el redondeo a 2 dígitos resulta 0.11 pero en cambio el redondeo a 5 dígitos resulta 0.1101**1**

↳ Cuando el bit más significativo descartado es 1 el resultado está en la mitad o pasada la mitad. Caso contrario al ser 0, el rdo. está necesariamente por debajo de la mitad.

↦ El *redondeo* siempre conduce al número máquina de **p** dígitos de precisión más próximo al resultado obtenido de **2p** dígitos.

↦ El máximo error que se puede cometer está acotado por la expresión:

$Err = b^{-p} / 2 = ½\ LSB$

Umbral se puede resolver inspeccionando sólo el bit más significativo entre los dígitos a ser descartados, más allá de la base adoptada

Máximo error

↳ Detectar el escenario en el que *peor* se desempeña el redondeo:

↦ Peor caso para le *truncado* -> se verifica cuando el rdo se aprox. a un nro máquina pero sin llegar a este; en ese caso la magnitud descartada representará a lo sumo 1 LSB

↦ Peor caso para el ***redondeo*** -> se verifica cuando el rdo equidista de dos nros máquina, descartando en ese caso 1/2 LSB

Teoría de redondeo

↳ Función de redondeo se define formalmente como el mapeo $\rho: \mathbb{R} \to \mathbb{M}$ tq $\forall$ a,b $\in \mathbf{R}$ se verifica que: $\rho(a) \leq \rho(b)$, toda vez que $a \leq b$

Redondeo óptimo

↳ Fc. *p* se denomina *óptima* sii. $\forall$ a $\in \mathbf{M}$ se verifica que *p*(a)=a.
↳ Asegura que $\forall$ a $\in \mathbf{R}$ con m_1 y m_2 dos números consecutivos en $\mathbf{M}$ tq. $m_1 < a < m_2$ se verifica siosi $p(a)=m_1$ o $p(a)=m_2$

Terminología

↳ *Redondeo p* es dirigido hacia arriba siii. $\forall$ a $\in \mathbf{R}$ se verifica que *p*(a) **>=** a .
↳ *Redondeo p* es dirigido hacia abajo siii. $\forall$ a $\in \mathbf{R}$ se verifica que *p*(a) **<=** a .
↳ Redondeo p es simétrico siii. $\forall$ a $\in \mathbf{R}$ se verifica que -*p*(-a) **=** *p*(a) .

↦ Redondeo óptimo dirigido hacia abajo
$$\nabla(a) = \max(\{m \in \mathbb{M} \mid m \leq a\})$$
↦ Redondeo óptimo dirigido hacia arriba
$$\Delta(a) = \min(\{m \in \mathbb{M} \mid m \geq a\})$$

Redondeos simétricos

↦ A partir de los redondeos óptimos es posible definir 3 redondeos simétricos que presentan un comportamiento interesante:

⇀ Truncado -> redondea pos. y neg. siempre hacia 0.

$$T(a) = \begin{cases} \nabla(a) \text{ cuando } a \geq 0 \\ \Delta(a) \text{ cuando } a < 0 \end{cases}$$

⇀ Aumentación -> redondea pos. y neg. siempre *alejándose* del 0.

$$A(a) = \begin{cases} \Delta(a) \text{ cuando } a \geq 0 \\ \nabla(a) \text{ cuando } a < 0 \end{cases}$$

⇀ Proximidad -> redondea siempre hacia el nro. máquina más próximo, eligiendo el de mayor magnitud en el caso límite, al equidistar entre dos nros. máquina consecutivos.

- Biased (sesgado)

$$P(a) = \begin{cases} \nabla(a) \text{ cuando } \nabla(a) \leq a < \text{umbral} \\ \Delta(a) \text{ cuando } \text{umbral} < a \leq \Delta(a) \\ \nabla(a) \text{ cuando } a = \text{umbral} \text{ y } a < 0 \\ \Delta(a) \text{ cuando } a = \text{umbral} \text{ y } a \geq 0 \end{cases}$$

$$\text{con umbral} = \frac{\nabla(a) + \Delta(a)}{2}$$

- Unbiased

$$P(a) = \begin{cases} \nabla(a) \text{ cuando } \nabla(a) \le a < \text{umbral} \\ \Delta(a) \text{ cuando } \text{umbral} < a \le \Delta(a) \\ \Delta(a) \text{ cuando } a = \text{umbral}, a < 0 \text{ y } p_0 = 0 \\ \nabla(a) \text{ cuando } a = \text{umbral} \text{ y } a < 0 \text{ y } p_0 = 1 \\ \nabla(a) \text{ cuando } a = \text{umbral} \text{ y } a \ge 0 \text{ y } p_0 = 0 \\ \Delta(a) \text{ cuando } a = \text{umbral} \text{ y } a \ge 0 \text{ y } p_0 = 1 \end{cases}$$

- ↦ El diseñador del sistema de FLP debe tomar decisiones que afectan tanto a la *velocidad de cómputo* como a la *exactitud.*
- ↦ La norma IEEE-754 implementa un redondeo por *proximidad unbiased,* prefiriendo en el caso límite los resultados pares.
  - ⇀ Esto disminuye el error acumulado producto de sucesivos redondeos.

IEEE-754

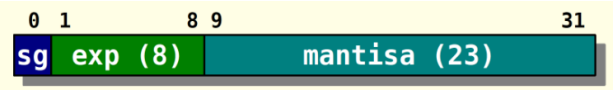
↳ La norma IEEE-754 de 1985 y su revisión del 2018 son el estándar de facto a la hora de implementar en hw una representación de FLP.

↳ Fue escrito en su mayor parte por el profesor W. M. Kahan.

Formatos de representación

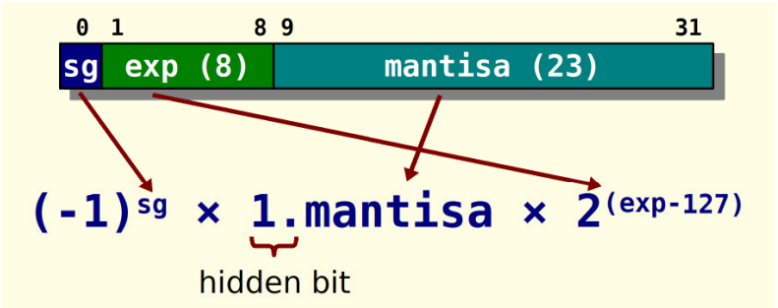
↳ La norma IEEE-754 contempla en principio dos formatos para los nros. FLP

- ↦ Precisión simple (32 bits)



- ↦ Precisión doble (64 bits)

- ↦ Características:
  - ⇀ En todos los casos se hace uso de una base b = 2
  - ⇀ La mantisa es representada usando la notación SM
  - ⇀ La *norma sanciona* una operación normalizada, por lo que la mantisa contará con un bit adicional de precisión <- Hidden bit .
  - ⇀ El exponente es representado usando notación exceso, que para el caso del formato precisión simple corresponde a exceso-127 (2^7-1 -> 8 bits para el exp -> 2^{e-1})



	exp = 0	0 < exp < 255	exp = 255
mantisa = 0	0	potencias de 2	±∞
mantisa ≠ 0	no normalizado (denormal)	números comunes	NaN (not a number)

Normalización

↳ La norma adopta una mantisa peculiar, con exactamente un dígito a la izquierda del punto.

↳ La norma tambn. estipula que opera sólo con mantisas normalizadas, por lo que debemos *adecuar su definición* a este contexto:

⇒ Mantisa normalizada cuando cuenta con exactamente un 1 la izq del punto

⇀ Ej. No normalizada -> 101.110 | Normalizada -> 1.01110

Conversión hacia la norma

↳ Procedimiento para convertir un nro en notacion FXP a la norma es bastante directo, abarca los siguientes pasos:

⇒ Convertir a binario la parte *entera* usando cualquiera de los métodos vistos.

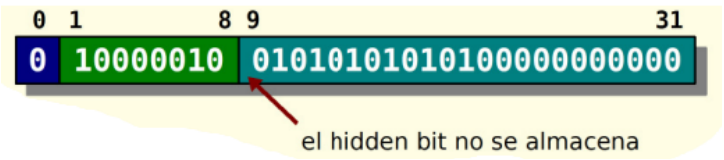
⇒ Convertir a binario la parte *decimal*, usando nuevamente cualquiera de los métodos vistos.

⇒ Normalizar el rdo obtenido anteriormente, ajustado el exponente de manera acorde.

⇒ Finalmente, expresar la mantisa y el exponente usando alguno de los formatos de la norma.

↳ Ej. X = (10.666015625)$_{10}$

1) Parte entera ⇀ (10)$_{10}$ = (1010)$_2$
2) Parte fraccionaria ⇀ (.666015625)$_{10}$ = (.101010101)$_2$
   ½ bis) ⇀ 1010.101010101 x 2^0
3) Normalizar ⇀ 1.010101010101 x 2^3
4) Exponente en exceso ⇀ 3 + 127 = 130$_{10}$ = 10000010$_2$



↳ Ej. Y = (-0.171875)$_{10}$

1) Parte entera ⇀ (0)$_{10}$ = (0)$_2$
2) Parte fraccionaria ⇀ (.171875)$_{10}$ = (.001011)$_2$
   ½ bis) ⇀ 0.001011 x 2^0
3) Normalizar ⇀ 1.011 x 2^{-3} (-3 pq hice corrimiento a der ent decremento exp)
4) Exponente en exceso : -3 +127 = 124_{10} = 01111100_2

⇨ Considerando el siguiente fragmento de código

```
int i; float f;
f = (float) i;
i = (int) f;
```

> ¿Pierdo precisión al convertir un int en un float?

↳ Puede haber pérdida de precisión al castear un entero a tipo flotante porque si consideramos la precisión simple (32 bits) el FLP destina parte de esos bits al exponente lo cual haría que no se usan la misma cantidad de dígitos para representar el número entero, por lo cual podría llegar a haber pérdida de precisión porque en el casteo podrían perderse bits del entero.

Denormals

↳ Existen combinación de valores para la mantisa y el exponente a los que se les reservó una interpretación especial.

↳ Los nros cuyo exponente es cero pero no su mantisa, se denominan *denormals.*

⇀ Los denormals admiten mantisas sin normalizar, por lo que permiten representar valores más pequeños que al utilizar mantisas normalizar.

⇀ Nótese que los denormals, al no estar normalizados no cuentan con el hidden bit.

Normalizados vs. Denormals

↳ El exponente asociado a un denormal es -126, más allá de que el valor codificado es cero.

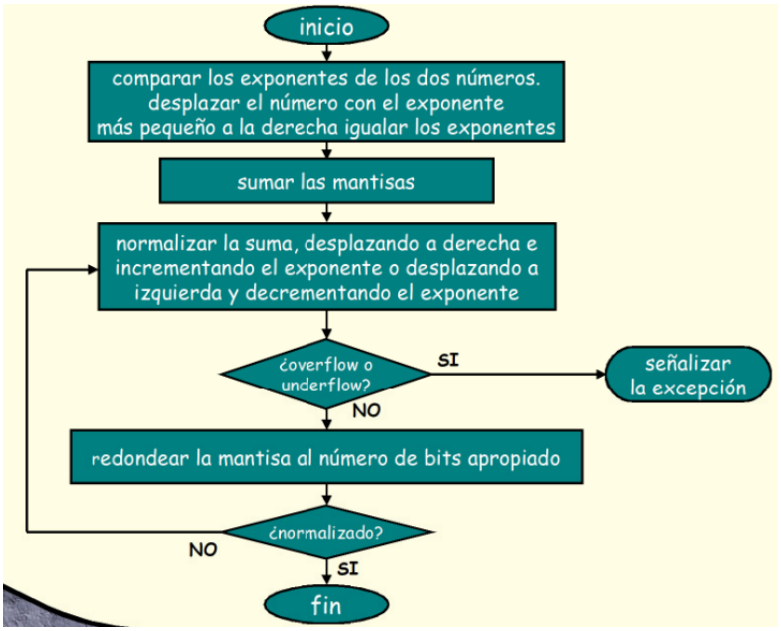
⇀ Es decir, los denormals no codifican su exponente en exceso, ya que en exceso, 0 representaría -126.

↳ ¿A qué se debe esta aparente inconsistencia?

⇀ La idea es permitir representar nros próximos a los nros ordinarios, por eso el exponente representado es el mismo (esto es, -126)

⇀ Naturalmente, el exponente almacenado difiere, para poder distinguir normalizados de no normalizados.

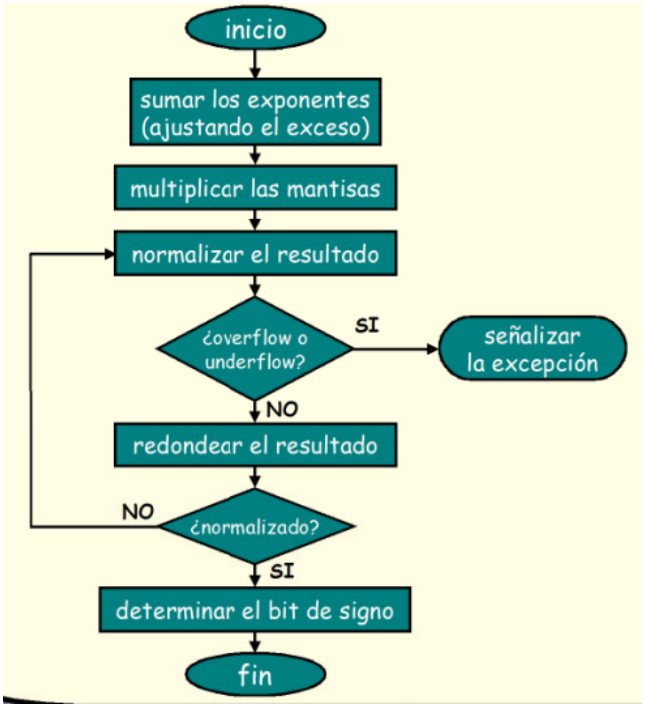
Suma



Multiplicación

↳ Algoritmo básico para la multiplicación de dos nros :

1) Sumar los **exponentes**, compensando el exceso.
2) Multiplicar las **mantisas**.
3) **Normalizar** el resultado y comprobar si se produjo ov.
4) **Redondear** la respuesta al nro correcto de dígitos significativos.
5) Normalizar nuevamente en caso de ser necesario.
6) Determinar el signo del resultado.



Redondeo

↳ La norma contempla 5 modalidades de redondeo:

⇒ Hacia arriba
⇒ Hacia abajo
⇒ Hacia cero
⇒ Proximidad unbiased (hacia los pares)
⇒ Proximidad biased (hacia afuera del cero)

⇀ Estas modalidades de redondeo requieren el uso de **bits adicionales de precisión**.

Bits de guarda

↳ Denominaremos bits de guarda a los bits de precisión adicionales a la derecha del LSB.

⇒ Los bits de guarda son conservados temporalmente hasta que sean requeridos.

⇒ Usualmente son empleados en la normalización y resultan fundamentales durante el redondeo.

⇒ La norma IEEE-754 contempla *3 bits de guard*

⇀ guard **(G)**
⇀ round **(R)**
⇀ sticky **(S)**

⇒ Esto implica que la ALU deba ensancharse 3 bits.

Round y Sticky

⇒ $x_1 + x_2 = 2.561 \times 10^0 + 2.34 \times 10^2$
$= 0.02561 \times 10^2 + 2.34 \times 10^2$
$= 2.36\,561 \times 10^2$
(R = 5 y S ≠ 0)

⇀ Para 0 <= R <= 4
⇀ Para 6 <= R <= 9
⇀ R = 5, se mira el estado de S
⇀ Sticky solo importa si es cero o no
> En la norma S se calcula haciendo **OR** entre todos los bits descartados (esto es, los que no forman parte del resultado ni tampoco de G ni R)

redondeo	resultado ≥ 0	resultado < 0
hacia $-\infty$		+1 LSB si (R OR S)
hacia $+\infty$	+1 LSB si (R OR S)	
hacia el 0		
proximidad hacia pares	+1 LSB si ((R AND p_0) OR (R AND S))	+1 LSB si ((R AND p_0) OR (R AND S))
proximidad afuera del 0	+1 LSB si R	+1 LSB si R

↳ Ej. Cálculo con precisión infinita:

$X_1 = 1.001000 \times 2^3$

$X_2 = +1.101001 \times 2^{-1}$

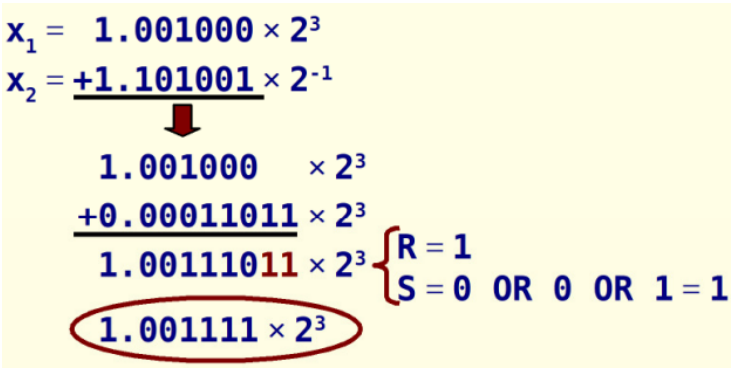
1.001000×2^3

0.0001101001×2^3

1.0011101001×2^3

-> 1.001111×2^3

Cálculo usando sólo R y S



Rol del bit de guarda

↳ El bit G se emplea al normalizar el resultado preliminar de las operaciones aritméticas (antes de aplicar el redondeo)

↳ En los casos en que no haga falta usar el bit **G** se deben ajustar los valores de los bits **R** y **S**

⇀ $R_{nuevo} = G_{anterior}$

⇀ $S_{nuevo} = R_{anterior}$ OR $S_{anterior}$

Implementación en hw

↳ Surgen consideraciones al implementar estos algoritmos en hw

⇒ Alcanzar un desempeño altamente eficiente (no tomar decisiones que comprometan el desempeño)

⇒ Minimizar la cantidad de hw necesario (reusar en la medida de lo posible el hw disponible)

Alg suma con GRS diapo 21-29 M06 N IEEE-754 pt2
Multip. 30-33 M06 N IEEE-754 pt2