3.6 Codificación Punto Flotante

Esta codificación nace por la necesidad de tener un rango más amplio de representatividad numérica, o **cobertura**. Los esquemas antes mencionados ofrecen un rango limitado de representatividad, por el hecho que su **resolución** (número de prototipos por largo de intervalo en la recta real) es uniforme independiente de la magnitud de los números representados.

El esquema de codificación de punto flotante permite una resolución variable. Mayor resolución para números de magnitud pequeña y menor resolución para números de magnitud mayor. El formato de sus campos se deriva de la notación científica,

$$+\text{-}\ S \times B^{+\text{-}E}$$

De esta forma, los campos distintivos de este esquema de codificación son:

- ***Signo (S)***
- ***Parte significativa o Mantisa (S)***
- ***Exponente (E)***

Se utiliza un bit para codificar el signo. Respecto a los bits utilizados para codificar la mantisa y el exponente, estos directamente afectan:

- La resolución del esquema o **precisión**, (los bits para codificar la magnitud)
- El rango o cobertura de representatividad del esquema (los bits para codificar el exponente).

3.6.1 Forma Normalizada

Este estándar presupone una representación normalizada. Es decir, los números a representar obedecen a la siguiente forma:

$$\pm\, 0{,}1bbbbbb....b \times 2^{\pm E}$$

Forma Normalizada

Esta forma es equivalente a la notación científica base 10, pero en este caso particular tanto las magnitudes como el exponente son base 2.

<u>Consideraciones de Síntesis</u>

-i) La mantisa se considera siempre distinta de cero. Esto justifica el 1 después de la coma en la forma normalizada. Por tanto con ***m bit*** de mantisa se codifica valores entre 0.5 y $1\text{-}2^{-(m+1)}$.

- ii) Para el caso del exponente se utiliza la llamada **representación sesgada**. Es decir con k-bit para representar el exponente, su valor numérico viene dado por la siguiente expresión:

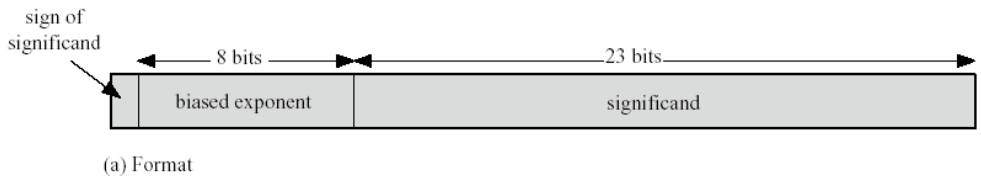
$$E = \sum_{i=0}^{k-1} e_i 2^i - \left(2^{k-1} - 1\right).$$

Es decir, la magnitud del exponente corresponde a interpretar los bits con su codificación entera sin signo, menos el valor promedio del rango de codificación alcanzado con k-bits, llamado **sesgo**. Por lo tanto los valores de exponente van de $-(2^{k-1}\text{-}1)$ a 2^{k-1}.

Justificación de la representación sesgada del exponente

La razón que justifica la interpretación sesgada es que mantiene la relación de orden observada en la codificación entera sin signo. En particular, el número más grande es el 11..1111 (considerando primero los bits del exponente y después los de la mantisa) y el más pequeño es 0000..00.

La siguiente figura muestra los campos de una codificación de punto flotante.



(a) Format

```
 0.11010001   2^10100  = 0 10010011 10100010000000000000000
-0.11010001   2^10100  = 1 10010011 10100010000000000000000
 0.11010001   2^-10100 = 0 01101011 10100010000000000000000
-0.11010001   2^-10100 = 1 01101011 10100010000000000000000
```

Figura f.1: *Codificación en punto flotante de 8 bit de mantisa y 23 de exponente*

(Propuesto 1)----------
En la figura f.1 se muestran las codificaciones de algunas representaciones numéricas. Muestre que comparar las magnitudes de dos números en punto flotante equivale a comparar utilizando la relación de orden de la codificación entera sin signo, donde los bits del exponente son los más significativos y los de la mantisa los menos significativos.

De esta forma la lógica de comparación entera sin signo se puede usar para comparar las magnitudes de números en punto flotante.

Rangos de Representatividad

El formato de la figura f.1 corresponde a la codificación en punto flotante en 32 bits, con 8 bit de exponente (valores entre –127 a 128) y 23 bit de mantisa (entre 0.5 a $1-2^{-24}$). Con este esquema de es posible cubrir el rango siguiente:

- ***Números negativos:*** $-(1-2^{-24})x2^{128}$ a $-0.5x2^{-127}$
- ***Números positivos:*** $0.5x2^{-127}$ a $(1-2^{-24})x2^{128}$

Se pueden distinguir varios escenarios de **overflow** (ver figura f.2)

- Números negativos menores que $-(1-2^{-24})x2^{128}$. Región de **desbordamiento negativo**.
- Números negativo mayores que $-0.5x2^{-127}$. Región de **desbordamiento a cero negativo**.
- Números positivos mayores que $(1-2^{-24})x2^{128}$, región de **desbordamiento positivo**.
- Números positivos menores que $0.5x2^{-127}$, región de **desbordamiento a cero positivo**.

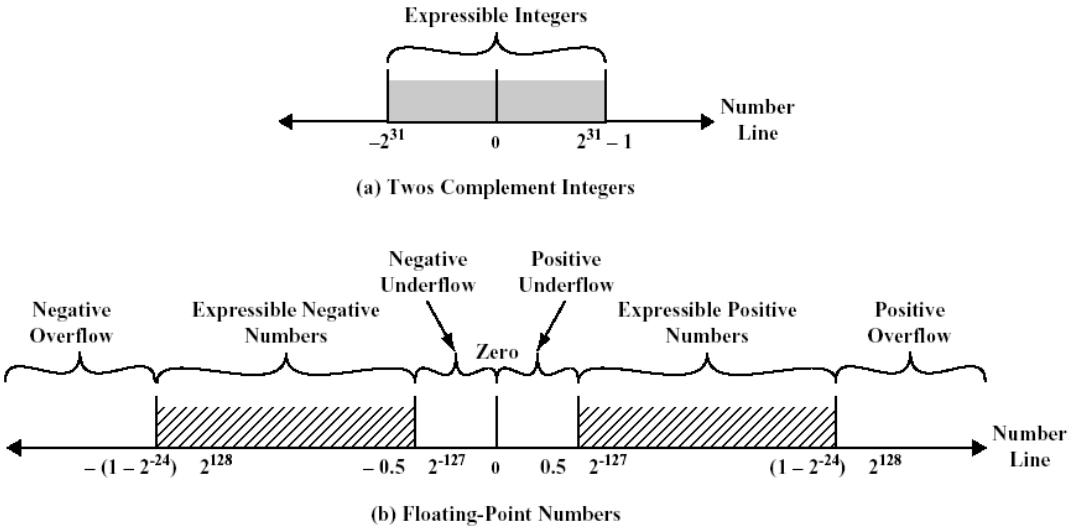


Figura f.2*: Rangos de representatividad numérica a) complemento dos b) punto flotante en 32bit*

Compromiso entre Cobertura y Precisión

La figura f.2 distingue los rangos de representatividad numéricas de la codificación de punto flotante y entera complemento dos. Se muestra que el rango de representatividad en punto flotante es sustancialmente mayor al de codificación entera sin signo. Dado que con 32 bit se representa la misma cantidad de prototipos, 2^{32}, esto implica necesariamente que la densidad promedio de puntos representables en algunos tramos de la recta real es mayor en la codificación complemento dos.

El aspecto que distingue este esquema de codificación es que la densidad de puntos varía de forma inversamente proporcional a la magnitud de los valores representados. La codificación entera, por otro lado, ofrece una densidad uniforme de prototipos. De hecho, la distancia entre cualquier prototipo es la misma (prototipos equi-espaciados).

De esta forma se puede decir que la codificación en punto flotante ofrece un mayor rango de representatividad, al costo de una menor precisión numérica, pues la distancia promedio de un valor arbitrario en la recta real (dentro del rango de representatividad) al prototipo más cercano aumenta con su magnitud.

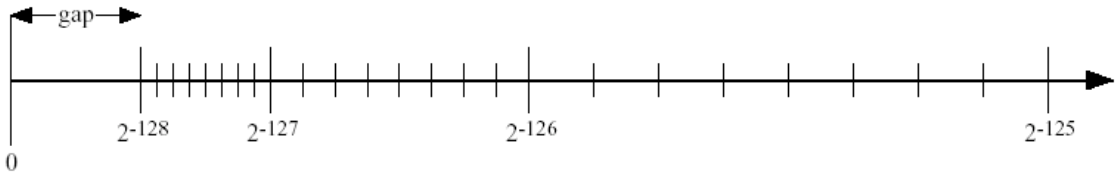
(Propuesto 2)----------

El propósito de este problema es ver en detalle la precisión de la codificación en punto flotante, Considere el esquema de codificación de la figura ***f.1***

i) Encuentre la cantidad de prototipos en el intervalo de magnitudes $[2^{n-1}, 2^n]$ en función de n para -128<n<=128. *¿Este valor depende del intervalo escogido?*

ii) Muestre que los prototipos están equi-espaciados en el intervalo y sobre este intervalo calcule el error de precisión promedio. Encuentre la relación de este error entre intervalos adyacentes.

iii) Finalmente justifique el esquema de la ***figura f.3*** y determine el error de precisión promedio sobre todo el rango de representatividad numérica. Obtenga una expresión genérica en función de M y E. (M: número de bit del exponente, E numero de bit de la mantisa)



(a) 32-bit format without denormalized numbers

Figura f.3: *Distribución de los prototipos de la codificación en punto flotante, con 3 bit de mantisa y 23 de exponente.*

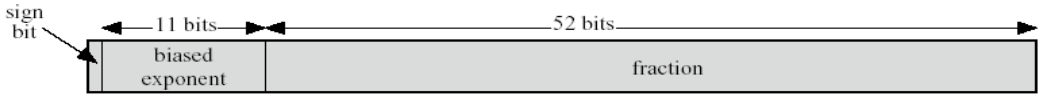
De la resolución del problema anterior, es posible ver que el rango de representatividad esta directamente relacionado con los bits del **exponente**, mientras la densidad de puntos en dicho rango esta asociado a los bits con los cuales se codifica la **mantisa**. De esta forma si se tiene una cantidad de bits fijos, existe un compromiso de diseño entre el rango de representatividad y la precisión numérica, para determinar finalmente cuantos bits son asociados a cada campo de la codificación.
Necesariamente para mejorar simultáneamente el rango y la precisión, se debe aumentar los bits totales de la codificación.

Del análisis anterior, se ve que el valor **cero** no tiene una representación explicita en este esquema de codificación. Dado que este valor es importante, para efectos de implementar instrucciones aritméticas y de comparación, el estándar mas adoptado**,** el **IEEE 754,** considera una codificación especial para poder representarlo.

3.6.2 Estándar IEEE 754

Corresponde al estándar de codificación en punto flotante que se encuentra implementado en prácticamente todos los procesadores o co-procesadores aritméticos actuales.
Este estándar define un formato simple y uno extendido como se ve en la figura.

(a) Single format



(b) Double format

Figura f.4*:* Formatos simples y de doble precisión, estándar IEEE 754

El formato de doble precisión se utiliza para resultados aritméticos intermedios, la idea es minimizar errores de precisión (truncamiento de los bits menos significativos) y los escenarios de ***overflow*** en una secuencia de operaciones aritméticas.

Adicionalmente el estándar define tramas de bits especiales que no se interpretan de la forma estándar, si no que tienen una síntesis especial. Estos específicamente corresponden a las palabras con exponente todo cero (000.00) y todo unos (111..1).

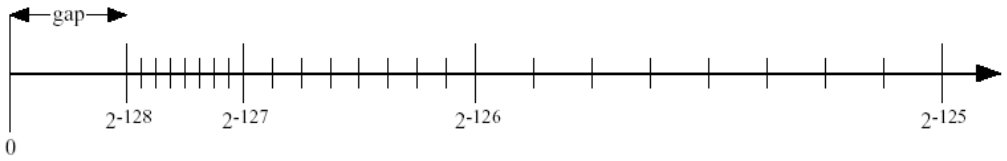
Los rangos son los siguientes:

- Códigos con exponentes distintos de (111.11) y (00.0), representan números en coma flotantes normalizados (***representación clásica***), los rangos de exponentes van de (-126 a 127, codificación simple y –1022 a 1023, doble precisión)
- Exponente (000.0) junto con mantisa (00..0), representan el **cero positivo y cero negativo**.
- Exponente (111.1) junto con mantisa (00..0), representa el **infinito positivo y negativo.**
- Exponente (00..0) junto a mantisa distinta de (000.0) representan los **valores de-normalizados**. En este caso el bit a la izquierda de la coma binaria es cero, por lo que los valores representables vienen dado por la expresión.

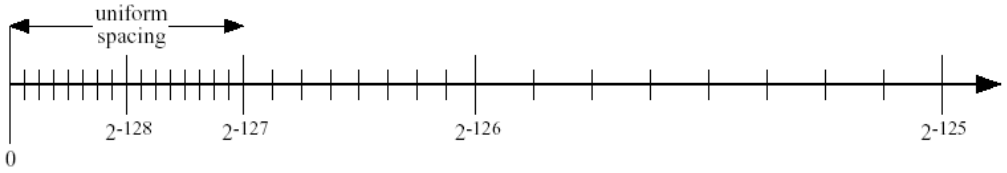
$$\pm 0,bbbbbb....b \times 2^{-127} \quad \textit{(simple)}$$

$$\pm 0,bbbbbb....b \times 2^{-1023} \quad \textit{(doble precisión)}$$

De esta forma se cubre de manera uniforme el intervalo entre [0, 2^{-127}] y se evita el rango de números sin representar entre 2^{-128} y 0 del formato clásico. (Ver figura f.5)



(a) 32-bit format without denormalized numbers



(b) 32-bit format with denormalized numbers

Figura f.5: *Distribución de prototipos esquema normalizado a) y de-normalizado b)*

- Números de exponente (111.11) y mantisa distinta de (00.00) se denominan números NaN (***not number***) y se utilizan para señalizar condiciones de excepción.

3.7 Aritmética punto flotante

En este punto veremos las consideraciones para poder implementar las operaciones aritméticas básicas en este formato de codificación.

Escenarios críticos

Antes de entrar en detalles, caractericemos los posibles escenarios críticos (**overflow**) producto de llevar a cabo operaciones aritméticas en este formato de codificación.

- **Desbordamiento del exponente*:*** producto de una operación aritmética, el exponente del resultado en la forma normalizada puede exceder el valor máximo representable (128, caso exponente con 8 bit) (**overflow**)
- **Desbordamiento a cero del exponente*:*** producto de una operación aritmética, el exponente del resultado en la forma normalizada puede ser menor que el valor mínimo representable (-127, caso exponente con 8 bit) En este caso no ocurre **overflow**, pues se asigna el valor especial de la codificación del cero (negativo o positivo)
- **Desbordamiento a cero de mantisa*:*** producto del proceso de ajustar la mantisa a la forma normalizada, se pueden perder los bits menos significativos de ésta. En consecuencia se debe efectuar un proceso de redondeo. El más barato es simplemente despreciar dicha información (desplazamiento a la derecha de la palabra)
- **Desbordamiento de mantisa*:*** producto de las operaciones aritméticas la mantisa de la codificación resultante puede presentar el fenómeno de acarreo al bit más significativo. Esto no necesariamente genera un **overflow**, pues se puede resolver con un proceso de ajuste de mantisa - exponente (incrementar exponente y desplazar la mantisa a la derecha)

Suma y Resta

Este proceso en la práctica resulta más complicado que la multiplicación, pues implica el proceso de ajuste de mantisa de tal forma de hacer coincidir las bases de los argumentos. Las etapas son:

1. Comprobar el caso de un argumento en cero.
2. ajuste de mantisa.
3. sumar o restar las mantisas.
4. Normalizar el resultado.

Una apreciación importante es que las mantisas de los argumentos que serán operadas por una unidad aritmética, deben usar de forma explicita los bit más significativos de los argumentos (implícito en la codificación).

Las etapas del algoritmo de suma son:

- **Ajuste de mantisa**: manipular los exponentes de tal forma que sean iguales en ambos argumentos. Esto implica aplicar desplazamientos a la representación de magnitud de la mantisa y incrementar o decrementar el exponente.

- **Sumar las mantisas**: puede ocurrir un desbordamiento de mantisa, en cuyo caso se incrementa el exponente y se desplaza ésta un bit a la derecha.

- **Proceso de normalización**: implica llegar a una representación de la mantisa con su bit más significativo en 1. De esta palabra se toman la n-bits menos significativos.

Multiplicación

El flujo-grama de la ejecución de la multiplicación se muestra en la siguiente figura.

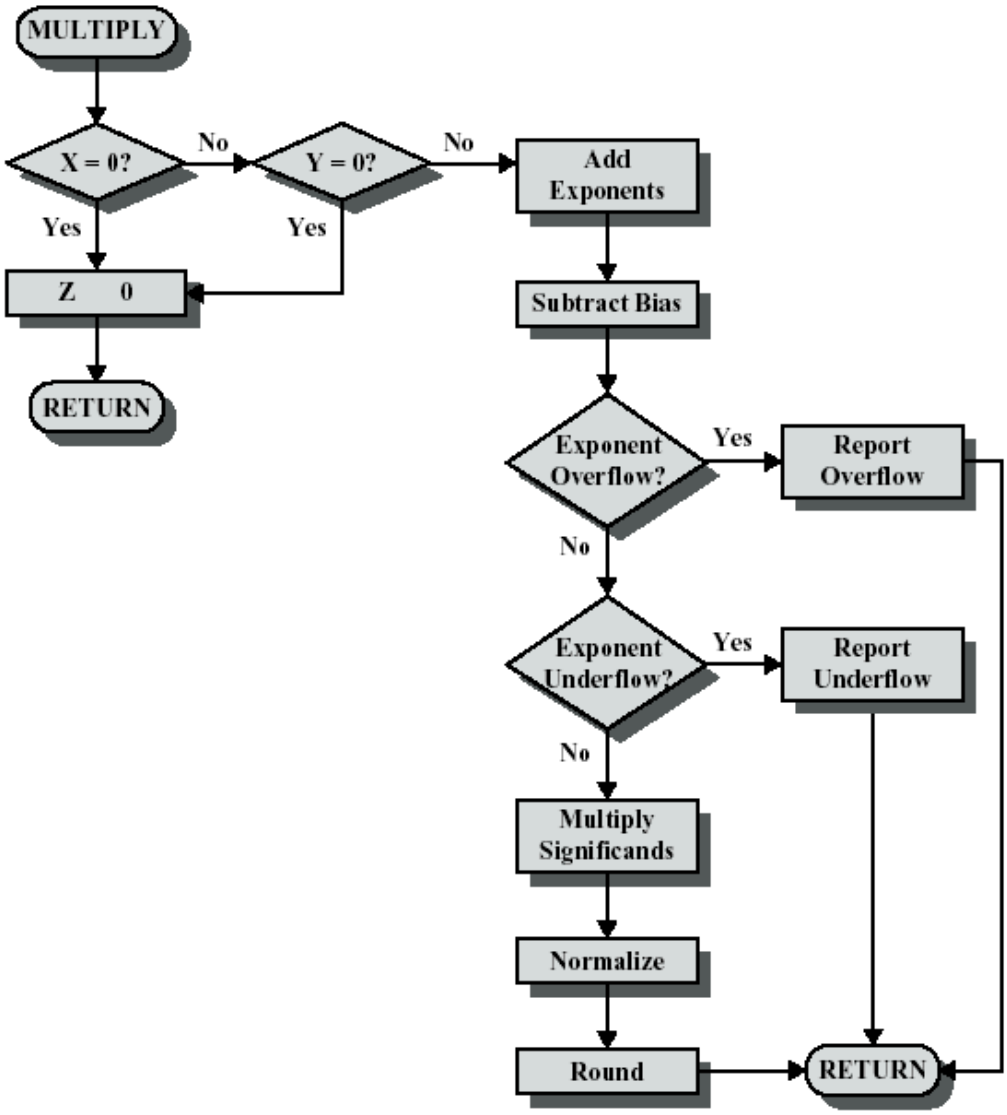


Figura f.7*:* Flujo-grama de la ejecución de multiplicación

Las etapas de este proceso son:

- Verificar si alguno de los argumentos es **cero**. En este caso se retorna **cero**.
- Sumar los exponentes: Dado que los exponentes tienen la representación sesgada la suma debe llevar en consideración un sesgo doble. Por lo tanto posteriormente a la suma es necesario restar el valor del sesgo. Esta aritmética sobre los exponentes puede implicar un **desbordamiento** (**overflow**) o un **desbordamiento a cero** del exponente (cero positivo o negativo), lo que da fin al algoritmo.

- Si el exponente resultante cae en rango de codificación, entonces el siguiente paso es multiplicar los exponentes como enteros sin signo. La palabra resultante puede requerir hasta 2n bits para ser representada.

- La mantisa resultante en 2n-bits es normalizada dejándola en n-bit, lo que implica el fenómeno de redondeo (eliminación de los n-bit menos significativos) y la actualización del exponente como consecuencia de los desplazamientos sobre la mantisa. Esta etapa puede llevar a un desbordamiento a cero del exponente.

- El bit de signo se ajusta con lógica elemental, utilizando un **or exclusivo** sobre los bits de signo de los argumentos.