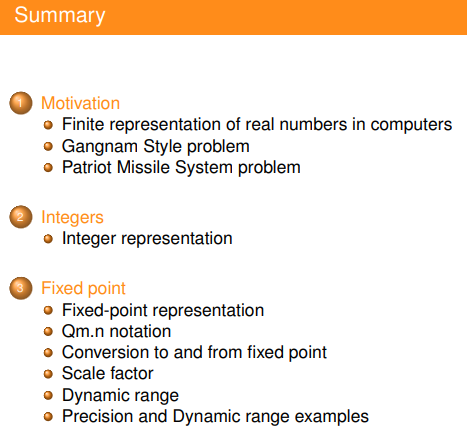
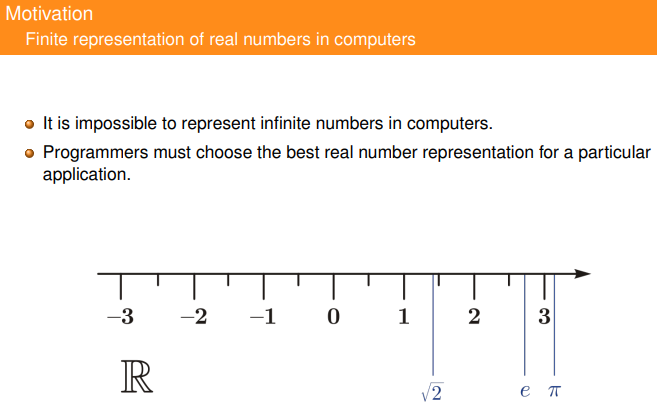
**Representación finita de números reales en formato punto fijo**





Existen limitaciones cuando queremos representar números que pertenecen a la recta de los reales en una computadora.

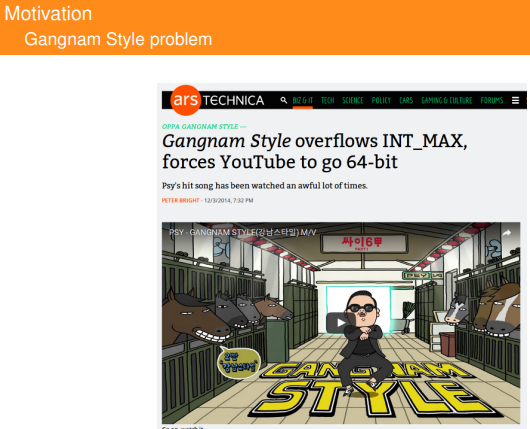
La recta tiene infinito números, y es imposible representar infinitos números en una computadora, porque necesitaríamos infinitos bits, y memoria infinita. Con la tecnología actual es imposible.

Por eso hablamos de representación FINITA de números.

Entonces lo que hacen los programadores cuando eligen representar números en una computadora, es ajustar o adecuar esta representación al problema que tiene que resolver, a la aplicación particular donde lo va a utilizar.

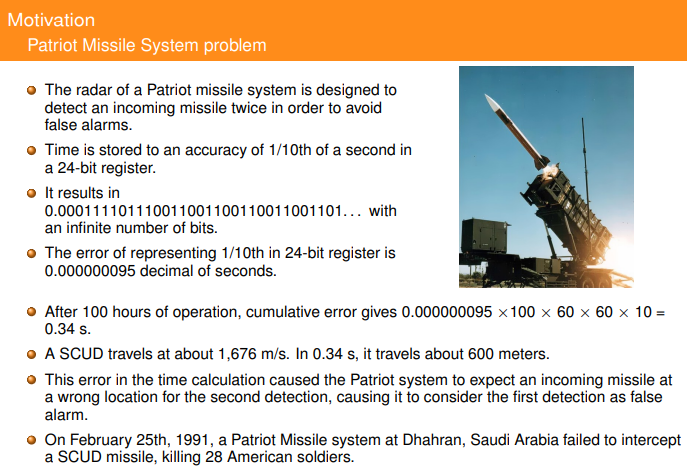
Todas las formas de representar números en una PC tienen ventajas y desventajas, Punto Fijo, Flotante o Enteros.

Ejemplos de 2 problemas que se produjeron por representar números de forma errónea.

Años 2012, cantidad de reproducimos 900 millones

Representaba el contador con un numero de 32 bits sin signo. Aproximadamente 4300 millones de reproducciones.

Paso de 32 a 64 ... 1.8x10^19 … Así se queda tranquilo

Sistema misiles patriot, derribar un misil con otro misil. El sistema tiene un radar que verifica si se aproxima un misil enemigo.

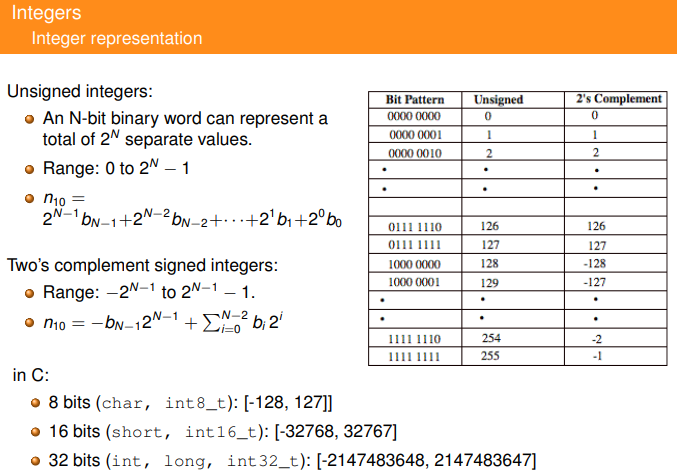
Si lo detecta, lo que hace es, como conoce la velocidad del misil detectado, midiendo tiempo predice la distancia que va a recorrer.

Entonces cuando realiza una primera detección, predice donde va a estar y realiza un enfoque en esa área para corroborar que se trate de un misil enemigo. Y acto seguido dispara el misil de defensa Patriot.

La base de tiempo debería ser muy precisa, ya que de esta depende el calculo de las distancias.

Aproximadamente 95 nanosegundo de error por representar con 24 bits. El error es acumulativo

Ejemplo de las consecuencias que puede traer representar un numero de forma inexacta en un sistema de base 2 de una computadora. ¿Cuál hubiera sido un mejor enfoque?

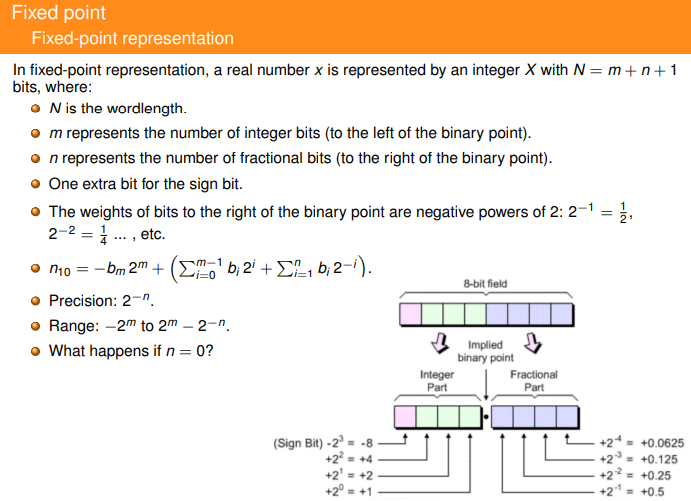
Repasemos como se representan los números enteros en una PC

El “-1” aparece para dedicarle una palabra para representar el 0. Bit b que corresponde a cada posición para pasarlo a decimal

Otra forma de representar los números enteros típicamente en una PC es complemento a 2. Muy utilizada por temas de eficiencia de hardware. Porque al operar con números representados de esta forma, se puede usar un sumador para sumar o restar.

La formula nos dice que todos los que empiecen con 1 son negativos.

Formas de representarlo en lenguaje C. Hay veces que cambia la cantidad de bits según el compilador.

Ahora vemos como representamos un numero real en formato punto fijo en una PC:

Vamos a utilizar una palabra de N bits, de los cuales tomas “m” bits para representar la parte entera y “n” para la parte fraccional, y uno extra para determinar el signo.

Vamos a ver que un numero en punto fijo es una generalización de representar un numero como entero. Ahora vamos a considerar las potencias negativas en base 2.

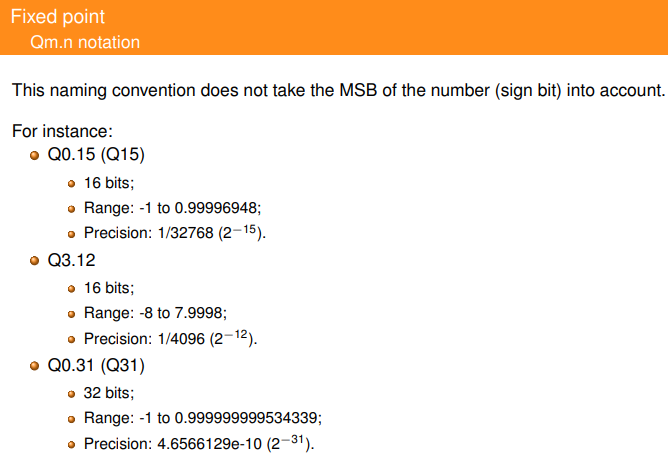
Fórmula para pasar a decimal: primer bit determina el signo, 1ra sumatoria de potencias + de 2, representa la parte entera y la 2da sumatoria de potencias de 2 negativas, la parte fraccional.

Acá es punto fijo, porque la cantidad de bits que asignamos a la parte fraccional no varía. Cuando el programador elije cuantos bits va a dedicar a la parte entera y a la decimal, quedan terminadas 2 variables que definen a su vez, el rango de presentación que hemos determinado como programadores. La Precisión y el Rango.

Si elegimos n=1, la precisión es 0,5 y nos indica que entre 1 y 1,5 no hay una representación posible, el mínimo numero que podemos representar va en pasos de lo que valga la precisión.

El rango va desde números negativos a positivos.

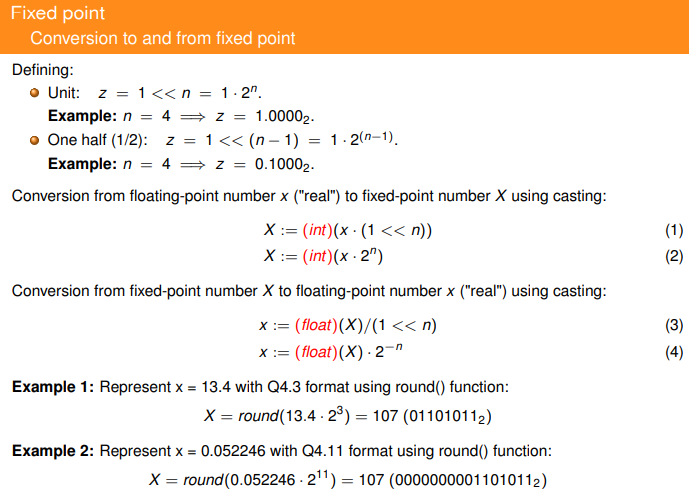
Los números enteros en complemento a 2, son un caso particular de la representación en punto fijo. Si n=0, la precisión =1, la que tenemos entre números enteros.

Notación Q: Es una forma de expresar rápidamente cuantos bits pertenecen a la parte entera y cuantos a la fraccional. Es una convención no un estándar.

¡Hay que sumar el bit de signo!

Vemos algunos ejemplos.

Se calcula la precisión y luego se resta.

Conversión de un numero real a uno en formato punto fijo y viceversa.

-Primero vemos un par de definiciones:

Como se define la unidad, Z=enteros, n=cantidad de bits de la parte decimal.

Agrego tantos ceros como la cantidad de bits que representen la parte decimal.

1<<n, operación de desplazamiento hacia la izquierda n bits. Es lo mismo que multiplicar por 2^n

Punto flotante es lo mas parecido a un real en una PC

Las expresiones son equivalentes. El numero en P. Flotante lo multiplicamos por 1 desplazado n lugares hacia la izquierda. (es la unidad representada en el formato punto fijo que definimos (cantidad de decimales) al cual queremos llevar el numero en P. Flotante.)

Este resultado, le indicamos al compilador que lo tiene que transformar en un entero, un casteo, lo fuerza a un entero.

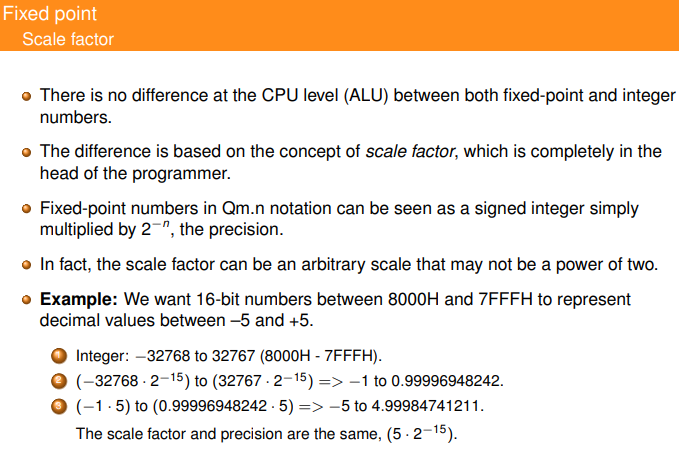
Operación inversa:

Tomamos el numero X en punto fijo, y lo “dividimos” por el 1 desplazado n lugares a la izquierda, que es lo mismo que multiplicar por 2^-n.

El resultado se castea a float.

Se observa en los ejemplos, que tenemos 2 números totalmente diferentes, que tienen la misma representación dentro de la PC, ambos valen 107 para la PC, un entero. Pero para el programador según el formato en punto fijo que elije, tienen diferentes significados, valen diferentes.

Es decir, un numero que tiene la misma representación en una PC tiene diferentes valores según el formato punto fijo que el programador este eligiendo.

FACTOR DE ESCALA: ¿Qué es?

Como vimos una PC, o en particular su ALU, no distingue si esta operando números en formato entero o punto fijo, solo el programador sabe con que tipo de formato está operando la CPU.

Porque el programador no tiene forma de indicarle a una computadora si una variable esta representada en formato punto fijo u entero. No existe un registro para ese fin.

En realidad, ambas variables (P. Fijo y un Entero) se definen como un INT cuando se escribe el código.

De esta diferencia nace el concepto del factor de escala y se dice que el factor de escala, solo esta en la cabeza del programador, porque la ALU solo opera con números enteros.

De acá, que podemos considerar que un numero en punto fijo, es un entero multiplicado por un factor de escala.

Si utilizamos notación Q, el factor de escala es la precisión. Es decir 2^-n

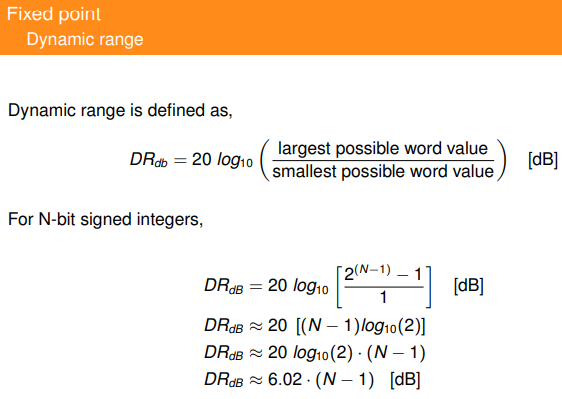
Pero, de hecho, el factor de escala puede ser un numero arbitrario, no necesariamente una potencia de 2.

Ver ejemplo. Entrada salida de un Conversor A/D y salida se muestra en un Display.

Pasos:

1. Tomamos los números enteros del rango de bits.
2. Al rango lo multiplicamos por la precisión.
3. Al nuevo rango lo multiplicamos por el rango deseado.

El factor de Escala y la precisión en realidad son las mismas.

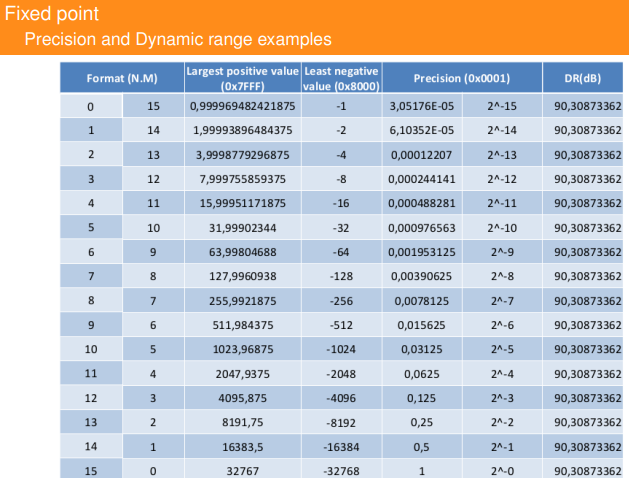
RANGO DINAMICO:

Nos da una idea de cuantos números podemos representar con una cantidad definida de bits.

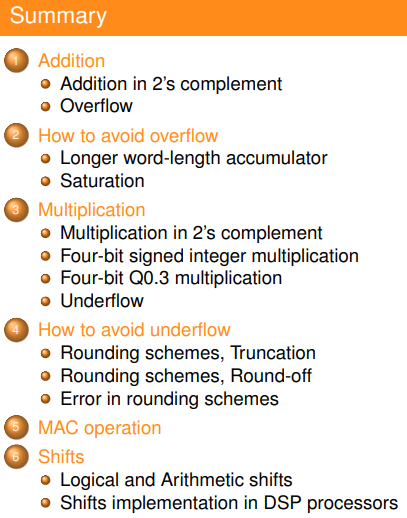
Este valor nos es útil cuando queramos comparar los rangos dinámicos de un numero representado en punto fijo, con el rango dinámico de un numero representado en punto flotante.

En esta formula N = Cantidad total de bits.

En la tabla se observan valores de precisión y rango dinámico, para diferentes formatos en punto fijo.



Se observa que el rango dinámico es el mismo, porque se define por la cantidad total de bits, y todos tienen 16 bits.

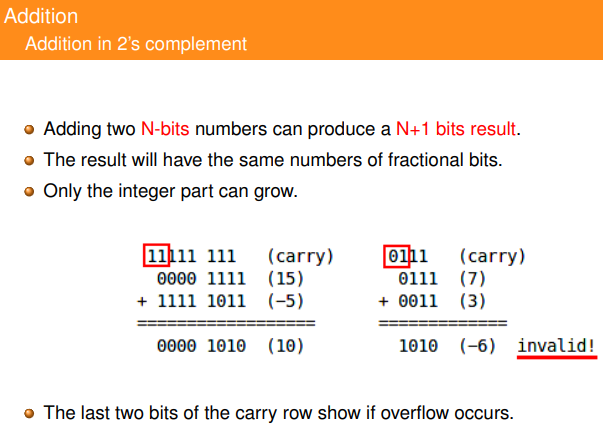
Operaciones Aritméticas Utilizando P. Fijo.

Error asociado a la operación de suma (en complemento a 2) y como evitarlo.

Multiplicación en complemento a 2, error asociado y como evitarlo.

Operación super importante en DPS.

Desplazamientos aritméticos y lógicos.

Suma:

* Cuando sumo w números de N bits, el resultado podrá tener cuando mucho N + 1 bits.
* Como estoy operando con números en punto fijo, el resultado mantiene la cantidad de bits para la parte decimal.
* Pero se puede incrementar la cantidad de bits que voy a necesitar para representar el resultado de la parte entera.

Comentarios: En una PC lo números típicamente se representan en formato complemento a 2, esto es así porque tiene la ventaja de que puedo usar un sumador en Hardware, tanto para sumar como para restar.

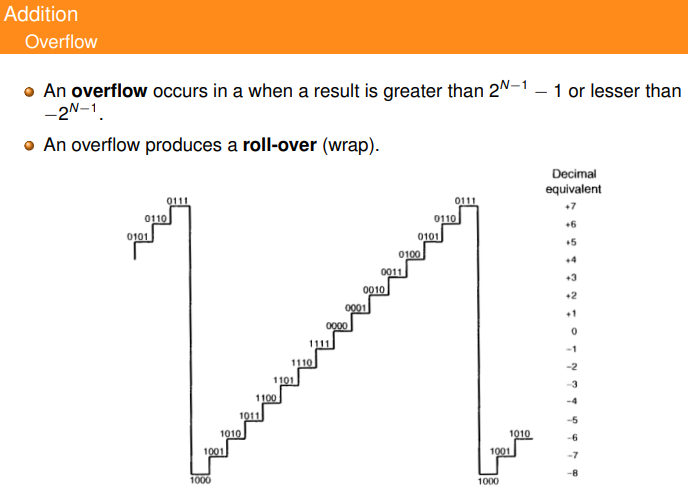
Los números enteros son un caso particular, de los números en P. Fijo. Por tanto, los ejemplos que están realizados con números enteros, se pueden generalizar para números en P. Fijo

En los ejemplos: Observar en la fila del Carry que los últimos 2 bits son iguales o distintos. IMPORTANTE!!!!

Overflow: Cuando se suman 2 números en complemento a 2 y el resultado se sale del rango, es decir necesito mas bits para representarlo, se obtiene un numero de diferente signo al esperado.

Este resultado es incorrecto y catastrófico.

Típicamente la arquitectura de una PC puede detectar si se produjo un overflow, leyendo los últimos 2 bits de la fila de acarreo, si son iguales no se produjo overflow, si son diferente sí. Cuando esto ocurre, la arquitectura de la Pc, generalmente tiene un Flag destinado para indicarlo, y un programador puede detectarlo leyendo este flag permanentemente.



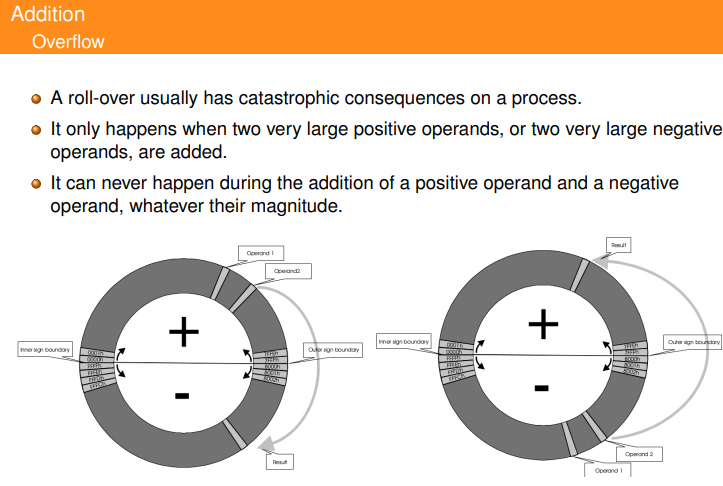
¿Cuándo se produce un Overflow?

Se produce cuando al realizar una suma o una resta el resultado sale del rango de representación de la cantidad de bits con la que estoy operando.

Ósea sumo 2 positivos o resto 2 negativos.

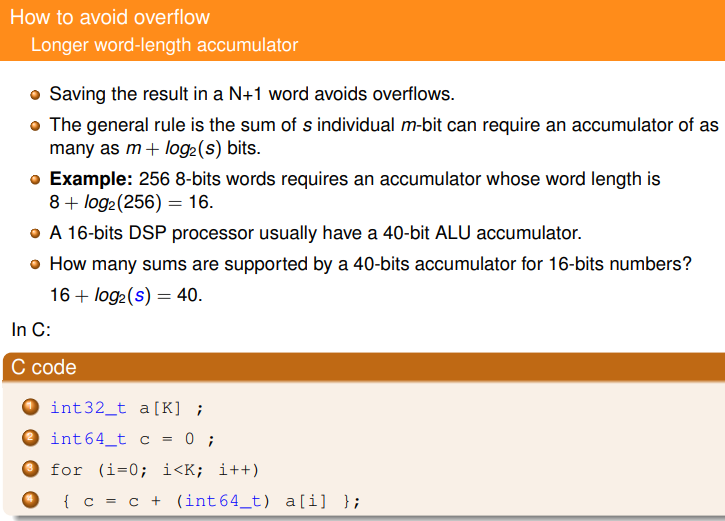
Al Overflow también se lo conoce como Roll-Over

Me caigo de la escalera y empiezo a subir del extremo inferior. (Extremo negativo del rango)



Se dice que un overflow puede tener consecuencias catastróficas porque al cambiar de signo, el resultado es todo lo contrario a lo que se esta esperando o debería ser.

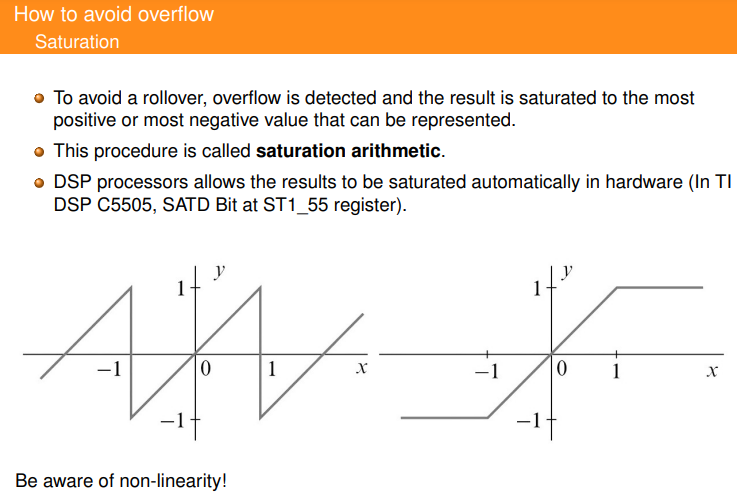
Observar que sumar 2 números muy positivos o negativos lleva a que el resultado caiga en la zona opuesta.

Formas de Evitar un Overflow:

La forma mas sencilla de evitar un overflow es guardar el resultado de la suma de 2 números de N bits, en un acumulador de N + 1 bits

Formula para calcular cuantos bits se necesitan para sumar “s” números de “m” bits cada uno.

Abría que calcular el tamaño máximo de K.

Otra manera de evitar un overflow, es implementando en nuestro sistema lo que se conoce como aritmética de saturación.

En la gráfica se observa el efecto roll-over.

Cuando el programador implementa aritmética de saturación, tenemos la curva derecha.

Ej: con 3 bits 7 + 3 = 7 … se satura al mayor valor de representación posible según el rango que se está utilizando de representación.

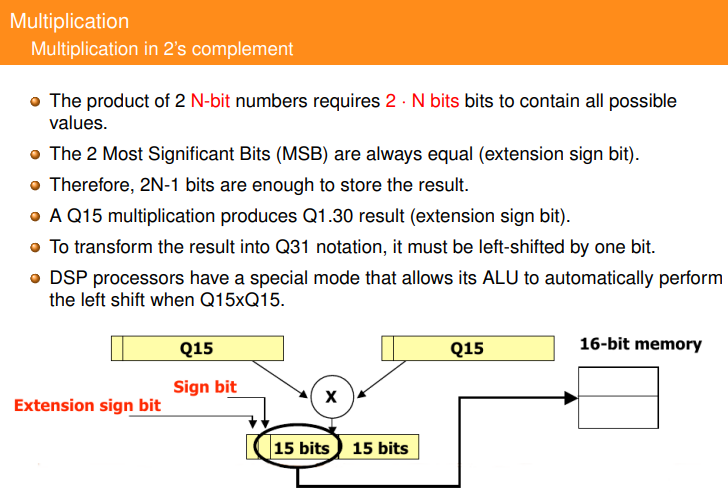
Pasamos de una situación catastrófica a una mala.

La curva representa un sistema ALINEAL

Entonces si implemento aritmético de saturación, y me la paso operando en los extremos, mi sistema se transforma en un sistema no lineal. Y si esto se implementa en el contexto de un filtro IIR o FIR, que son sistemas lineales, hay que tener cuidado … entonces pase a de tener un filtro lineal a uno no lineal.

Algunos procesadores DSP permiten implementar aritmética de saturación por hardware. Es decir, el programador se olvida del tema…. Pero típicamente en un procesador, como un ARM o una PC personal, la aritmética de saturación se tiene que implementar a través de funciones que debe desarrollar el programador.

MULTIPLICACIÓN en complemento a 2:

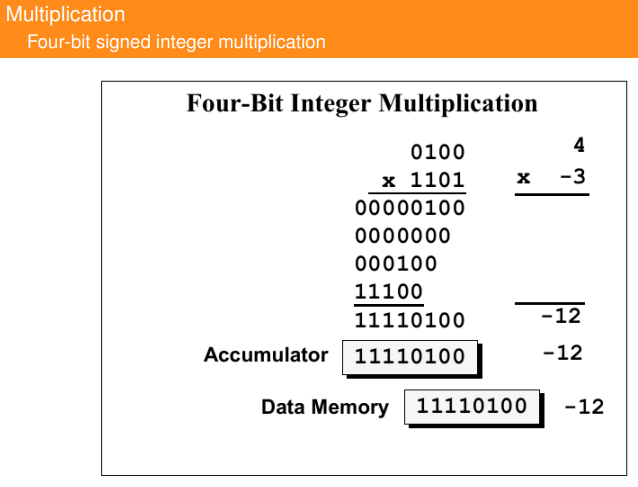
Cuando se multiplican dos números de N bits, se van a requerir para almacenar el resultado como mucho 2 x N bits. Es decir, se duplica la cantidad de bits.

Cuando se hace la multiplicación el resultado tiene 2 bits de signo, uno de signo y lo que se conoce como una extensión del bit de signo.

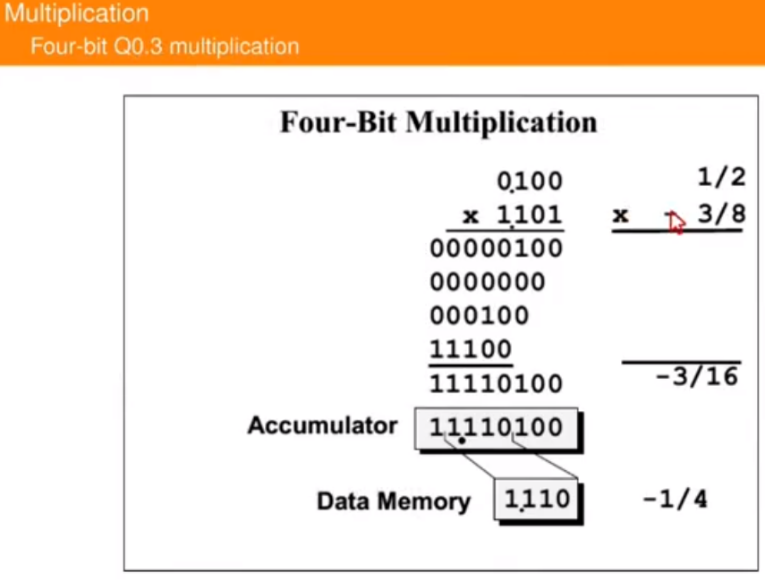
Cuando se hace una multiplicación, la ALU generalmente realiza un desplazamiento hacia la izquierda. Y elimina el bit de signo adicional.

Hay que tener presente que al duplicar la cantidad de bits para la parte decimal la precisión del resultado se duplico, el resultado es mas exacto.

En algún momento el resultado tiene que ir a la memoria, de 16 bits en este caso, entonces se tiene que plantear el programador que hacer con los bits extras que se obtuvieron al realizar la multiplicación.

Ejemplos:

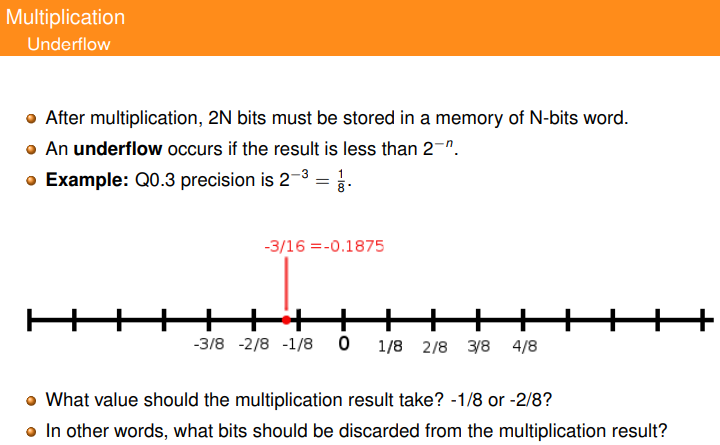
Resultado, el programador decide guardar en la memoria todos los bits.



El resultado de la multiplicación en el acumulador es de 8 bits.

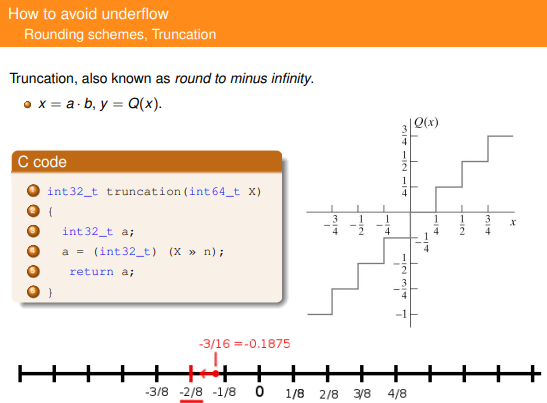
El programador decide solo tomar 4 bits, es decir volver al formato Q0.3.

Pasa de un numero que vale -0.1875 a -0.25

Al fenómeno de disminuir la precisión luego de hacer una multiplicación se lo conoce como Underflow.

En general esto sucede cuando el resultado, ve reducida la magnitud de su precisión. Es decir, aumentamos la cantidad de bits de la parte decimal. De n a 2n bits.

Se observa que el valor obtenido en el ejemplo se ubica entre 2 valores que podían representarse en el formato Q0.3. Por lo que hay que elegir que valor toma el resultado, -1/8 o -2/8.

A los métodos que definen cual de 2 posibles valores debe tomar el resultado de una multiplicación luego de producirse un underflow, se los conoce como esquemas de redondeo.

Vamos a ver 2 esquemas.

El 1ro se lo conoce como truncación o redondeo a menos infinito.

Al resultado de multiplicar 2 números, le aplicamos la función truncación Q.

X es el argumento de Q

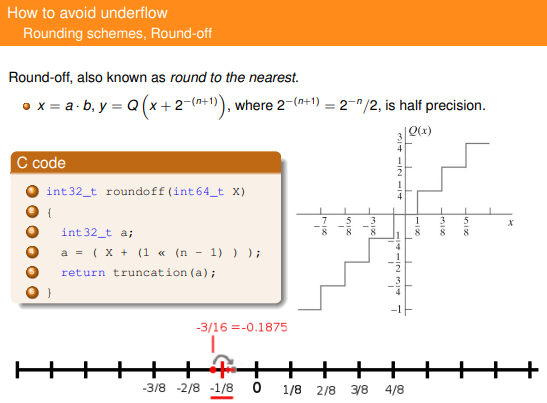
Y tiene el mismo formato que a y b

Lo que hace Q, el método de truncación es eliminar los bits extras que se generaron en la parte decimal del resultado luego de hacer una multiplicación.

En la grafica se observa la truncación para la precisión de 1/4

Si x es un numero real, los infinitos valores que se encuentre entre 0 y ¼, x será 0, en caso siguiente al aplicar la truncación x=1/4. Esto quiere decir que la salida de la truncación siempre resuelve hacia le valor de su izquierda o hacia menos infinito.

Se observa el ejemplo en rojo.

2do esquema de redondeo. Round-Off

Redondeo al más cercano.

Al valor de X se le suma 2^-(n+1), la mitad de la precisión y luego se aplica la truncación.

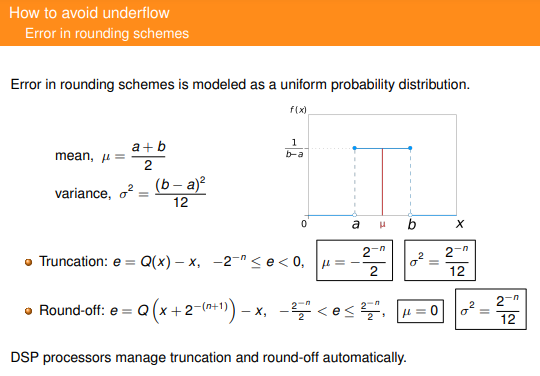
Se observa la recta del ejemplo anterior.

Lo que hace el redondeo round off, cuando se suma media precisión, lo que sucede es que el resultado pasa a estar a la derecha de -1/8 entonces al aplicar Q, y pasa a valer -1/8 … Es decir, el resultado pasa a tomar el valor mas cercano, de los 2 valores posibles con los cuales se puede representar.

Ejemplo de una posible implementación en C.

Para entender la línea 4 del código. Supongamos que n vale 16, es decir que el formato original de a y b era Q15.16, entonces la precisión de este formato esta dada por 2^-16 y la mitad (divido por 2) de esta precisión seria 2^-17.

Para sumar media precisión, se le tiene que sumar a x un 1, en la posición 15. Este resultado se guarda en a y se aplica truncación.



Luego de ver estos 2 esquemas de redondeo, truncación y round off, se puede intuir que el ultimo presenta el menor error.

El error en estos métodos se modela como una distribución de probabilidad uniforme. Que esta definida por los externos a y b, y la media dada por la letra mu (u).

Se dan las expresiones de la media y varianza.

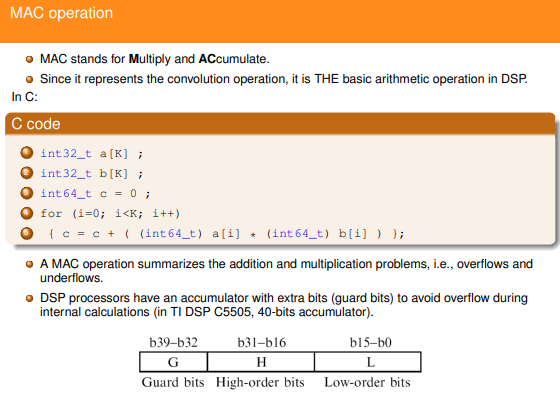
Se analizan los valores para ambos métodos.

Se define el error.

En la truncación la media tiene un sesgo.

En ronund-off la media vale cero y la varianza es igual. No tiene sesgo, ofrece menos error.

Operación MAC.

Operación aritmética fundamental cuando se trabaja con procesamiento de señales digitales (DPS)

MAC: Multiplicar y ACumular.

Esta operación implementa la convolución entre 2 señales. Y además es una operación esencial en la transformada rápida de Fourier, que es, a su vez, una implementación optimizada de la transformada discreta de Fourier

Lo que hace la operación MAC es multiplicar el contenido de 2 vectores, elemento a elemento e ir acumulando el resultado de la multiplicación.

Ejemplo de un posible código en C.

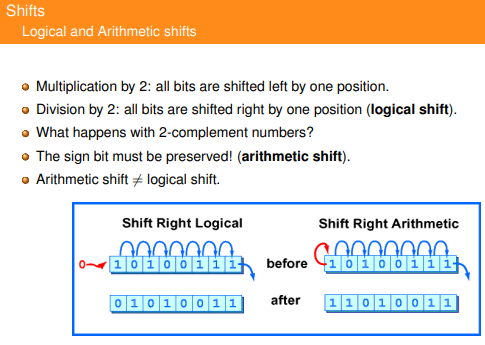
Como la operación MAC realiza tanto multiplicaciones como sumas, es susceptible de que se produzcan tanto overflows como underflows.

Como se mencionó la operación MAC es la operación básica cuando se trabaja en DSP, por lo que los procesadores DSP están optimizados para implementar esta operación.

Generalmente la implementan en un ciclo de reloj. Y además cuenta con un acumulador en su ALU que permite abordar de mejor manera los problemas que producen tanto overflows como underflows.

En la imagen se observa un esquema de un acumulador de un procesador DSP, que opera con números de 16 bits. Y el acumulador tiene 40 bits, de los cuales, 32 son para almacenar el resultado de una multiplicación, y además cuenta con 8 bits extras para poder manejar correctamente los overflows que se puedan producir.

Desplazamientos:

Estas operaciones son importantes porque desplazar un numero binario hacia la izquierda un lugar, implica multiplicar por 2, y desplazarlo hacia la derecha un lugar dividirlo por 2.

Entonces las operaciones de desplazamiento son importantes, porque desde el punto de vista del esfuerzo de cálculo, es mucho más fácil realizar un desplazamiento que multiplicar en la ALU por 2.

Los desplazamientos se dividen en Lógicos y Aritméticos.

En un Dto. Lógico se desplaza el numero hacia la derecha un lugar, y el numero mas significativo se rellena con un cero.

En el ejemplo, si el numero esta en complemento a 2, es un numero negativo, si ponemos un cero pasaría a ser un numero positivo y estamos cometiendo un error.

A partir de esto surge la necesidad de implementar un desplazamiento aritmético, en el cual se desplaza el numero hacia la derecha, pero en el bit mas significativo se vuelve a colocar el mismo valor previo. Si hay 1, queda en 1. Si hay 0 queda en 0.

