

Unidad 1.B. Representación finita de números reales en punto fijo

Dr. Ing. Hernán Garrido

Control y sistemas
Universidad Nacional de Cuyo, Facultad de Ingeniería

carloshernangarrido@gmail.com

Agosto de 2023



1 Representación en punto fijo

- Definición
- Números enteros y punto fijo
- Notación $Q_m.n$ Rango y precisión.
- Conversión de punto flotante a punto fijo y viceversa.
- Escala de representación. Rango dinámico.

2 Aritmética de punto fijo

- Suma complemento a 2. Overflow. Saturación. Acumulador, bits de guarda.
- Multiplicación en complemento a 2. Underflow. Esquemas de redondeo, Truncamiento y round-off.
- Desplazamientos lógico y aritmético.

Motivación

- Es imposible representar infinitos números en una computadora.
- El programador debe elegir la mejor representación de acuerdo a la aplicación.

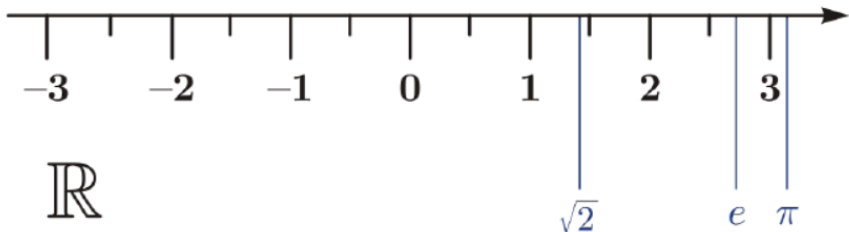


Figura: Recta real.

ars TECHNICA


BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE STORE

OPPA GANGNAM STYLE —

Gangnam Style overflows INT_MAX, forces YouTube to go 64-bit

Psy's hit song has been watched an awful lot of times.

ARS STAFF · 12/3/2014, 7:32 PM




Mirar en  YouTube

Figura: El problema de Gangnam Style.

Representación en binario

Una palabra binaria de N bits $b_{N-1} \dots b_2 b_1 b_0$ puede representar un total de 2^N valores distintos.

- Enteros sin signo:
 - Rango: desde 0 hasta $2^N - 1$.
 - El número en base 10 se recupera mediante:

$$n_{10} = 2^{N-1} b_{N-1} + \dots + 2^1 b_1 + 2^0 b_0$$

- Enteros con signo en complemento a 2:
 - Rango: desde -2^{N-1} hasta $2^{N-1} - 1$
 - El número en base 10 se recupera mediante:

$$n_{10} = -b_{N-1} 2^{N-1} + 2^{N-2} b_{N-2} \dots + 2^1 b_1 + 2^0 b_0$$

Números enteros: Ejemplos

Binario	Decimal sin signo	Decimal en complemento a dos
000	0	0
001	1	1
010	2	2
011	3	3
100	4	-4
101	5	-3
110	6	-2
111	7	-1

Cuadro: Comparación de enteros sin signo y en complemento a dos (3 bits).

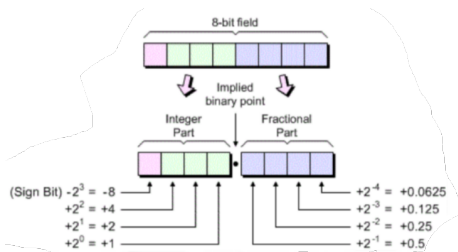
En lenguaje C:

- 8 bits (char, int8_t): [-128, 127]
- 16 bits (short, int16_t): [-32768, 32767]
- 32 bits (int, long, int32_t): [-2147483648, 2147483647]

Representación en punto fijo

Un número real x se representa mediante un entero X con $N = m + n + 1$ bits, donde

- N es el tamaño de la palabra binaria
- m es el número de bits de la parte entera (a la izquierda del punto binario)
- n es el número de bits de la parte fraccionaria (a la derecha del punto binario)
- 1 bit para el signo

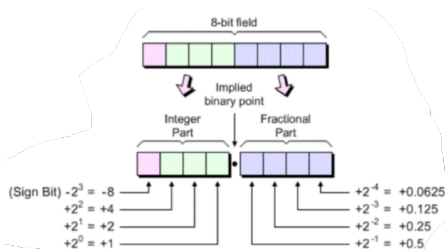


Representación en punto fijo

- Los pesos de los bits fraccionarios son potencias negativas de 2: $1/2$, $1/4$, $1/8$, ...
- El número decimal se recupera mediante:

$$n_{10} = -b_m 2^m + \left(\sum_{i=0}^{m-1} b_i 2^i + \sum_{i=1}^n b_i 2^{-i} \right)$$

- Rango: desde -2^m hasta $2^m - 2^{-n}$
- Precisión: 2^{-n}



Representación en punto fijo: Ejemplos

Notación Qm.n: m bits para la parte entera, n bits para la parte fraccionaria, 1 bit para el signo. Por ejemplo: Q1.1 es un número de 3 bits que permite representar:

Binario	Decimal
00.0	0.0
00.1	0.5
01.0	1.0
01.1	1.5
10.0	-2.0
10.1	-1.5
11.0	-1.0
11.1	-0.5

- Rango: desde $-2^m = -2^1 = -2$ hasta $2^m - 2^{-n} = 2^1 - 2^{-1} = 2 - 0,5 = 1.5$
- Precisión: $2^{-n} = 2^{-1} = 0.5$

Representación en punto fijo: Ejemplos

- Q0.15 (Q15)
 - 16 bits;
 - Rango: desde -1 hasta 0.99996948;
 - Precisión: $\frac{1}{32768}$ (2^{-15}).
- Q3.12
 - 16 bits;
 - Rango: desde -8 hasta 7.9998;
 - Precisión: $\frac{1}{4096}$ (2^{-12}).
- Q0.31 (Q31)
 - 32 bits;
 - Rango: desde -1 hasta 0.999999999534339;
 - Precisión: $4.6566129 \times 10^{-10}$ (2^{-31}).

Conversión a y desde punto flotante

Definición de desplazamiento a la izquierda

Unidad (1): $1 \ll n = 1 \cdot 2^n$

Un medio (1/2): $1 \ll (n - 1) = 1 \cdot 2^{n-1}$

Conversión de número de punto flotante x (real) a número de punto fijo X mediante casting:

```
X = (int)(x * (1 << n));
```

```
X = (int)(x * pow(2, n)); // equivalente pero
```

```
↪ ineficiente
```

Conversión de número de punto fijo X a número de punto flotante x (real) mediante casting:

```
x = (float)X / (1 << n);
```

```
x = (float)X * pow(2, -n); // equivalente pero
```

```
↪ ineficiente
```

Factor de escala

- No hay diferencia a nivel de CPU (ALU) entre los números de punto fijo y los números enteros.
- La diferencia se basa en el concepto de factor de escala, que está completamente en la cabeza del programador.
- Los números de punto fijo en notación $Q_m.n$ pueden verse como un número entero con signo simplemente multiplicado por 2^{-n} , la precisión.
- De hecho, el factor de escala puede ser una escala arbitraria que no sea una potencia de dos.

Por ejemplo: Queremos números de 16 bits entre 8000H y 7FFFH para representar valores decimales entre -5 y +5.

- Entero: -32768 a 32767 (8000H - 7FFFH).
- (-32768×2^{-15}) a $(32767 \times 2^{-15}) = -1$ a 0.99996948242.
- (-1×5) a $(0,99996948242 \times 5) = -5$ a 4.99984741211.

El factor de escala y la precisión son iguales (5×2^{-15}).

Definición

$$DR_{dB} = 20 \log_{10} \left(\frac{\text{valor-mas-grande-possible}}{\text{valor-mas-chico-possible}} \right) \text{ [dB]}$$

Para enteros con signo de N bits:

$$DR_{dB} = 20 \log_{10} \left(\frac{2^{N-1} - 1}{1} \right)$$

$$DR_{dB} \approx 20 \log_{10} (2^{N-1})$$

$$DR_{dB} \approx 20(N - 1) \log_{10} 2$$

$$DR_{dB} \approx 6,02(N - 1)$$

Regla de pulgar: 6 dB por bit útil.

Precisión y rango dinámico: Ejemplos

Ejemplos de representaciones en punto fijo de 16 bits con signo (todos tienen el mismo rango dinámico de 90.3 dB):

Qm.n	Precisión	V_{\min}	V_{\max}
Q15.0	$2^0=1$	-32768	32767
Q10.5	$2^{-5}=0.03125$	-1024.00000	1023.96875
Q1.14	$2^{-14}=0.0000610$	-2.000000000000000	1.99993896484375
Q0.15	$2^{-15}=0.0000305$	-1.000000000000000	0.999969482421875

1 Representación en punto fijo

- Definición
- Números enteros y punto fijo
- Notación $Q_m.n$ Rango y precisión.
- Conversión de punto flotante a punto fijo y viceversa.
- Escala de representación. Rango dinámico.

2 Aritmética de punto fijo

- Suma complemento a 2. Overflow. Saturación. Acumulador, bits de guarda.
- Multiplicación en complemento a 2. Underflow. Esquemas de redondeo, Truncamiento y round-off.
- Desplazamientos lógico y aritmético.

Suma complemento a 2

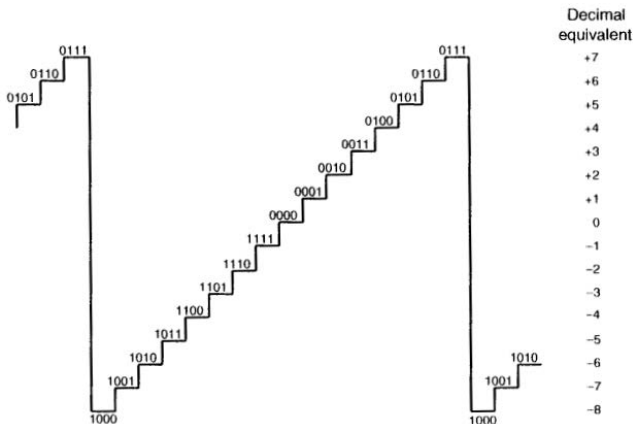
- Sumar dos números de N bits puede producir un resultado de hasta $N + 1$ bits.
- El resultado tiene el mismo número de bits en la parte fraccional.
- Sólo la parte entera puede crecer en 1 bit.

11 111 111 (carry)	01 11 (carry)
0000 1111 (15)	0111 (7)
+ 1111 1011 (-5)	+ 0011 (3)
=====	
0000 1010 (10)	1010 (-6) <u>invalid!</u>

Si los últimos dos bits del acarreo son diferentes, hubo overflow.

Suma en complemento a 2: Overflow

- El overflow ocurre cuando el resultado es menor que -2^{N-1} o mayor que $2^{N-1} - 1$. Esto es viendo el número como un entero con signo de N bits en total.
- El overflow produce una caída en escalera:



Suma en complemento a 2: Overflow

Características del overflow:

- El efecto del overflow no controlado es *catastrófico*, ya que el resultado es muy diferente del esperado. En el caso de un sistema de control, esto implica una discontinuidad.
- Sólo sucede cuando se suman dos números positivos o dos números negativos con valores absolutos muy grandes.
- Nunca ocurre si los sumandos son de signo opuesto.

Técnicas para tratar el overflow: Acumulador largo

Solución simple: Un acumulador para el resultado más largo que los sumandos.

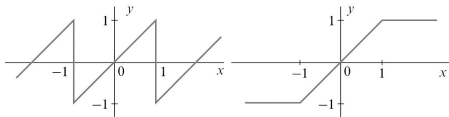
- Si son dos sumandos de N bits, un acumulador de $N + 1$ bits es suficiente.
- Si son s sumandos de N bits, el acumulador debe tener: $N + \log_2(s)$.
 - Por ejemplo, para sumar 256 números de 8 bits, se necesita un acumulador de $8 + \log_2(256) = 8 + 8 = 16$ bits.

```
int32_t a[K];
int64_t c = 0;
/* Asignación de valores de a */
for (i=0; i<K; i++)
{
    c = c + (int64_t) a[i];
}
```

Técnicas para tratar el overflow: Saturación

Otra solución: La saturación no evita el overflow pero sí la caída en escalera o rollover. Esto es menos catastrófico.

- Introduce una no linealidad pero evita la discontinuidad.
- De todos modos se requiere operar primero con un acumulador más largo.



```
#include <stdint.h>
int32_t saturate(int64_t x) {
    if (x > INT32_MAX) { return INT32_MAX; }
    else if (x < INT32_MIN) { return INT32_MIN; }
    else { return (int32_t)x; }
}
```

Multiplicación en complemento a 2: Enteros

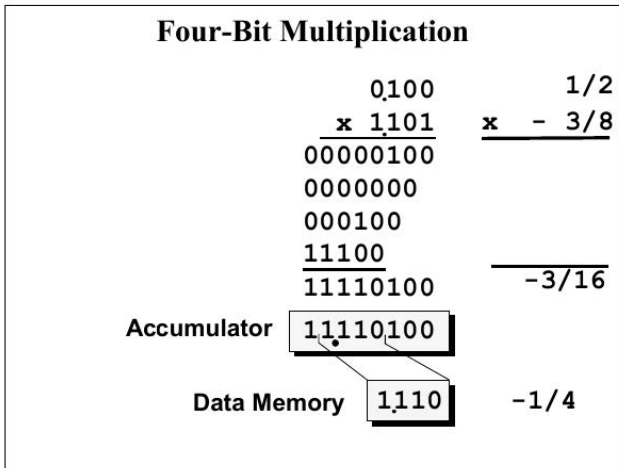
Multiplicar dos números de N bits puede producir un resultado de hasta $2N$ bits.

Four-Bit Integer Multiplication

	0100	4
	<u>x 1101</u>	<u>x -3</u>
	00000100	
	0000000	
	000100	
	<u>11100</u>	
	11110100	-12
Accumulator	11110100	-12
Data Memory	11110100	-12

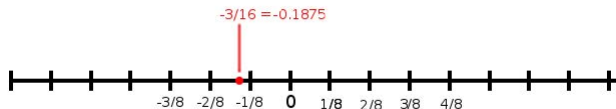
Multiplicación en complemento a 2: Punto fijo

Multiplicar dos números de N bits puede producir un resultado de hasta $2N$ bits. En punto fijo *se duplica el número de bits de la parte fraccional*.



Multiplicación en complemento a 2: Underflow

- Después de multiplicar dos números de N bits, se obtiene un número de $2N$ bits que normalmente debe almacenarse en memoria con N bits.
- El *underflow* ocurre si el resultado de la multiplicación es menor a 2^{-n} .
 - Al truncar los n bits que aparecen, se obtiene exactamente 0.
- Ejemplo: La precisión de Q0.3 es $2^{-3} = 1/8$



- ¿Qué número habría que almacenar $-1/8$ o $-2/8$?
- Siempre será necesario algún *esquema de redondeo* si se desea usar el mismo tamaño para los factores y el resultado.

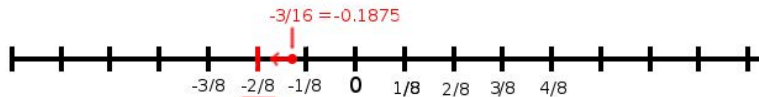
Esquemas de redondeo: Truncamiento

Truncamiento

$$y = Q(x)$$

Es un redondeo a $-\infty$.

```
#include <stdint.h>
/* n es el número de bits de la parte fraccionaria */
int32_t truncation(int64_t X)
{
    int32_t a;
    a = (int32_t) (X >> n);
    return a;
}
```



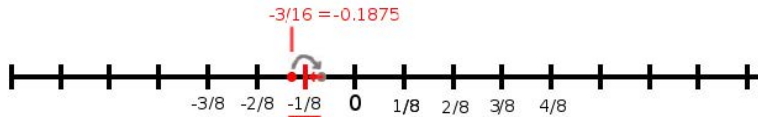
Esquemas de redondeo: Redondeo

Redondeo

$$y = Q(x + 2^{-(n+1)})$$

Es un redondeo al valor más cercano. Se suma la mitad de la precisión y luego se trunca.

```
int32_t roundoff(int64_t X)
{
    int32_t a;
    a = X + (1 << (n-1));
    return truncation(a);
}
```



Operación MAC

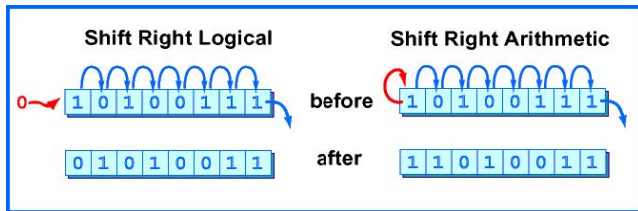
La operación MAC (multiplicaciones y acumulación) es la operación básica en los procesadores digitales de señales.

```
int32_t a[K];
int32_t b[K];
int64_t c = 0;
for (i=0; i<K; i++){
    c = c + ( (int64_t) a[i] * (int64_t) b[i] );
};
```

- Es la implementación en código de la *convolución*.
- Los Procesadores Digitales de Señales (DSP) la ejecutan por hardware (directamente en la ALU). Los DSP tienen:
 - acumuladores con bits de guarda para evitar overflow, y
 - manejan los esquemas de redondeo automáticamente.

Desplazamientos lógico y aritmético

- Multiplicación por 2: todos los bits se desplazan hacia la izquierda en una posición.
- División por 2: todos los bits se desplazan hacia la derecha en una posición (desplazamiento lógico).
- ¿Qué sucede con números en complemento a 2?
- ¡El bit de signo debe ser preservado! (desplazamiento aritmético).
- Desplazamiento aritmético \neq desplazamiento lógico.



- 1 Richard G. Lyons. Understanding Digital Signal Processing, 3rd Ed. Prentice Hill. 2010. Chapter 12.
- 2 Bruno Paillard. An Introduction To Digital Signal Processors, Chapter 5.