

Representación de datos en el sistema de cómputo

Las computadoras almacenan en memoria dos tipos de información: datos e instrucciones. Para ello utiliza el sistema binario por dos razones fundamentales: el dispositivo se encuentra en uno de dos posibles estados y es más fácil identificar cual es este estado. Distinto sería el caso si fuera necesario identificar uno de diez posibles, en el caso de trabajar en base diez.

Dentro del sistema de cómputo estos unos y ceros pueden representar a números enteros con y sin signo, números reales con y sin signo, caracteres, instrucciones, imágenes, etc.

Vamos a ver como en el sistema de cómputo se representan números enteros, en principio sin signo y luego con signo.

Representación de números enteros sin signo (BSS = binario sin signo)

El número es representado en base 2 en la forma conocida hasta ahora. Si el número está formado por n bits, donde bits es cada uno de los dígitos binarios que forman al número, se pueden escribir 2^n números distintos. El rango de este sistema va desde 0 hasta $2^n - 1$.

Ejemplo: un sistema BSS con $n = 3$ bit. Números distintos = $2^n = 2^3 = 8$

Decimal	Representación sin signo
0	000
1	001
2	010
..
7	111

Ejemplo: un sistema BSS con $n = 8$ bits. Números distintos = $2^n = 2^8 = 256$

Decimal	Representación sin signo
0	00000000
..
128	10000000
..
254	11111110
255	11111111

La cantidad de números distintos (representaciones) está depende de la base y el número de dígitos.

Representación de números enteros con signo

Vamos a ver distintos sistemas que cumplen con el objetivo: Binario con signo (BCS) ó Módulo y signo (MyS), Complemento a la base reducida (Complemento a uno = Ca1), Complemento a la base (Complemento a dos = Ca2) y Exceso.

Binario con signo (BCS) ó Módulo y signo (MyS)

Un número con n bits en este sistema tiene el siguiente formato:



El bit n-1, el del extremo izquierdo, representa sólo al signo y no forma parte del valor del número. El resto de los bits, 0 a n-2, representan el valor (módulo) del número en BSS.

Un 0 en el bit de signo indica que el número es positivo. Un 1 en el bit de signo indica que el número es negativo.

Ejemplos: con n = 8 bits $+32_{10} = 00100000$ $-32_{10} = 10100000$

En el primer ejemplo vemos que +32 empieza con 0 indicando que es positivo y en el segundo ejemplo -32 empieza con 1 indicando que es negativo. En ambos ejemplos el resto de los bits son iguales pues representan al número 32 (módulo).

Otros ejemplos donde se pueden observar los mismos conceptos:

$+7_{10} = 00000111$	$-7_{10} = 10000111$
$+41_{10} = 00101001$	$-41_{10} = 1010100$

El rango de este sistema va desde el número más pequeño $-(2^{n-1} - 1)$ hasta el más grande $+(2^{n-1} - 1)$, teniendo un cero +, 000000000 y otro cero -, 100000000. La cantidad de números distintos, en realidad la cantidad de representaciones distintas sigue siendo 2^n y no depende del sistema de representación sino de la base y la cantidad de dígitos.

Ejemplo: un sistema BCS con n = 3 bit. Números distintos = $2^n = 2^3 = 8$

Decimal	Representación con signo
-3	$111 = -(2^{n-1} - 1)$
-2	110
-1	101

Computación – Lic. en Física Médica y Lic. en Física
Curso 2024 - Prof. Jorge Runco

-0	100
+0	000
+1	001
+2	010
+3	011 = $+(2^{n-1} - 1)$

Ejemplo: un sistema BCS con $n = 8$ bits. Representaciones distintas = $2^n = 2^8 = 256$.

Números negativos	{	11111111	$-(2^{n-1} - 1) = -127$
		...	
		10000000	- 0
Números positivos	{	00000000	+0
		...	
		01111111	$+(2^{n-1} - 1) = +127$

De los ejemplos anteriores vemos que en realidad no “inventamos” combinaciones de unos y ceros para representar a los números negativos. Las mismas combinaciones que representan a los negativos en este sistema (BCS), están presentes en BSS, pero representan a otros números. Como la cantidad de representaciones es la misma para ambos sistemas (con el mismo número de bits), lo que cambia necesariamente es el rango de cada sistema. A modo de ejemplo recordemos con $n = 8$ bits, en ambos casos hay $2^8 = 256$ representaciones distintas, en BSS el rango va desde 0 hasta 255 (256 representaciones), mientras que en BCS el rango va desde -127 a +127 y con dos ceros (256 representaciones).

Si bien este mecanismo es sencillo para pasar de un número positivo a negativo ó al revés, basta con cambiar el dígito de signo como vimos en los ejemplos anteriores, este sistema presenta problemas a la hora de realizar álgebra. El primer bit no puede ser “sumado directamente” en una operación aritmética, debe ser tratado como un “símbolo” pues representa sólo al signo y no forma parte del valor del número. Otro problema es que este sistema tiene dos ceros.

Técnica de Complementos

Nos ocuparemos ahora de otro sistema de representación de números positivos y negativos dentro del sistema de cómputo. Para eso primero vamos a ver algunas definiciones: complemento a la base y complemento a la base reducida.

El complemento a un número N de un número A (A menor que N) es igual a la cantidad que le falta a A para ser N:

Complemento a N de A = $N - A$

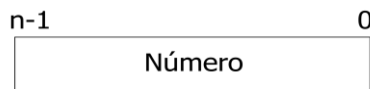
También es llamado complemento a la base.

De manera más general en un sistema de base b y con n dígitos $N = (base)^n$ y la definición anterior se llama complemento a la base. Si la base es 2 el complemento a la base se llama complemento a 2 (Ca2).

Si $N = (base)^n - 1$ la definición anterior se llama complemento a la base reducida y si la base es dos se llama complemento a 1 (Ca1).

Representación en complemento a 1 (Ca1)

Un número con n bits en este sistema tiene el siguiente formato:



A diferencia del sistema anterior el bit más significativo (bit del extremo izquierdo) forma parte del número, sigue dando información del signo del número, pero no es sólo el signo como en BCS.

Si el número es positivo los n bits tienen la representación binaria del número, es decir en este sistema los números positivos coinciden con la representación en BCS y BSS.

Si el número es negativo, el mismo es representado por el complemento a 1 del número deseado. Para obtener el Ca1 de un número podemos aplicar la definición anterior ó utilizar la siguiente regla práctica: invertir todos los bits.

Como consecuencia de lo dicho, los positivos empiezan con 0 y los negativos con 1.

Ejemplos: con $n = 8$ bits $+32_{10} = 00100000$

Para representar -32 en Ca1, se invierten todos los bits de $+32$

$-32_{10} = 11011111$

Otros ejemplos:

$+7_{10} = 00000111$ $-7_{10} = 11111000$

$+41_{10} = 00101001$ $-41_{10} = 11010110$

El rango de este sistema va desde el número más pequeño $-(2^{n-1} - 1)$ hasta el más grande $+(2^{n-1} - 1)$, teniendo un cero +, 00000000 y otro cero -, 11111111. La cantidad de números distintos, en realidad la cantidad de representaciones distintas sigue siendo 2^n y no depende del sistema de representación sino de la base y la cantidad de dígitos.

Ejemplo: un sistema Ca1 con $n = 3$ bit. Números distintos = $2^n = 2^3 = 8$

Computación – Lic. en Física Médica y Lic. en Física
Curso 2024 - Prof. Jorge Runco

Decimal	Representación con signo
-3	$100 = -(2^{n-1} - 1)$
-2	101
-1	110
-0	111
+0	000
+1	001
+2	010
+3	$011 = +(2^{n-1} - 1)$

Ejemplo: un sistema Ca1 con $n = 8$ bits. Representaciones distintas = $2^n = 2^8 = 256$.

Números negativos	$\left\{ \begin{array}{ll} 10000000 & -(2^{n-1} - 1) = -127 \\ \dots & \\ 11111111 & -0 \end{array} \right.$
Números positivos	$\left\{ \begin{array}{ll} 00000000 & +0 \\ \dots & \\ 01111111 & +(2^{n-1} - 1) = +127 \end{array} \right.$

Veamos ahora el siguiente problema: dada una cadena de bits ¿cómo determinamos a qué número decimal corresponde?

Si el número es positivo

$$01100000 = 1 \times 2^6 + 1 \times 2^5 = 64 + 32 = 96$$

Aplicamos lo que ya sabíamos del Teorema Fundamental de la Numeración, darle el peso correspondiente a cada dígito que vale 1.

Si el número es negativo puedo hacer tres cosas:

- Hacer el Ca1 del número negativo, nos va a dar un número positivo cuyo valor podemos calcular como en el caso anterior, pero debemos recordar que estábamos calculando el valor del negativo, es decir hay que cambiar de signo al valor calculado

Ejemplo: ¿A qué número decimal representa la siguiente cadena de bits 11100000?

Como es negativo (empieza con 1) aplicamos la regla de complemento a la base disminuida que es invertir todos los dígitos: 00011111. Como el número ahora es positivo podemos darle peso a cada uno de sus dígitos

$$00011111 = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 16 + 8 + 4 + 2 + 1 = +31$$

En resultado que buscábamos en realidad es -31 .

- Otro método consiste en darle peso a todos los dígitos, inclusive al bit que indica el signo. Dijimos que este bit formaba parte del número por lo tanto debe tener un peso, nuestro problema es saber cuánto vale dicho peso. Sin demostración diremos que el peso es uno menos al que le corresponde a esa posición y luego haciéndolo negativo, es decir $-(2^{n-1} - 1)$. El resto de los dígitos tienen los pesos conocidos hasta ahora.

Ejemplo: resolveremos el ejemplo anterior con este nuevo método.

$$11100000 = -1 \times (2^7 - 1) + 1 \times 2^6 + 1 \times 2^5 = -127 + 64 + 32 = -31$$

- El último de los tres métodos consiste en aplicar la definición de complemento a la base disminuida

$$Ca1 = b^n - 1 - N^o$$

Lo que tenemos que hacer es despejar N^o pues todo lo demás es conocido. Este método puede prestar a confusión porque el número en $Ca1$, debe ser tratado como BSS.

Ejemplo: $11100000 = 2^7 + 2^6 + 2^5 = 128 + 64 + 32 = 224$

Volvamos a la definición $224 = 2^8 - 1 - N^o = 256 - 1 - N^o$ $224 - 256 + 1 = N^o$

$$N^o = -31$$

Si bien con este sistema ($Ca1$) solucionamos el problema del álgebra pues el bit más significativo forma parte del número (tiene peso) y por lo tanto puede intervenir en las operaciones aritméticas. Nos falta solucionar el problema de los dos ceros.

Representación en complemento a 2 ($Ca2$)

Un número con n bits en este sistema tiene el siguiente formato:



Al igual que en $Ca2$ el bit más significativo (bit del extremo izquierdo) forma parte del número y sigue dando información del signo del número.

Si el número es positivo los n bits tienen la representación binaria del número, es decir en este sistema los números positivos coinciden con la representación en Ca1, BCS y BSS.


Si el número es negativo, el mismo es representado por el complemento a 2 del número deseado. Para obtener el Ca2 de un número podemos aplicar la definición de complemento a la base $Ca2 = b^n - N^o$. Otra forma sería: invertir todos los bits (hacer el Ca1) y sumarle 1. Alternativamente también se podría utilizar la siguiente regla práctica: mirando desde la derecha al número, dejar todos los dígitos iguales hasta el primer 1 inclusive y luego invertir el resto de los dígitos.


Notemos que en BCS, Ca1 y Ca2, los números positivos se representan de la misma manera, lo que cambia para cada sistema es la representación de los negativos.

Como consecuencia de lo dicho, los positivos empiezan con 0 y los negativos con 1.

Ejemplos: con $n = 8$ bits $+32_{10} = 00100000$

Para representar -32 en Ca2, aplicando el primer método

$+32_{10} = 00100000$  “mirando desde la derecha” copiamos hasta el primer 1 inclusive e invertimos los siguientes

 $-32_{10} = 11100000$

Otros ejemplos:

$+7_{10} = 00000111$ $-7_{10} = 11111001$

$+41_{10} = 00101001$ $-41_{10} = 11010111$

El rango de este sistema va desde el número más pequeño $-(2^{n-1})$ hasta el más grande $+(2^{n-1} - 1)$, teniendo un cero +, 00000000 (sólo uno). La cantidad de números distintos, en realidad la cantidad de representaciones distintas sigue siendo 2^n y no depende del sistema de representación y si de la base y la cantidad de dígitos. Vemos que hay un negativo más, al no tener unos cero negativos hemos recuperado una representación que empieza con uno, por lo tanto hay un número negativo más.

Ejemplo: un sistema Ca2 con $n = 3$ bit. Números distintos $= 2^n = 2^3 = 8$

Decimal	Representación con signo
-1	111
-2	110
-3	101
-4	100 = $-(2^{n-1})$

+0	000
+1	001
+2	010
+3	011 = $+(2^{n-1} - 1)$

Ejemplo: un sistema Ca2 con $n = 8$ bits. Representaciones distintas = $2^n = 2^8 = 256$.

Números negativos	{	11111111	- 1
		...	
		10000000	$-(2^{n-1}) = -128$
Números positivos	{	00000000	+0
		...	
		01111111	$+(2^{n-1} - 1) = +127$

Este sistema tiene un solo cero por lo tanto está solucionado el problema del álgebra. Más adelante veremos como hace las cuentas el sistema de cómputo.

Como hicimos en Ca1, veamos ahora el siguiente problema: dada una cadena de bits en Ca2 ¿cómo determinamos a qué número decimal corresponde?

Si el número es positivo

$$01100000 = 1 \times 2^6 + 1 \times 2^5 = 64 + 32 = 96$$

Aplicamos lo que ya sabíamos del Teorema Fundamental de la Numeración, darle el peso correspondiente a cada dígito que vale 1.

Si el número es negativo puedo hacer tres cosas:

- Hacer el Ca2 del número negativo, nos va a dar un número positivo cuyo valor podemos calcular como en el caso anterior, pero debemos recordar que estábamos calculando el valor del negativo, es decir hay que cambiar de signo al valor calculado

Ejemplo: ¿A qué número decimal representa la siguiente cadena de bits 11100000 en Ca2?

Como es negativo (empieza con 1) aplicamos la regla de complemento que ya mencionamos, mirando desde la derecha copiar hasta el primer 1 inclusive y luego invertir los dígitos que siguen: 00100000. Como el número ahora es positivo podemos darle peso a cada uno de sus dígitos

$$00100000 = 1 \times 2^5 = +32$$

En resultado que buscábamos en realidad es -32 .

- Otro método consiste en darle peso a todos los dígitos, inclusive al bit que indica el signo. Dijimos que este bit formaba parte del número por lo tanto debe tener un peso, nuestro problema es saber cuánto vale dicho peso. Sin demostración diremos que el peso es el que le corresponde a esa posición pero con signo negativo, es decir $-(2^{n-1})$. El resto de los dígitos tienen los pesos conocidos hasta ahora, es decir positivos.

Ejemplo: resolveremos el ejemplo anterior con este nuevo método.

$$11100000 = -1 \times (2^7) + 1 \times 2^6 + 1 \times 2^5 = -128 + 64 + 32 = -32$$

- El último de los tres métodos consiste en aplicar la definición de complemento a la base

$$Ca2 = b^n - N^\circ$$

Lo que tenemos que hacer es despejar N° pues todo lo demás es conocido. Este método puede prestar a confusión porque el número en $Ca2$ (en esta cuenta), debe ser tratado como BSS.

Ejemplo: $11100000 = 2^7 + 2^6 + 2^5 = 128 + 64 + 32 = 224$

Volvamos a la definición $224 = 2^8 - N^\circ = 256 - N^\circ$ $224 - 256 = N^\circ$

$$N^\circ = -32$$

Hasta ahora vimos tres sistemas distintos para representar números positivos y negativos: BCS, $Ca1$ y $Ca2$. En los tres sistemas los números positivos se representan de la misma manera coincidiendo con el sistema BSS y lo que cambia para cada sistema es la representación de los negativos. Seguidamente vamos a ver un sistema llamado Exceso donde cambian las representaciones tanto de los números positivos como de los negativos.

Representación en Exceso

Para representar un número en exceso primero debemos representar el número en $Ca2$ y luego sumarle una cantidad fija que se llama exceso. Si el número es de 8 bits el exceso es 10000000, si el número es de 4 bits el exceso es 1000 o sea para un número dado de bits el exceso se forma con un 1 en el bit de más a la izquierda seguido de todos ceros. Más adelante vamos a ver otros excesos.

Ejemplo: escribir +32 en exceso con $n = 8$ bits

Primero escribimos +32 en Ca2 00100000

Luego sumamos el exceso 10000000

$$\begin{array}{r} 00100000 \\ + \\ 10000000 \\ \hline 10100000 \end{array} \Longrightarrow +32 \text{ en exceso}$$

Lo primero que notamos es en exceso los números positivos empiezan con 1.

Ejemplo: escribir - 32 en exceso con n = 8 bits

Primero escribimos -32 en Ca2 \Longrightarrow +32 en Ca2 = 00100000

-32 en Ca2 = 11100000

Luego sumamos el exceso 10000000

$$\begin{array}{r} 11100000 \\ + \\ 10000000 \\ \hline 1 \leftarrow 01100000 \end{array} \Longrightarrow -32 \text{ en exceso}$$

Para la representación en exceso tomamos la misma cantidad de bits que el número original, o sea 8 bits y descartamos el carry generado. A partir de este ejemplo notamos que en exceso los números negativos empiezan con 0.

Como hicimos con los sistemas anteriores, veamos ahora el siguiente problema: dada una cadena de bits en exceso ¿cómo determinamos a qué número decimal corresponde? Así como para representar un número en exceso partimos de la representación en Ca2 y le sumamos una cantidad fija, inversamente dada una cadena de bits en exceso debemos restarle el exceso y así obtener la cadena de bits en Ca2. Cuando tratamos Ca2 se explicaron distintas maneras de obtener el número en base 10.

Ejemplo: dada la siguiente cadena de bits en exceso 11100000, calcular el valor decimal.

$$\begin{array}{r} 11100000 \\ - \\ 10000000 \\ \hline 01100000 \end{array} \Longrightarrow \text{El número está en Ca2} = +32$$

Bits de Condición (Banderas)

Son bits que el procesador establece en forma automática acorde al resultado de cada operación realizada. Después de la ejecución de cada instrucción estos bits cambian, debemos aclarar que no

todas las instrucciones afectan a todos los bits de condición. En particular nos ocuparemos de aquellos que cambian luego de realizar una suma ó una resta a los que llamaremos banderas aritméticas. Los valores que tomen estos bits nos permitirán tomar decisiones como:

- determinar la relación entre dos números (mayor, menor, igual)
- realizar ó no una transferencia de control

Más adelante estudiaremos las instrucciones que nos permitirán conocer el valor de estos bits de condición. Antes de seguir adelante vamos a definirlos y ver ante que condición cambian sus valores, mencionando primeramente que el sistema de cómputo sólo hace cuentas en Ca2 ó BSS.

Z (cero): este bit vale 1 si el resultado de la suma ó resta son todos bits 0.

C (carry): en la suma este bit vale 1 si hay acarreo en el bit más significativo y en la resta vale 1 si hay borrow hacia el bit más significativo. Cuando la suma ó resta involucra a números sin signo (BSS), C=1 indica una condición de fuera de rango (desborde), quiere decir que la cantidad de bits disponibles para expresar el resultado no son suficientes.

N (negativo): igual al bit más significativo del resultado. Si el resultado empieza con 1, entonces N=1 y el número es negativo (Ca2).

V (overflow): cuando la suma ó resta involucra a números con signo (Ca2), V=1 indica una condición de fuera de rango (desborde), quiere decir que la cantidad de bits disponibles para expresar el resultado no son suficientes.

Suma en Ca2

Para sumar dos números en Ca2 se suman los n bits directamente.

Si sumamos dos números + y el resultado es – ó si sumamos dos – y el resultado es + hay overflow, en otro caso no lo hay.

Si los N°s son de distinto signo nunca puede haber overflow.

Resta en Ca2

Para restar dos números en Ca2, se restan los n bits directamente. También se puede hacer Ca2 el sustraendo y transformar la resta, en suma.

Si a un N° + le restamos un N° – y el resultado es – ó si a un N° – le restamos un + y el resultado es + hay overflow en la resta.

Si son del mismo signo nunca hay overflow.

Bits de condición para la suma

Operación	NZVC	Ca2	Sin signo
0101	1010	+5	+5
+		+	+
0111		+7	+7
<hr/>		<hr/>	<hr/>
1100		-4 overf.	+12

El resultado de la operación empieza con 1, entonces N=1. Es distinto de cero, entonces Z=0. Sumamos dos números + y el resultado es -, entonces hay overflow V=1. No hay acarreo en el bit más significativo, entonces C=0.

✓ Ca2 incorrecto, sin signo correcto.

Operación	NZVC	Ca2	Sin signo
1101	0101	-3	13
+		+	+
0011		+3	3
<hr/>		<hr/>	<hr/>
1 0000		0	carry 0

✓ Ca2 correcto, sin signo incorrecto.

Operación	NZVC	Ca2	Sin signo
1001	0011	-7	9
+		+	+
1100		-4	12
<hr/>		<hr/>	<hr/>
1 0101		V +5	C 5

✓ Los dos resultados son incorrectos.

Bits de condición para la resta

Operación	NZVC	Ca2	Sin signo
1 0101	1001	+5	5
-		-	-
0111		+7	7
<hr/>		<hr/>	<hr/>
1110		-2	B 14

✓ Ca2 correcto, sin signo incorrecto.

Computación – Lic. en Física Médica y Lic. en Física
Curso 2024 - Prof. Jorge Runco

Operación	NZVC	Ca2	Sin signo
1001	0010	-7	9
-		-	-
0100		+4	4
<hr/>			
0101		V +5	5

✓ Ca2 incorrecto, sin signo correcto.

Números en punto (coma) flotante

Vamos a estudiar ahora la representación de números reales dentro del sistema de cómputo. Vimos como representar números con coma en punto fijo, para comprender la necesidad de introducir el concepto de representación en punto flotante debemos pensar en el orden de magnitud de los números con los cuales vamos a trabajar. Muchas disciplinas en las ciencias requieren el manejo de órdenes tan distintos como 10^{-27} ó 10^{+27} . Si bien podemos representar

estos números en los sistemas de punto fijo, deberíamos tener en cuenta la cantidad de bits necesarios para dicho rango y con que resolución deseamos representar dichos números. Es conocida la forma de expresar constantes físicas como :

Esta notación es conocida como notación científica donde M es la mantisa, E el exponente y 10 es la base numérica. Veamos un ejemplo en base 10:

$$N1= 9,33 \times 10^{-31} \quad N2=9,33 \times 10^{+31}$$

La mantisa está formada por tres dígitos decimales y el exponente por dos dígitos. Si bien ambos números parecen tener la coma en la misma posición, el exponente nos está diciendo que esto no es cierto. Si siempre trabajamos en base 10 no sería necesaria almacenarla y se necesitarían menos dígitos para almacenar mantisa y exponente que todo el número completo. En nuestro caso ocuparía menos espacio en memoria almacenar los números por separado 9,33 y -31 que el número completo $9,33 \times 10^{-31}$.

Utilizando estas ideas para números binarios, podemos representar un número en coma (punto) flotante como:

$$\pm M \times 2^{\pm E}$$

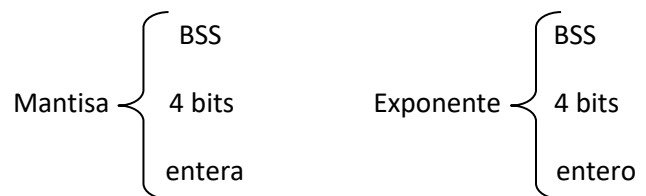
Donde M y E son dos números binarios representados en alguno de los sistemas en punto fijo vistos (BCS, CA1, CA2...). El sistema de cómputo trabaja en base 2 y no es necesaria almacenarla, es la misma para todos los números, por lo tanto, el número se almacenará como una palabra binaria de dos campos: mantisa y exponente.

exponente	mantisa
-----------	---------

La figura muestra un formato típico de punto flotante.

Veamos todas estas ideas con algunos ejemplos.

Ejemplo 1: supongamos el siguiente formato en punto flotante



Determinar rango y resolución.

$$N_2 = \text{Número máximo} = \text{Mantisa máxima} \times 2^{\text{Exponente máximo}} = 1111 \times 2^{1111} = 15 \times 2^{15}$$

Con 4 bits en BSS el número entero más grande es 1111 que es el número 15 en base 10. El mismo razonamiento seguimos con el exponente.

N_1 = Número mínimo = Mantisa mínima $\times 2^{\text{Exponente mínimo}} = 0$

En BSS el número entero más chico es cero.

El rango está representado por el conjunto de números desde el mínimo al máximo.

Rango = $[0, \dots, 15 \times 2^{15}] = [0, \dots, 491520]$

La resolución es la distancia entre dos números consecutivos.



Resolución en el extremo superior será la distancia entre N_2 y N_4 . Donde N_2 es el número máximo y N_4 es el inmediato anterior. Así:

$N_2 = 1111 \times 2^{1111} = 15 \times 2^{15}$ Calculado anteriormente.

Para encontrar el inmediato anterior, cambiamos la mantisa en el bit menos significativo manteniendo el mismo exponente. No cambiamos el exponente pues hacerlo produciría un cambio mayor, entonces:

$N_4 = 1110 \times 2^{1111} = 14 \times 2^{15}$

Finalmente, la resolución en el extremo superior será:

$$R_2 = N_4 - N_2 = (15 - 14) \times 2^{15} = 1 \times 2^{15}$$

Resolución en el extremo inferior será la distancia entre N_3 y N_1 . Donde N_1 es el número más chico y N_3 el inmediato siguiente.

$N_1 = 0$ Calculado anteriormente

Para encontrar el inmediato posterior, como hicimos anteriormente cambiamos la mantisa en el bit menos significativo manteniendo el mismo exponente:

$N_3 = 1 \times 2^0 = 1$

Finalmente, la resolución en el extremo inferior será:

$$R_1 = (1 - 0) \times 2^0 = 1$$

Veamos otro ejemplo que nos ayudará a introducir nuevas ideas.

Ejemplo 2. Consideremos el siguiente formato en punto fijo: 8 bits, enteros, BSS. Calcular rango y resolución.

$$N_2 = \text{Número máximo} = 11111111 = 255$$

$$N_1 = \text{Número mínimo} = 00000000 = 0$$

$$\text{Rango} = [0, 255]$$



Resolución en el extremo superior

$$N_4 = 11111110 = 254$$

$$R_2 = N_2 - N_4 = 255 - 254 = 1$$

Resolución en el extremo inferior

$$N_3 = 00000001 = 1$$

$$R_1 = N_3 - N_1 = 1 - 0 = 1$$

La resolución en todo el rango la resolución es la misma e igual a 1.

Si comparamos los ejemplos 1 y 2 podemos sacar algunas conclusiones

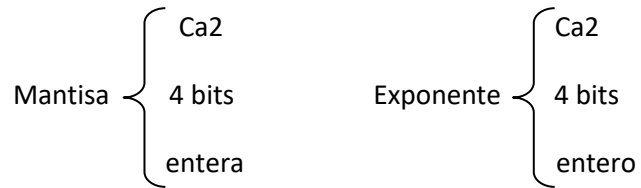
- ✓ el rango en punto flotante es mayor
- ✓ la cantidad de combinaciones binarias distintas es la misma en ambos sistemas $2^8 = 256$
- ✓ en punto flotante la resolución no es constante a lo largo del intervalo, como lo es en el segundo ejemplo.

En el sistema de punto flotante el rango es mayor. Podemos representar números más grandes ó más pequeños que en un sistema de punto fijo (para igual cantidad de bits), pero pagamos el precio que los N° s no están igualmente espaciados, como en punto fijo. Hay que comprender bien esta última idea: en punto flotante el número más grande es mayor que el número más grande en punto fijo (misma cantidad de bits en ambos sistemas), si en los dos sistemas existe la misma cantidad de representaciones ¿cómo es posible esto? Dicho de otra manera, en el sistema de punto fijo la resolución es constante a lo largo de todo el rango, en cambio en el sistema de punto flotante la resolución varía a lo largo del rango; por esto podemos llegar a escribir números más grandes ó más chicos. Como existen la misma cantidad de representaciones en ambos sistemas

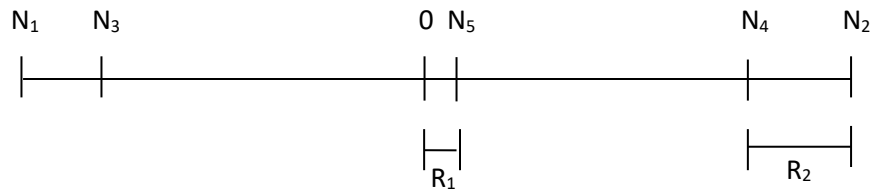
podemos obtener números en punto flotante “más grandes” pagando el precio de estar los mismos más separados.

Vamos a ver un ejemplo donde el sistema representa a números con signo. La mantisa es la que tiene que estar representada por alguno de los sistemas ya vistos (BCS, Ca1, Ca2, Exceso) para que el número en punto flotante tenga signo. El exponente también puede estar en BCS, Ca1, Ca2 ó Exceso, pero esto no hace que el número sea positivo ó negativo.

Ejemplo 3: supongamos el siguiente formato en punto flotante



Determinar rango y resolución.



$$\text{Máximo} = N_2 = 0111 \times 2^{0111} = +7 \times 2^{+7}$$

El número más grande se obtiene con la mantisa mayor positiva y con el exponente mayor también positivo. Con 4 bits en Ca2 el entero más grande positivo es 0111 = 7.

$$\text{Mínimo} = N_1 = 1000 \times 2^{0111} = -8 \times 2^{+7}$$

El número más chico se obtiene con la mantisa menor (negativa), con 4 bits en Ca2 el entero más chico es 1000 = - 8 y con el exponente mayor 0111 = 7.

$$\text{Rango} = [N_1, N_2] = [-8 \times 2^{+7}, \dots, +7 \times 2^{+7}]$$

Resolución en el extremo superior

$$N_2 = 0111 \times 2^{0111} = +7 \times 2^{+7}$$

$$N_4 = 0110 \times 2^{0111} = +6 \times 2^{+7}$$

$$R_2 = N_2 - N_4 = (7 - 6) \times 2^7 = 1 \times 2^7$$

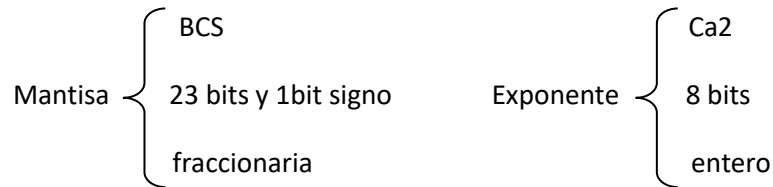
Resolución en el origen

$$N_5 = 0001 \times 2^{1000} = 1 \times 2^{-8}$$

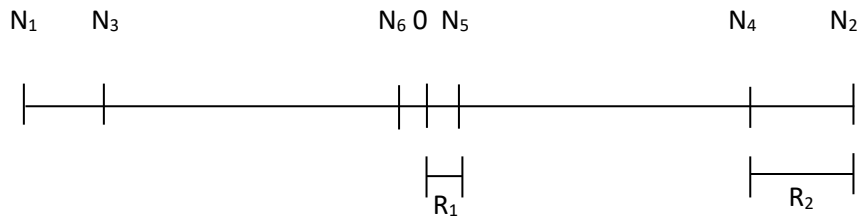
El número positivo más cercano a 0 es N_5 , que se obtiene con la mantisa positiva más chica (1) y con el exponente menor (-8).

$$R_1 = N_5 - 0 = (1 \times 2^{-8} - 0) = 1 \times 2^{-8}$$

Ejemplo 4: supongamos el siguiente formato en punto flotante



Determinar rango y resolución.



$$\text{Máximo positivo} = N_2 = 0,111\dots111 \times 2^{01111111} = +(1-2^{-23}) \cdot 2^{+127}$$

$$\text{Mínimo positivo } (\neq 0) = N_5 = 0,000\dots001 \times 2^{10000000} = +(2^{-23}) \cdot 2^{-128}$$

$$\text{Máximo negativo } (\neq 0) = N_6 = 1,000\dots001 \times 2^{10000000} = -(2^{-23}) \cdot 2^{-128}$$

$$\text{Mínimo negativo} = N_1 = 1,111\dots111 \times 2^{01111111} = -(1-2^{-23}) \cdot 2^{+127}$$

La resolución más grande, entendiendo por aquella donde los números están más separados, es

$$R_2 = N_2 - N_4$$

$N_4 = 0,111\dots110 \times 2^{01111111}$ Como antes cambiamos el bit menos significativo de la mantisa para obtener el más cercano a N_2 , entonces:

$$R_2 = (0,1111\dots111 - 0,1111\dots110) \times 2^{01111111} = 0,000\dots001 \times 2^{+127} = 2^{-23} \times 2^{+127}$$

La resolución más pequeña, entendiendo por aquella donde los números están más cercanos, es

$$R_1 = N_5 - 0$$

$N_5 = 0,000\dots001 \times 2^{10000000} = 2^{-23} \times 2^{-128}$ Como antes cambiamos el bit menos significativo de la mantisa para obtener el más cercano a N_2 , entonces:

$$R_2 = 0,000\dots001 \times 2^{10000000} - 0 = 0,000\dots001 \times 2^{-128} = 2^{-23} \times 2^{-128}$$

Normalización

Veamos el siguiente ejemplo en base 10:

$$40 \times 10^0 = 4 \times 10^1 = 0,4 \times 10^2 = 400 \times 10^{-1}$$

Existen distintos valores de mantisa y exponente para representar un mismo número. Esto también sucede en base 2. Con el objetivo de tener un único par de valores de mantisa y exponente para un número, se introduce la normalización. Con el objetivo anterior, las mantisas fraccionarias se definen de la forma:

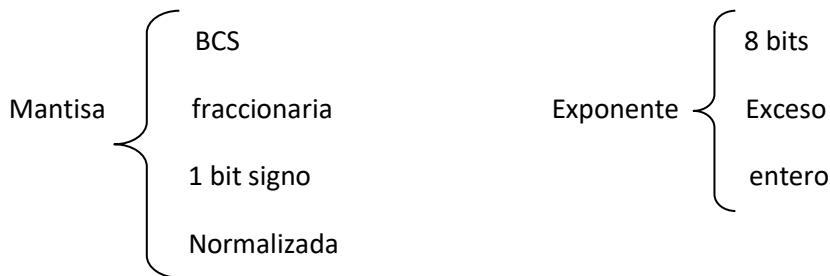
$$0,1\ldots\ldots\ldots$$

es decir, todas las mantisas empiezan con 0,1 y luego continúan con los dígitos binarios que correspondan (0 ó 1). Esta normalización es para mantisas expresadas en BCS ó BSS, prestemos atención que la misma empieza siempre con 0, sólo es posible en números donde el signo está aparte del módulo del número (BCS) ó donde no hay signo (BSS).

Si la mantisa estuviera escrita en Ca2 por ejemplo, la normalización sería otra y no ésta.

Veamos el siguiente ejemplo para aclarar estas ideas.

Formato en punto flotante



Determinar el rango y resolución

Máximo positivo

$$0 \ 0,111\ldots111 \times 2^{11111111} = +(1 - 2^{-23}) \cdot 2^{+127}$$

El primer 0 indica que el número es +, la mantisa más grande fraccionaria es $0,111\ldots11 (1 - 2^{-23})$ y el exponente más grande en Ca2 es 01111111 (+127), luego de sumar el exceso 10000000, queda igual a 11111111 que es +127 en exceso

Mínimo positivo ($\neq 0$)

$$0 \ 0,100\ldots000 \times 2^{00000000} = +(0,5) \cdot 2^{-128}$$

La mantisa normalizada más pequeña es 0,100...000 (0,5) y el exponente más pequeño es 2^{-128} , luego de sumar el exceso 10000000, queda igual a 00000000 que es -128 en exceso.

Máximo negativo ($\neq 0$)

$$1 \ 0,100...000 \times 2^{00000000} = -(0,5) \cdot 2^{-128}$$

Este número es simétrico al anterior con la diferencia que empieza con 1 (-) indicando que es negativo.

Mínimo negativo

$$1 \ 0,111...111 \times 2^{11111111} = -(1-2^{-23}) \cdot 2^{+127}$$

Este número es simétrico respecto del máximo positivo pero empieza con 1 (-) indicando que es negativo.

Finalmente un formato posible para almacenar la mantisa y el exponente en memoria, sería

0 1	8 9	31
S	Exponente	Mantisa

El máximo negativo distinto ($\neq 0$)

1	00000000	1000.....00
---	----------	-------------

Dibujemos ahora la recta numérica:



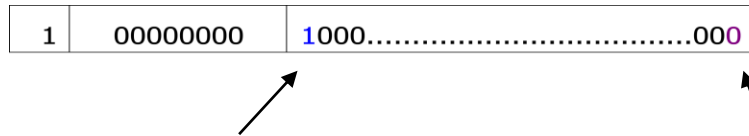
Igual que en otros sistemas, dada una cantidad fija de bits hay un número máximo que es posible representar, pero debido a la normalización aquí hay una zona alrededor del cero de la recta numérica que no es posible representar, pues la mantisa nunca se hace cero.

En nuestro ejemplo entre $-0,5 \cdot 2^{-128}$ y $+0,5 \cdot 2^{-128}$ no es posible representar ningún número, incluido el 0.

Bit Implícito

Como todas las mantisas comienzan con 0,... vimos que no era necesario almacenar el 0, a la hora de guardar en memoria la mantisa y el exponente. Al normalizar el número, todas las mantisas empiezan con 0,1. Como este 1 siempre está presente podríamos pensar de la siguiente manera: si no lo almacenamos podemos “adicionar” un bit más a la mantisa. El bit no almacenado se conoce como bit implícito.

El formato anterior quedaría:



Dibujemos ahora la recta numérica para el sistema anterior pero con bit implícito



Por el uso de bit implícito hemos ganado un bit más en la mantisa y por lo tanto mayor precisión al momento de representar un determinado número.

¿Cómo se escribe un número en punto flotante normalizado?

Una manera sencilla de hacerlo es siguiendo estos tres pasos:

1. Se escribe el N° en el sistema propuesto para la mantisa.
2. Se desplaza la coma y se cambia el exponente hasta obtener la forma normalizada.
3. Se convierte el exponente al sistema propuesto para él.

Ejemplo: escribir $N = -13,5$ en el formato normalizado anterior (sin bit implícito)

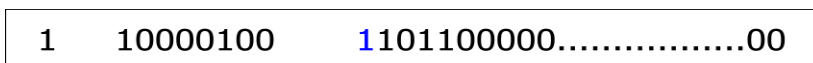
1) $N = 1 \ 1101,100..0 = 1 \ 1101,100..0 \times 2^0$

2) $N = 1 \ 0,110110..0 \times 2^4$

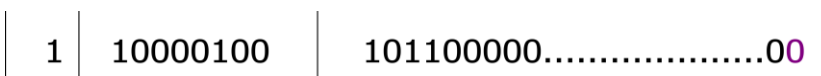
3) Exponente 4 en Ca2=00000100 \implies Exponente 4 en Exceso=10000100

Finalmente

Sin bit implícito



Con bit implícito



Error Absoluto

Recordemos que la resolución es la diferencia entre dos representaciones sucesivas y que en punto flotante la resolución varía a lo largo del intervalo (rango) y no es constante como en el caso de punto fijo.

Dado un sistema numérico no siempre es posible representar en forma exacta un número dado. Pensemos un ejemplo sencillo en punto fijo: si los números son enteros no podremos representar 1,2 en forma exacta, será aproximado por 1. Otro ejemplo 1,7 será aproximado por 2. En ambos casos cometimos error en la representación del número porque por la definición del sistema no permite números con coma. El error cometido en ambos casos es:

$$\text{Error} = 1,2 - 1 = 0,2$$

$$\text{Error} = 2 - 1,7 = 0,3$$

En punto flotante también sucede lo mismo, por más que la mantisa tenga coma no significa que puedo representar en forma exacta cualquier número. Definimos el error absoluto como la diferencia entre el valor representado (el que se pudo) y el valor a representar.

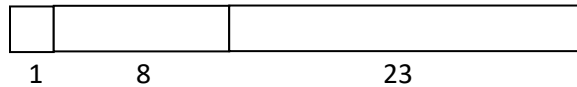
IEEE 754

El estándar IEEE 754 fue definido por el Instituto de Ingenieros Eléctricos y Electrónicos (Institute of Electrical and Electronics Engineers, IEEE) y establece dos formatos básicos para representar a los números reales dentro del sistema de cómputo. Estos dos formatos son conocidos como simple precisión y doble precisión.

IEEE 754 simple precisión

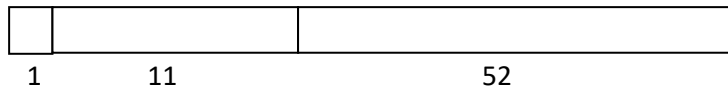
En este formato de simple precisión para escribir un número real se usan 32 bits (4 bytes): 1 bit para el signo (**S**) del número, 23 bits para la mantisa (**M**) y 8 bits para el exponente (**E**). La mantisa está expresada en BCS (MyS), fraccionaria con la coma después del primer bit que siempre es 1 pues está normalizada y el exponente en exceso.

La normalización de la mantisa es distinta a la vista anteriormente 0,1...; ahora es 1,... Y el exceso sumado al exponente es 01111111 también distinto al visto anteriormente 10000000 para 8 bits, también llamado exceso $2^{n-1} - 1$. Finalmente, el formato será:



IEEE 754 doble precisión

En este formato de doble precisión para escribir un número real se usan 64 bits (8 bytes): al igual que en simple precisión la mantisa está representada en BCS, normalizada pero en 52 bits y el exponente en exceso en 11 bits (exceso = 0111111111). Finalmente, el formato será:



Casos especiales

En este sistema, como en cualquier otro normalizado, la mantisa nunca es cero. Por lo tanto, el cero no puede ser representado por este formato normalizado, requiere una representación especial. A continuación, veremos otros casos especiales además del cero.

En el formato IEEE 754 hay dos exponentes que son usados para representar casos especiales: todos los dígitos del exponente son unos ó ceros. Quiere decir que si al analizar un número escrito en este formato encontramos que el exponente son todos unos ó todos ceros, el número corresponde a un caso especial tanto para simple como para doble precisión.

En simple precisión un exponente con todos sus bits en cero (00000000) visto como número sin signo es el cero. Recordemos que el exponente en realidad está representado en exceso, por lo tanto, visto como un número con signo en exceso debemos restar el exceso para saber a qué número representa, entonces $00000000 - 01111111 = 10000001$. Recordemos que al restar el exceso el número queda expresado en Ca2, por lo tanto $10000001 = -127$.

Computación – Lic. en Física Médica y Lic. en Física
Curso 2024 - Prof. Jorge Runco

En simple precisión un exponente con todos sus bits en uno (11111111) visto como número sin signo es el 255. Visto como un número con signo en exceso debemos restar el exceso para saber a qué número representa, entonces $11111111 - 01111111 = 10000000$. Recordemos que al restar el exceso el número queda expresado en Ca_2 , por lo tanto $10000000 = -128$.

Resumiendo, podemos decir que los exponentes usados para casos especiales son 0 y 255, vistos como números sin signo; ó podemos decir que estos exponentes son -128 y -127, vistos como números con signo en exceso. En cualquiera de los dos casos, BSS ó Exceso, son los exponentes 00000000 y 11111111.