Números Negativos. Codificación

Sistemas de Procesamiento de Datos
Tecnicatura Superior en Programación.
UTN-FRA

Autores: *Ing. Darío Cuda*

Revisores: *Lic. Mauricio Dávila*

Versión : 1

Esta obra está bajo una Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional.

Codificación.

Es sabido que dentro de un procesador, microprocesador, memoria o cualquier otro medio de almacenamiento de datos, solo se utiliza el sistema binario.

Esta situación es particularmente útil cuando queremos realizar un dispositivo que solo opere con números, (por ejemplo una calculadora) pero resulta evidente que en una memoria, o procesador en la que solo pueden procesarse y almacenarse o leerse binarios, debemos encontrar alguna forma de representar números negativos. (No existen celdas de memoria o espacios en el micro en los que "poner" un signo "-"

Exsisten entonces varios códigos que pueden utiliarse a tal fin de los cuales describiremos tres.

1. Signo magnitud o bit de signo.

Se utiliza un bit para representar el signo. Este bit, normalmente es el bit más significativo o MSB (de sus siglas en inglés, Most Significant Bit) y, por convención: un 0 denota un número positivo, y un 1 denota un número negativo; de esta manera, los (n-1)-bits restantes se utilizan para representar el **significando** (o sea la magnitud del número en valor absoluto)
Esta idea, que se conoce como "Signo y Magnitud" es directamente comparable a la forma habitual de mostrar el signo colocando "+" o "-" al lado de la magnitud del número, pero colocando un cero si es positivo o un uno si es negativo.

Algunas de las primeras computadoras binarias (la IBM 7090) utilizaron esta representación, quizás por su relación obvia con la práctica habitual.

El formato Signo y Magnitud es además el habitual para la representación del **significando** en números en punto flotante (del que hablaremos mas adelante)

Ejemplos del sistema de "bit de signo", son los siguientes:

Binario de 8 Bits	Natural (N) Sin Signo	Entero (Z) Con signo
00000001	1	1
10000001	129	-1
111111111	255	-127
011111111	127	127
10000000	128	0
00000000	0	0

El primer problema al que nos enfrentamos en la tabla anterior, queda demostrado en la ambigüedad del cero.
En la representación "Signo magnitud" de números negativos, el Cero posee dos representaciones (como si existiesen un cero negativo y un cero positivo).

La segunda, menos evidente hasta ahora, es que el código pierde la virtud de "operar" o de ser operable, es decir de "hacer operaciones aritméticas" con ellos directamente.

Veamos un ejemplo:

En decimal: 120 - 100 = 20
Expresado en "signo magnitud", deberíamos poder hacer 120 + (-100):

01111000 + 11101000 = 1 01100000, Sin considerar el bit de acarreo el resultado es +96 y considerándolo es -96, en ambos casos incorrecto.

El ejemplo anterior muestra además la necesidad de saber de antemano, cuántos bits debo contar para establecer mi número, aunque esta característica es común a todos los métodos de codificación numérica en computadoras.

NOTA: Algunos sistemas de computadoras antiguos utilizaban esta codificación y resolvían el problema anterior de la siguiente manera:

Para realizar una suma, primero hay que determinar si los dos números tienen el mismo signo, y en caso de que sea así, realizar la suma de la parte significativa, pero en caso contrario, restar el mayor del menor y asignar el signo del mayor.

2. Complemento a la (base -1) o complemento a 1:

Se trata de representar números negativos usando el complemento a la base menos uno. En el caso de los binarios, cuya base es 2, se trata de realizar el complemento a uno y su forma, es hacer un "NOT" (invertir bit a bit) al número, es decir, la inversión de unos por ceros y ceros por unos. De esta forma tendremos:

Binario de 8 Bits	Natural (N) Sin Signo	Entero (Z) Con signo
00000001	1	1
10000001	129	-126
111111111	255	0
011111111	127	127
10000000	128	-127
00000000	0	0

Esta representación tiene la particularidad de; al igual que la forma signo y magnitud" de ser simétrica, es decir de poder representar en una cantidad determinada de bits, la misma cantidad de números negativos que positivos (en el caso de 8 bits, desde el -127 al +127) por ejemplo, pero tiene el inconveniente de la doble representación del cero al igual que en el caso anterior.

Esto significa que su operabilidad es restringida, es decir que exige una algoritmia adicional al momento de operar, que permita determinar si el número a tratar es positivo, negativo o cer, para luego transformarlo y realizar la mencionada operación.

2. Complemento a la base (En binarios, complemento a 2)

Es tal vez la mas moderna de las representaciones de números con signo, (de hecho es la que utilizan los microprocesadores y microcontroladores en la actualidad).

Sin entrar en una definición académica rigurosa, diremos que el complemento a dos de un número cualquiera se obtiene de sumarle "1" al número convertido en complemento a 1.

En otras palabras, por ejemplo el número -100 decimal se convierte a complemento a 2 haciendo:

1º) Tomar el valor absoluto del número (en este caso 100) y convertirlo a Ca1 (complemento a 1)

100d = 01100100b → (Ca1) = 10011011

2º) Al número complementado, se le suma una unidad (en el sistema de numeración que corresponda) en nuestro aso binario:

10011011b + 1b = 10011100b

Con lo que el número -100d se representa en Ca2 (Complemento a 2) = 10011100b

Veamos que pasa con el cero en ocho bits (1 byte):

0d = 00000000b → Ca1 = 11111111b

Si le sumamos 1b = 11111111b + 1b = 1 00000000b (Descartando el noveno bit -acarreo- obtenemos el mismo número que al principio.

Esta última es quizá, la mayor fortaleza del Ca2: el cero posee una representación única al igual que todos los números representables.

Ahora, esto además lleva a la conclusión de que el sistema en Ca2 es asimétrico (de los 128 números con el MSB en 0, se usan 127 para representar los positivos y 1 para el cero, mientras que los que poseen el MSB en 1, van del -1 al -128).

Binario de 8 Bits	Natural (N) Sin Signo	Entero (Z) Con signo
00000000	0	0
00000001	1	1
00000010	2	2
---	---	---
01111111	127	127
10000000	128	-128
10000001	129	-127
10000010	130	-126
---	---	---
11111110	254	-2
11111111	255	-1

Para finalizar, veamos que sucede cuando queremos operar utilizando Ca2:

120d - 100d = 20d

01111000 + 10011100 = 1 00010100b (Descartando el noveno bit -acarreo- obtenemos el número esperado, 20-)