UNIVERSIDAD DISTRITAL
FRANCISCO JOSE DE CALDAS

Aritmética de Enteros y Flotantes

2013
Transversal de Programación Básica
Proyecto Curricular de Ingeniería de Sistemas

1. Introduccion

La aritmética de enteros es aritmética modular en complemento a dos, es decir, si un valor excede el rango de su tipo es reducido a su módulo. Por tanto, la aritmética de enteros nunca produce desbordamientos ni subdesbordamientos. La división entre enteros trunca el resto con la siguiente regla: (x/y)*y + x%y == x. La división por cero o el resto por cero no son válidos.

La aritmética de caracteres es aritmética de enteros ya que un char es convertido implícitamente a int.

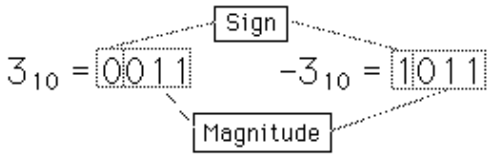
Java usa el estándar IEEE 754-1985 tanto para la representación como para la aritmética de números en punto flotante. De acuerdo con él, podemos producir desbordamientos hacia el infinito o subdesbordamientos a cero. También se puede representar los resultados de expresiones no válidas, NaN, como la división por cero.

2. Representación de Números Negativos

Existen diversos sistemas para representar números negativos:

a. **SIGNO MAGNITUD.**

En este sistema el bit más significativo es el bit de signo (0 es +, 1 es –) y el resto de los bits corresponden a la magnitud.



Por ejemplo:

510 = 0000 0101
-510 = 1000 0101

La suma en Signo Magnitud se efectúa sobre la magnitud de los sumandos que deben ser de igual signo. El signo del resultado es el signo de los dos sumandos.

0 0101 (510) 1 1010 (-1010)
0 0011 (310) 1 0011 (-310)

_____ _____
0 1000 (810) 1 1101 (-1310)

¿Cuándo ocurre un desbordamiento u *overflow*?
Cuando hay un acarreo del bit más significativo de la magnitud.

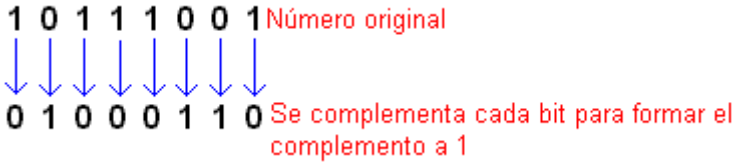
0 1011 (1110)
0 1000 (810)

_____
0 10011 desborde

Esta representación presenta varias limitaciones. Una de ellas es que la suma y la resta requieren tener en cuenta tanto los signos de los números como sus magnitudes relativas para llevar a cabo la operación en cuestión. Otra limitación es que hay dos representaciones del número cero.

b. **COMPLEMENTO A UNO**

Los números positivos se representan igual que en Signo Magnitud. Para obtener un número negativo se toma el positivo correspondiente y se niega cada uno de sus bits inclusive el bit de signo.



Por ejemplo:

510 = 0000 0101
-510 = 1111 1010

La operación para obtener el complemento a uno de un número es equivalente a restar el valor absoluto del número de 2^n-1.

Para sumar dos números en complemento a uno simplemente consiste en sumar todos los dígitos de los números y cuando hay un acarreo desde el bit más significativo entonces se debe sumar uno (1) al resultado.

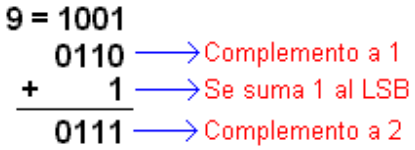
0 0111 (710) 1 1110 (-110)
0 0101 (510) 0 0010 (210)

_____ _____
0 1100 (1210) 10 0000 (010) error
1

_____
0 0001 (110)

c. COMPLEMENTO A DOS.

Los números positivos se representan igual que en Signo-Magnitud. Para obtener un número negativo, se substrae su valor absoluto de 2^n. Una forma sencilla consiste en tomar el positivo correspondiente y cada 0 se sustituye por 1 y cada 1 por 0, y posteriormente se le suma 1 al resultado. En esta suma si hay un acarreo final se desecha el bit.



Por ejemplo:

310 = 0011
-310 = 1101

Esta representación utiliza el bit más significativo (izquierda) como bit de signo, facilitando la comprobación de si el entero es positivo o negativo. Lo que difiere con la representación de Signo-Magnitud y Complemento a Uno es la forma de interpretar el resto de los bits.

Consideremos un entero de *n* bits, *A* representado en Complemento a Dos. Si *A* es positivo, el bit de signo, an-1 es cero. Los restantes bits representan la magnitud del número de la misma forma que en la representación Signo-Magnitud.

$$A = \sum_{i=0}^{n-2} 2^i a_i \qquad para\ A > 0$$

El número cero se representa como positivo y por lo tanto tiene un bit de signo 0 y una magnitud de 0. Para un número negativo *A*, el bit de signo, an-1 es 1. Los n-1 bits restantes pueden tomar cualquiera de las 2^{n-1} combinaciones.

Por lo tanto, el rango de números enteros negativos que pueden ser representados van desde -1 hasta -2^{n-1}. Una asignación conveniente de valores es hacer que los bits an-1 an-2 ... a2 a1 a0 sean iguales al número positivo 2^{n-1} + A.

$$2^{n-1} + A = \sum_{i=0}^{n-2} 2^i a_i \qquad A = -2^{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

El cuarto sistema que para números de *m* bits se conoce como exceso de 2^{m-1}, representa un número almacenándolo como la suma del mismo número y 2^{m-1}.

	Signo Magnitud	Complemento a uno	Complemento a dos	Exceso de 8
-8	No existe	No existe	1000	0000
-7	1111	1000	1001	0001
-6	1110	1001	1010	0010
-5	1101	1010	1011	0011
-4	1100	1011	1100	0100
-3	1011	1100	1101	0101
-2	1010	1101	1110	0110
-1	1001	1110	1111	0111
0	1000 y 0000	0000 y 1111	0000	1000
1	0001	0001	0001	1001
2	0010	0010	0010	1010
3	0011	0011	0011	1011
4	0100	0100	0100	1100
5	0101	0101	0101	1101
6	0110	0110	0110	1110
7	0111	0111	0111	1111

Observando la tabla anterior, los sistemas tanto de signo magnitud como el de complemento a uno tienen dos representaciones para el cero. Esta situación no es deseable.

El complemento a dos y el exceso no tienen este problema. Sin embargo, presentan un número desigual de números positivos y negativos.

De todos estos sistemas de representación de números negativos, el complemento a dos es el mejor método en términos de eficiencia en la implementación de las operaciones de suma y resta.

3. Aritmética Binaria

La tabla de suma para números binarios es:

Sumando 0 0 1 1
Sumando +0 +1 +0 +1

_____

Suma 0 1 1 0
Acarreo 0 0 0 1

Dos números binarios pueden sumarse comenzando con el bit menos significativo (extrema derecha) y sumando los bits correspondientes de los dos sumandos. Si se genera un acarreo, se lleva una posición a la izquierda igual que en la aritmética decimal. En aritmética de complemento a uno, un acarreo generado por la suma de los bits más significativos (izquierda) se suma al bit de la extrema derecha. En aritmética de complemento a dos, un acarreo generado por la suma de los bits de la extrema izquierda simplemente se desecha.

Las reglas que gobiernan la suma y la resta de números de n bits usando el sistema de representación en complemento a dos son:

1. Para sumar dos números, sume sus representaciones. El resultado será correcto siempre y cuando el resultado esté dentro del rango -2_{n-1} y $2_{n-1}-1$

2. Para restar dos números X y Y, obtenga la representación de –Y en complemento a dos y sume X + (-Y)

Si los sumandos tienen signos opuestos, no puede haber un error de desborde. Si tienen el mismo signo y el resultado es de signo opuesto entonces ha ocurrido un error de desborde y la respuesta es incorrecta.

Ejemplos:

```
(-7)    1001          (-4)    1100
(+5)    0101          (+4)    0100
-----------           ---------------
(-2)    1110                1 0000

(+3)    0011          (-4)    1100
(+4)    0100          (-1)    1111
-----------           ---------------
        0111                1 1011

(+5)    0101          (-7)    1001
(+4)    0100          (-6)    1010
-----------           ---------------
Desborde 1001         Desborde 1 0011
```

Ejemplo: Dado el número 10012 en Complemento a Dos usando 4 bits, qué número representa en decimal

$10012 = -2^3 + 1*2^0 = -8 + 1 = -7$

Ejemplo: Cómo se representa -2 usando Complemento a 2 con 32 bits?

```
2     =     0000 0000 0000 0000 0000 0000 0000 0010
            1111 1111 1111 1111 1111 1111 1111 1101 +
            0000 0000 0000 0000 0000 0000 0000 0001
            ____________________________________________
-2    =     1111 1111 1111 1111 1111 1111 1111 1110
```

Ejemplo: Cómo se representa -17 usando Complemento a 2 con 32 bits?

```
17    =     0000 0000 0000 0000 0000 0000 0001 0001
            1111 1111 1111 1111 1111 1111 1110 1110 +
            0000 0000 0000 0000 0000 0000 0000 0001
            ____________________________________________
-17   =     1111 1111 1111 1111 1111 1111 1110 1111
```

4. Punto Flotante

Muchas aplicaciones requieren trabajar con números que no son enteros. Existen varias formas de representar números no enteros. Una de ellas es usando un punto o coma fijo. Este tipo de representación ubica siempre el punto o coma en alguna posición a la derecha del dígito menos significativo.

Otra alternativa comúnmente usada es la que se conoce como representación en punto flotante. Bajo este esquema, un número puede ser expresado mediante un exponente y una mantisa. Por ejemplo el número 10.75 puede ser expresado como

10.75 x 10^{0}
1.075 x 10^{1}
Mantisa exponente

En general, un número en punto flotante puede ser representado como $\pm d0.d1d2d3...dk \times b^{exp}$ donde *d0.d1d2d3...dk* se conoce como la mantisa, *b* es la base y *exp* es el exponente.

¿Qué se necesita para representar un número en punto flotante?
- el signo del número.
- el signo del exponente.
- Dígitos para el exponente.
- Dígitos para la mantisa.

Dado que un número en punto flotante puede expresarse de distintas formas que son equivalentes, es necesario establecer una única representación. Es por ello que se trabaja con números *normalizados*. Decimos que un número está normalizado si el

dígito a la izquierda del punto o coma está entre 0 y la base (0< dígito a la izquierda del punto < *b*).

En particular, decimos que un número binario está normalizado si el dígito a la izquierda del punto es igual a 1.

1.00×10^{-1} normalizado
0.01×10^{2} no normalizado

4.1 ESTÁNDAR IEEE-754 PARA REPRESENTACIÓN DE PUNTO FLOTANTE

Este estándar se desarrolló para facilitar la portabilidad de los programas de un procesador a otro y para alentar el desarrollo de programas numéricos sofisticados. Este estándar ha sido ampliamente adoptado y se utiliza prácticamente en todos los procesadores y coprocesadores aritméticos actuales. El estándar del IEEE define el formato para precisión simple de 32 bits y para precisión doble de 64 bits.

a. Precisión Simple

El formato para los números de *precisión simple* es de 32 bits.

signo	exponente con signo	Mantisa
1	8	23

La representación de un número en precisión simple en el formato IEEE-754 consta de las siguientes partes:

• *Signo* se encuentra en el bit más significativo, de esta manera podemos usar la misma circuitería (de enteros) para llevar a cabo comparaciones con respecto al cero.

• *Exponente con signo*. Está conformado por los siguientes 8 bits. Esta ubicación del exponente en la palabra facilita las comparaciones de números. Si los números se encuentran normalizados, comparamos los exponentes. Si son iguales pasamos a comparar las mantisas. Pero, ¿ qué representación es más conveniente usar para el exponente?. Si utilizamos Complemento a Dos, los exponentes negativos aparecerán como mayores que los exponentes positivos al usar la circuitería de enteros.

C2(-1) = 1111 1111
C2(0) = 0000 0000
C2(1) = 0000 0001

Para evitar este inconveniente, se utiliza una representación en exceso *N* de forma que el exponente más negativo posible quede en 0000 0001 y el más grande de los positivos en 1111 1110. El estándar IEEE 754 usa como exceso 127 para precisión simple.

Exponente más negativo representable:
x + 127 = 0000 0001
x = -126

Exponente más grande representable
x + 127 = 1111 1110
x = 127

• *Mantisa.* Está formada por el resto de los bits en la palabra (23). Como los números se representan de manera normalizada entonces siempre tendremos un 1 a la izquierda del punto. Por lo tanto este dígito no es necesario almacenarlo en la palabra y se tiene de manera implícita. La mantisa consiste en 24 bits de precisión.

Ejercicio

Representar según el estándar IEEE de punto flotante los siguientes valores

- 7

  - Convertimos el número a binario.

    $7_{10} = 111_2$

  - Normalizamos el número.

    $1.11_2 \times 10_2^2$

  - Calculamos el exponente con exceso 127 para precisión simple.

    $2 + 127 = 129_{10} = 1000\ 0001_2$

  - El número 7_{10} en el estándar IEEE es representado como:

0	10000001	11000000000000000000000
signo	exponente en exceso	mantisa

- 21

$21_{10} = 10101_2 = 1.0101_2 \times 10_2^4$

exponente $4 + 127 = 131_{10} = 1000\ 0011_2$

0	10000011	01010000000000000000000

b. Precisión Doble

La representación de un número en precisión doble en el formato IEEE-754 consta de las siguientes partes:

• *Signo* se encuentra en el bit más significativo

• *Exponente en exceso*. Está conformado por los siguientes 11 bits. Se utiliza una representación en exceso 1023 de forma que el exponente más negativo posible quede en 000 0000 0001 y el más grande de los positivos en 111 1111 1110.

• *Mantisa.* Está formado por 52 bits más el bit implícito (53).

signo	exponente en exceso	Mantisa
1 bit	11 bits	52 bits

Casos Especiales

Para valores de exponente desde 1 hasta 254 en el formato simple y desde 1 a hasta 2046 en el formato doble, se representan números en punto fijo normalizados. El exponente está en exceso, siendo el rango del exponente de -126 a +127 en el formato simple y de -1022 a +1023 en el doble.

Un número normalizado debe contener un bit 1 a la izquierda del punto binario; este bit está implícito, dando una mantisa efectiva de 24 bits para precisión simple o 53 bits para precisión doble.

Un exponente cero junto con una parte fraccionaria cero representa el cero positivo o negativo, dependiendo del bit de signo. Es útil tener una representación del valor 0 exacto.

Precisión Simple

Exponente en exceso	Mantisa	Valor
0	0	Cero
0	<>0	Número no normalizado (0. + Mantisa) x 2^{-126}
1 .. 254		(1. + Mantisa) x $2^{exp-127}$
255	0	Infinito
255	<>0	Not a Number

Precisión Doble

Exponente en exceso	Mantisa	Valor
0	0	Cero
0	<>0	Número no normalizado (0. + Mantisa) x 2^{-1022}
1 .. 2046		(1. + Mantisa) x $2^{exp-1023}$
2047	0	Infinito
2047	<>0	Not a Number

Conversión de un número en Punto Flotante Decimal a Binario

Un número *Numb= d0.d1d2d3*... en base *b* representa

$$Num_{10} = d_0 + d_1 * b^{-1} + d_2 * b^{-2} + d_3 * b^{-3} + \ldots + d_n * b^{-n}$$

podemos reescribirlo de la siguiente forma:

$$Num_{10} = d_0 + b^{-1}\left(d_1 + d_2 * b^{-1} + d_3 * b^{-2} + \ldots + d_n * b^{-n-1}\right)$$

$$Num_{10} = d_0 + b^{-1}\left(d_1 + b^{-1}\left(d_2 + d_3 * b^{-1} + \ldots + d_n * b^{-n-2}\right)\right)$$

$$Num_{10} = d_0 + b^{-1}\left(d_1 + b^{-1}\left(d_2 + d_3 * b^{-1}\left(d_4 + \ldots + b^{-1}\left(d_{n-1} + d_n * b^{-1}\right)\right)\right)\right)$$

De la última expresión podemos deducir el algoritmo de conversión de punto flotante decimal a cualquier base

Dado un número Num_{10} en punto flotante decimal y una base *b*

d_0= parte entera(Num_{10})
$Num_{10} = (Num_{10} - d_0) * b$
i=1
Repetir desde i=1 hasta N
  d_i=parte entera (Num_{10})
  Num_{10}=($Num_{10} - d_i$) * *b*

$$Num_{10} = d_0.d_1d_2d_3d_4...d_{N\,b}$$

Ejemplos

a.) Convertir 0.510 a binario y hallar su representación en IEEE precisión simple

0.50
(0.50-0) * 2 = 1 d0=0
(1.00-1) * 2 = 0 d1=1
$0.5010 = 0.12 = 1.0 \times 2^{-1}$

Exponente en exceso= -1 + 127 = $126_{10} = 0111\ 1110_2$
0 01111110 00000000000000000000000

0	01111110	00000000000000000000000
signo	exponente en exceso	mantisa

b.) Convertir 3.7510 a binario y hallar su representación en IEEE precisión simple

3.75
(3.75-3) * 2 = 1.50 d0=3
(1.50-1) * 2 = 1.00 d1=1
(1.00-1) * 2 = 0.00 d2=1
$3.7510 = 11.112 = 1.111 \times 2^{1}$

Exponente en exceso= 1 + 127 = $128_{10} = 1000\ 0000_2$

0	1000 0000	11100000000000000000000
signo	exponente en exceso	mantisa

c.) Convertir 0.310 a binario y hallar su representación en IEEE precisión simple
0.3

(0.3-0) * 2 = 0.6 d0=0
(0.6-0) * 2 = 1.2 d1=0
(1.2-1) * 2 = 0.4 d2=1
(0.4-0) * 2 = 0.8 d3=0
(0.8-0) * 2 = 1.6 d4=0
(1.6-1) * 2 = 1.2 d5=1
0.310 = 0.01001001001...2 = 1.001001001... x 2^{-2}

Exponente en exceso= -2 + 127 = 125_{10} = 0111 1101 $_2$

0	0111 1101	00100100100100100100100
signo	exponente en exceso	Mantisa

Esta representación es una aproximación. No puede ser escrito en forma precisa. Los números punto flotante son normalmente aproximaciones. La razón de esto es que existe un número infinito de números reales entre dos números dados.

Lecturas de Profundización:

- http://www.infor.uva.es/~fernando/asignaturas/estruct/leccion7.pdf
- IEEE Coma Flotante http://es.wikipedia.org/wiki/IEEE_coma_flotante

Imágenes:
Las imágenes fueron tomadas de www.google.com

Referentes:

- http://ldc.usb.ve/~adiserio/ci3815/clases/AritmeticaEnteros.pdf
- http://mate.dm.uba.ar/~pdenapo/apuntes-algebraI/enteros.pdf
- http://mmc2.geofisica.unam.mx/cursos/mcst-2007-II/arch/SisNum.pdf

- http://www.ac.usc.es/pag_personal/fran/Tecnologia/Tema2b.pdf
- http://people.hpc.cl/~parce/cc1/clase18-RP.html
- http://hyperphysics.phy-astr.gsu.edu/hbasees/electronic/number2.html
- http://www.iafe.uba.ar/e2e/metodosnumericos/algoritmos/RepresentacionNumerica.html
- http://grouper.ieee.org/groups/754/