Aritmética de Enteros

La aritmética de los computadores difiere de la aritmética usada por nosotros. La diferencia más importante es que los computadores realizan operaciones con números cuya precisión es finita y fija. Otra diferencia es que casi todos los computadores utilizan el sistema binario en lugar del decimal.

En casi todos los computadores, la cantidad de memoria con que se cuenta para almacenar un número se fija en el momento en que se diseña la máquina. La naturaleza finita del computador nos obliga a manejar sólo números que se puedan representar con un número fijo de dígitos. Por ejemplo, supongamos que estamos trabajando con el sistema decimal y fijamos como tamaño tres dígitos decimales. Este conjunto tiene exactamente 1000 valores distintos. Con la restricción de tamaño impuesta antes no podemos representar:

- Números mayores que 999
- Números negativos
- Fracciones
- Números irracionales
- Números complejos

Una propiedad importante de la aritmética del conjunto de los enteros es que es cerrado con respecto a las operaciones de suma, resta y multiplicación. El conjunto no es cerrado con respecto a la división pues existen valores que no pueden expresarse como enteros.

El conjunto de los números enteros finitos no es cerrado con respecto a ninguna de las cuatro operaciones:

600 + 700 = 1300	(demasiado grande)
003 – 010 = -007	(demasiado pequeño)
007 / 002 = 3.5	(no es entero)

Los problemas anteriores pueden clasificarse como error de desbordamiento, error de subdesbordamiento y no pertenencia al conjunto de valores.

En el álgebra de los números de precisión finita también se presenta el problema de que no se cumple ni la ley asociativa y ni la ley distributiva

Sistemas Numéricos

A diario usamos el sistema decimal para representar números. Este sistema utiliza los dígitos decimales o símbolos: 0,1,2,3,4,5,6,7,8,9. Podríamos usar otro sistema de numeración pero para nuestra conveniencia usamos el decimal.

Un sistema de numeración puede ser definido como un medio que se utiliza para representar una cantidad usando símbolos. La cantidad de símbolos distintos que se emplean se conoce como la base del sistema. El sistema binario contiene dos símbolos: 0 y 1, por lo tanto su base es igual a dos. Además cada símbolo en un número posee un valor dependiendo de su posición en el número. Por ejemplo, considere lo que significa el número 283

$$283 = 2*100^2 + 8*10^1 * 3$$

Esto significa que cada dígito del número ó símbolo se multiplica por 10 elevado a la potencia correspondiente a la posición de dicho dígito. Lo anterior también se aplica para un número representado en otra base. El número 11010_2 representa el número 26_{10} en base decimal.

$$11010_2 = 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0$$
$$= 16 + 8 + 0 + 2 + 0$$
$$= 26_{10}$$

En general, un número $d_m\ d_{m-1} \ldots . d_0$ en base b representa al número decimal

$$\sum_{i=0}^{n-1} d_i * b^i \quad \forall i, 0 \le d_i < b$$

Ejemplo: Convertir 137_8 a decimal

$137_8 = 1*8^2 + 3*8^1 + 7*8^0$

$= 64 + 24 + 7 = 95_{10}$

Conversión de Decimal a cualquier Base

Dado un número en decimal queremos obtener su representación en base *b*. Usando la regla de Horner podemos expresar un número Num de la siguiente manera:

$Num_{10} = d_0\ b^0 + d_1\ b^1 + d_2 b^2 + \ldots + d_{n-2}\ b^{n-2} + d_{n-1}\ b^{n-1}$

$Num_{10} = d_0 + d_1\ b^1 + d_2 b^2 + \ldots + d_{n-2}\ b^{n-2} + d_{n-1}\ b^{n-1}$

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

$Num_{10} = d_0 + b*(d_1 + b*(d_2 + \ldots + d_{n-2}\ b^{n-4} + d_{n-1}\ b^{n-3}))$

$Num_{10} = d_0 + b*(d_1 + b*(d_2 + b*(d_3 + \ldots + b*(d_{n-2} + d_{n-1}\ b)...)))$

Reescribimos Num_{10} como

$Num_{10} = d_0 + b*\ q_1$ donde $q_1 = d_1 + b*(d_2 + b*(d_3 + \ldots + b*(d_{n-2} + d_{n-1}\ b)...)))$

Para obtener q_1 utilizamos una división entera, y para obtener d_0 usamos la función módulo.

$q_1 = Num_{10} / b$ (división entera) y

$d_0 = Num_{10}$ mod b (módulo)

q_1 a su vez es un número expresable como $q_1 = d_1 + b * q_2$, de aqui obtenemos el siguiente algoritmo de conversión.

Algoritmo:

i := 0

Mientras Num_{10} <> 0 hacer

d_i := Num_{10} módulo b

$Num_{10} = Num_{10} / b$

i:= i+1

Las bases más importantes son 2 (binario), 8 (octal) y 16 (hexadecimal). El sistema octal tiene ocho símbolos distintos (0,1,2,3,4,5,6 y 7). El sistema hexadecimal posee 16 símbolos distintos (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E y F). La conversión entre números octales o hexadecimales y números binarios es fácil. Para convertir un número binario en octal basta con formar grupos de tres bits empezando de derecha a izquierda. Cada grupo de tres bits se puede convertir directamente a un solo dígito octal. La conversión de binario a hexadecimal es equivalente al octal sólo que se agrupan de 4 bits.

La conversión de octal/hexadecimal a binario es también fácil. Se toma cada dígito y se lleva a su equivalente en binario.

Representación de Números Negativos

Existen diversos sistemas para representar números negativos. El primero se conoce como 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.

5_{10} = 0000 0101

-5_{10} = 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

En el segundo sistema, llamado 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.

5_{10} = 0000 0101

-5_{10} = 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)
```

El tercer sistema se conoce como 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.

```
 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, a_{n-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, a_{n-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 $a_{n-1}\ a_{n-2}\ \ldots\ a_2\ a_1\ a_0$ 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.

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.

Consideremos la suma módulo N. Un dispositivo gráfico para la descripción de la suma módulo N de números positivos es un círculo con N valores 0 .. N-1 marcados a lo largo del perímetro del círculo. Supongamos N=16. La operación (7+4) mod 16 es igual a 11. Para realizar la operación gráficamente localizamos el valor 7 en el círculo y luego nos desplazamos 4 unidades para obtener el resultado. (9+14) mod 16 = 7 Localizamos el valor 9 y luego nos desplazamos 14 unidades obteniendo el valor 7.

Consideremos una interpretación distinta del círculo mod 16 (Figura 1). Representemos los valores del 0 al 15 como números binarios de 4 bits. Reinterpretemos estos valores para que representen los números con signo del –8 al 7 usando el método de complemento a dos. Apliquemos ahora la técnica de suma módulo 16. Por ejemplo si queremos sumar +7 y –3. La representación en complemento a dos de estos números es 0111 y 1101 respectivamente. Para sumar localizamos en el círculo el valor 0111 y luego nos desplazamos 13 unidades (1101) y nos lleva a la respuesta correcta +4

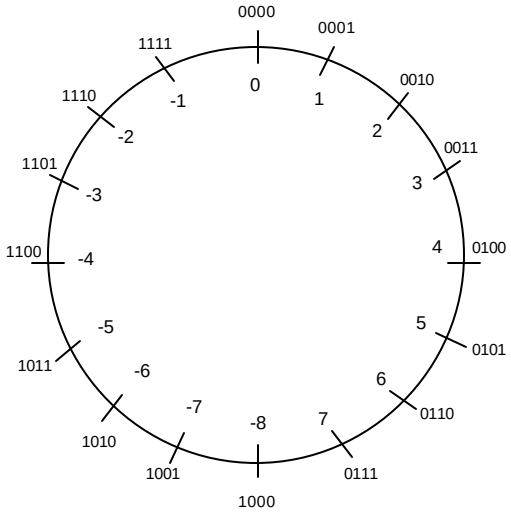

Figura 1. Círculo módulo 16

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 1001_2 en Complemento a Dos usando 4 bits, qué número representa en decimal

$1001_2 = -2^3 + 1*2^0 = -8 + 1 = -7$

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
```

Ejercicios propuestos

1. Completar la siguiente tabla de conversions entre bases

2	3	4	8	10	16
100100111					
	212				
		230			
			175		
				658	
					1AC

2. Representar los siguientes números en Complemento a Dos de 16 bits

256_{10}	
-256_{10}	
-945_{10}	

3. ¿A qué número decimal corresponde cada una de las siguientes representaciones en Complemento a Dos de 32 bits?

$ffff\ fffa_{16}$	
$0000\ 0027_{16}$	

4. Usted cuenta con un computador que trabaja con números en hexadecimal, posee una palabra de 16 bits y emplea el complemento a 2 para representar los valores que va a operar. En función de estos datos y haciendo los cálculos como los haría el computador, indique cuál será el resultado al realizar las siguientes operaciones.

   Operandos:
   a = $(-6)_{10}$ b = $(32781)_{10}$ c = $(7FFF)_{16}$ d = $(B8C2)_{16}$
   e = $(742)_{8}$ f = $(126)_{5}$ g = $(8001)_{16}$

   - a + c + f
   - e – f
   - c + f
   - a + b – e
   - (c – d) + a
   - a + g

Justifique cada una de sus respuestas.

5. ¿Cuáles de estas propiedades se cumplen y cuáles no en la suma de enteros en un computador de 32 bits que emplee complemento a 2. Justifique claramente su respuesta.
    - propiedad conmutativa.
    - propiedad asociativa.