Aritmética de Enteros

Elaborado por Prof. Ricardo González
A partir de Materiales de las Profesoras
Angela Di Serio
María Blanca Ibañez

Contenido

- Problemas relacionados a espacios finitos de almacenamiento
  - Representación de Enteros
    - Sistemas: Decimal, Binario, Octal, Hexadecimal, otros
    - Cambios de Base
  - Sistema Signo-Magnitud
  - Sistema Complemento a *Uno*
  - Sistema Complemento a Dos
  - Notación a Exceso 2^{m-1}
    - Suma *y* Resta

Números decimales

- La forma que empleamos para representar números es una notación posicional con base decimal
- En esta notación se emplean como símbolos los números arábigos (1, 2, 3, 4, 5, 6, 7, 8, 9, 0)
- El valor real de cada uno de estas cifras depende de su ubicación dentro del número escrito.

Por ejemplo para el número Mil quinientos treinta y dos

$$1532 = 1000 + 500 + 30 + 2$$

$$1 * 1000 = 1 * 10^3$$

$$5 * 100 = 5 * 10^2$$

$$3 * 10 = 3 * 10^1$$

$$2 * 1 = 2 * 10^0$$

Motivación

Hechos:

- Los números son infinitos.
- El computador tiene capacidad de almacenamiento finita y trabajan con números de precisión finita y fija. Por lo general los números se almacenan en una, dos o tres palabras.
- Trabajamos en el sistema decimal
- El computador trabaja en sistema binario.

Motivación

Preguntas:

- ¿Cuántos números podemos representar?
- ¿Se cumplen las reglas conocidas de algebra?
- ¿Cómo representar números negativos, y números reales?

Situación

Si tenemos tres (3) posiciones para almacenar números decimales. 

- Podemos representar los números enteros:

  000, 001, 002, ... , 998, 999,

  En total podemos representar o nominar a 1000 enteros diferentes con tres dígitos

El conjunto de números de precisión finita no es cerrado respecto a las cuatro operaciones básicas:

600 + 600 = 1200 (demasiado grande)
003 - 005 = -2 (demasiado pequeño) } Desbordamiento
050 x 050 = 2500 (demasiado grande)

007 / 002 = 3.5 (no es entero) ← No pertenece al conjunto

Situación

(propiedad asociativa)

$\forall$ a, b, c ε Z : a + (b - c) = (a + b) -c

Verifiquemos la propiedad asociativa para los valores: a =700, b =400 y c =300

700 + (400 -300) = (700 + 400) – 300

700 + 100 $\neq$ desbordamiento – 300

800 $\neq$ desbordamiento

Situación

(propiedad distributiva)

$\forall$ a, b, c ε Z : a x (b - c) = (a x b) - (a x c)

Verifiquemos la propiedad asociativa para los valores: a = 5, b= 210 y c = 195

5 x (210 - 195) = (5 x 210)- (5 x 195)

5 x 15 $\neq$ desbordamiento - 975

75 $\neq$ desbordamiento

Sistemas de Numeración

- Un sistema de numeración puede ser definido como un medio que se utiliza para representar una cantidad usando para ello unos símbolos.
- La cantidad de símbolos distintos que se emplean se conoce como la base del sistema.
- El sistema decimal posee 10 símbolos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 por lo tanto su base es igual a diez.
- El sistema binario posee 2 símbolos 0, 1 por lo tanto su base es igual a dos.
- El sistema hexadecimal posee 16 símbolos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D , F por lo tanto su base es igual a dieciséis.
- El sistema octal posee 8 símbolos 0, 1, 2, 3, 4, 5, 6, 7 por lo tanto su base es igual a ocho.

Sistemas de Numeración

Decimal	Binario	Hexadecimal	Octal	Base 5
00	0000	00	00	00
01	0001	01	01	01
02	0010	02	02	02
03	0011	03	03	03
04	0100	04	04	04
05	0101	05	05	10
06	0110	06	06	11
07	0111	07	07	12
08	1000	08	10	13
09	1001	09	11	14
10	1010	0A	12	20
11	1011	0B	13	21
12	1100	0C	14	22
13	1101	0D	15	23
14	1110	0E	16	24
15	1111	0F	17	30
16	10000	10	20	31
17	10001	11	21	32
18	10010	12	22	33
19	10011	13	23	34
20	10100	14	24	40
21	10101	15	25	41

Conversiones de cualquier Base a Base 10

$$d_n \ldots d_2\ d_1\ d_o$$

$$\text{Número}_{10} = \sum_{i=0}^{n} d_i \times b^i \qquad \forall\ i \quad 0 \leq d_i < b$$

- Decimal

  Se representan con los dígitos: 0 1 2 3 4 5 6 7 8 9

  $2001_{10} = 2 \times 10^3 + 0 \times 10^2 + 0 \times 10^1 + 1 \times 10^0$

  $= 2000 + 0 + 0 + 1$

- Binario

  Se representan con los dígitos: 0 1

$1\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1_2 \Rightarrow$

$1x2^{10} + 1x2^9 + 1x2^8 + 1x2^7 + 1x2^6 + 0x2^5 + 1x2^4 + 0x2^3 + 0x2^2 + 0x2^1 + 1x2^0$

$1024 + 512 + 256 + 128 + 64 + 0 + 16 + 0 + 0 + 0 + 1$

$= 2001$

Regla de Horner

Conversiones de cualquier Base a Base 10

$$d_n \ldots d_4\ d_3\ d_2\ d_1\ d_o$$

$$\text{Número}_{10} = \sum_{i=k}^{n} d_i \times b^i \qquad \forall i \quad 0 \leq d_i < b$$

- Octal

Se representan con los dígitos: 0 1 2 3 4 5 6 7

$3721_8 \Rightarrow 3 \times 8^3 + 7 \times 8^2 + 2 \times 8^1 + 1 \times 8^0$

$= 3 \times 512 + 7 \times 64 + 2 \times 8 + 1 \times 1$

$= 1536 + 448 + 16 + 1 = 2001_{10}$

- Hexadecimal

Se representan con los dígitos: 0 1 2 3 4 5 6 7 8 9 A B C D E F

10 11 12 13 14 15

$7D1_{16} \Rightarrow 7 \times 16^2 + D \times 16^1 + 1 \times 16^0$

$= 7 \times 256 + D \times 16 + 1 \times 1$

$= 1792 + 208 + 1$

$= 2001_{10}$

Conversiones de Base 10 a cualquier otra Base

- Dado un número en decimal queremos obtener su representación en base b. Usando la regla de Horner podemos expresar un Numero de la siguiente forma:

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

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

$$Num_{10} = \quad d_0 + b *(d_1 b^0 + d_2 b^1 + \ldots\ldots + d_{n-2} b^{n-3} + d_{n-1} b^{n-2})$$

$$Num_{10} = \quad d_0 + b * (d_1 + b * (d_2 b^0 + \ldots\ldots + d_{n-2} b^{n-4} + d_{n-1}\ ^{bn-3})$$

$$Num10 = \quad d_0 + b\ q_1$$

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

Para obtener q_1 basta con usar la 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 expresable como $q_1 = d_1 + b * q_2$

q_2 a su vez es expresable como $q_2 = d_2 + b * q_3$

Conversiones de Base 10 a cualquier otra Base

Algoritmo

i := 0

Mientras Num_{10} <> 0 hacer

$d_i := Num_{10}$ modulo b

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

i := i + 1

Num_{10} = 20 b = 2				
i	Num_{10}	d_i		Num'_{10}
0	20	0	d0	10
1	10	0	d1	5
2	5	1	d2	2
3	2	0	d3	1
4	1	1	d4	0
5	0			

$d_4\ d_3\ d_2\ d_1\ d_o$

1 0 1 0 0

$$\sum d_i b^i = 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 0x2^0$$

$$16 + 0 + 4 + 0 + 0$$

$$20$$

Conversiones de Base 10 a Base 2

dividendo | divisor

resto cociente

Num	Resto al dividir entre 2
20	0
10	0
5	1
2	0
1	1

```
20 | 2
 0  10 | 2
       0  5 | 2
            1  2 | 2
                 0    1
```

1 0 1 0 0

Conversiones de Base 10 a Base 2

Cocientes	Residuos
1492	
746	0
373	0
186	1
93	0
46	1
23	0
11	1
5	1
2	1
1	0
0	1

1 0 1 1 1 0 1 0 1 0 0

Conversiones de Base 10 a Base 2

dividendo	divisor	cociente	Residuo d_i	2^i	
1492	2	746	0	1	0
746	2	373	0	2	0
373	2	186	1	4	4
186	2	93	0	8	0
93	2	46	1	16	16
46	2	23	0	32	0
23	2	11	1	64	64
11	2	5	1	128	128
5	2	2	1	256	256
2	2	1	0	512	0
1	2		1	1024	1024
					1492

$1492_{10} = 1\,0\,1\,1\,1\,0\,1\,0\,1\,0\,0_{2}$

Conversiones de una Base potencia de 2 a otra

Hexadecimal 4 8 B 6

Binario 0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 0

00__ _____ _____ _____ _____ _____

Octal 0 4 4 2 6 6

0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 0 B

0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 0 H

0 0 0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 0 O

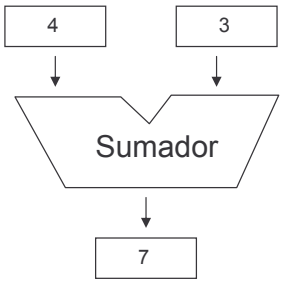
Para pasar de base x a base y

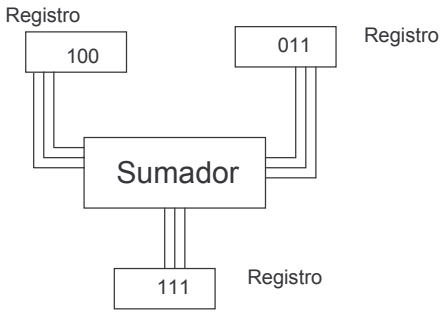
basex -> base 10 -> base y

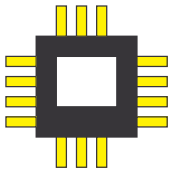
Binario	Hexadecimal	Octal
0000	0	00
0001	1	01
0010	2	02
0011	3	03
0100	4	04
0101	5	05
0110	6	06
0111	7	07
1000	8	10
1001	9	11
1010	A	12
1011	B	13
1100	C	14
1101	D	15
1110	E	16
1111	F	17

Representación de números binarios

- Existen limitaciones al espacio empleado para cada número en el computador.







=>Tamaño de la palabra del computador

Representación de números binarios negativos

Dado que tenemos una cantidad fija de bits para representar un número, debemos ahora idear una estrategia para representar a los números negativos, que nos permita realizar cálculos con cierta facilidad.

Signo Magnitud

Vamos a dedicar al bit más significativo o el que esta más a la izquierda del número para representar al signo

Si tengo 3 dígitos d_2 representará al signo y d_1 y d_0 el valor absoluto del numero a representar. El cero se emplea para números positivos y el 1 para los negativos. El resto de los bits corresponden a la magnitud que representa el número.

+ 3 = 0 1 1

Cambiamos el bit de signo

1 1 1 = - 3

Valor	S / M d2 d1 d 0
+ 3	0 1 1
+ 2	0 1 0
+ 1	0 0 1
+ 0	0 0 0
- 0	1 0 0
- 1	1 0 1
- 2	1 1 0
- 3	1 1 1

Representación de números binarios negativos

Complemento a 1

En esta representación el bit de la izquierda puede usarse como un indicativo de signo, pero los números negativos son construidos a partir de un número positivo en el cual se han cambiado todos los valores 0 por valores 1 y todos los valores 1 por 0.

+ 3 = 0 1 1

Cambiamos los 0 por 1 y los 1 por 0

1 0 0 = - 3

Valor	CA1
+ 3	0 1 1
+ 2	0 1 0
+ 1	0 0 1
+ 0	0 0 0
- 0	1 1 1
- 1	1 1 0
- 2	1 0 1
- 3	1 0 0

Representación de números binarios negativos

Complemento a 2

En esta representación también el bit de la izquierda puede usarse como un indicativo de signo, con valor de 0 para los números positivos y 1 para los negativos, pero ahora construir un número negativo es un proceso de dos fases:

primera fase: todos los valores 0 son cambiados por valores 1 y todos los valores 1 por 0.

segunda fase: se le suma un 1 al resultado anterior. Si hay un acarreo al final de esta suma se desecha el bit

+ 3 = 0 1 1

Cambiamos 0 por 1 y 1 por 0

1 0 0

Sumamos 1 + 1

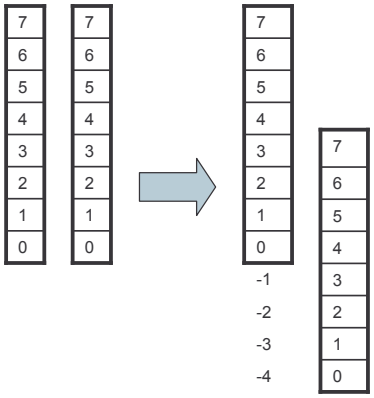
1 0 1 = - 3

Valor	CA2
+ 3	0 1 1
+ 2	0 1 0
+ 1	0 0 1
+ 0	0 0 0
- 1	1 1 1
- 2	1 1 0
- 3	1 0 1
- 4	1 0 0

Representación de números binarios negativos

Exceso

Un sistema que se emplea para números de m bits se conoce como exceso 2^{m-1} , en este formato se representa un número almacenándolo como la suma de él mismo con 2^{m-1}



Valor	Exceso 4
+ 3	1 1 1
+ 2	1 1 0
+ 1	1 0 1
+ 0	1 0 0
- 1	0 1 1
- 2	0 1 0
- 3	0 0 1
- 4	0 0 0

Representación de números binarios negativos

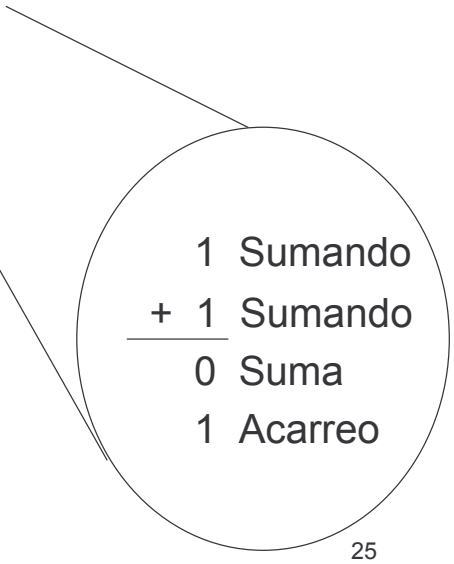
Decimal	S / M	C A 1	C A 2	Exceso a 8
-8	No existe	No existe	1000	0000
-7	1111	1000	1001	0001
-6	1110	1001	1010	0010
-5	1101	1010	1011	0011
-4	1100	1011	1100	0100
-3	1011	1100	1101	0101
-2	1010	1101	1110	0110
-1	1001	1110	1111	0111
0	1000 y 0000	1000 y 0000	0000	1000
1	0001	0001	0001	1001
2	0010	0010	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

- S / M y C A 1 poseen dos representaciones para el cero
- C A 2 y Exceso m representan más números negativos que positivos
- C A 2 es la representación más eficiente en la implementación de las operaciones de suma y resta

Suma de números binarios

$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 1 \\ \hline 10 \end{array}$$

+	0	1
0	0	1
1	1	10



Resta de números binarios

pedir prestado

$$\begin{array}{r} 0 \\ -\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ -\ 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ -\ 1 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ -\ 1 \\ \hline \end{array} \qquad \begin{array}{r} 10 \\ -\ 1 \\ \hline 1 \end{array}$$

pedir prestado

Suma Módulo N con N = 16

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.

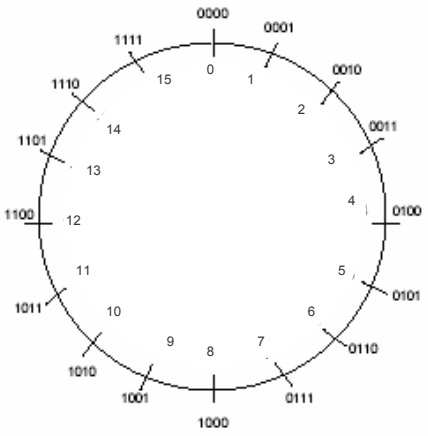


Figura 1. Circulo módulo 16

Suma Módulo N con N = 16

Consideremos una interpretación distinta del círculo mod 16

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)

(7 – 3) y nos lleva a la respuesta correcta +4

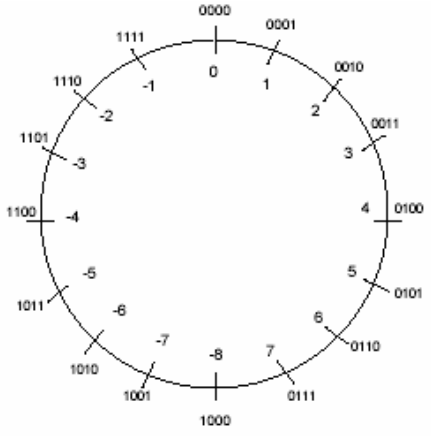


Figura 1. Circulo módulo 16

3. Bases de la aritmética binaria en coma fija

Las operaciones aritméticas en binario se realizan según tablas más sencillas que las equivalentes en el sistema decimal.

Suma binaria

SUMA BINARIA (+)	A 0	A 1
B 0	0	1
B 1	1	10

Resta binaria

RESTA BINARIA (-)	A 0	A 1
B 0	0	1
B 1	1	11

acarreo

1 acarreos	111	
9	1001	83 — 1 0 1 0 0 1 1 minuendo
+ 15	+ 1111	- 21 — - 1 0 1 0 1 sustraendo
24	11000	acarreos 111
		62 — 0 1 1 1 1 1 0 diferencia

Indicadores de resultado

El **desbordamiento** (*overflow*) es la circunstancia que sucede cuando el resultado de una operación aritmética está fuera del rango de representación.

- Desbordamiento positivo: el número es positivo y mayor que el más grande representable.
- Desbordamiento negativo: el número es más negativo (menor) que el extremo inferior del rango de los negativos.

Es necesario detectar la condición de desbordamiento (¡el resultado obtenido es erróneo!)

Subdesbordamiento (*underflow*): sucede cuando el número que queremos representar está demasiado cercano a 0 y se confunde con él.

- Subdesbordamiento positivo: el número es positivo.
- Subdesbordamiento negativo: el número es negativo.

6.a. Aritmética de binario puro

Sus reglas son las de la aritmética binaria ya estudiada, con la limitación del tamaño de los operandos (**n = p+q**).

Desbordamiento: puede darse al realizar sumas, restas, multiplicaciones y divisiones.

- **Suma**: el resultado puede tener n+1 bits (**acarreo superior C = 1**)

acarreo

```
    1101    13
  + 1111   +15
  ------   ---
  1 1100   28
```

DESBORDAMIENTO POSITIVO

- **Resta**: el resultado puede ser negativo (**acarreo superior C = 1**)

```
    1101    13
  - 1111   - 15
  ------   ---
  1 1110   - 2
```

acarreo

DESBORDAMIENTO NEGATIVO: sustraendo mayor que minuendo

- **Producto**: al multiplicar números de n bits el resultado puede necesitar hasta 2n bits (¡puede salirse de rango!).

- **División**: hay desbordamiento si el divisor es 0.

Sistema Signo-Magnitud

El bit más a la izquierda es para el signo:

0 para positivos

1 para negativos. Ejemplos:

$+5_{10} = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1_2$

$-5_{10} = 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1_2$

Suma en Signo-Magnitud

- Sume sólo enteros de igua1 signo .
- El signo del resultado es el signo de los sumandos
- Realice la suma sobre la magnitud de los números

Suma en Signo-Magnitud

0 0101 (5_{10})	1 1010 (-10_{10})	0 1010 (10_{10})
+ 0 0011 (3_{10})	+ 1 0011 (-3_{10})	+ 1 0011 (-3_{10})
0 1000 (8_{10})	1 1101 (-13_{10})	0 0111 (7_{10})

¿Cuándo ocurre *overflow* o desbordamiento?

- Cuando los signos son iguales y hay un acarreo del bit más significativo de la magnitud.

$$\begin{array}{rr} & 0\ 1\ 1\ 1\ (7_{10}) \\ + & 0\ 0\ 0\ 1\ (1_{10}) \\ \hline & 0\ 1\ 0\ 0\ 0 \end{array}$$

Problemas de los Sistema Signo-Magnitud

- Existen dos representaciones diferentes para el número 0.
- La suma y la resta requieren tener en cuenta tanto los signos de los números como sus magnitudes relativas para llevar a cabo las operaciones.

6.b. Aritmética en magnitud y signo

Este sistema equivale al que los humanos usamos para operar.

- ➲ Diferencia: opera en binario y no en base 10.

Las reglas básicas son similares a las del binario puro.

- ➲ Diferencia: es preciso tratar por separado signos y magnitudes.

Suma de R=A+B: casos posibles

- ➲ Signo(A) = Signo(B):
  - •Signo(R) = signo(A) = signo(B)
  - •$|R| = |A| + |B|$
- ➲ $A \geq 0$ y $B \leq 0$:
  - •Si $|A| \geq |B| \Rightarrow$ signo(R) = 0 y $|R| = |A| - |B|$
  - •Si $|A| < |B| \Rightarrow$ signo(R) = 1 y $|R| = |B| - |A|$
- ➲ $A \leq 0$ y $B \geq 0$: igual que el caso anterior cambiando A por B.

Resta: similar a la suma, cambiando el signo del segundo operando.

Aritmética en magnitud y signo

Por tanto, al sumar o restar con módulo y signo se debe hacer lo siguiente:

1. Observar los signos y decidir qué operación se va a realizar.
2. Ordenar los módulos si hay que restar.
3. Operar con los módulos y detectar el posible desbordamiento.
4. Colocar el signo al resultado.

Producto:

1. Se separan el signo y el módulo del multiplicando y del multiplicador.
2. Se multiplican los módulos (da un resultado de hasta 2n-2 bits).
3. Si los signos del multiplicando y el multiplicador son iguales, el resultado es positivo, y si no es negativo.

El resultado (¡puede salirse de rango!).

6.c. Aritmética en complemento

Para simplificar el diseño de los circuitos aritméticos del computador sería muy bueno **que la suma y la resta pudieran ser tratadas sin distinciones**, es decir, que la resta pudiera realizarse como si fuese una suma y no con un circuito radicalmente distinto.

En matemáticas se suele tratar a la resta como la suma de un opuesto, es decir, A-B = A+(-B), pero aún así hay que utilizar la tabla de la resta.

Ejemplo: base r=10, n=2 dígitos.

```
  23
– 02
  21
```

⇒ la suma no sirve para hacer la resta

Complementando el sustraendo y sumando

```
  23
+ 98
1 21
```

⇒ la **suma casi sirve para hacer la resta** a excepción de un 1 como bit más significativo.

25

Aritmética en complemento

Suma en complemento a 2:

Además de permitir la resta mediante la operación de suma, los números representados en complemento a la base permiten **calcular la suma operando con todos los bits de igual modo, sin hacer distinciones con el bit de signo**. Esto hace que la representación en complemento a 2 sea muy utilizada.

- **Si A >0 y B >0**: aritmética binaria pura. Ejemplo:

```
 0100    4
+0010  + 2
 0110    6
```

- **Si A >0 y B <0**: dos posibles casos dependiendo del valor absoluto de A y B
  - Si |A| ≥ |B| ⇒ R positivo, signo(R) = 0, se calcula R = |A| – |B|

    A se representa normal
    B se representa en C_2

    Ejemplo: 6 + (-4) = 2

```
 0110    6
+1100  - 4
10010    2
```

27

Aritmética en complemento

Resta en complemento a 2:

Toda operación de resta en complemento a la base puede reducirse a un caso de suma, **sin más que complementar previamente el sustraendo**.

Ejemplo: $A = 6_{|10} = 0110_{|C2}$, $B = 4_{|10} = 0100_{|C2}$, $A-B = 2_{|10}$, $n = 4$, $q = 0$

Primero: complementar el sustraendo $-B_{|C2} = C2(B_{|C2}) = 1100_{|C2}$

Segundo: sumar A+(-B)

El acarreo superior se desprecia, y el resultado es positivo

$$\begin{array}{r} 0110 \\ +\ 1100 \\ \hline 10010 \end{array}$$

Ejemplo: $A = -7_{|10} = 1001_{|C2}$, $B = -3_{|10} = 1101_{|C2}$, $A-B = -4_{|10}$, $n = 4$, $q = 0$

Primero: complementar el sustraendo $-B_{|C2} = C2(B_{|C2}) = 0011_{|C2}$

Segundo: sumar A+(-B)

$$\begin{array}{r} 1001 \\ +\ 0011 \\ \hline 1100 \end{array}$$

Aritmética en complemento

➲ **En sumas y restas en complemento a 2, el bit de acarreo superior siempre se desprecia, y el resultado obtenido siempre es correcto (salvo que se produzca desbordamiento).**

➲ **Desbordamiento en sumas y restas:** se detecta porque el resultado presenta un signo erróneo.

La presencia de desbordamiento se revela poniendo el indicador V a 1.

Puede producirse desbordamiento al sumar dos números de igual signo o al restar dos números de distinto signo.

Nunca puede haber desbordamiento al sumar números de distinto signo o al restar números de igual signo.

El posible acarreo superior resultante en una suma o una resta no indica desbordamiento.

También puede producirse desbordamiento en productos y divisiones.

Aritmética en complemento

Ejemplos de sumas con desbordamiento

$A = 6_{L10} = 0110_{LC2}$, $B = 3_{L10} = 0011_{LC2}$, $A+B = 9_{L10}$, n = 4, q = 0

0110	6
+ 0011	+ 3
1001	¡¡-7!!

La suma de dos números positivos no puede producir un número negativo: V = 1

$A = -3_{L10} = 1101_{LC2}$, $B = -7_{L10} = 1001_{LC2}$, $A+B = -10_{L10}$, n = 4, q = 0

1101	-3
+ 1001	+ -7
10110	¡¡6!!

La suma de dos números negativos no puede producir un número positivo: V = 1

El acarreo superior se desprecia, y el resultado es positivo

Suma y resta en CA2

Las reglas que gobiernan la suma y la resta de números de n bits usando el sistema de representación en complemento a 2 (CA2) son:

- Para sumar dos números, sume sus representaciones. El resultado será correcto siempre y cuando el resultado esté dentro del rango $-(2^{n-1})$ y $2^{n-1}-1$.
- Para restar dos números X - Y, obtenga la representación de –Y en complemento a 2 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 a los operandos entonces ha ocurrido un error de desborde (overflow) y la respuesta númerica obtenida es incorrecta.

Sistemas de Complemento a 1 y a 2

Sistema Complemento a 1

Comentarios

- Hay dos representaciones diferentes del 0 .
- Este sistema hoy en día es obsoleto.

Sistema Complemento a 2

Comentarios.

- Unica representación del 0.
- Es fácil realizar las operaciones aritméticas bajo esta representación.

6.g. Extensión de signo

Es una operación consistente en que, dado un número A representado con **n** bits, pasamos a representarlo con **m** bits, siendo **n < m**.

➲ **Binario puro**: se rellenan los bits sobrantes en el destino con 0.

Ejemplo: extender X = 0110_{I2} de 4 a 8 bits

0110

00000110

➲ **Magnitud y signo:** se desplaza a la izquierda el bit de signo, y el hueco en el destino se rellena con bits a 0.

Ejemplo 1: extender X = 100110_{IMS} de 6 a 8 bits

1̲00110

1̲**00**00110

Ejemplo 2: extender X = 010011_{IMS} de 6 a 8 bits

0̲10011

0̲**00**10011

41

Extensión de signo

➲ **Complemento a 2**: se replica el bit de signo hacia la izquierda.

Ejemplo 1: extender X = $100110_{|C2}$ de 6 a 8 bits

100110

11100110

Ejemplo 2: extender X = $010011_{|C2}$ de 6 a 8 bits

010011

00010011

➲ **Complemento a 1**: se hace igual que en complemento a 2.

➲ **Exceso a M**: no se suele hacer, ya que implica un cambio en el valor del exceso.

2. Operaciones lógicas

De acuerdo con los axiomas del álgebra de Boole, las operaciones lógicas toman bits individuales como operandos.

Sin embargo, los computadores realizan operaciones lógicas tomando datos completos de **n** bits.

➲ Operación lógica NOT: se invierten todos los bits del operando.

•Ejemplo: n=4 bits, A=0110.

A = 0 1 1 0

NOT A = 1 0 0 1

➲ Operaciones binarias: se realizan bit a bit con dos operandos.

•Ejemplo: n=4 bits, A=0110, B=1100.

A = 0 1 1 0
B = 1 1 0 0
A OR B = 1 1 1 0

A = 0 1 1 0
B = 1 1 0 0
A AND B = 0 1 0 0

A = 0 1 1 0
B = 1 1 0 0
A EXOR B = 1 0 1 0

5. Operaciones de desplazamiento

Son operaciones unarias en las que los bits del operando se desplazan hacia la izquierda o hacia la derecha.

La longitud del desplazamiento será **s** ≥ 1, generándose un "hueco" de **s** bits.

Según el criterio utilizado para dar valores a los **s** bits del hueco hay varios tipos de desplazamientos:

- Lógicos.
- Aritméticos.
- Circulares (rotaciones).

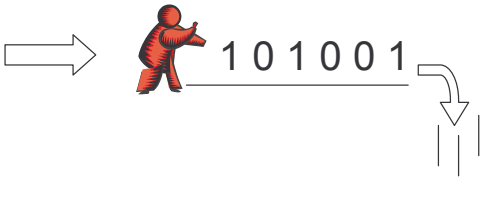
En el MC68000 los desplazamientos y rotaciones suelen involucrar a algún indicador de resultado.

13

Operaciones de Desplazamiento

1 0 1 0 0 1 1

Desplazamiento a la derecha



Desplazamiento lógico

En este caso los bits del hueco se rellenan con ceros.

- El último bit que sale va al indicador C (y/o al X).

Ejemplos: n = 6 bits, A = 001011

- Desplazamientos lógicos a la derecha

Longitud s = 1	Longitud s = 2	Longitud s = 3
A' = **0**00101	A' = **00**0010	A' = **000**001
C,X = 1	C,X = 1	C,X = 0

- Desplazamientos lógicos a la izquierda

Longitud s = 1	Longitud s = 2	Longitud s = 3
A' = 01011**0**	A' = 1011**00**	A' = 011000
C,X = 0	C,X = 0	C,X = 1

Instrucciones de desplazamiento lógico en ensamblador:

- MC68000: LSR (derecha); LSL (izquierda), cumplen las reglas anteriores.
- MIPS: SRL, SRLV (derecha); SLL, SLLV (izquierda).

Desplazamiento aritmético

Se usa cuando se considera que el dato es un número en complemento a 2.

- Hacia la derecha: se replica el bit de signo.
- Hacia la izquierda: se rellena con ceros, y si se modifica el bit de signo en el proceso el indicador V se pone a 1.
- El último bit que sale va al indicador C (y/o al X).

Ejemplos de desplazamientos aritméticos a la derecha

- n = 6 bits, A = 001011

Longitud s = 1	Longitud s = 2	Longitud s = 3
A' = **0**00101	A' = **00**0010	A' = **000**001
C,X = 1	C,X = 1	C,X = 0

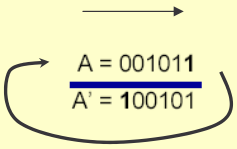
- n = 6 bits, B = 100101

Longitud s = 1	Longitud s = 2	Longitud s = 3
A' = **1**10010	A' = **11**1001	A' = **111**100
C,X = 1	C,X = 0	C,X = 1

Rotación

Los bits que salen por un extremo entran por el otro.

- El último bit que sale va al indicador C (y/o al X).



Ejemplos: n = 6 bits, A = 001011

➲ Rotación a la derecha

Longitud s = 1	Longitud s = 2	Longitud s = 3
A = 00101**1**	A = 0010**11**	A = 001**011**
A' = **1**00101	A' = **11**0010	A' = **011**001
C,X = 1	C,X = 1	C,X = 0

➲ Rotación a la izquierda

Longitud s = 1	Longitud s = 2	Longitud s = 3
A = **0**01011	A = **00**1011	A = **001**011
A' = 01011**0**	A' = 101**00**	A' = 011**001**
C,X = 0	C,X = 0	C,X = 1

17

Multiplicación y División por la base del sistema

En el sistema decimal multiplicar por una potencia de 10 equivale a desplazar un número tantas veces a la izquierda.

Dividirlo entre una potencia de 10 equivale a desplazarlo a la derecha tantas veces como la potencia de 10 lo indique.

$15_{10} \times 10^2 = 1500_{10}$ $\quad$ $12000_{10} / 10^1 = 1200_{10}$

En el sistema binario la situación es similar

$000101_2 \times 2^2 = 010100_2$ $\quad$ $01000_2 / 2^1 = 00100_2$

5_{10} 4_{10} 20_{10} $\quad$ 8_{10} 2_{10} 4_{10}

Representación de Caracteres

- El computador trabaja y almacena números binarios.
- Las personas entendemos los números decimales y las palabras escritas con caracteres alfabéticos.
- Para poder almacenar y procesar caracteres alfabéticos se hace necesario establecer una correspondencia entre caracteres y números para que el computador pueda procesarlos
- Para ello se han creado tablas de códigos
  - ASCII (código americano estándar para el intercambio de informacion)
  - EBCDIC
  - Unicode

Tabla ASCII

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
0	0	NUL	16	10	DLE	32	20	(space)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	□