

4

Aritmética de la computadora

Contenido

4.1 Introducción	74
4.2 Aritmética binaria.....	74
4.3 Aritmética decimal	79
4.4 Resumen.....	84
4.5 Ejercicios propuestos.....	85
4.6 Contenido de la página Web de apoyo.....	86

Objetivos

- Formular los convenios para la representación de cantidades signadas.
- Determinar los rangos de representación en formato de n bits.
- Evaluar la validez de las operaciones aritméticas.



En la página Web de apoyo encontrará un breve comentario de la autora sobre este capítulo.

4.1 Introducción

El desarrollo de este capítulo permite el análisis de las operaciones básicas de la aritmética binaria, implementadas en la unidad de coma fija para números enteros. Las operaciones en BCD corresponden a la aritmética decimal o de “base diez codificada en binario”, implementada en los microprocesadores en la unidad de coma flotante (aun cuando no se trata de la misma forma de operatoria) y en unidades específicas de coma flotante decimal en *mainframes*. Los números “decimales codificados en binario” (BCD o *Binary Coded Decimal*) se utilizan en aplicaciones donde deba considerarse importante un error por conversión; por ejemplo, si $0,2_{(10)}$ es igual en binario a $0,0011$ periódico en las cuatro cifras fraccionarias, significa que la conversión a este binario se aproxima pero nunca será igual al original en base 10. Es de destacar que la aritmética decimal es más compleja que la binaria y que la representación interna de los datos ocupa mucho mayor espacio de almacenamiento.

4.2 Aritmética binaria

Físicamente, una unidad aritmética es un circuito asociado con uno o más registros, que contienen los operandos en la operación. Este circuito tiene un diseño distinto de una computadora a otra, que depende fundamentalmente de las operaciones que realice. Algunas computadoras sólo tienen implementadas en hardware operaciones muy simples, como la suma, el desplazamiento y el complemento. Las operaciones más complejas, como la multiplicación o la radicación, se realizan por medio de la ejecución de un software, que logra la solución con operaciones simples implementadas en hardware. Estas rutinas constituyen lo que se denomina **implementación en software** de operaciones aritméticas, de modo que el programador no debe preocuparse por implementarlas cuando las necesite. Por ejemplo, dada una sentencia de un lenguaje de programación que defina una multiplicación, se llamará a la rutina correspondiente que puede resolverla por sumas sucesivas y desplazamientos. El proceso insumirá cierto tiempo de ejecución, por lo que se deduce que las implementaciones en software simplifican el diseño de hardware a expensas de mayor tiempo de ejecución.

Recordemos que una ALU puede operar en base 2 cuando los datos se definen como enteros (signados o no), o como reales, y en base 10 cuando se pueden definir como decimales codificados en binario (BCD).

En una computadora cada operando será transferido de una o varias locaciones de memoria a uno de los registros de la CPU, que tiene una capacidad fija y limitada. La ALU no reconoce el formato de los operandos, sino que ellos se establecen de acuerdo con convenciones definidas para cada tipo de dato en los programas que los utilizan y sólo estos últimos son los encargados de interpretarlos, resultando transparentes para los componentes físicos de la ALU.

Describiremos ahora los algoritmos para la resolución de los problemas aritméticos elementales que puede utilizar la ALU en la operatoria de datos binarios enteros, signados, en su representación de punto fijo, y para datos binarios reales, en su representación de punto flotante.

4.2.1 Representación de datos en punto fijo (binarios enteros)

Para un operando declarado como entero, la coma mantiene una posición fija a extrema derecha de su magnitud y los números se representan en un convenio denominado representación de datos en punto fijo. El punto no está presente en el registro donde se almacena el operando, sino que su presencia se supone manteniendo esta posición fija.

Los números positivos se representan en la forma de magnitud con signo y los negativos pueden representarse de tres maneras diferentes. En una misma computadora, sólo los negativos se tratarán de una de estas formas. Por ejemplo, sea el número $-3_{(10)}$ podemos representarlo en un formato de 8 bits así:

1. en magnitud con signo
- 1 0000011
2. en complemento a 1 (con signo)
- 1 1111100
3. en complemento a 2 (con signo)
- 1 1111101

4.2.2 Operaciones aritméticas con enteros signados

Considere que los registros que contendrán los operandos que han de intervenir en la operación almacenan 8 bits y los negativos se representan en su forma de complemento a 2. Consideremos los cuatro casos de sumas posibles:

1. $(+A) + (+B)$
2. $(+A) + (-B)$
3. $(-A) + (+B)$
4. $(-A) + (-B)$

En todos los casos se realizará una suma binaria entre los pares de bits de igual peso y, en ocasiones, estas sumas parciales afectarán el par de bits de la siguiente columna con un acarreo (carry) igual a 1, como se puede observar en los ejemplos que se presentan a continuación.

$(+A) + (+B)$	$(+A) + (-B)$	$(-A) + (+B)$	$(-A) + (-B)$
$(+3) + (+3) = +6$	$(+10) + (-7) = +3$	$(-7) + (+1) = -6$	$(-64) + (-32) = -96$
00000011	00001010	11111001	11000000
+ 00000011	+ 11111001	+ 00000001	+ 11100000
00000110	00000011	11111010	10100000

En este procedimiento hay algunos conceptos importantes que destacar:

- Los operandos negativos están complementados a la base y, por lo tanto, los resultados negativos que se obtienen están también complementados.
- El signo forma parte de la operación, sumándose como cualquier otro bit.
- En las operaciones se consideran todos los bits del formato, aun cuando sean 0 no significativos (a izquierda).

En el primer caso los operandos (ambos positivos) están en su forma de magnitud binaria real y signo. Se obtuvo el resultado esperado sin lugar a dudas.

En el segundo caso el primer operando es positivo y el segundo, negativo (complementado); se obtuvo el resultado esperado y el acarreo final se desprecia.

En el tercer caso el primer operando es negativo (complementado) y el segundo es positivo; sin embargo, el resultado obtenido no aparenta ser -6. La razón es que al ser negativo, el valor obtenido es su complemento a la base, en este caso el número de 7 bits obtiene su complemento de la base $2^7 = 128$. Si se convierte el valor obtenido a decimal se obtiene 122, que es precisamente lo que le falta a 6 para llegar a 128, por lo tanto, el resultado es correcto. La ALU entrega como resultado la secuencia de bits obtenida y es función del programa interpretar que cuando el signo es negativo el valor está enmascarado en su complemento. Si quiere realizar la verificación, simplemente calcule el complemento y obtendrá la magnitud buscada.



El acumulador es un registro de cálculo referenciado por la mayor parte de las instrucciones aritméticas.

<i>s</i>	<i>Magnitud complementada</i>
1	1111010 (invierto y sumo 1)
	<i>Magnitud binaria real</i>
	0000110

En el cuarto caso ambos operandos son negativos y, por lo tanto, están complementados; sin embargo, puede parecer que el primero no lo está, eso ocurre porque la magnitud complementada coincide con la magnitud real. El complemento a la base de 64, en 7 bits, es lo que le falta a 64 para llegar a $2^7 = 128$, que es exactamente 64.

	<i>Magnitud binaria real</i>
	1000000 (invierto y sumo 1)
<i>s</i>	<i>Magnitud complementada</i>
1	1000000

Nótese que en este caso también hay acarreo final que se desprecia.
Las sumas vistas pueden efectuarse en un sumador con las características que se presentan en la figura 4.1.

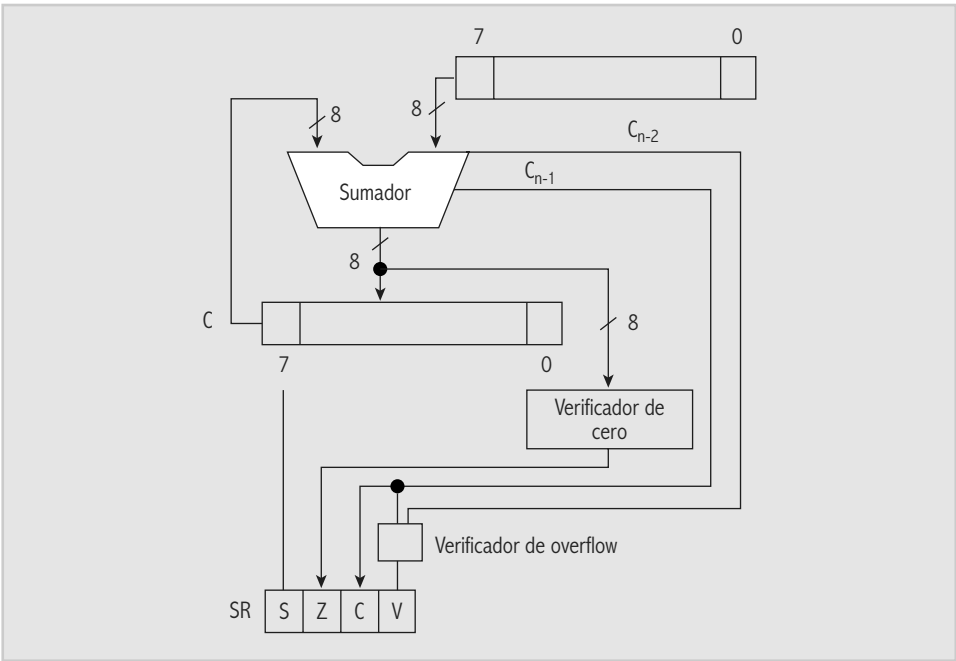


Fig. 4.1. Esquema de registros para suma de 8 bits.

El registro de estado almacena las banderas o flags aritméticas y otras que representan ciertas condiciones que pueden ser verdaderas o falsas.

El esquema presentado en la figura 4.1 es un bosquejo parcial de una unidad aritmética formado por:

- Un registro de 8 bits acumulador (AC), que almacenará el primer operando antes de la operación y el resultado después de ella.
- Un registro de 8 bits, que almacenará el segundo operando.

- Un registro de control del que consideraremos, sólo por ahora, 4 *flags* o banderas, llamado Registro de Estado o *Status Register* (SR).

Los *flags* son:

flag V: indica si hubo o no *overflow* o desborde de registro, después de haberse llevado a cabo la operación. Si almacena un valor “1” significa que hubo *overflow*; si almacena un valor “0” significa que no hubo *overflow*.

flag C: indica el acarreo o *carry* que se produce en la operación. Se pone en “1” si C_n es igual a “1”, o se pone en “0” si C_n es igual a “0”.

flag S: indica el signo del resultado de la operación. Se pone en “1” si el resultado es negativo, o se pone en “0” si el resultado es positivo.

flag Z: indica si el resultado de la operación es 0 o distinto de 0. Se pone en “1” si el resultado es 0, o se pone en “0” si el resultado es distinto de 0.



El acarreo o *carry* es el 1 producido cuando se suma 1+1 en base 2.

4.2.2.1 *Overflow, sobreflujo o bit de desborde*

Se dice que hay un sobreflujo cuando se excede la capacidad de un registro de almacenamiento, lo que provoca la invalidez del resultado obtenido. El *overflow* se verifica cuando, después de realizada una operación, se determina que el último acarreo es distinto del penúltimo.

Para un formato de 8 bits, el primero es para el signo y los 7 restantes para la magnitud; por lo tanto, el rango de variabilidad está comprendido entre -128 y +127. Para que se produzca *overflow*, bastaría con sumar +1 al operando +127. No obstante, observemos los casos extremos:

	$\begin{array}{c} C_7 C_6 \\ \downarrow \downarrow \\ 0 \ 1 \end{array}$	$\begin{array}{c} 7 \\ \downarrow \\ 1 \end{array}$	$\begin{array}{c} 6 \\ \downarrow \\ 1 \end{array}$	$\begin{array}{c} 5 \\ \downarrow \\ 1 \end{array}$	$\begin{array}{c} 4 \\ \downarrow \\ 1 \end{array}$	$\begin{array}{c} 3 \\ \downarrow \\ 1 \end{array}$	$\begin{array}{c} 2 \\ \downarrow \\ 1 \end{array}$	$\begin{array}{c} 1 \\ \downarrow \\ 1 \end{array}$	$\begin{array}{c} 0 \\ \downarrow \\ 1 \end{array}$
+127									01111111
+									
+127									01111111
+254									11111110
									nueva magnitud

	$\begin{array}{c} C_7 C_6 \\ \downarrow \downarrow \\ 1 \ 0 \end{array}$	$\begin{array}{c} 7 \\ \downarrow \\ 0 \end{array}$	$\begin{array}{c} 6 \\ \downarrow \\ 0 \end{array}$	$\begin{array}{c} 5 \\ \downarrow \\ 0 \end{array}$	$\begin{array}{c} 4 \\ \downarrow \\ 0 \end{array}$	$\begin{array}{c} 3 \\ \downarrow \\ 0 \end{array}$	$\begin{array}{c} 2 \\ \downarrow \\ 0 \end{array}$	$\begin{array}{c} 1 \\ \downarrow \\ 0 \end{array}$	$\begin{array}{c} 0 \\ \downarrow \\ 0 \end{array}$
-128									10000000
+									
-128									10000000
-256									00000000
									nueva magnitud

La primera es la suma de los positivos más grandes que admite el formato. Siempre, cuando hay *overflow* de positivos, $C_7 = 0$ y $C_6 = 1$; el resultado correcto se logra si se considera que C_7 es signo del resultado y los ocho bits obtenidos de resultado son la magnitud binaria real. La segunda es la suma de los negativos más grandes que admite el formato. Cuando hay *overflow* de negativos, $C_7 = 1$ y $C_6 = 0$; el resultado correcto se logra si se considera que C_7 es signo del resultado y los ocho bits obtenidos son el complemento de la magnitud binaria real. En este caso es lógico analizar que si la combinación de 10000000 (8 bits) es -128, entonces la combinación de 100000000 (9 bits) corresponde a -256, que es la base siguiente.

Se debe destacar que el *overflow* sólo puede producirse con operandos del mismo signo, ya que en el caso contrario (un positivo y un negativo), el resultado obtenido siempre “entrará” en el formato utilizado (en realidad, uno se resta del otro).

Como ya dijimos, una ALU que tiene sólo un circuito sumador, como en este caso, realiza las restas binarias transformándolas en sumas de la siguiente manera:

$$\begin{aligned} (+A) - (+B) &= (+A) + (-B) \\ (+A) - (-B) &= (+A) + (+B) \\ (-A) - (+B) &= (-A) + (-B) \\ (-A) - (-B) &= (-A) + (+B) \end{aligned}$$

O sea que, para efectuar una resta, la ALU debe recibir el complemento del segundo operando calculado previamente; así, un sustraendo positivo pasa a ser un operando negativo y un sustraendo negativo pasa a ser un operando positivo, quedando la resta implementada como una suma.

Una rutina software de resta debe ordenar las siguientes operaciones simples:

1. Invertir los bits del sustraendo (cálculo del complemento restringido).
2. Incrementarlos en una unidad (cálculo del complemento auténtico).
3. Sumarles el minuendo.

Delimitemos los pasos con el siguiente ejemplo:

$$(+A) - (-B)$$

$$(+10) - (-7) = +17$$

$$+10 = 00001010 \quad -7 = 11111001$$

1. 000000110

2. 00000111

3. 00000111

00010001

4.2.3 Operaciones aritméticas en punto flotante

Habiendo dejado en claro que todas las operaciones para realizar en una ALU elemental serán transformadas en sumas, falta aclarar que, para sumar dos operandos en representación de punto flotante (RPFte), se debe tener en cuenta el alineamiento del punto, esto es, que ambos exponentes sean iguales. Si este procedimiento lo realiza el usuario escribiendo sobre un papel, alinear parte entera con parte entera y parte fraccionaria con parte fraccionaria es bastante sencillo, pero para el procesador no lo es.

Quando un operando en punto flotante está desalineado respecto del otro, los exponentes son distintos, por lo tanto, la operación de suma se ve complicada con un paso previo. Una rutina software, semejante a la que se describe a continuación, servirá de apoyo para que el hardware de la ALU lleve a cabo la suma:

- Enviar a la ALU las entidades representativas de los exponentes de ambos operandos.
- Comparar los exponentes, por ejemplo, restando uno de otro. Si el resultado es igual a 0, significa que los exponentes son iguales, por lo tanto, las mantisas tienen su punto alineado. Cuando ambos exponentes son distintos, significa que se debe desplazar una de las mantisas para alinearla con la otra; el criterio que sigue el programa para decidir cuál de las dos se desplaza depende del convenio de punto flotante utilizado.
- Enviar a la ALU las mantisas para ser sumadas.

La unidad aritmética está capacitada para desplazar la información de algunos registros asociados con ella, de manera que se puede cargar en ellos la mantisa de uno de los operandos y, así, desplazarla los lugares necesarios hasta alinearla con la otra.

En los convenios en exceso las características de ambos operandos son binarias sin signo. Si se comparan, la menor característica indicará que la correspondiente mantisa se debe desplazar a la derecha y el valor que se obtiene del resultado de la comparación señalará cuántos lugares se debe

desplazar. El hecho de desplazar el operando que tiene menor característica responde a que un desplazamiento a derecha implica la pérdida de los dígitos menos significativos y, a lo sumo, el resultado se verá afectado por un redondeo. En caso contrario, un desplazamiento a izquierda representará la pérdida de los dígitos más significativos de la mantisa y, por ende, el resultado será erróneo.

A continuación, se presentan dos ejemplos para dar una “idea” de la operatoria de la ALU, dejando aclarado que las operaciones se realizan en binario signado, en punto flotante, con un exceso igual a 2^{p-1} , con mantisa fraccionaria normalizada, siendo $m = \delta$ y $p = \delta$.

$$\begin{aligned}
 1) \quad & (+32)_{(10)} + (+16)_{(10)} = \\
 & 32_{(10)} = 00100000_{(2)} = 0.1000000 \cdot 10^{+110} \\
 & + \underline{16}_{(10)} = 00010000_{(2)} = 0.1000000 \cdot 10^{+101} \\
 & 48_{(10)}
 \end{aligned}$$

$$\begin{array}{rcl}
 10000110 & \left[\begin{array}{l} 0.1000000 \\ 0.1000000 \\ 0.0100000 \end{array} \right] \\
 10000101 & + & 0.1000000 \\
 10000110 & \left[\underline{0.0100000} \right] \\
 10000110 & 0.1100000 \\
 \hline
 & + & 48_{(10)}
 \end{array}$$

En este ejemplo se muestra el desplazamiento de la mantisa, correspondiente al decimal 16, un lugar a la derecha para igualar las características.

$$\begin{aligned}
 2) \quad (+1)_{(10)} + (+6)_{(10)} &= \\
 1_{(10)} &= 00000001_{(2)} = 0.1000000 \cdot 10^{+1} \\
 + \underline{6_{(10)}} &= 00000110_{(2)} = 0.1100000 \cdot 10^{+1} \\
 7_{(10)} &
 \end{aligned}
 \quad
 \boxed{
 \begin{array}{r}
 10000001 \quad 0.1000000 \\
 10000011 \quad \lceil 0.1100000 \\
 10000011 + \underline{0.0010000} \\
 10000011 \quad \lfloor 0.1110000 \\
 \hline
 \quad \quad \quad + \quad 7_{(10)} \quad \quad
 \end{array}
 }$$


En este ejemplo el desplazamiento de la mantisa, correspondiente al decimal 1, es de dos lugares a la derecha para llegar a la misma característica.

La suma de dos mantisas normalizadas suele provocar un *overflow* de resultado y éste puede subsanarse con facilidad si se realiza un corrimiento del acarreo-signo-mantisa a derecha y se le suma un 1 a la característica. Si la suma no produjo *overflow*, pero el resultado se encuentra desnormalizado, se deberá efectuar un corrimiento a izquierda y restar del exponente el número representativo de los corrimientos efectuados. Si como consecuencia de los corrimientos, y a causa de representar la mantisa números muy grandes, se produjera sobreflujo en la característica, éste activará la bandera de *overflow* del SR.

4.3 Aritmética decimal

Como se indicó antes, una computadora digital puede realizar operaciones aritméticas en BCD (decimal codificado en binario). Los operandos pueden ser números con signo o sin él; la diferencia que hay entre ambos es que los números signados agregarán a su formato un dígito BCD, destinado a contener en el último bit el signo del operando. Es obvio que si se trabaja con un formato fijo se deberá tener en cuenta que la capacidad de almacenamiento se verá restringida a un dígito BCD menos, dedicado a contener el signo.

Desarrollaremos aquí el algoritmo utilizado por la rutina que se encarga de operar números BCD sin signo, dejando asentado que la operatoria para el manejo de los números BCD signados es similar a la de los binarios signados.



BCD (*Binary Coded Decimal*): es un código que representa un dígito decimal con 4 bits.

4.3.1 Operaciones con operandos BCD

4.3.1.1 Suma

Consideremos dos números decimales A y B , siendo a_i y b_i los dígitos respectivos. Si queremos efectuar la suma de los dígitos de a pares, debemos tener en cuenta, también, un posible acarreo c_{i-1} (*carry_{i-1}*), que a lo sumo tendrá valor “1”

C_n

A

$+ B$

S

C_{n-1}

$a_n \dots$

$b_n \dots$

C_n

C_{i-1}

$a_i \dots$

$b_i \dots$

$S_{i-1} \dots$

C_0

a_0

b_0

S_0

La situación más desfavorable que se puede presentar es que

$a_i = 9_{(10)} = 1001_{(BCD)}$

$que \quad b_i = 9_{(10)} = 1001_{(BCD)}$

$y \quad que \quad c_{i-1} = 1_{(10)} = 0001_{(BCD)}$

$cuyo \text{ resultado sería } S_i = 19_{(10)}$

Tabla 4-1. Tabla de conversión		
Decimal	Binario	BCD
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	$6 + \boxed{1010}$	$-\boxed{0001\ 0000}$
11	1011	0001 0001
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101
16	$\boxed{10000}$	0001 0110
17	10001	0001 0111
18	10010	0001 1000
19	10011	0001 1001

Si observamos los valores de la tabla 4-1 y los consideramos posibles S_i , vemos que mientras no excedan $9_{(10)}$ la correspondencia binaria-BCD es idéntica.

¿Qué pasa entonces con los $S_i > 9_{(10)}$?

Nótese que la tabla binaria está desfasada de la tabla S_i BCD en 6 unidades. Por lo tanto, si $S_i > 9_{(10)}$, entonces varía entre 10 y 19, pues la suma máxima es $9 + 9 + 1$.

$10 \leq S_i \leq 19,$

o cuando $S_i \leq 9 \text{ pero } C_i = 1$

se suma 6 (codificado en BCD) para obtener el resultado correcto; además, se debe tener en cuenta que se genera un acarreo $C_i = 0001_{(BCD)}$ para la suma del siguiente par de dígitos de los operandos BCD.

Observemos la operatoria para la suma en BCD en los siguientes dos casos:

<i>Decimal</i>	<i>BCD</i>
123	0001 0010 0011
+ 456	+ 0100 0101 0110
579	0101 0111 1001
456	0100 0101 0110
+ 789	+ 0111 1000 1001
1245	1011 1101 1111
	+ 0110 0110 0110
	+ 0001 0001 0001 10101
	0001 10010 10100 5
	1 2 4

4.3.1.2 Resta

Caso en que el minuendo (M) > sustraendo (S)

Siendo la resta $M - S = R$ (resto o resultado), demostraremos que si sumamos el minuendo al complemento del sustraendo obtenemos el mismo resultado incrementado en 10^n , siendo n la cantidad de dígitos de M y de S :

$$M + C_s = R + 10^n$$

Dada la resta $70 - 40 = 30$

entonces $70 + (10^2 - 40) = 30 + 10^2$

$$70 + 60 = 130$$

eliminando 10^2 de R obtenemos 30 , que es el resultado correcto.

De la misma manera: $M + C'_s + 1 = R + 10^n$

Caso en que el minuendo < sustraendo

Si el minuendo es menor que el sustraendo, el resultado obtenido es el “complemento de la diferencia real”, luego:

$$M + C_s = C_{R(\text{resultado})}^B$$

Dada la resta $20 - 30 = -10$

$$20 + (10^2 - 30) = 10^2 - 10$$

$$20 + 70 = 90$$

y 90 es el complemento de la magnitud del resultado. Por lo tanto, 90 representa -10 , que es la diferencia real.

También: $M + C_s + 1 = C_{R(\text{resultado})}^{B-1}$

Utilizando el complemento a la base 0 a la base -1 , evitamos la implementación de un circuito que efectúe la resta dentro de la unidad aritmética, dado que se puede transformar en una suma.

El código BCD 8421 no es autocomplementario, ya que no obtenemos el complemento a 9 de un número (complemento restringido BCD) simplemente invirtiendo sus bits (procedimien-

to utilizado en el sistema de base 2), sino que debemos incluir una corrección. Para hallar el complemento de un dígito BCD, la rutina de tratamiento de la operación en curso, se puede utilizar el algoritmo que enunciamos a continuación.

Algoritmo para el cálculo del complemento restringido de un dígito BCD natural:

*Al dígito BCD se le suman 6 unidades (0110)
y al resultado se le invierten los bits.*

Este algoritmo basa su lógica en considerar que la base de un operando BCD es 10, por lo tanto, el complemento restringido del mismo se obtendrá de la base -1, o sea, 9. Si al dígito BCD lo llamamos N se calculará como 9 - N.

A su vez, un dígito BCD se representa en una computadora con cuatro dígitos binarios y, entonces, su complemento restringido se obtendrá de 1111₍₂₎ [equivalente a 15₍₁₀₎], o sea, 1111₍₂₎ - N, que es lo mismo que expresar 15₍₁₀₎ - N.

Para equiparar las dos relaciones enunciadas antes, (9 - N) y (15 - N), es necesario introducir una corrección al segundo enunciado, que se detalla a continuación:

$$9 - N = (15 - 6) - N, \text{ operando podemos indicar que}$$
$$9 - N = 15 - (6 + N).$$

La última ecuación define el algoritmo enunciado precedentemente. Como ya se demostró, esta resta equivale al complemento binario de 6 + N, razón por la cual una vez que ésta se llevó a cabo, se invierten los bits resultantes.

Veamos la operatoria utilizada para la resta:

Decimal	BCD 8421	Regla de resta
456	0100 0101 0110	0 - 0 = 0
- 123	- 0001 0010 0011	0 - 1 = 1 con acarreo
		1 - 0 = 1
333	001100110011	1 - 1 = 0

Proponemos:

999	456	suma del minuendo más
- 123	+ 876	el complemento del sustraendo
876	1 332	
	+ 1	más 1 para llegar a la base por ser
	1 333	complemento restringido
		se desprecia
		el exceso de 10 ⁿ

Según el algoritmo, se le suma 6 (0110) a cada dígito y luego se invierten:

	suma de 0110	inversión	
C'0001 =	0001 + 0110 = 0111	= 1000	= 8 ₍₁₀₎
C'0010 =	0010 + 0110 = 1000	= 0111	= 7 ₍₁₀₎
C'0011 =	0011 + 0110 = 1001	= 0110	= 6 ₍₁₀₎

luego, se realiza la suma en BCD 8421:

$$\begin{array}{r}
 0100 \ 0101 \ 0110 \\
 + \underline{1000 \ 0111 \ 0110} \\
 1100 \ 1100 \ 1100 \\
 + \underline{0110 \ 0110 \ 0110} \\
 \underline{0001} \overline{0001} \overline{10010} \\
 1 \ 0011 \ 10011 \\
 + \qquad \qquad \qquad 0001 \text{ para llegar a la base} \\
 \underline{1} \overline{0011 \ 0011 \ 0011} \\
 \underline{1} \overline{3 \ 3 \ 3} \\
 \text{se desprecia}
 \end{array}$$

A continuación, resolvemos algunos ejercicios para afianzar los procedimientos:

1) $25 + 2 =$

$$\begin{array}{r}
 25 \qquad \qquad \qquad 0010 \ 0101 \\
 + \quad \underline{2} \qquad \qquad + \underline{0000 \ 0010} \\
 27 \qquad \qquad \qquad \underline{0010} \ \underline{0111} \\
 \qquad \qquad \qquad \underline{2} \quad \underline{7}
 \end{array}$$

2) $662 + 1045 =$

$$\begin{array}{r}
 662 \qquad \qquad \qquad 0110 \ 0110 \ 0010 \\
 + \underline{1045} \quad + \underline{0001 \ 0000 \ 0100 \ 0101} \\
 1707 \qquad \underline{0001 \ 0110 \ 1010 \ 0111} \\
 \qquad \qquad \underline{0001} \overline{0110} \\
 \qquad \qquad \underline{0111} \ \underline{10000} \\
 \qquad \underline{1} \quad \underline{7} \quad \underline{0} \quad \underline{7}
 \end{array}$$

3) $234 - 152 =$

$$\begin{array}{r}
 234 \quad 234 \quad C'0001 = \boxed{0001 + 0110 = 0111} = \boxed{1000} = 8_{(10)} \\
 - \underline{152} \quad + \underline{847} \quad C'0101 = \boxed{0101 + 0110 = 1011} = \boxed{0100} = 4_{(10)} \\
 082 \quad 1 \ 081 \quad C'0010 = \boxed{0010 + 0110 = 1000} = \boxed{0111} = 7_{(10)} \\
 + \quad \underline{1} \\
 \underline{1} \overline{082} \\
 \qquad \qquad \qquad 0010 \ 0011 \ 0100 \\
 + \underline{1000 \ 0100 \ 0111} \\
 1010 \ 0111 \ 1011 \\
 + \underline{0110} \qquad \underline{0110} \\
 1 \ 0000 \ \underline{0001} \overline{10001} \\
 1000 \\
 \qquad \qquad \qquad + \underline{0001} \\
 \underline{1} \overline{0000 \ 1000 \ 0010} \\
 \underline{1} \overline{0 \ 8 \ 2}
 \end{array}$$

$$4) \quad 623 - 54 =$$

$$\begin{array}{r}
 623 \\
 - 54 \\
 \hline
 569
 \end{array}
 \quad
 \begin{array}{r}
 623 \\
 945 \\
 + 1 \\
 \hline
 1568
 \end{array}$$

$$\begin{array}{r}
 0110 \ 0010 \ 0011 \\
 + 1001 \ 0100 \ 0101 \\
 \hline
 1111 \ 0110 \ 1000 \\
 + 0110 \quad + 0001 \\
 \hline
 1 \overline{)0101} \quad \overline{)0101} \quad \overline{)1001} \\
 \quad \quad \quad 5 \quad \quad \quad 6 \quad \quad \quad 9
 \end{array}$$

4.4 Resumen

La denominación de “número de punto o coma fija” obedece a que se trata de números enteros. Por lo tanto, la “coma separadora de decimales” se supone alojada en la posición derecha extrema del dígito de orden inferior. Son ejemplos de ello los números $2,0_{(10)}$ o $35,0_{(10)}$ o $234\,768,0_{(10)}$. En cualquiera de estos casos “la parte decimal” es 0, por lo que se elimina junto con la coma “que queda” (en el supuesto lugar “fija”. En representación binaria de coma fija, un entero sumado o restado de otro entero da un número de igual cantidad de bits que los operandos. En el caso de la multiplicación, se debe considerar que si los operandos son de n bits, el resultado puede llegar a ocupar $2 \cdot n$, o sea que se expande al doble. En el caso de la división de dos enteros de n bits, el cociente se considerará otro entero de n bits y el resto, otro también de n bits.

En el mundo, la mayoría de la gente utiliza aritmética decimal (base 10) y, cuando los valores son muy grandes o muy pequeños, se emplean los exponentes en potencias de diez, mientras que los números en coma flotante (base 2) sólo pueden aproximar números decimales comunes. En aplicaciones financieras o comerciales, donde no se admiten errores cuyo origen sea la conversión entre sistemas, caso presentado en el número $0,2_{(10)}$ aproximadamente igual a $0,0011_{(2)}$, se deberán utilizar lenguajes que permitan declarar variables “tipo decimal”. Los cálculos sobre este tipo de datos se operan usando la aritmética decimal, que es más eficiente también con respecto a la precisión, ya que para los reales utiliza un entero como mantisa y un entero que representa una potencia de diez como exponente.

En Java la “Clase BigDecimal” representaría el número $8.9999996E-10$ con esta precisión, mientras que el mismo número para el tipo de dato “Java float” se representaría $9E-10$. Es preocupación del diseñador de la aplicación ver si la diferencia es significativa o no para la tarea en cuestión.

Como contrapartida, los números decimales utilizan 20% más de almacenamiento que la representación en binario puro. Recuérdese que el número $255_{(10)}$ en formato de 8 bits en representación de coma fija sin signo es “11111111” (se almacena en un byte), mientras que el mismo número en BCD desempaquetado ocupa tres bytes: el primero “0000 0010” para el 2, el segundo “0000 0101” para el primer 5 y el tercero “0000 0101” para el último 5.

Los cálculos en decimal requieren 15% más de “circuitería”, debido a las correcciones que surgen de operar “en binario pero en base diez”, y por esta razón son más lentos. Recuérdese que la suma de “9+1” en BCD requiere la corrección con el número $6_{(10)}$ o “0110”, pues el código BCD está definido sólo hasta el dígito decimal 9.

4.5 Ejercicios propuestos

1) Realizar en representación de punto fijo, tal como lo haría la ALU, las siguientes sumas para un formato de 6 bits. Verificar el resultado en decimal, teniendo en cuenta que los negativos están representados en la forma de complemento a 2.

- a) $001110_{(2)} + 110010_{(2)}$
- b) $010101_{(2)} + 000011_{(2)}$
- c) $111001_{(2)} + 001010_{(2)}$
- d) $101011_{(2)} + 111000_{(2)}$

2) Realizar las siguientes restas en representación de punto fijo, tal como lo haría la ALU, para un formato de 6 bits y utilizar negativos complementados a la base. Verificar el resultado en decimal.

- a) $010101_{(2)} - 000111_{(2)}$
- b) $001010_{(2)} - 111001_{(2)}$
- c) $111001_{(2)} - 001010_{(2)}$
- d) $101011_{(2)} - 100110_{(2)}$

3) Efectuar la siguiente operación como lo haría la ALU en representación de punto fijo con formato de 8 bits y negativos complementados a la base. Indicar el valor final de los flags S, Z, V, C y qué representa cada uno de ellos.

$(-85)_{(10)} - (+53)_{(10)}$

4) Considerar que los siguientes datos hexadecimales están representando operandos binarios enteros signados. Realizar su suma. Comprobar en decimal.

$20; F9$

5) Dadas las siguientes operaciones en decimal, efectuarlas como lo haría la ALU, en un formato de 8 bits, con negativos complementados a la base. Indique el valor del resultado y el de los flags S, Z, V y C. Compruebe el resultado obtenido convirtiéndolo a decimal.

- a) $101 + 29$
- b) $101 + (-29)$
- c) $(-101) + 29$
- d) $(-101) + (-29)$
- e) $101 - 29$
- f) $101 - (-29)$
- g) $(-101) - 29$
- h) $(-101) - (-29)$

6) Sea la siguiente sentencia de un lenguaje de alto nivel (llamado x):

$C = A - B$

- Indicar un código que la represente en el programa fuente y cuántos octetos ocuparía en memoria.
- Si $A = +3_{(10)}$ y $B = -54_{(10)}$ indicar cómo llegarían los datos a la ALU si fueron definidos como binarios enteros signados.
- Operar en formato de 16 bits, en complemento a la base, e indicar el estado de los flags S, Z, V, C.
- Indicar los valores a partir de los cuales hay *overflow* para enteros naturales y para enteros signados en C_2 , según el formato especificado en el párrafo anterior.

7) Suponga que desea realizar la siguiente operación $C = A + B$ en binario de punto fijo en complemento a la base, como lo haría una ALU de 16 bits, siendo:

$A = +102_{(10)}$

$B = -48_{(10)}$

- ¿Cuál es el resultado de la operación?
- ¿Cuál es el estado de los flags?
- Realice la comprobación en decimal del resultado.
- Exprese el resultado en binario de punto flotante, con un exceso igual a $2^p - 1$, siendo $m = 32$ y $p = 8$ bits, y con mantisa fraccionaria normalizada.

8) Realizar la siguiente operación en BCD puro y comprobar en decimal:

$2400_{(10)} + 1625_{(10)}$

9) Realizar las siguientes sumas en decimal codificado en binario y comprobar en decimal:

$23 + 4$
 $680 + 1027$

10) Realizar las siguientes restas en BCD y comprobar en decimal:

a) $234 - 152$
b) $725 - 435$

11) Multiplicación en complemento a dos. Algoritmo de Booth

El algoritmo de Booth utiliza cuatro elementos de almacenamiento. Éstos pueden pensarse como variables o registros de cálculo y se denominan A para contener el multiplicando, M para contener el multiplicador y Q como variable o registro auxiliar (que además de contener la misma cantidad de bits que A y M, utiliza un bit más al que identificaremos como Q-1, dado que se relaciona con Q0 en los desplazamientos). Para el ejemplo utilizaremos un formato de 4 bits, donde el bit de orden superior es siempre el signo del operando y los números

para multiplicar serán -4 y +2. El resultado esperado es -8 y, obviamente, estará complementado por ser negativo; se debe considerar que el resultado de la multiplicación queda alojado en A y Q. A continuación, se describen los pasos del algoritmo.

- 1. Se inicializan A y Q-1 con 0 y una variable denominada cuenta con 4, que es la cantidad de bits del formato.
- 2. Se carga Q con el multiplicando y M con el multiplicador.
- 3. Se compara Q0 bit menos significativo de Q y Q-1.
- 4. Si son iguales a "0" "1", se suma A con M y se almacena en A; luego se salta al paso 7.
- 5. Si son iguales a "1" "0", se resta M de A y se almacena en A. Cuidado, recuerde que se debe realizar la resta con el complemento a la base de M. Luego se salta al paso 7.
- 6. Si los pasos 4 y 5 fueron falsos, significa que el resultado de la comparación fue "1" "1" o "0" "0", en cuyo caso se salta en forma directa al paso 7.

- 7. Se desplazan a la derecha los bits del grupo A, Q y Q-1, manteniendo el signo en A3, y se pierde el valor actual de Q-1 a causa del desplazamiento. Esto se conoce como desplazamiento aritmético a derecha (SAR o *Shift Arithmetic Right*), dado que se conserva el signo.
- 8. Se decrementa cuenta en una unidad.
- 9. Si cuenta no llegó a 0, se vuelve al paso 3, caso contrario se termina la secuencia de pasos, ya que se logró el objetivo Se pide:
 - A. Realizar un diagrama de flujo que represente los pasos enunciados.
 - B. Indicar en cada paso cuál es el verbo *assembler* 80x86 que se utilizaría si se deseara programar el algoritmo.
 - C. Observar la tabla descripta a continuación para comprender el funcionamiento del algoritmo.
 - D. Realizar una tabla idéntica para los operandos -3 y +2.

Tabla de secuencia de una multiplicación con el algoritmo de Booth.							
	Cuenta	A	Q	Q ₋₁	M		
1 y 2	4	0000	1100	0	0010	Inicialización de los registros o las variables	
3						Comparar Q0 y Q-1	
7 y 8	3	0000	0110	0	0010	Sólo desplazamiento	Primer ciclo
7 y 8	2	0000	0011	0	0010	Sólo desplazamiento	Segundo ciclo
5		1110	0011	0	0010	A A-M	Tercer ciclo
7 y 8	1	1111	0001	1	0010	Desplazar A,Q,Q-1	
7 y 8	0	1111	1000	1	0010	Sólo desplazamiento	Cuarto ciclo
9		1111	1000	1	0010	Fin del algoritmo resultado 11111000 igual -8	

4.6 Contenido de la página Web de apoyo



El material marcado con asterisco (*) sólo está disponible para docentes.

Resumen gráfico del capítulo

Simulación

Resuelve el algoritmo de Booth paso a paso

Autoevaluación

Video explicativo (02:13 minutos aprox.)

Audio explicativo (02:13 minutos aprox.)

Evaluaciones Propuestas*

Presentaciones*