Organización del Computador
Representaciones de punto fijo

B. Gonzalez Kriegel - N. Wolovick

12 de marzo de 2001

1 Representaciones para $\mathbb{N}$

Los desarrollos s-adicos nos permiten expresar un número natural como suma finita de potencias de una base.

Teorema 1 (desarrollos s-adicos) *Para todo número $a \in \mathbb{N}$, para toda base $b > 1$, existen únicos $n, a_{n-1} \dots a_0$ con $0 \leq a_i \leq b$ y $0 < a_{n-1}$ tal que*

$$a = \sum_{i=0}^{n-1} a_i b^i$$

Resulta sencillo obtener los coeficientes a_i respecto a la base b de un número utilizando este resultado y el algoritmo de la división.

Ejemplo 1 *Queremos obtener los a_i del número 20 para $b = 2$ (binario). Por desarrollos s-adicos sabemos que $20 = \sum_{i=0}^{n-1} a_i 2^i$ luego por el algoritmo de la división tenemos $20 = 2(\sum_{i=0}^{n-2} a_{i+1} 2^i) + a_0$ con lo cual escribimos $20 = 2 \cdot 10 + 0$, obteniendo asi a_0=0. Continuamos el proceso para obtener los siguientes a_i:*

$$\begin{array}{rcl} 10 & = & 2 \cdot 5 + 0 \\ 5 & = & 2 \cdot 2 + 1 \\ 2 & = & 1 \cdot 2 + 0 \\ 1 & = & 0 \cdot 2 + 1 \end{array}$$

luego $20 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$.

Notación 1 *Denotamos $a = \sum_{i=0}^{n-1} a_i b^i$ como $(a_{n-1} \dots a_0)_b$.*

Vemos que:

- la representación no es única, por ejemplo $(10100)_2 = (010100)_2 = (0 \dots 010100)_2$.
- para $b = 10$ la representación coincide con la que normalmente usamos $123 = (123)_{10}$
- toda transformación de base es posible hacerla pasando por la base decimal.
- cuando $b > 10$ utilizamos letras mayúsculas además de los símbolos numéricos. $(FA)_{16} = 15 \cdot 16^1 + 10 \cdot 16^0 = (240)_{10}$
- existen trucos para convertir de manera directa entre bases "parientes".

2 Representaciones para $\mathbb{Z}$

Ahora nos concentraremos en representaciones binarias para $\mathbb{Z}$ Vamos a ver las 4 soluciones más comunes a este problema.

2.1 Magnitud con Signo - *MGS*

En la representación el bit más significativo (*MSB*) indica el signo, 1 para negativo y 0 para positivo, mientras que los bits restantes son la magnitud. La expresión que relaciona la *representación* con el número *representado* es la siguiente:

$$(a_{n-1}\dots a_0)_{MGS} = (-1)^{a_{n-1}}\sum_{i=0}^{n-2} a_i 2^i$$

Aunque muy sencilla, la *MGS* tiene el problema de duplicar la representación del 0, además de tener formas complicadas (*case analisis*) para realizar sumas de enteros.

2.2 Complemento radical disminuído - *CPL₁*

Una forma alternativa a la magnitud con signo es el complemento a 1's, donde el bit de mayor peso indica el signo, y los restantes la magnitud los cuales estarán complementados si $a_{n-1} = 1$.

Si la representación es de un positivo

$$(0a_{n-2}\dots a_0)_{CPL_1} = \sum_{i=0}^{n-2} a_i 2^i$$

en caso de un negativo

$$(1a_{n-2}\dots a_0)_{CPL_1} = -\sum_{i=0}^{n-2} (1-a_i) 2^i$$

reescribiendo tenemos

$$(0a_{n-2}\dots a_0)_{CPL_1} = \sum_{i=0}^{n-2} (a_i - 0) 2^i \qquad (1a_{n-2}\dots a_0)_{CPL_1} = \sum_{i=0}^{n-2} (a_i - 1) 2^i$$

entonces

$$(a_{n-1}a_{n-2}\dots a_0)_{CPL_1} = \sum_{i=0}^{n-2} (a_i - a_{n-1}) 2^i$$

Nuevamente tenemos dos representaciones para el 0, pero ahora las operaciones de suma y resta se realizan mediante el algoritmo tradicional de la suma, salvo el problema del ***end around carry***.

2.3 Complemento radical - *CPL₂*

Una computadora real puede manejar una cantidad finita de bits, este hecho produce que toda la aritmética entera sea módulo 2^n con n igual a la cantidad de bits que maneja. Sea $-2^{n-1} \le a < 0$ el número negativo a representar, por lo anterior tenemos $2^{n-1} \le 2^n + a < 2^n$, luego

$$\textit{representacion} = 2^n + \textit{representado}$$

Supongamos $-2^{n-1} \leq a < 0$, entoces veamos como calcular su representación

$$a = \textit{representado} = \textit{representacion} - 2^n = \sum_{i=0}^{n-1} a_i 2^i - 2^n$$

dado que la representación está en el intervalo $[2^{n-1}, 2^n)$ tenemos $a_{n-1} = 1$, luego

$$\textit{representado} = -2^n + 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i = -2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

Para números positivos $0 \leq a < 2^{n-1}$ utilizamos su representación binaria

$$\textit{representado} = \sum_{i=0}^{n-2} a_i 2^i$$

juntando los casos negativos y no-negativos obtenemos:

$$(a_{n-1} \dots a_0)_{CPL_2} = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

Esta es la representación más usada pues por un lado aprovecha todo el rango y el procedimiento de suma tradicional funciona para todas las combinaciones de representaciones positivas y negativas, el único problema reside en la multiplicación signada, pero se soluciona mediante el ***método de Booth***.

2.4 Exceso - $EX - e$

Si al número a representar le sumamos un *exceso* o *sesgo* e, trasladamos la porción de números negativos $[-e, k]$ a $[0, k + e]$ para luego representarlo en binario.

$$(a_{n-1} \dots a_0)_{EX-e} = \sum_{i=0}^{n-1} a_i 2^i - e$$

Esta traslación es una solución sencilla al problema de representar enteros en binario y resulta muy utilizada en los formatos de punto flotante.

2.5 Ejercicios

1. Obtenga una tabla de *representación* vs. *representado* para números binarios de 4 bits con respecto a todas las representaciones vistas.

2. Demuestre que $(a_{n-1} \dots a_0)_{CPL_1} = -a_{n-1}(2^{n-1} - 1) + \sum_{i=0}^{n-2} a_i 2^i$. Compare con la expresión para el $CPL2$ y obtenga conclusiones respecto a los nombres.

Referencias

[1] Computer Science and Engineering Handbook, Chp.19

[2] Computation Structures, Stephen Ward, Robert Halstead Jr., Chp.2