División Rápida SRT

Aunque la técnica de correr sobre ceros no se usa mayormente en la actualidad resulta instructivo considerar su posible aplicación en el proceso de división. El tiempo de ejecución resulta dependiente de los operandos, y se puede decir en tal sentido que la ventaja no es mucha y más aun no resulta conveniente la idea de tiempos variables, tanto a nivel del control de hardware como de los propios compiladores optimizantes. En la multiplicación correr sobre 0´s (*shift*) implica que cuando se detecta que el bit del multiplicador es 0 solo se corre, sin sumar y más aun, con unidades de corrimiento variable se podrá detectar cadenas de 0´s y saltar (*skip*) sobre los 0´s reduciendo el número de iteraciones efectivo.

Ahora bien, qué respecto a corrimientos y saltos sobre los 0´s del cociente con la división. Se vio para la división sin restoring que en cada iteración siempre se suma o se resta, independientemente del valor que adoptase el bit del cociente en ese paso. No parece haber oportunidad para saltear dicha operación. Lograrlo demanda un algoritmo de división que se aparte del método convencional de prueba y error aplicado en la división.

Esto es lo que proporciona el algoritmo **SRT**, desarrollado simultánea aunque "independientemente" por Sweeney, Robertson, y Tocher. Para computar **a**/**b** substraemos múltiplos de **b** a **a**, para luego establecer cuantas substracciones fueron realizadas. De forma convencional aunque el dividendo fuese un pequeño positivo uno resta **b** (para la prueba) pero supóngase en cambio que solo corremos y la resta la dejamos para el próximo paso. En la medida de que el dividendo parcial resultante tenga un tamaño que así lo permita no habría inconvenientes, vale decir que pueda ser representado con los dígitos del cociente restantes (menos significativos). Esto va a requerir un cambio en la forma de computar el número de veces que **b** se ha restado.

Haciendo uso de notación redundante dígito signado para el cociente veremos como esto es posible, y es precisamente el algoritmo **SRT**. Veamos primero el algoritmo con base 2, un ejemplo de aplicación, y finalmente su fundamentación a nivel teórico.

Sea el cociente entre dos números no signados: **a** dividendo y **b** divisor. Determinar Q = **a** / **b** y resto r. Hacemos MQ ← **a**; B ← **b**; A ← 0

1. Si B tiene k 0´s adelante desplazamos ambos, A.MQ y B, k lugares a izquierda (dividendo y divisor).
2. Para i = 0 hasta n-1
   a) Si los tres bits más significativos de A son iguales se hace $q_i = 0$ y se corre A.MQ una posición a izquierda.
   b) Si los tres bits más significativos de A son distintos y A es negativo se hace $q_i = -1$, corremos A.MQ un lugar a izquierda y le sumamos B.
   c) En caso contrario, A.MQ positivo, se hace $q_i = 1$, corremos A.MQ un lugar a izquierda y le restamos B.
3. Si el resto final es negativo, corregimos sumando B y además corregimos el cociente restándole 1. Luego para el resto r este dividendo deberá desplazarse k lugares a derecha y finalmente se deberá convertir a binario el cociente, restándole a la parte positiva la negativa.

Veámoslo en un ejemplo trabajando con n = 4, con **a** = 1000 y **b** = 11 lo que se esquematiza y detalla a continuación.

A	MQ	
00000	1000	
00010	0000	Paso 1, b tiene 2 0´s, corremos 2 lugares
00100	000**0**	Paso 2a, los 3 bits iguales, $q_i = 0$ y corro
01000	00**01**	Paso 2c, los 3 bits primeros distintos y A > 0
10100		
11100	00**01**	Corro, $q_i = +1$ y resto B
11000	0**010**	Paso 2a, los 3 bits iguales, $q_i = 0$ y corro
10000	**0101**	Paso 2b, los 3 bits primeros distintos y A < 0
01100		
11100	**010-1**	Corro, $q_i = -1$ y sumo B
01100		Paso 3, resto negativo, sumo B y Q = Q - 1
01000	**01-10**	Se computa el Q en binario, y corre a
R = 00010	Q = 0010	derecha, se deshace el corrimiento inicial

¿Cual es el fundamento teórico de este algoritmo? Vamos a comenzar el estudio trabajando en base 2. En tal caso los dígitos para el cociente serán -1, 0, ó 1. Debemos reiterar que el algoritmo se basa en decidir de forma anticipada el dígito del cociente, con la sola restricción de que el dividendo resultante para la próxima etapa pueda ser representado, sin pérdida, con los dígitos restantes del cociente.

Así, estando en la iteración i, el dividendo parcial para la siguiente iteración, esto es R_{i+1}, estará acotado dada la representación posicional dígito signado entre los valores extremos dados por los valores mínimos y máximos de los dígitos del cociente:

$$(0.\text{min min min}\ldots\ldots.)B \leq R_{i+1} \leq (0.\text{max max max}\ldots\ldots.)B \qquad \textbf{(I)}$$

Como se vio, el min = -1 y el max = 1, por lo cual la ecuación anterior queda:

$$(0.\text{-1-1-1}\ldots\ldots..\text{-1})B \leq R_{i+1} \leq (0{,}111\ldots\ldots..1)B$$

Podemos reescribirla como:

$$-B/2(1{,}111\ldots\ldots.1) \leq R_{i+1} \leq B/2(1{,}111\ldots\ldots.1)$$

Recordando que dada la serie $x^0 + x^1 + x^2 + \ldots + x^n$, con n □□, converge a $1/(1 - x)$ para todo $x < 1$, en este caso $x = 1/2$, y que para un dividendo parcial negativo resultará posible su corrección mediante la suma de B en la posición menos significativa, con lo cual extremo negativo es el que se puede codificar con el min para cada dígito del cociente más $-(B)(1/2)^n$, resultará luego el rango para el próximo dividendo parcial:

$$-B \leq R_{i+1} < B \quad \textbf{(II)}$$

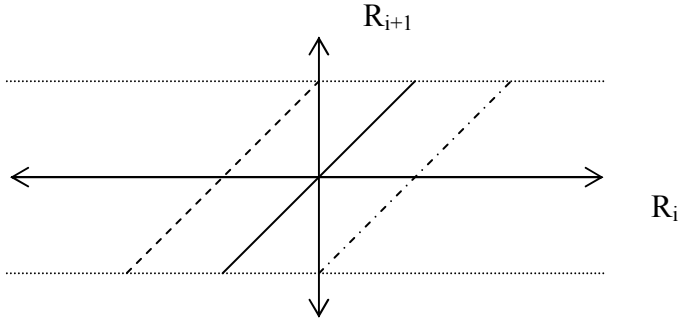
Adoptado un dígito del cociente q_{i+1} y dado un dividendo parcial R_i, a partir de **(II)** que fija el rango para el próximo dividendo R_{i+1} se tendrá:

$$-B \leq (2R_i - q_{i+1}B) < B \textbf{ (III)}$$

Claramente se está computando el dividendo parcial del próximo paso, que no es otra cosa que el actual dividendo parcial desplazado un lugar, por la menor significancia para la siguiente posición, al que se le restará B, se le sumará B, o se deja como está, en función del valor que asuma q_{i+1}, esto es 1, -1, o 0 respectivamente.

Para la primer iteración, i = 0, con el registro A.MQ desplazado un lugar a izquierda, estaremos computando el q_{n-1} (recordar el garbage que se introducía al comienzo de la división para comenzar con el cómputo del q_{n-1}). Así seguiremos avanzando con sucesivos corrimientos hasta arribar a definir el q_0 que permite computar el último dividendo. En caso de ser negativo este dividendo indica que la elección fue incorrecta, y análogamente a la división sin restoring deberá corregirse sumándole B para un resto mayor que 0. Recordar en tal sentido que los dividendos parciales están acotados, según la ecuación **(II)**, en un intervalo [-B,B). Además, y aquí se diferencia con la división sin restoring, se deberá corregir el cociente dado que las decisiones son en avance, el bit del cociente se fija de forma adelantada, Q = Q – 1, y además convertir Q a binario.

Una vez que hemos visto la teoría en la que se sustenta el algoritmo, vamos a analizar que propone el algoritmo enunciado más arriba. Para ello grafiquemos de forma parametrizada la función **(III)**, con q_{i+1} asumiendo los tres valores posibles, -1, 0, 1:



El objetivo de la división **SRT** es aprovechar, toda vez que se pueda, a correr sobre los 0´s del cociente. Esto es en la medida de que $q_{i+1} = 0$ solo habría que correr para el paso del algoritmo que computa R_{i+1}. Como se deduce de **(III)**, y queda ilustrado, se puede asumir el valor $q_{i+1} = 0$ para un rango del dividendo $-B/2 \leq R_i < B/2$.

Asumamos, sin pérdida de generalidad B fraccionario, el punto a la izquierda. Luego del eventual corrimiento del primer paso podemos asegurar que $B \geq 1/2$. La decisión de adoptar $q_{i+1} = 0$ para el caso de los tres bits más significativos iguales se traduce en un rango para el dividendo $1.11xx \leq R_i \leq 0.00xx$, es decir $-1/4 \leq R_i < 1/4$. Dado que el menor valor de B vimos es 1/2 de la anterior se deduce $-B/2 \leq R_i < B/2$ que es lo que se demostró debía verificar dividendo R_i para admitir $q_{i+1} = 0$.

Los casos de los tres bits más significativos distintos se resuelven, como se desprende del diagrama, con $q_{i+1} = -1$ en caso de dividendo negativo $R_i < -1/4$, o con $q_{i+1} = 1$ en caso de dividendo positivo $R_i \geq 1/4$.

Más aún, considerando estos valores extremos para el dividendo con $q_{i+1} = 0$, el dividendo $R_{i+1} = R_i - q_{i+1}$ para el próximo paso se encontrará en el rango [-1/2 , 1/2) esto es en notación complemento el rango de valores es [1.1xxx , 0.0xxx] cualquiera sean los valores que asuma x, de lo que se infiere que el dígito de signo es redundante. Luego será suficiente analizar los dos primeros bits de la magnitud. De tal forma el hardware requerido no es de n+1 bits, como en el caso de la división convencional, sino de n bits sin el agregado de un bit para signo.

Para bases distintas de 2, por ejemplo base 4, se razona de igual manera en cuanto a que el valor del dígito del cociente que se asuma deberá ser tal que el dividendo parcial resultante del siguiente paso pueda ser representable. Si asumimos para los dígitos signados en base 4

los valores -2, -1, 0, 1, 2 resulta el mínimo = -2 y el máximo = 2, luego según **(I)** el rango del dividendo es

$$(0.-2-2-2 \ldots\ldots -2)B \leq R_{i+1} \square (0.222\ldots\ldots.2)B$$

Con i número de iteración y valores desde 0 hasta n/2 - 1.

Reelaborando la anterior queda

$$-2/4(1.010101\ldots\ldots 01)B \leq R_{i+1} \leq 2/4(1.010101\ldots\ldots 01)B$$

Dado que una serie $x^0 + x^1 + x^2 + \ldots + x^n$ con x =1/4 para n $\square\square$ converge a 1/(1 –x) = 4/3, y que como se ha visto el dividendo negativo se podrá corregir sumándole B al dividendo, el menor R_{i+1} representable es el que se obtiene de restar $(B)(1/2)^n$ al negativo con todos los dígitos del cociente -2, extendiendo por ende el extremo negativo de forma que el rango del dividendo verifica:

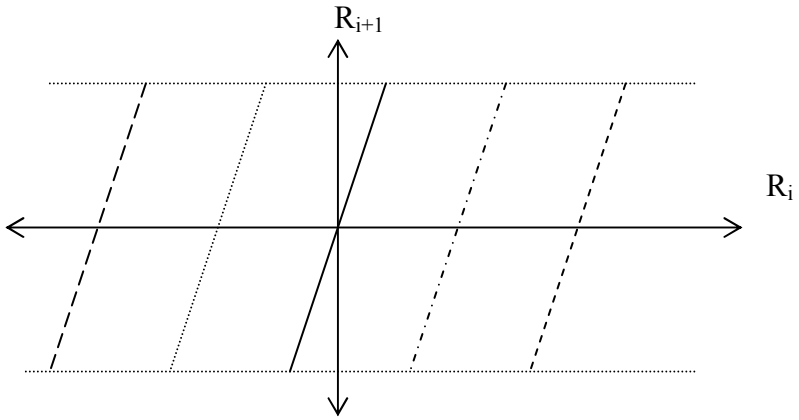
$$-2/3B \leq R_{i+1} < 2/3B$$

La fórmula para computar R_{i+1} trabajando en base = 4 es:

$$R_{i+1} = 4R_i - q_{i+1}B$$

Cada dígito q_{i+1} codifica dos posiciones binarias. Para la primer iteración, i = 0, el valor asumido de q_1 corresponde a las posiciones binarias n-1 y n-2 y así se avanza hasta llegar para la última iteración al dígito del cociente para las posiciones 1 y 0.

Multiplicar el dividendo R_i por 4, desplazándolo dos lugares a izquierda, tiene que ver con que en cada iteración se avanza de a dos posiciones. Esta función se podrá graficar parametrizada en q_{i+1}, con valores -2,-1,0,1,2



En la tabla del apéndice A del libro *"Computer Architecture - A Quantitative Approach"* de Hennessy y Patterson (Fig. A.34) para la determinación del q_{i+1}, en función de R_i y B, se verifica que se accede con los cuatro bit más significativos de B y los 6 más significativos de R_i, resultando esto información suficiente para la búsqueda del q_{i+1}.

Como comentario adicional, los primeros Pentium fabricados fallaban con "ciertas divisiones", esto es para particulares combinaciones de operandos, al punto que Intel terminó retirando, reemplazando, miles de estos micros que fallaban. La razón para esta falla *tan selectiva* es que el origen del error era una mala programación de la tabla, lo que motivaba esa falla que, aunque consistente, se daba en contadas circunstancias.

Es de observar que la acción inicial de desplazar el divisor eliminando los 0´s redundantes solo persigue un interés práctico en la determinación el valor del cociente.

Por último, la respuesta del porqué estudiamos la división SRT la encontraremos en que hace posible emplear bases mayores que 2 en la división, dado que no es el esquema clásico de prueba y error, con la ventaja de menos iteraciones, y además facilita el empleo de CSA. Respecto a esto último podemos apreciar que dado que la determinación del dígito del cociente solo demanda conocer el dividendo parcial con una cierta precisión (vimos 6 bits para b = 4) resulta posible llevar adelante el cómputo de los dividendos parciales mediante los CSA y realizar la verdadera suma solo para los bits superiores que determinan el ingreso a la tabla.