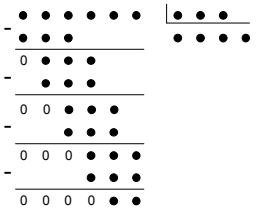
División

Martín Vázquez
Arquitectura I - Curso 2013
UNICEN

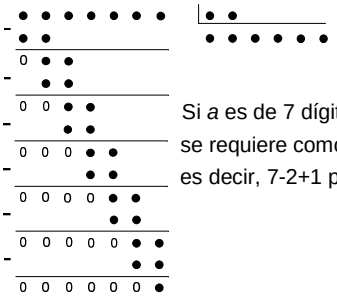
INSTITUTO DE INVESTIGACIÓN EN TECNOLOGÍA INFORMÁTICA AVANZADA

División

Algunos ejemplos de división de números naturales en notación *dot, a=q.b + r,* (*lápiz y papel*)



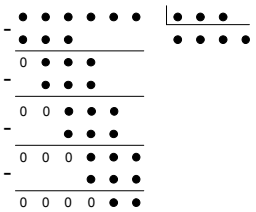
Si *a* es de 6 dígitos y *b* de 3 dígitos,
se requiere como máximo 4 pasos
es decir, 6-3+1 pasos



Si *a* es de 7 dígitos y *b* de 2 dígitos,
se requiere como máximo 6 pasos
es decir, 7-2+1 pasos

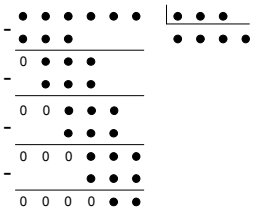
**Generalizando, para *a* de *n* dígitos y *b* de *m* dígitos,
se requieren como máximo *m*-*n*+1 pasos**

División



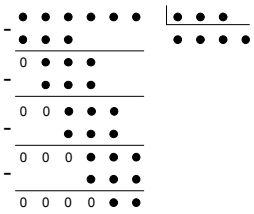
- En el ejemplo empezamos tomando los 3 primeros dígitos del dividendo, no siempre es así

División



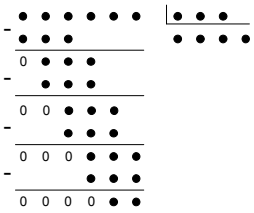
- En el ejemplo empezamos tomando los 3 primeros dígitos del dividendo, no siempre es así
  - Ej: $101001_2/110$, se debe comenzar tomando los 4 primeros bits

División



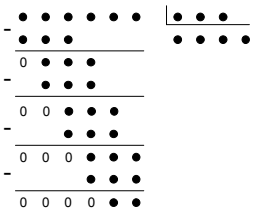
- En el ejemplo empezamos tomando los 3 primeros dígitos del dividendo, no siempre es así
  - Ej: 101001_2/110, se debe comenzar tomando los 4 primeros bits
- Si el divisor es 110_2, se requieren 4 pasos, si el divisor fuera 010_2, se requieren 5 pasos. Depende de los 0's iniciales del divisor

División



- En el ejemplo empezamos tomando los 3 primeros dígitos del dividendo, no siempre es así
  - Ej: 101001_2/110, se debe comenzar tomando los 4 primeros bits
- Si el divisor es 110_2, se requieren 4 pasos, si el divisor fuera 010_2, se requieren 5 pasos. Depende de los 0's iniciales del divisor
- También los 0's iniciales en el dividendo alteran la cantidad de pasos del algoritmo

División

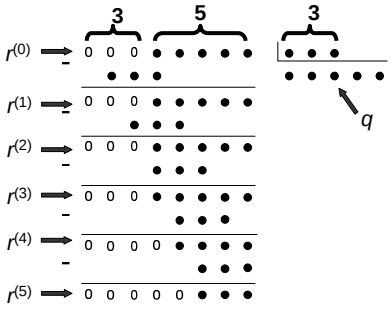


- En el ejemplo empezamos tomando los 3 primeros dígitos del dividendo, no siempre es así
  - Ej: $101001_2/110$, se debe comenzar tomando los 4 primeros bits
- Si el divisor es 110_2, se requieren 4 pasos, si el divisor fuera 010_2, se requieren 5 pasos. Depende de los 0's iniciales del divisor
- También los 0's iniciales en el dividendo alteran la cantidad de pasos del algoritmo
- Es muy dependiente de la cantidad de dígitos utilizados para almacenar o representar los operandos

División

Estrategia para abordar la división de números naturales en base ***B***, $a=q.b + r$,

- Ejemplo, con *a* de 6 dígitos y *b* de 3 dígitos



- Sea *a* de *n* dígitos y *b* de *m* dígitos
  - En cada paso se determina un resto parcial $r^{(i)}$ de *n*+*m* dígitos
  - Se requieren *n* pasos para efectuar la división completa
  - q_i en {0,1, 2, . . ,***B***-1}

División con restauración

- Ejemplo en base 10,
  - $q = a/b$, con $a = 3425_{10}$ y $b = 234_{10}$

```
0 0 0 3 4 2 5 | 2 3 4
  2 3 4       | 0 0 1 4
0 0 0 3 4 2 5
    2 3 4
0 0 0 3 4 2 5
      2 3 4
0 0 0 1 0 8 5
        9 3 6
          1 4 9
```

3425 = 234.14 + 149

División con restauración

- Ejemplo en base 10,
  - $q = a/b$, con $a = 3425_{10}$ y $b = 234_{10}$

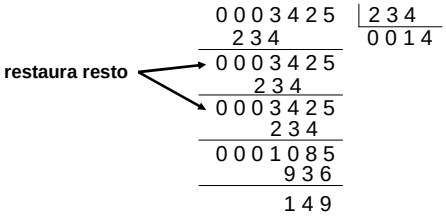
```
0 0 0 3 4 2 5 | 2 3 4
  2 3 4       | 0 0 1 4
0 0 0 3 4 2 5
    2 3 4
0 0 0 3 4 2 5
      2 3 4
0 0 0 1 0 8 5
        9 3 6
          1 4 9
```

restaura resto

3425 = 234.14 + 149

División con restauración

- Ejemplo en base 10,
  - $q = a/b$, con $a = 3425_{10}$ y $b = 234_{10}$



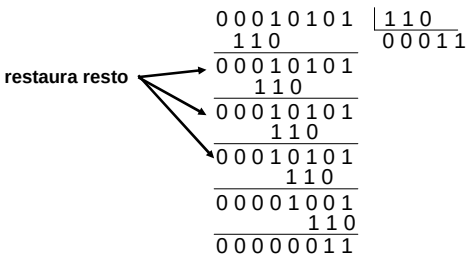
3425 = 234.14 + 149

Inicializa $r^{(0)} = a$
En cada paso computa $r^{(i)} = r^{(i-1)} - q_i.b$
queda $r = r^{(n)}$, $0 \le q_i < 10$

Algoritmo clásico de división con restauración

División con restauración

- Ejemplo en base 2,
  - $q = a/b$, con $a = 10101_2$ y $b = 110_2$



$10101_2 = 110_2.11_2 + 11_2$

Inicializa $r^{(0)} = a$
En cada paso computa $r^{(i)} = r^{(i-1)} - q_i.b$
queda $r = r^{(n)}$, $0 \le q_i < 2$

Algoritmo clásico de división con restauración

División con restauración

Con esta estrategia se puede perfectamente agregar fraccionarios al resultado

- Ejemplos,
  - $q = a/b$ forma $XXXX.YY$, con $a = 3425_{10}$ y $b = 234_{10}$
  - $q = a/b$ forma $XXXXX.Y$, con $a = 10101_2$ y $b = 110_2$

```
0 0 0 3 4 2 5 0 0 | 2 3 4
  2 3 4           | 0 0 1 4, 6 3
---------
0 0 0 3 4 2 5
    2 3 4
---------
0 0 0 3 4 2 5
      2 3 4
---------
0 0 0 1 0 8 5
        9 3 6
---------
      1 4 9 0 0
      1 4 0 4
---------
          8 6 0
          7 0 2
---------
          1 5 8
```

3425 = 234.14,63 + 1,58

```
0 0 0 1 0 1 0 1 0 | 1 1 0
  1 1 0           | 0 0 0 1 1,1
-----------
0 0 0 1 0 1 0 1
    1 1 0
-----------
0 0 0 1 0 1 0 1
      1 1 0
-----------
0 0 0 1 0 1 0 1
        1 1 0
-----------
0 0 0 0 1 0 0 1
          1 1 0
-----------
0 0 0 0 0 0 1 1 0
            1 1 0
-----------
            0 0
```

$10101_2 = 110_2.11,1_2$

División con restauración

Entre números naturales y resultado con fraccionarios (base 2)

- Sea $q = a/b$ forma $X..X,Y..Y$, con a de n bits, b de m bits, y k bits de parte fraccionaria, entonces
  - tamaño de restos parciales $r^{(i)}$ es de $m+n+k$ bits
  - $r^{(0)}$ se inicializa con $0..0\&a\&0..0$ (concatenación de m 0's, a y k 0's al final)
  - La cantidad de pasos requerida para hacer la diivisión completa es de $n+k$ pasos

División con restauración

Entre números naturales y resultado con fraccionarios (base 2)

- Ejemplo,
  - $q = a/b$, con $a = 111_2$, $b = 11_2$ y 2 bits de parte fraccionaria en la solución (q tiene forma $XXX.YY$)

$r^{(0)} = 0011100_2$
$r^{(1)} = r^{(0)} - b.2^{n+k-1=4} = 0011100_2 - 110000_2 < 0$ restaura, $r^{(1)} = r^{(0)} = 0011100_2$, $q = 0\_\_\_\_$
$r^{(2)} = r^{(1)} - b.2^3 = 0011100_2 - 11000_2 = 0000100_2$ $r^{(2)} = 0000100_2$, $q = 01\_\_\_$
$r^{(3)} = r^{(2)} - b.2^2 = 0000100_2 - 1100_2 < 0$ restaura, $r^{(3)} = 0000100_2$, $q = 010\_\_$
$r^{(4)} = r^{(3)} - b.2 = 0000100_2 - 110_2 < 0$ restaura, $r^{(4)} = 0000100_2$, $q = 0100\_$
$r^{(5)} = r^{(4)} - b = 0000100_2 - 11_2 = 0000001_2$ $r^{(5)} = 0000001_2$, $q = 01001_2$

$111_2 = 11_2.10{,}01_2 + 0{,}01$

División con restauración

Entre números naturales y resultado con fraccionarios (base 2)

- Ejemplo,
  - $q = a/b$, con $a = 111_2$, $b = 11_2$ y 2 bits de parte fraccionaria en la solución (q tiene forma $XXX.YY$)

$r^{(0)} = 0011100_2$
$r^{(1)} = r^{(0)} - b.2^{n+k-1=4} = 0011100_2 - 110000_2 < 0$ restaura, $r^{(1)} = r^{(0)} = 0011100_2$, $q = 0\_\_\_\_$
$r^{(2)} = r^{(1)} - b.2^3 = 0011100_2 - 11000_2 = 0000100_2$ $r^{(2)} = 0000100_2$, $q = 01\_\_\_$
$r^{(3)} = r^{(2)} - b.2^2 = 0000100_2 - 1100_2 < 0$ restaura, $r^{(3)} = 0000100_2$, $q = 010\_\_$
$r^{(4)} = r^{(3)} - b.2 = 0000100_2 - 110_2 < 0$ restaura, $r^{(4)} = 0000100_2$, $q = 0100\_$
$r^{(5)} = r^{(4)} - b = 0000100_2 - 11_2 = 0000001_2$ $r^{(5)} = 0000001_2$, $q = 01001_2$

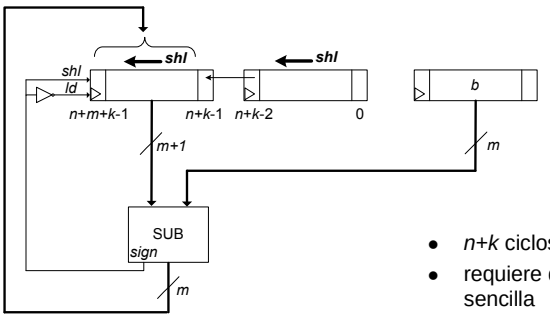
$111_2 = 11_2.10{,}01_2 + 0{,}01$

- El punto está fijo luego de los k bits menos significativos de r y q
- En cada paso, se puede realizar desplazamiento de $r^{(i)}$ y restas del divisor desplazado
  - $r^{(i)} = 2.\ r^{(i-1)} - b.2^4$
- El cociente también se puede generar mediante desplazamientos

División con restauración

Entre números naturales y resultado con fraccionarios basada en desplazamientos (base 2)

Implementación secuencial o serial



- *n*+*k* ciclos de latencia
- requiere de unidad de control sencilla

División con restauración

Normalización de operandos

- En el algoritmo de división anterior los operandos no estaban normalizados
- Cuando los operandos no están normalizados es difícil manejar precisiones específicas
  - si dividendo *n* bits, divisor *m* bits y se desea *l* bits de precisión en resultado con *k* bits de parte fraccionaria
    - tanto el dividendo como el divisor puede empezar con 0's
    - en implementación anterior, el resultado tiene precisión fija de *n*+*k* bits, con *k* bits para fraccionaria.
- El dividendo y divisor, en el algoritmo anterior, están en punto fijo con formato *x*., es decir sin fraccionarios. Muchas veces es deseable que los operandos posean números fraccionarios

División con restauración

Normalización de operandos

- Una estrategia es operar con operandos normalizados en 0,1*X*
  - se "interpreta" a los operandos de la misma manera. Todos poseen el punto en el mismo lugar.
  - es directo operar con mantisas de números representados en punto flotante
  - el proceso de normalización, implica que los exponentes del número tengan un tratamiento especial
    - Ej. $101,10_2 = 0,10110_2.2^{-3}$
  - si dividendo es menor a divisor, entonces el resultado tiene forma 0,1*X*

División con restauración

Algoritmo de división con restauración (*restoring*) y operandos normalizados (base 2)

Considerando

- dividendo *a* y divisor *b* normalizados 0,1*X* ($0,1 \leq a,b < 1$), $a \leq b$ y *k* bits de precisión en el resultado

```
r(0) = a
for i: 1 to k loop
        r(i) = 2.r(i-1) – b
        if r(i) ≥ 0 then
                q(i) = 1
        else
                q(i) = 0
                r(i) = 2.r(i-1)
        end if
end loop
```

Formato de resultado

$$q = 0,q_1q_2 \bullet\bullet q_k$$

División con restauración

Algoritmo de división con restauración (*restoring*) y operandos normalizados (base 2)

Considerando

- dividendo *a* y divisor *b* normalizados 0,1*X* (0,1≤*a*,*b*<1), *a*≤*b* y *k* bits de precisión en el resultado

```
r(0) = a
for i: 1 to k loop
        r(i) = 2.r(i-1) – b
        if r(i) ≥ 0 then
                q(i) = 1
        else
                q(i) = 0
restaura resto ⟶ r(i) = 2.r(i-1)
        end if
end loop
```

Formato de resultado

$$q = 0,q_1q_2 \bullet \bullet q_k$$

División con restauración

Algoritmo de división con restauración (*restoring*) y operandos normalizados (base 2)

Considerando

- dividendo *a* y divisor *b* normalizados 0,1*X* (0,1≤*a*,*b*<1), *a*≤*b* y *k* bits de precisión en el resultado

```
r(0) = a
for i: 1 to k loop
        r(i) = 2.r(i-1) – b
        if r(i) ≥ 0 then
                q(i) = 1
        else
                q(i) = 0
                r(i) = 2.r(i-1)
        end if
end loop
```

Cada paso computa $2.r(i\text{-}1) = b.q(i) + r(i)$,

donde $q(i)$ puede ser 0 o 1

División con restauración

Algoritmo de división con restauración (*restoring*) y operandos normalizados (base 2)

$r(0) = a$

$2.r(0) = b.q(1) - r(1)$

$2.r(1) = b.q(2) - r(2)$

$\vdots$

$2.r(k\text{-}1) = b.q(k) - r(k)$

$$\Rightarrow \quad a = b.\left(\frac{q(1)}{2} + \frac{q(2)}{4} + \frac{q(3)}{8} + \bullet\bullet\bullet + \frac{q(k)}{2^k}\right) + \frac{r(k)}{2^k}$$

División

Normalización de los operandos (base 2)

Sea dividendo *a* y divisor *b*, $1 \le a$ y $1 \le b$;

entonces

- $a' = a.2^{-s}$, con $0{,}1 \le a' < 1$ y $s > 0$
- $b' = b.2^{-f}$, con $0{,}1 \le b' < 1$ y $f > 0$

significa que para hacer *a*/*b*

$$\frac{a}{b} = \frac{a'.2^s}{b'.2^f} = \frac{a'}{b'}.2^{s-f}$$

se realiza *a'*/*b'* y luego se desplaza *s*-*f* lugares, o bien expresar el resultado en términos de mantisa y exponente

División

Normalización de los operandos (base 2)

Sea dividendo *a* y divisor *b*, $1 \le a$ y $1 \le b$;

entonces

- $a' = a.2^{-s}$, con $0,1 \le a' < 1$ y $s > 0$
- $b' = b.2^{-f}$, con $0,1 \le b' < 1$ y $f > 0$

Similar para $a<0,1$ o $b<0,1$ pero con $s<0$ y $f<0$

significa que para hacer *a*/*b*

$$\frac{a}{b} = \frac{a'.2^{s}}{b'.2^{f}} = \frac{a'}{b'}.2^{s-f}$$

se realiza *a'*/*b'* y luego se desplaza *s*-*f* lugares, o bien expresar el resultado en términos de mantisa y exponente

División

Normalización de los operandos (base 2)

Ejemplos

- con dividendo $a = 10011_2$ y $b = 101_2$

  $a' = 10011_2.2^{-5}$ y $b' = 101_2.2^{-3}$

$$\frac{a}{b} = \frac{a'}{b'}.2^{2}$$

- con dividendo $a = 1101_2$ y $b = 1010101_2$

  $a' = 1101_2.2^{-4}$ y $b' = 1010101_2.2^{-7}$

$$\frac{a}{b} = \frac{a'}{b'}.2^{-3}$$

División

Normalización de los operandos (base 2)

Ejemplos

- con dividendo a =0,0011$_2$ y b = 10$_2$

  a' = 0,0011$_2$.2^2 y b´ = 101$_2$.2^{-2}

$$\frac{a}{b} = \frac{a'}{b'}.2^{-4}$$

División

Normalización de los operandos (base 2)

Si luego de normalizar a 0,X el dividendo a' es mayor al divisor b', se debe desplazar y ajustar exponente de a' para cumplir con las hipótesis del algoritmo

$a'' = a'.2^{-1}$

entonces

$$\frac{a}{b} = \frac{a''}{b'}.2^{s-f}.2 = \frac{a''}{b'}.2^{s-f+1}$$

Ejemplo

con dividendo a =1101$_2$ y b = 1010101$_2$

a' = 1101$_2$.2^{-4} y b´ = 1010101$_2$.2^{-7} , pero como a'>b' entonces $a'' = a'.2^{-1}$

$$\frac{a}{b} = \frac{a''}{b'}.2^{-2}$$

División con *restoring*

Ejemplo de división con (*restoring*) y operandos normalizados

Considerar

- dividendo $a = 1001010_2$ y divisor $b = 1000_2$
- 4 bits de precisión en el resultado

entonces

$a' = 1001011_2.2^{-7}$ y $b' = 1000_2.2^{-4}$, pero como $a'>b'$ entonces
$a'' = a'.2^{-1}$

$$\frac{a}{b} = \frac{a''}{b'}.2^4$$

por consiguiente, se realiza la división a''/b', es decir, entre dividendo $0{,}01001011_2$ y divisor $0{,}1000_2$

División con *restoring*

Ejemplo, dividendo $0{,}01001010_2$ y divisor $b' = 0{,}1000_2$

Paso	Operaciones	Comentarios	Salida	Cociente *q*
1)	0,1001010 - 0,1000000 0,0001010	2.*r*(0) -*b'* 2.*r*(0) – *b'* > 0	*r*(1)=2.*r*(0)-*b'* *q*(1)=1	0,1
2)	0,0010100 - 0,1000000 negativo	2.*r*(1) -*b'* 2.*r*(1) – *b'* < 0	*r*(2)=2.*r*(1) *q*(2)=0	0,10
3)	0,0101000 - 0,1000000 negativo	2.*r*(2) -*b'* 2.*r*(2) – *b'* < 0	*r*(3)=2.*r*(2) *q*(3)=0	0,100
4)	0,1010000 - 0,1000000 0,0010000	2.*r*(3) -*b'* 2.*r*(3) – *b'* > 0	*r*(4)=2.*r*(3)-*b'* *q*(1)=1	0,1001

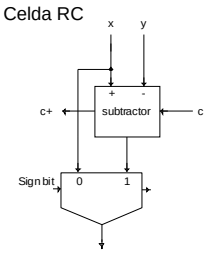
$$\frac{a}{b} = \frac{a''}{b'}.2^4$$

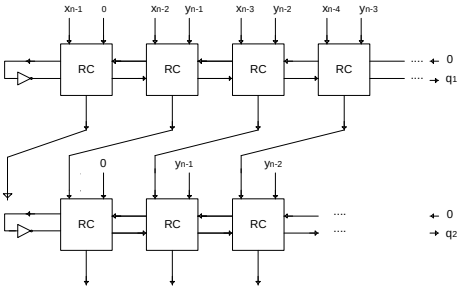
División con *restoring*

División con *restoring* y operandos normalizados

Implementación combinacional o paralela.

Celda RC





División sin restauración

Algoritmo sin restauración (*non-restoring*) y operandos normalizados (base 2)

Considerando

- dividendo *a* y divisor *b* normalizados 0,1*X* (0,1≤*a*,*b*<1), *a*≤*b* y *k* bits de precisión en el resultado

```
r(0) = a
for i: 1 to k loop
        if r(i-1) ≥ 0 then
                r(i) = 2.r(i-1) – b
                if (r(i) ≥ 0) then
                        q(i)=1
                else
                        q(i)=0
                end if;
        else
                r(i) = 2.r(i-1) + b
                if (r(i) ≥ 0) then
                        q(i)=1
                else
                        q(i)=0
                end if;
        end if
end loop
```

Formato de resultado

$$q = 0,q_1q_2 \bullet \bullet q_k$$

División sin restauración

Algoritmo sin restauración (*non-restoring*) y operandos normalizados (base 2)

Considerando

- dividendo *a* y divisor *b* normalizados 0,1*X* (0,1≤*a*,*b*<1), *a*≤*b* y *k* bits de precisión en el resultado

```
r(0) = a
for i: 1 to k loop
        if r(i-1) ≥ 0 then
                r(i) = 2.r(i-1) – b
                if (r(i) ≥ 0) then
                        q(i)=1
                else
                        q(i)=0
                end if;
        else
                r(i) = 2.r(i-1) + b
                if (r(i) ≥ 0) then
                        q(i)=1
                else
                        q(i)=0
                end if;
        end if
end loop
```

Formato de resultado

$q = 0,q_1q_2 \bullet \bullet q_k$

evita restauración de resto

División
restoring y *non-restoring*

En cada paso, ambos algoritmos realizan el mismo cómputo

Restoring

paso *i*

$r(i)=2.r(i-1) – b < 0$, entonces

$r(i)=2.r(i-1)$ y $q(i)=0$

paso *i*+1

$r(i+1) = 2.r(i) – b = 4.r(i-1) – b$

si $r(i+1)>0$ entonces $q(i+1) = 1$
sino $q(i+1) = 0$

Non-restoring

paso *i*

$r(i)=2.r(i-1) – b < 0$, entonces

$q(i)=0$

paso i+1

$r(i+1) = 2.r(i) + b = 4.r(i-1) – b$

si $r(i+1)>0$ entonces $q(i+1) = 1$
sino $q(i+1) = 0$

División *non-restoring*

División *non-restoring* y operandos normalizados

Ejemplo, dividendo $0{,}01001010_2$ y divisor $b' = 0{,}1000_2$

Paso	Operaciones	Comentarios	Salidas	Cociente *q*
1)	00,1001010 +11,1000000 00,0001010	2.*r*(0) -*b*' 2.*r*(0)-*b*'>0	*r*(1)=2.*r*(0)-*b*' *q*(1)=1	0,1
2)	00,0010100 +11,1000000 11,1010100	2.*r*(1) -*b*' 2.*r*(1)-*b*'<0	*r*(2)=2.*r*(1)-*b*' *q*(2)=0	0,10
3)	11,0101000 +00,1000000 11,1101000	2.*r*(2) +*b*' 2.*r*(2)+*b*'<0	*r*(3)=2.*r*(2)+*b*' *q*(3)=0	0,100
4)	11,1010000 +00,1000000 00,0010000	2.*r*(3) +*b*' 2.*r*(2)+*b*'>0	*r*(4)=2.*r*(3)+*b*' *q*(4)=1	0,1001

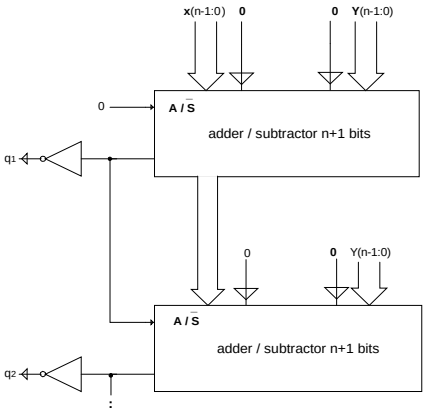
b' = 00,1000000
-*b*' = 11,1000000

$$\frac{a}{b} = \frac{a''}{b'} . 2^4$$

División con *non-restoring*

División *non-restoring* y operandos normalizados

Implementación combinacional o paralela.



División
restoring vs. *non-restoring*

- *Restoring*
  - Trabaja con operandos, resto y cociente positivos
  - No requiere sumador/restador.
    - Se puede restaurar mediante signo de restador y multiplexor
- *Non-restoring*
  - Trabaja con operandos y cociente positivos
  - El resto puede ser positivo o negativo (entero)
  - Requiere sumador/restador
  - Cada paso
    - Menor tiempo de cómputo
    - Menor ocupación de área

División
por método de Newton-Raphson

- Algoritmo de convergencia,
  - iteración funcional
- Basado en técnica de cálculo numérico para resolver ecuaciones. Aproximaciones de los ceros o raíces de una función.
- Posee una convergencia mejor que lineal
- Cada paso es más costoso que algoritmos vistos

División
por método de Newton-Raphson

$D = d.q + r$

Si r (resto) = 0 → $q = D / d = D.1/d$

Newton-Raphson calcula primero el recíproco de d

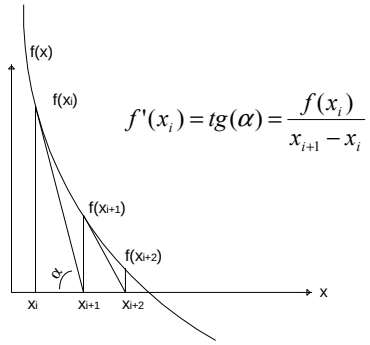
$x = 1/d \rightarrow 1/x = d \rightarrow 1/x - d = 0$

$\rightarrow \boldsymbol{f(x) = 1/x - d = 0}$

Para solucionar $f(x)=0$ se usa la función

$$\boldsymbol{x_{i+1} = x_i - f(x_i) / f'(x_i)}$$

en forma recursiva para evaluar x



$$f'(x_i) = tg(\alpha) = \frac{f(x_i)}{x_{i+1} - x_i}$$

Luego multiplica el recíproco obtenido por D

División
por método de Newton-Raphson

Sea $x0$ ($\neq 0$) la primera aproximación al resultado, entonces

$f(x_0) = 1/x_0 - d$ y $f'(x_0)=[df/dx]_{x=0} = -1/x_0^2$

entonces

$(1/x_0 - d)' = (1/x_0 - d) / (x_0 - x_1)$

$-1/x_0^2 = (1/x_0 - d) / (x_0 - x_1)$

$-((x_0 - x_1) / x_0^2) = 1/x_0 - d$

$x_1 - x_0 = x_0^2 (1/x_0 - d)$

$x_1 = x_0 + x_0 - d.x_0^2$

$x_1 = 2.x_0 - d.x_0^2$

$x_1 = x_0.(2 - d.x_0)$

y

$x_2 = x_1.(2 - d.x_1)$

…

$$\boldsymbol{x_{i+1} = x_i.(2 - d.x_i)} \qquad \textbf{con } 1/B \leq d < 1,\ 1 < 1/d \leq B$$

División
por método de Newton-Raphson

Seleccionando x_0 tal que $1<x_0 \leq B$, se asegura convergencia cuadrática

Algoritmo

$x(0) = LUT(d)$
for i: 0 to k-1 **loop**
    $x(i+1) = x(i).(2-d.x(i))$
end loop

Ejemplo

Sea d = 0,161828 (base 10)

siendo 32 la precisión deseada para 1/d.

Se supone disponible una tabla LUT de 4 dígitos de precisión.

x_0 = LUT (d) = **6,179**

$d \cdot x_0$ = 0.161828 · 6,179 = 0,99993521

x_1 = 6,179 · (2 – 0,99993521) = **6,1794003**

$d \cdot x_1$ = 0,161828 · 6,1794003 = 0,9999999917484

x_2 = 6,1794003 · (2 - 0.9999999917484) = **6,179400350989939**

$d \cdot x_2$ = 0,161828 · 6,179400350989939 = 0,99999999999999848492

x_3 = 6,179400350989939 · (2 – 0,99999999999999848492)
= **6,17940035098993993622858837778 37**

División
por método de Newton-Raphson

Seleccionando x_0 tal que $1<x_0 \leq B$, se asegura convergencia cuadrática

Algoritmo

$x(0) = LUT(d)$
for i: 0 to k-1 **loop**
    $x(i+1) = x(i).(2-d.x(i))$
end loop

Si la precisión requerida es p y la precisión de $x(0)$ (desde LUT) es t

$p = t.2^k$

$k = \log_2 p/t$

Ejemplo

Sea d = 0,161828 (base 10)

siendo 32 la precisión deseada para 1/d.

Se supone disponible una tabla LUT de 4 dígitos de precisión.

x_0 = LUT (d) = **6,179**

$d \cdot x_0$ = 0.161828 · 6,179 = 0,99993521

x_1 = 6,179 · (2 – 0,99993521) = **6,1794003**

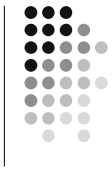
$d \cdot x_1$ = 0,161828 · 6,1794003 = 0,9999999917484

x_2 = 6,1794003 · (2 - 0.9999999917484) = **6,179400350989939**

$d \cdot x_2$ = 0,161828 · 6,179400350989939 = 0,99999999999999848492

x_3 = 6,179400350989939 · (2 – 0,99999999999999848492)
= **6,17940035098993993622858837778 37**

División
por método de Newton-Raphson



Normalización (base 2)

Cuando *d* se encuentra fuera de rango, 1≤*d*, hay que normalizar y ajustar

$$d' = d.2^{-s}$$

entonces

$$x' = \frac{1}{d'} = \frac{2^s}{d} = 2^s.\frac{1}{d} = 2^s.x$$

Ejemplo

$d = 101_2$

entonces

$d' = 101_2.2^{-3} = 0{,}101_2$ y $x' = 2^3/101_2 = 8_{10}/5_{10}=1{,}6 \approx 1.10011_2$

como $x = x'.2^3$,

entonces

$x = 0.00110011_2 \approx 0{,}2_{10} = 1/d = 1/5$