Division entière

Alain MÉRIGOT

Université Paris Saclay

Plan

Division par soustraction-décalage

Soit Dd le dividende, Dv le diviseur.
Nous supposerons $0 \leq Dv, Dd < 1$ et $Dd < Dv$
Les nombres seront codés en virgule fixe par des puissances négatives de 2

Si $Dv = (d_{n-1}, \ldots, d_1, d_0)$

$$\begin{aligned} Dv &= 0.d_{n-1}d_{n-2}\cdots d_1 d_0 \\ &= d_{n-1} \times 2^{-1} + d_{n-2} \times 2^{-2} + \cdots + d_1 \times 2^{-n-1} + d_0 \times 2^{-n} \\ &= \sum_{i=n-1}^{0} d_i \times 2^{i-n} \\ &= 2^{-n} \times \sum_{i=n-1}^{0} d_i \times 2^{i} \end{aligned}$$

$Dd \div Dv$ produit deux nombres : le quotient Q et le reste R avec :

$$Dd = Q \times Dv + R$$
$$\text{et } 0 \leq R < Dv$$

Si $0 \leq Dv, Dd < 1$ et $Dd < Dv$, nous aurons aussi $0 \leq Q, R < 1$.

Si Dv et Dd sont codés sur n bits, Q et R sont aussi codés avec la même précision.

Division par soustraction-décalage (cont.)

Exemple de division en décimal

$0.26 \div 0.87$

	0.26	0.87	
		0	
	2.6	0.**3**	$3 \times 0.87 = 2.61$ (non car > 2.6)
	2.6	0.**2**	$2 \times 0.87 = 1.74$ (OK)
−	1.74		
	0.86		
	8.6	0.2**9**	$9 \times 0.87 = 7.83$
−	7.83	0.29	
	0.77		

Algorithme de division.

Détermination des restes successifs : comparaison avec le diviseur et détermination des bits successifs du quotient

Reste à la la $i$ème itération $R^{(i)}$

$$\begin{aligned} R^{(0)} &= Dd \\ R^{(i+1)} &= 2 \times R^{(i)} - q_{n-i+1} \times Dv \end{aligned}$$

$$\begin{aligned} R^{(1)} &= 2 \times R^{(0)} - q_{n-1} \times Dv \\ R^{(2)} &= 2 \times R^{(1)} - q_{n-2} \times Dv \\ &= 2^2 \times R^{(0)} - (2 \times q_{n-1} + q_{n-2}) \times Dv \\ R^{(n)} &= 2^n \times R^{(0)} - (2^{n-1} \times q_{n-1} + 2^{n-2} q_{n-2} + \cdots + q_0) \times Dv \\ 2^{-n} R^{(n)} &= Dd - \underbrace{(2^{-1} \times q_{n-1} + 2^{-2} q_{n-2} + \cdots + 2^{-n} q_0)}_{Q} \times Dv \end{aligned}$$

$$Dd = Q \times Dv + 2^{-n} \times R^{(n)}$$

ce qui correspond à la définition de la division.

Plan

Division avec restauration

Mise en oeuvre directe de la formule précédente.
On cherche le plus grand quotient qui maintienne un reste positif.

Algorithme 1 division avec restauration

1 $R^{(0)} \leftarrow D_d$
2 **pour** $i \leftarrow 0$ **à** $n-1$
3 **si** $2R^{(i)} - D \geq 0$
4 $R^{(i+1)} \leftarrow 2R^{(i)} - D$
5 $q_{n-1-i} \leftarrow 1$
6 **sinon**
7 $R^{(i+1)} \leftarrow 2R^{(i)}$
8 $q_{n-1-i} \leftarrow 0$
9 **fin si**
10 **fin pour**
11 $R_{final} = R \times 2^{-n}$

Division avec restauration (cont.)

Mise en oeuvre matérielle :

- suppression des indices
- un registre contient le reste courant

Algorithme 2 division avec restauration

1 $R \leftarrow D_d$
2 **pour** $i \leftarrow 0$ **à** $n-1$
3 $R \leftarrow 2R - D$ ▷ *calcul du nouveau reste en supposant* $q = 1$
4 **si** $R \geq 0$
5 $q_{n-1-i} \leftarrow 1$
6 **sinon**
7 $R \leftarrow R + D$ ▷ *Restauration si reste négatif*
8 $q_{n-1-i} \leftarrow 0$
9 **fin si**
10 **fin pour**
11 $R_{final} = R \times 2^{-n}$

Problèmes : irrégularité suivant les bits et possibilité de deux additions par itérations

Division avec restauration (cont.)

Division de $Dd = 0.011 = 0.375$ par $Dv = 0.101 = 0.625$ (ou $\frac{3/8}{5/8} = \frac{3}{5}$).

Initialement : $R = Dd = 0.011$

$Dv = 0.101$ $-Dv = 1.011$

Etape 1 :

Calcul de $R \leftarrow 2 \times R - Dv$

		$^{11\ 10}$
$2 \times R$		0.110
$-Dv$	$+$	1.011
$R \leftarrow 2 \times R - Dv$	$=$	$0.001 \quad \geq 0$

$\Rightarrow \quad q_{-1} \leftarrow 1 \quad R \leftarrow 0.001$

Etape 2 :

Calcul de $R \leftarrow 2 \times R - Dv$

		$^{00\ 10}$
$2 \times R$		0.010
$-Dv$	$+$	1.011
$R \leftarrow 2 \times R - Dv$	$=$	$1.101 \quad < 0$

$\Rightarrow \quad q_{-2} \leftarrow 0 \quad R \leftarrow R + Dv = 1.101 + 0.101 = 0.010$

Etape 3 :

Calcul de $R \leftarrow 2 \times R - Dv$

		$^{00\ 00}$
$2 \times R$		0.100
$-Dv$	$+$	1.011
$R \leftarrow 2 \times R + Dv$	$=$	$1.111 \quad < 0$

$\Rightarrow \quad q_{-3} \leftarrow 0 \quad R \leftarrow R + Dv = 1.111 + 0.101 = 0.100$

$Q = 0.100 = 0.5, R = 0.100 \times 2^{-3} = 0.0625$ $(0.375 = 0.625 \times 0.5 + 0.0625)$

Plan

Division sans restauration

Suppression de l'étape de restauration.
Le reste R peut devenir négatif.
Le signe de R détermine le quotient et l'opération à effectuer
Quotient codé *en digits signés* : $q_i \in \{-1, +1\}$

Algorithme 3 division sans restauration

▷ *On suppose* $D > 0$
$R \leftarrow D_d$
pour $i \leftarrow 0$ **à** $n-1$
  si $R \geq 0$
    $R \leftarrow 2R - D$
    $q_{n-1-i} \leftarrow +1$
  sinon
    $R \leftarrow 2R + D$
    $q_{n-1-i} \leftarrow -1$
  fin pour
si $R < 0$
  $R \leftarrow R + D$ ▷ *Correction d'un reste négatif*
  $Q \leftarrow Q - 2^{n-1}$ ▷ *Enlever 1 au bit de poids faible de Q*
fin si
$R_{final} = R \times 2^{-n}$

Division sans restauration (cont.)

Si un bit à -1 du quotient (parfois noté $\bar{1}$) est codé par un bit à 0,
on retrouve la valeur du quotient par :
$Q = \sum_{i=n-1}^{0} 2^i \times q_i - \sum_{i=n-1}^{0} 2^i \times \overline{q_i}$

Certaines valeurs du quotient ne sont pas codables avec des ± 1, par exemple si le LSB du quotient doit être à 0.
Dans ce cas, le dernier reste sera < 0

Pour obtenir un résultat avec $R \geq 0$:

- on remplace $q_0 = 1$ par $q_0 = 0$ (et donc on retranche 2^{-n} au quotient)
- on corrige le dernier reste en ajoutant D (car la dernière opération aurait du être $R \leftarrow 2R - q_0 \times D$ avec $q_0 = 0$ et non $q_0 = +1$)

Division sans restauration (cont.)

Division de $Dd = 0.011 = 0.375$ par $Dv = 0.101 = 0.625$.
Initialement : $R = Dd = 0.011$ $Dv = 0.101$ $-Dv = 1.011$

Etape 1:
$R \geq 0 \rightarrow$ Calcul de $R \leftarrow 2 \times R - Dv$

		$^{11\ 10}$	
$2 \times R$		0.110	
$-Dv$	$+$	1.011	
$R \leftarrow 2 \times R - Dv$	$=$	0.001	≥ 0

$\Rightarrow \quad q_{-1} \leftarrow 1 \quad R \leftarrow 0.001$

Etape 2 :
$R \geq 0 \rightarrow$ Calcul de $R \leftarrow 2 \times R - Dv$

		$^{00\ 10}$	
$2 \times R$		0.010	
$-Dv$	$+$	1.011	
$R \leftarrow 2 \times R - Dv$	$=$	1.101	< 0

$\Rightarrow \quad q_{-2} \leftarrow 1 \quad R \leftarrow 1.101$

Etape 3 :
$R < 0 \rightarrow$ Calcul de $R \leftarrow 2 \times R + Dv$

		$^{00\ 00}$	
$2 \times R$		1.010	
$+Dv$	$+$	0.101	
$R \leftarrow 2 \times R - Dv$	$=$	1.111	< 0

$\Rightarrow \quad q_{-3} \leftarrow \bar{1} \quad R \leftarrow 1.001$

...

Division sans restauration (cont.)

...

Soit un résultat final
$Q = 0.11\bar{1} = 0.110 - 0.001 = 0.110 + 1.111 = 0.101 = 0.625$
$R = 1.111 \times 2^{-3} = -0.001 \times 2^{-3} = -0.015625$
$Dd = 0.375 = Dv \times Q + R = 0.625 \times 0.625 - 0.015625$

Mais, comme $R < 0$, une correction du résultat est nécessaire :

1/ on soustrait 2^{-n} à Q
$Q \leftarrow Q - 0.001 = 0.101 - 0.001 = 0.100$
2/ on ajoute D à R
$R \leftarrow R + D = 1.111 + 0.101 = 0.100$

Finalement $D = 0.100 = 0.5_d$ et $R = 0.100 \times 2^{-3} = 0.0625_d$
(identique au cas précédent)

Plan

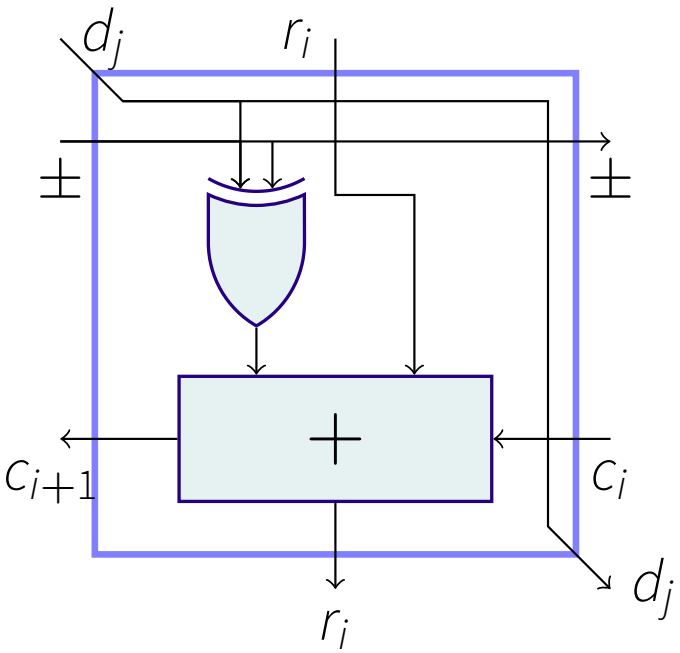
Mise en oeuvre matérielle

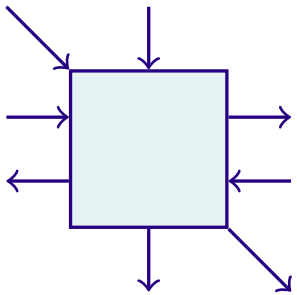
Opération de base de la division sans restauration :
addition/soustraction suivant le signe de R

Contrairement à la multiplication, il n'est *pas* possible d'utiliser des additionneurs à sauvegarde de retenue, car le signe du résultat est requis.

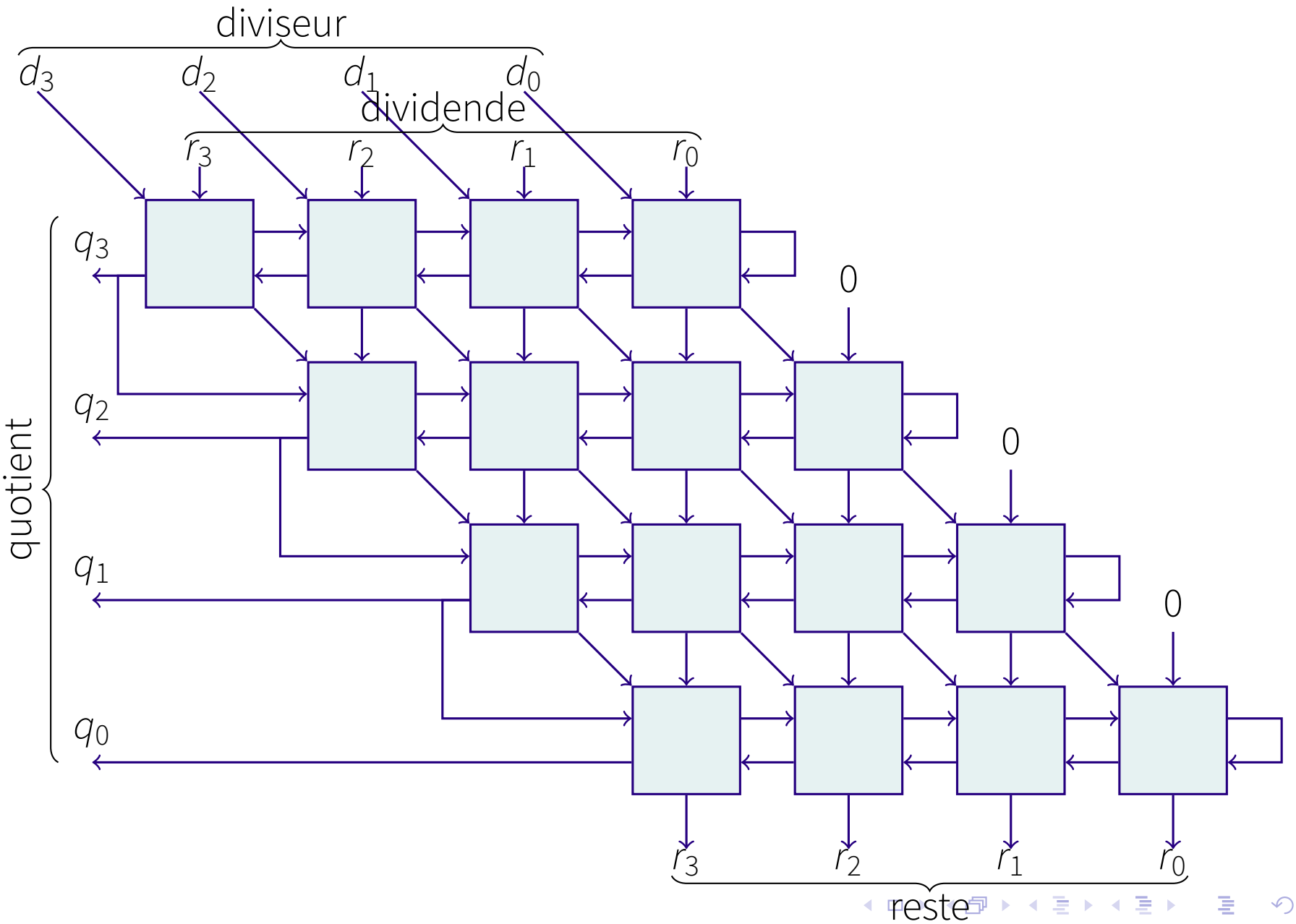
Utilisation d'additionneur/soustracteur à propagation de retenue

Cellule de base :





Mise en oeuvre matérielle (cont.)



Mise en oeuvre matérielle (cont.)

Principal problème de cette approche : l'addition à propagation prend un temps n et le temps total est en n^2.

Les diviseurs des processeurs actuels utilisent une méthode dérivée SRT (Sweeney, Robertson, Tocher).

- le calcul est fait en base 4 ou 8
- les additions/soustractions sont à sauvegarde de retenue
- on cherche à *prédire* le signe du reste avec les bits de poids fort des opérandes (mais *sans* le calculer)
  si la prédiction est incorrecte, la méthode permet quand même de retrouver le bon résultat aux itérations suivantes

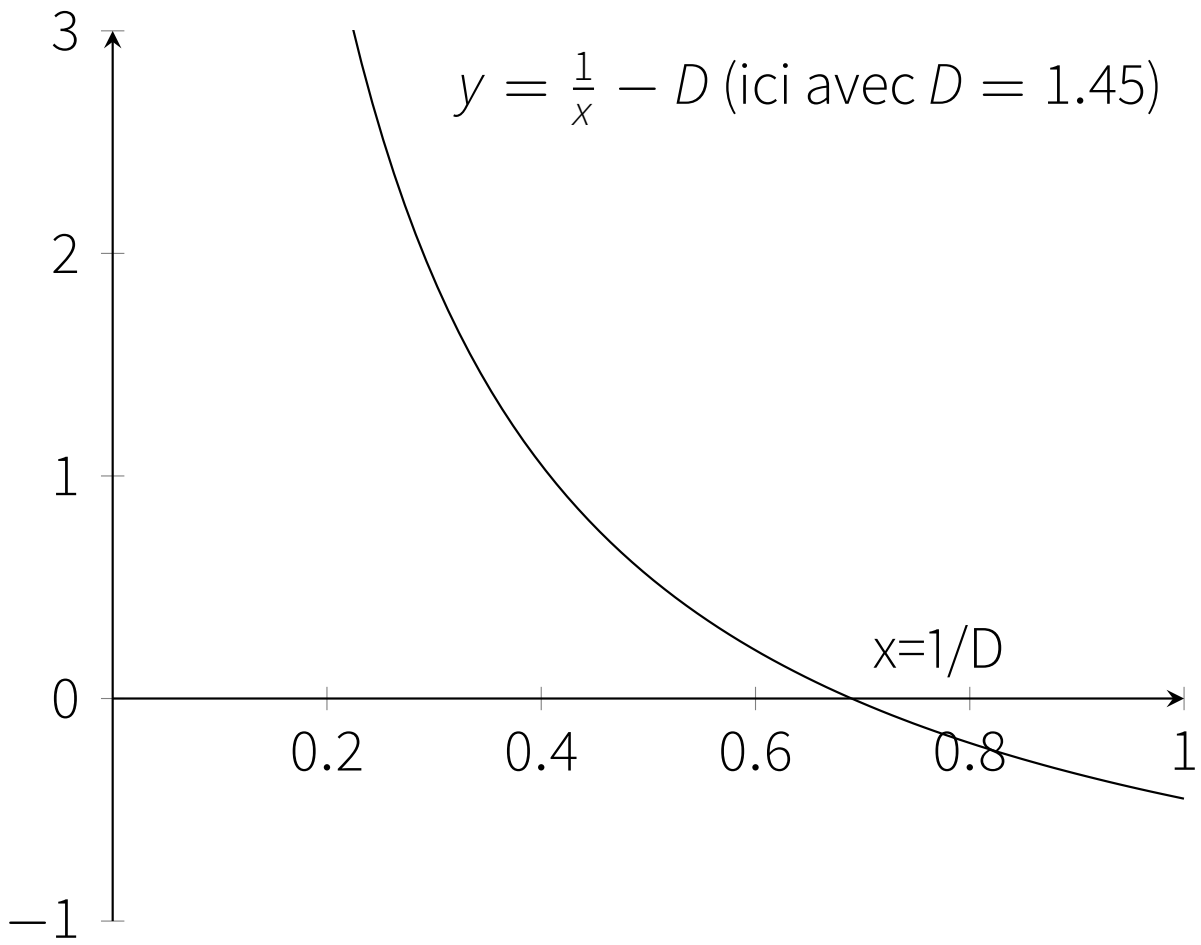
La division est une opération coûteuse : 10–30 cycles sur les processeurs actuels

Plan

Division itérative

Utilisation de méthodes d'analyse numérique.

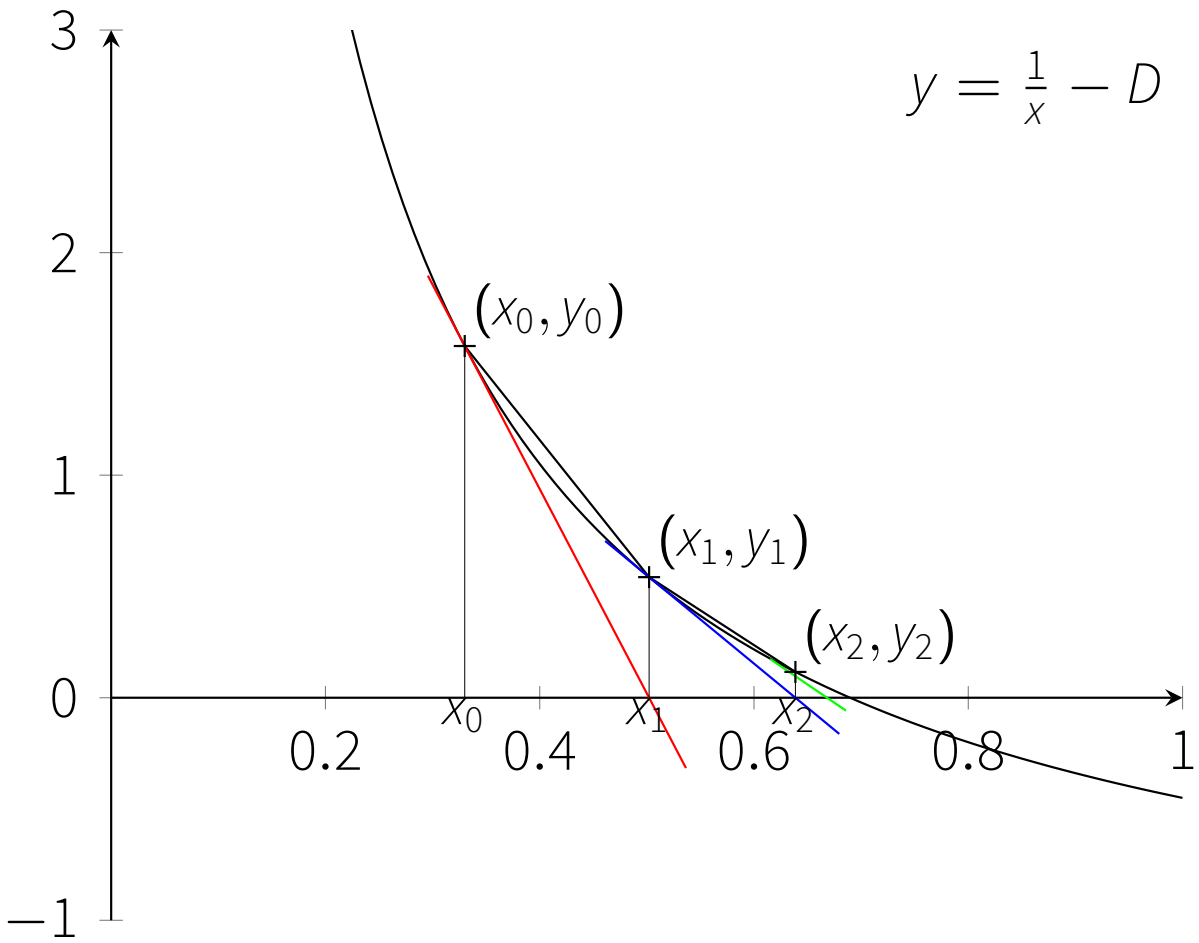
On cherche le point où la fonction $y = \frac{1}{x} - D$ passe par 0.



Pour calculer $N \div D$, il suffit ensuite de multiplier N par $1/D$.

Division itérative (cont.)

Utilisation de méthodes itératives pour trouver le « zéro » de $f = 1/x - D$.



1. on part d'une approximation initiale de la solution x_0
2. autour du point (x_0, y_0), on approxime f par sa tangente
3. on cherche le point x_1 d'intersection de la tangente avec l'axe des x
   x_1 est une meilleure approximation de la solution
4. on itére 2–3 pour trouver les points x_2, x_3, x_4, etc...

Division itérative (cont.)

Equation de la tangente en x_0

$$y = f(x_0) + f'(x_0)(x - x_0)$$

La solution de cette équation en $y = 0$ donne une nouvelle approximation de la solution x_1

$$x = x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

On itére un certain nombre de fois pour trouver la solution de $f(x) = 0$ (méthode de Newton-Raphson).

Pour calculer l'inverse $f(x) = 1/x - D$ qui s'annule en $x = 1/D$.
Nous supposons que $D \in [0.5, 1]$ et donc $1/D \in [1, 2]$.
$f'(x) = -1/x^2$
itération de Newton-Raphson :

$$\begin{aligned} x_{i+1} &= x_i - f(x_1)/f'(x_i) \\ &= x_i - \frac{1/x_i - D}{-1/x_i^2} \\ &= x_i \times (2 - D \times x_i) \end{aligned}$$

Le calcul de x_{i+1} ne nécessite que + et $\times$ $\Longrightarrow$ utilisation possible pour calculer $1/D$

Nous supposons $x_i = \frac{1}{D}(1+\epsilon_i)$.
ϵ_i l'erreur relative algébrique quand on approxime $\frac{1}{D}$ par x_i.
A l'ordre $i+1$,

$$\begin{aligned} x_{i+1} &= x_i(2 - Dx_i) \\ &= \frac{1}{D}(1+\epsilon_i)(2 - D \times \frac{1}{D}((1+\epsilon_i)) \\ &= \frac{1}{D}(1-\epsilon_i^2) \\ \epsilon_{i+1} &= -\epsilon_i^2 \end{aligned}$$

Erreur ϵ_i

- réduite de manière *quadratique*
- toujours négative (approximation par défaut).

Si $|\epsilon_i| = 2^{-k}$ (k bits significatifs), $|\epsilon_{i+1}| = 2^{-2k}$ ($2k$ bits significatifs).
Nombre de bits significatifs doublé à chaque étape,
Nombre maximal d'étapes pour obtenir n bits de précision est $\mathbf{log}_2\, n$.

Division itérative (cont.)

approximation initiale

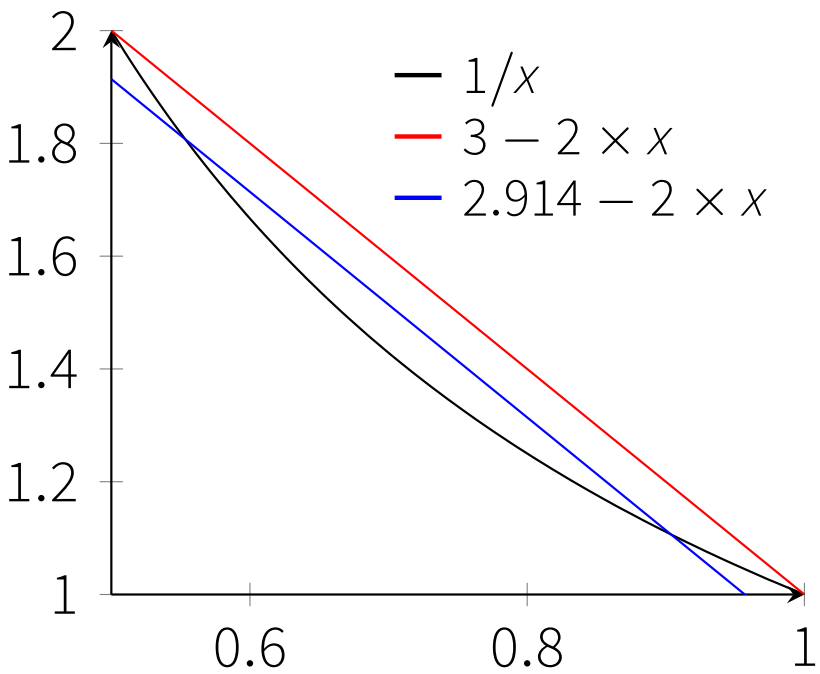
Deux méthodes pour obtenir l'approximation initiale :

valeurs tabulées (*look-up table*)

On approxime la fonction $1/D$ par un tableau avec valeur précalculées

En partant, par exemple, d'une table des inverses sur 4 bits, 32 bits de précision en 3 étapes

Approximation par une droite de la fonction $1/x$ dans l'intervalle considéré ($D \in [0.5, 1]$)



La droite qui passe par $(0.5, 2)$ et $(1, 1)$ a pour équation $y = 3 - 2 \times x$

Une meilleure approximation est $y = k - 2 \times x$ avec $k = 2.914\ldots$ pour minimiser l'erreur d'approximation.

On peut également utiliser des polynomes d'ordre 2.

Conclusion

La division itérative permet d'obtenir de bonnes performances sans diviseur matériel.
Mais il est difficile de respecter les contraintes de la norme IEEE-754 avec cette méthode (à cause des arrondis dans le calcul de $Dd \times (1/Dv)$).
De ce fait, elle n'est plus utilisée dans les processeurs standard, mais c'est une des meilleures manières d'effectuer une division, par exemple sur un FPGA.

Une autre méthode itérative (division de GoldSchmidt) est utilisée dans les processeurs AMD et elle permet de calculer directement Dd/Dv.

Mais la plupart des processeurs actuels (pentium, arm, etc) ont un diviseur basé sur la méthode SRT.