

Calcul et arithmétique des ordinateurs
traité hermes

Jean-Michel MULLER et Jean-Claude BAJARD

version 3.0
25 mars 2004

Table des matières

Introduction	15
0.1. Bibliographie	20
PREMIÈRE PARTIE. REPRÉSENTATION EN MACHINE ET ÉVALUATION . .	23
Chapitre 1. Représentation des nombres	25
Marc Daumas et Jean-Michel Muller	
1.1. Introduction	25
1.2. Représentation de position des entiers	25
1.2.1. Représentation « de position » des entiers positifs	25
1.2.2. Représentation de position des entiers signés	29
1.2.2.1. Représentation par signe et valeur absolue	29
1.2.2.2. Représentation en complément à la base	29
1.2.2.3. Représentations biaisées des entiers signés	32
1.2.3. Représentations redondantes	32
1.2.3.1. Représentations d'Avizienis	33
1.2.3.2. Représentations « carry save » et « borrow save »	34
1.2.4. Représentations modulaires des entiers	37
1.3. La représentation virgule flottante	39
1.3.1. Quelques généralités	39
1.3.2. Les modes d'arrondi	41
1.3.3. Les formats spécifiés par la norme IEEE-754	43
1.3.4. Les exceptions et leur traitement	45
1.3.4.1. Valeurs infinies et nombres dénormalisés	45
1.3.4.2. Quantités Not a Number	49
1.3.4.3. Tests portant sur des NaN	50
1.3.4.4. Codage des valeurs particulières	50
1.3.5. Conversions	51
1.3.6. La multiplication-accumulation	52

1.3.7. Tester son environnement virgule flottante	53
1.3.8. Quelques lectures	54
1.4. Bibliographie	54
Chapitre 2. Méthodes générales d'addition et multiplication	59
Jean-Michel Muller	
2.1. Introduction	59
2.2. L'addition en binaire : généralités	60
2.3. Le principe d'addition avec retenue conditionnelle	61
2.4. Le théorème de Winograd	62
2.5. Les fonctions magiques : <i>Generate</i> et <i>Propagate</i>	64
2.6. Les additionneurs à retenue anticipée	66
2.7. Les additionneurs <i>parallel prefix</i>	66
2.8. D'autres solutions	70
2.9. La multiplication	72
2.10. Multiplication par réseau cellulaire	72
2.11. Décomposition récursive de la multiplication	73
2.12. Multiplication arborescente en temps logarithmique	74
2.13. Le recodage de Booth	76
2.14. Bibliographie	79
Chapitre 3. Evaluation des Fonctions élémentaires	83
Jean-Michel Muller	
3.1. Introduction	83
3.2. La réduction d'argument	86
3.3. Mettre au point des approximations polynomiales	88
3.3.1. Obtenir de «bonnes» approximations	88
3.4. Un exemple : le calcul de l'exponentielle	93
3.4.1. Réduction d'argument	94
3.4.2. Approximation choisie	94
3.5. Le dilemme du fabricant de tables	98
3.6. L'algorithme CORDIC	100
3.6.1. Une méthode simple pour peser du pain	101
3.6.2. De la pesée du pain vers l'évaluation des fonctions trigonométriques	102
3.6.3. L'algorithme CORDIC généralisé	103
3.7. Conclusion	105
3.8. Bibliographie	105
Chapitre 4. Opérateurs sur circuits FPGA	109
Arnaud Tisserand et Jean-Luc Beuchat	
4.1. Introduction	109
4.2. Circuits FPGA	109

4.2.1. Architecture générale des FPGA	112
4.2.2. Exemples de circuits actuels : les familles Virtex et Spartan de Xilinx	115
4.3. Opérations de base	118
4.3.1. Addition	118
4.3.1.1. Additionneurs séquentiels rapides sur FPGA	119
4.3.2. Multiplication	120
4.3.3. Génération des produits partiels	121
4.3.4. Réduction des produits partiels	122
4.3.5. Addition finale	123
4.3.6. Multiplication/addition fusionnée	123
4.3.7. Carré	124
4.3.8. Petits blocs de multiplication câblée	124
4.4. Fonctions algébriques et élémentaires	125
4.4.1. Division	126
4.4.2. Racine carrée	129
4.4.3. Évaluation des fonctions élémentaires sur FPGA	129
4.4.4. Évaluation de polynômes sur FPGAs	129
4.4.5. Algorithmes à base d'additions et de décalages	130
4.4.6. Méthodes à base de tables et d'additions	130
4.5. Arithmétique sérielle	131
4.5.1. Modes de transmission des données	132
4.5.2. Arithmétique sérielle classique	134
4.5.2.1. Addition et soustraction sérielles	134
4.5.2.2. Multiplication et élévation au carré sérielles	135
4.5.2.3. Multiplication parallèle-série	137
4.5.3. Arithmétique en-ligne	139
4.5.3.1. Opérations de base et fonctions algébriques	139
4.5.3.2. Évaluation en-ligne de fonctions élémentaires	141
4.6. Arithmétique modulaire sur FPGA	142
4.6.1. Addition modulaire	143
4.6.2. Multiplication modulaire	145
4.6.2.1. Opérateurs parallèles	145
4.6.2.2. Opérateurs parallèle-série	148
4.7. Bibliographie	150
DEUXIÈME PARTIE. EXTENSIONS	153
Chapitre 5. Arithmétique multiprécision	155
Laurent Imbert	
5.1. Introduction	155
5.2. Comment représenter un grand nombre ?	156
5.2.1. Représentation des entiers	156

5.2.2. Représentation des nombres réels	157
5.3. Opérations arithmétiques élémentaires sur les grands entiers	158
5.3.1. La normalisation	158
5.3.2. L'addition et la soustraction	158
5.3.3. La multiplication par un petit entier	159
5.3.4. La division par un petit entier	159
5.3.5. La multiplication de deux grands entiers par l'algorithme usuel	159
5.4. Comment multiplier plus rapidement ?	160
5.4.1. Les méthodes de Knuth, Karatsuba et Toom-Cook	160
5.4.2. La méthode de Schönhage et Strassen	164
5.4.2.1. La transformée de Fourier discrète	165
5.4.2.2. L'algorithme FFT	166
5.4.2.3. Application à la multiplication de grands entiers	166
5.4.2.4. Implantation	167
5.5. Division et racine carrée	168
5.5.1. La méthode de Newton-Raphson	168
5.5.2. La méthode de Goldschmidt	169
5.6. Fonctions élémentaires	170
5.6.1. Les approximations polynomiales	170
5.6.2. La moyenne arithmético-géométrique de Gauss-Legendre	173
5.6.3. La « Binary Splitting Method »	175
5.6.4. Quelques mots sur des méthodes mixtes	175
5.7. Ressources	176
5.8. Bibliographie	177
Chapitre 6. Systèmes modulaires de représentation	181
Laurent-Stéphane Didier	
6.1. Introduction	181
6.2. Les principes de la représentation modulaire	182
6.2.1. Les systèmes modulaires de représentation (Residue Number Systems)	182
6.2.2. Opérations de base	185
6.2.3. Conversions	186
6.2.3.1. Conversion des systèmes de numération de position vers les systèmes modulaires	187
6.2.3.2. Conversion des systèmes modulaires vers les systèmes de numération de position	187
6.3. Diverses applications de la conversion	191
6.3.1. Extension et changement de base	191
6.3.1.1. Extension de base de Szabo et Tanaka	192
6.3.1.2. Extension de base de Shenoy et Kumaresan	192
6.3.2. Comparaison	193
6.3.2.1. Comparaison par changement de base	193

6.3.2.2. Comparaison de Chiang et Lu	193
6.3.2.3. Comparaison de Dimauro, Impedovo et Pirlo	194
6.4. Autres opérations dans les systèmes modulaires	194
6.4.1. Division	195
6.4.1.1. Cas particulier de la division exacte (sans reste)	195
6.4.1.2. Division de Gamberger	196
6.4.2. Multiplication modulaire dans un système modulaire	197
6.5. Conclusion	201
6.6. Bibliographie	201
Chapitre 7. Calcul sur les corps finis	207
Jean-Claude Bajard	
7.1. Des généralités	208
7.2. La multiplication dans $GF(2^m)$	209
7.2.1. Algorithme de Montgomery	210
7.2.1.1. Présentation de l'algorithme original	210
7.2.1.2. Algorithme de Montgomery sur les corps finis	211
7.2.1.3. Version itérative	213
7.2.2. Méthode de Mastrovito	214
7.2.3. Utilisation d'une base normale	216
7.2.4. Bases duales	219
7.3. La division	222
7.3.1. Utilisation de l'algorithme d'Euclide	222
7.3.2. Utilisation du petit théorème de Fermat	224
7.4. Perspectives	225
7.5. Bibliographie	225

Introduction

Nous avons tous appris, à l'école primaire, à effectuer des additions, des multiplications et des divisions. Pour cette raison, la plupart des utilisateurs d'ordinateurs ne se sont jamais demandé comment leurs machines effectuaient les opérations arithmétiques, pensant que les mêmes méthodes (ou tout au moins de légères variantes, adaptées par exemple à l'emploi de la base 2) étaient utilisées, ce qui est nous le verrons souvent faux.

En ce qui concerne l'évaluation de fonctions plus complexes (sinus, cosinus, logarithme, exponentielle...), nombre d'entre nous se sont demandés, au lycée, quelles étaient les méthodes utilisées par nos calculatrices de poche. Cette interrogation a en général pris fin lorsque nous avons appris ce qu'est un développement de Taylor. Nous avons alors cru savoir.

Un des buts de cet ouvrage est de montrer que les algorithmes utilisés en arithmétique des ordinateurs sont souvent plus complexes qu'on ne croit, et qu'ils sont en général très différents de ceux que l'on utilise pour faire des calculs « à la main ».

Un autre point important concerne la fiabilité des calculs effectués sur ordinateur. Tout d'abord parce que les circuits et programmes arithmétiques peuvent, comme les autres, comporter des erreurs. Ensuite parce que même avec des circuits et programmes arithmétiques corrects, le résultat d'une opération arithmétique n'est pas forcément exactement représentable en machine : il faut l'arrondir au « nombre machine » le plus proche. On commet alors une toute petite *erreur d'arrondi*. En général, l'influence de ces erreurs d'arrondi sur le résultat final d'un calcul est faible, mais ce n'est pas toujours le cas. Donnons tout d'abord deux exemples d'erreur de conception dans des circuits ou programmes arithmétiques :

– le diviseur de la première version du processeur Pentium d'Intel donnait un résultat faux dans environ un cas sur 4×10^{10} (en simple précision). Par exemple, le calcul de

$$8391667/12582905$$

donnait $0.666869\dots$ au lieu de $0.666910\dots$. Il est d'ailleurs amusant de constater que l'erreur réside dans l'algorithme lui-même, et non dans son implantation [Mul95a];

– dans la version 7.0 du système de calcul formel Maple, si l'on calcule

$$\frac{5001!}{5000!}$$

on obtient 1 au lieu de 5001. Dans la version précédente (6.0) du même système, si on entrait :

$$21474836480413647819643794$$

la « quantité » affichée et mémorisée était $413647819643790)+'-.(-. (.$

Il est très facile¹ de construire des exemples numériques pour lesquels l'accumulation des erreurs d'arrondi finit par conduire à un résultat inacceptable. Considérons l'exemple suivant, construit par l'un d'entre nous [Mul95b] : en partant d'une valeur initiale

$$u_0 = e - 1 = 1.71828182845904523536028747\dots$$

où e est la base des logarithmes naturels, on construit la suite u_n définie par

$$u_n = nu_{n-1} - 1.$$

Le problème est de calculer, par exemple, u_{25} . C'est *a priori* facile : on n'effectue que des multiplications par de tous petits entiers, et des soustractions du nombre 1. Pourtant, selon le système utilisé, on obtiendra des résultats radicalement différents. Le tableau 1 donne quelques exemples obtenus sur une arithmétique de base 10, en faisant varier le nombre de décimales utilisées lors des calculs. On voit que sur cet exemple, il faut une très grande précision des calculs intermédiaires pour obtenir un résultat final correct.

Le présent ouvrage a été conçu pour donner au lecteur une idée de ce qu'est l'arithmétique des ordinateurs ainsi qu'un inventaire de propriétés et de solutions dans lequel il pourra venir piocher pour résoudre certains problèmes qu'il peut rencontrer.

Par « arithmétique des ordinateurs » nous entendons la discipline qui étudie les systèmes de représentation des nombres utiles au calcul, ainsi que les algorithmes permettant d'effectuer les opérations arithmétiques et de calculer les principales fonctions mathématiques. Le mot « ordinateur » est peut être malencontreux ici, car le problème abordé est bien plus vieux que l'informatique. Il y a environ 5000 ans, les sumériens

1. Peut être même trop facile : on voit circuler de nombreux exemples « bricolés » (comme celui que nous présentons ici, d'ailleurs !) ce qui peut conduire à surestimer le risque d'erreur encouru lors d'un calcul correspondant à un problème réel.

n	u_{25}
5	2.8188×10^{20}
10	-7120347065000000.0
15	73905327361.8675
20	615989.41313938063360
25	-7.271311422812782592000000
30	0.0398974311112776515584000000
valeur exacte	0.03993872967323 ...

Tableau 1. Résultats obtenus lorsqu'on calcule u_{25} en utilisant une arithmétique de base 10 (celle de Maple, mais ce système n'est pas en cause dans cet exemple) sur n chiffres, en faisant varier n .

inventaient un système de numération, le premier dont on ait gardé la trace. Au début il s'agissait probablement juste de *mémoriser* des quantités. Les hommes ont plus tard appris à *calculer*. Ils ont conçu alors des systèmes de numération mieux adaptés à ce besoin. Les babyloniens ont par exemple inventé un système de base 60 dont nous gardons encore la trace dans notre manière de représenter le temps et les angles. Ce système a permis le développement de techniques de calcul sophistiquées. Fowler et Robson [FR98] donnent un algorithme babylonien d'approximation de racines carrées qui n'est autre, réécrit en langage moderne, que l'itération

$$x_{n+1} = \frac{1}{2} \left(x_n + \frac{A}{x_n} \right)$$

de calcul de \sqrt{A} , communément attribuée à Héron d'Alexandrie (env. 10 – env. 75), quand ce n'est pas à Newton, puisqu'il s'agit de l'itération de Newton-Raphson

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

dans le cas particulier de la recherche des zéros de la fonction $f(x) = x^2 - A$. Comment les babyloniens ont-ils pu construire cet algorithme? Voilà comment Fowler et Robson l'expliquent. On cherche la racine carrée d'un nombre A , et on en connaît une approximation a , telle que $a^2 < A$. Le cas $a^2 > A$ est assez similaire et nous le laissons à la sagacité du lecteur. On peut donc (voir figure 1) construire un carré, de côté a , et dont la surface a^2 est légèrement inférieure à A . Chercher une meilleure approximation de \sqrt{A} , c'est chercher à construire un nouveau carré, dont la surface est plus proche de A . On peut faire ceci comme indiqué sur la figure : on adjoint au carré de côté a deux rectangles de longueur a et de largeur $(A - a^2)/(2a)$, de sorte que la somme des surfaces du carré et des deux rectangles fasse exactement A . En les disposant comme indiqué sur la figure, on obtient un nouveau carré, qui est de surface

légèrement supérieure à A (car pour faire un carré il a fallu ajouter un petit carré au carré initial et aux deux rectangles), et de côté

$$\frac{1}{2} \left(a + \frac{A}{a} \right).$$

Ce côté constitue notre nouvelle approximation de \sqrt{A} . Cet exemple illustre le degré

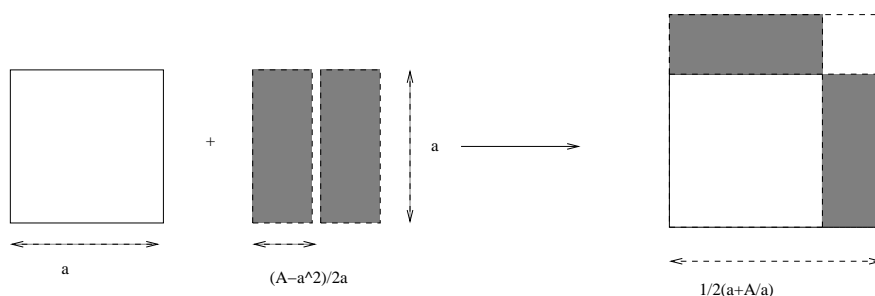


Figure 1. Explication par Fowler et Robson de la méthode Babylonienne d'amélioration d'une approximation de \sqrt{A} .

de sophistication de la science babylonienne, en particulier dans le domaine du calcul. En 1700 avant J.C., les babyloniens connaissaient la durée de l'année avec une précision d'un millièmme (voir http://www-obs.univ-lyon1.fr/~ga/hom_ciel/homciel.html).

Vers 300 avant J.C., Euclide propose son algorithme de calcul du PGCD de deux entiers. Eratosthène (env. 276 – env. 194 avant J.C.) invente sa méthode du crible, qui permet de construire la suite des nombres premiers.

L'invention, probablement en Inde au VIII^{ème} siècle de notre ère, de notre système de numération de position décimal (voir l'ouvrage de Geneviève Guitel [Gui75] pour plus d'information) a considérablement simplifié les calculs arithmétiques. Le traité d'arithmétique d'Al-Khwarizmi (env. 780 – env. 850), dont on ne possède maintenant qu'une version latine du XIII^{ème} siècle [DDP86], a joué un rôle majeur dans l'introduction de ce système en occident. Le terme « algorisme », dérivé de son nom, a désigné en Europe ce système de numération et les méthodes opératoires associées, avant de devenir notre moderne « algorithme ». Les ouvrages d'Al-Khwarizmi ont aidé Fibonacci et Pacioli à fonder l'arithmétique moderne [All99] et à inventer notre méthode actuelle de multiplication « à la main ».

Si l'automatisation du calcul a fait des progrès considérables dans la deuxième moitié du XX^{ème} siècle, il ne faut pas oublier que des machines arithmétiques ont été inventées bien avant : on pense bien entendu immédiatement aux machines de

Schickard (1623), Pascal (1642) et de Leibniz (1673) [Ifr94], mais les abaques² et les bouliers sont déjà des dispositifs conçus pour faciliter les calculs arithmétiques.

Au XVII^{ème} siècle, l'invention des logarithmes par Neper (ou Napier, 1550 – 1617) a constitué un immense progrès, permettant de remplacer des multiplications et divisions par des additions et soustractions. Ceci a conduit entre autres à l'invention de la règle à calcul par Gunter et Oughtred en 1632. Neper est également, avec le néerlandais Snellius, à l'origine de la notation décimale actuelle des nombres non entiers, qui a remplacé la notation fractionnaire. Là où auparavant on écrivait

$$25\frac{33}{100}$$

on écrit depuis

$$25,33.$$

Neper a également inventé un dispositif pour simplifier les multiplications, appelé les « réglettes (ou bâtons) de Neper ». Briggs (1561 – 1630) a publié en 1624 des tables de logarithmes et de certaines fonctions trigonométriques d'une précision de 14 décimales, ce qui constituait pour l'époque un tour de force impressionnant. Ces tables ont rendu de nombreux calculs astronomiques possibles, et pour les construire Briggs a inventé un algorithme qui est de la même famille que notre moderne algorithme CORDIC de calcul des principales fonctions élémentaires.

L'*Arithmomètre* de Thomas de Colmar, inventé en 1822, fût la première calculatrice très largement diffusée [Ifr94]. Au début du XIX^{ème} siècle Charles Babbage (1791–1871) a proposé deux machines, la *Difference Engine* puis plus tard l'*Analytical Engine* dédiées au calcul numérique. La première devait permettre de construire des tables de fonctions par interpolation. Quand à la seconde elle est l'ancêtre des ordinateurs actuels, elle est programmable via des cartes perforées, possède une unité arithmétique (en la concevant, Babbage a d'ailleurs inventé l'algorithme d'addition à *retenue bondissante*, ou *carry-skip*, que nous évoquerons au paragraphe 2.8) et une unité de contrôle. Augusta Ada Lovelace a écrit pour cette machine un programme de calcul de la suite des nombres de Bernoulli. Babbage n'a pu faire construire que des parties de ses machines. La première complètement réalisée (la *difference engine* numéro 2), l'a été en 1985, à la demande du Science Museum britannique. Le lecteur intéressé par l'histoire des machines modernes de calcul pourra parcourir l'ouvrage de Wilkes [Wil95].

La première partie de cet ouvrage est consacrée à la représentation des nombres en machine, ainsi qu'aux algorithmes arithmétiques élémentaires (addition, multiplication, principales fonctions mathématiques). Nous terminons cette partie par une

2. Certaines abaques babyloniennes datent de 3000 ans avant J.C.

description détaillée de l'implantation d'opérateurs arithmétiques sur circuits de type « FPGA » (*field programmable gate arrays*). Composés d'un réseau de fonctions logiques ou de transistors, ces circuits configurables sont programmables par leur utilisateur.

Dans une seconde partie, nous abordons des aspects moins « classiques » de l'arithmétique des ordinateurs. Par exemple, nous verrons comment étendre les représentations présentées dans la première partie pour répondre à certains besoins de précision. En effet, si les formats virgule flottante « simple précision », « double précision » et « quadruple précision » que nous manipulons au quotidien suffisent à traiter convenablement la plupart des problèmes numériques actuels (pour peu qu'on y apporte un peu de soin), il arrive que l'on ait besoin de manipuler des nombres de très grande taille. Au moment où nous écrivons ces lignes, le record est probablement détenu par Y. Kanada, de l'université de Tokyo, qui avec son équipe a calculé les 1.241.100.000.000 premiers chiffres décimaux de π , en utilisant les relations [Bai03]

$$\begin{aligned}\pi &= 48 \arctan \frac{1}{49} + 128 \arctan \frac{1}{57} - 20 \arctan \frac{1}{239} + 48 \arctan \frac{1}{110443} \\ \pi &= 176 \arctan \frac{1}{57} + 28 \arctan \frac{1}{239} - 48 \arctan \frac{1}{682} + 96 \arctan \frac{1}{12943}.\end{aligned}$$

Ceci a demandé un total de 600 heures de calcul sur un calculateur parallèle Hitachi à 64 processeurs, et la mémoire nécessaire a été de 10^{12} octets.

Nous proposons ensuite un survol d'une représentation un peu plus exotique que sont les systèmes « modulaires » de représentation des nombres (*Residue Number Systems*). Ils présentent certaines caractéristiques propices au parallélisme (en particulier ils rendent l'addition et la multiplication très facile), et peuvent répondre à des attentes rencontrées en traitement du signal ou en cryptographie. Nous terminons en présentant des éléments de calcul sur les corps finis, actuellement très utilisés en cryptographie.

0.1. Bibliographie

- [All99] A. Allard. La révolution arithmétique du moyen-âge. *La Recherche*, août 1999. Hors-série No 2.
- [Bai03] D. Bailey. Some background on kanada's recent pi calculation. Technical report, Lawrence Berkeley National Laboratory, 2003. <http://crd.lbl.gov/~dhbailey/dhbpapers/index.html>.
- [DDP86] A. Dahan-Dalmedico et J. Peiffer. *Histoire des mathématiques*. Editions du Seuil, 1986.
- [FR98] Fowler et Robson. Square root approximations in old babylonian mathematics : Ybc 7289 in context. *Historia Mathematica*, 25 :366–378, 1998.

- [Gui75] G. Guitel. *Histoire comparée des numérations écrites*. Flammarion, 1975.
- [Ifr94] G. Ifrah. *Histoire universelle des chiffres*. Robert Laffont (coll. Bouquins), Paris, 1994.
- [Mul95a] J. M. Muller. Algorithmes de division pour microprocesseurs : illustration à l'aide du "bug" du pentium. *Technique et Science Informatiques*, 14(8), octobre 1995.
- [Mul95b] J. M. Muller. Ordinateurs en quête d'arithmétique. *La Recherche*, juillet 1995.
- [Wil95] M. Wilkes. *Computing perspectives*. Morgan Kaufman Publishers, 1995.