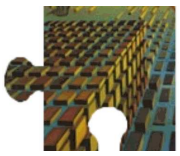
Implémentation de l'opérateur ADD2

David Defour, Bernard Goossens



DALI

Digits, Architectures et Logiciels Informatiques

Équipe de Recherche DALI

Laboratoire LP2A, EA 3679
Université de Perpignan

Implémentation de l'opérateur ADD2

David Defour et Bernard Goossens 3 décembre 2004

Rapport de recherche N° RR2004-03

Université de Perpignan
52 avenue Paul Alduy, 66860 Perpignan cedex, France
Téléphone : +33(0)4.68.66.20.64
Télécopieur : +33(0)4.68.66.22.87
Adresse électronique : dali@univ-perp.fr

Implémentation de l'opérateur ADD2

David Defour et Bernard Goossens

3 décembre 2004

Résumé

Cet article présente une implantation d'une unitée flottante capable d'effectuer deux additions flottantes consécutives. Cet opérateur prend trois nombres flottants double précision en entrée et retourne l'arrondi de la somme/soustraction de ces trois nombres. Cette unité propose pour l'unique arrondi final l'un des quatres modes d'arrondi exigés par la norme IEEE-754.

Mots-clés: Instructions, ILP, FMA, ADD2

Abstract

This paper presentsan implementation of a floating point unit able to perform two floating point additions with one rounding. This operator take two floating point input numbers in double precision arithmetic and return as result hte sum or subtrcation of this three numbers.

Keywords: Instructions, ILP, FMA, ADD2

Table des matières

1 Introduction

Le standard IEEE pour l'arithmétique flottante [6] a été développé pour standardiser les calculs entre ordinateurs. Lors de la création de cette norme, les principaux opérateurs flottants étaient alors l'addition, la multiplication, la division et la racine carrée. De façon à avoir un résultat identique d'une machine à une autre, la norme à imposée la façon d'arrondir les résultats produit par ces opérations. Mais depuis la définition de cette norme, un nouvel opérateur est apparu : le *Fused Multiply and Add* ou FMA capable d'une addition et d'une multiplication. La particularité de cet opérateur est de n'avoir qu'un seul arrondi final pour la multiplication et l'addition, ce qui le rend non conforme avec la norme IEEE754. Cependant il permet d'avoir des résultats plus précis et c'est une des raisons pour lequel son comportement sera normalisé dans la future révision de la norme flottante.

Le fait que les multiplications étaient souvent associées à des additions (produit matriciel, évaluation de polynome) à conduit à l'intégration du FMA dans les processeurs. Une récente étude [2] a été conduite sur les programmes issus du benchmarks SPEC2000fp, sensé être représentatifs d'application flottantes. Cette étude montre que l'opérateur ADD2 effectuant 2 additions flottantes consecutives est un bon candidat, après le FMA, pour accéler les calculs flottants. Il permet, comme le FMA, aussi d'avoir des calculs plus précis et plus rapides [11]. Cependant il n'existe à notre connaissance aucune proposition d'implémentation de cet opérateur.

Nous présentons dans une première section l'opérateur le plus proche du ADD2 : le FMA. La deuxième section introduira les mathématiques nécessaires à notre proposition pour l'implémentation du ADD2. La description de l'implémentation fera l'objet de la troisième section.

Nous noterons que de nombreux détails ne sont pas présentés dans cet version courte, mais apparaitrons dans la version finale.

2 Le FMA

L'apparition du FMA dans un processeur du commerce date de 1984 avec le RS/6000 de chez IBM [9]. Depuis, le constructeur IBM propose cette operation dans tout les descendants du RS/6000 : cela va des PowerPC utilisés par Apple dans ses Macs, au z990 [3] en passant par le S/390 G5 [12].

Depuis l'expérience d'IBM, de nombreux autre constructeurs ont intégré cette opération dans leurs processeurs. SGI a par exemple décliné cette opération sous deux formes différentes. Le MIPS R8000 [5] implemente le FMA en une seule instruction et un seul arrondi final. À l'inverse, son successeur, le MIPS R10000 [15] propose une instruction *UnFused Multiply ADD (uFMADD)* qui correspond à multiplication et une addition en une seule instruction mais avec un arrondi après la multiplication et un arrondi après l'addition. Cette dernière instruction qui ne nécessite pas de matériel spécifique, contrairement qu FMA, est également présente dans le SPARC64 V [10] de chez Sun.

Le FMA apparait également dans le HP PA 8000 [4] de chez Hewlett Packard. L'Itanium 1 et 2, qui est le fruit d'une collaboration entre Intel et Hewlett Packard, a hérité de deux de ces opérateurs pouvant opérer en parallèle sur le format double étendu d'Intel.

Un défaut important lié à cet opérateur est d'être peu supporté par les langages de programmation. Le langage C [1] est à notre connaissance le seul langage de programmation de haut niveau offrant l'instruction $fma(a, b, c) = a \times b + c$. Ce manque de support est en partie lié au manque de standard sur la façon d'évaluer certaine expression comme $a \times b + c \times d$.

XXXXXXXXXXXXXXXXXXX Comme nous pouvons l'observer sur le tableau XXX, les constructeurs ont plutot mis l'accent sur la latence que sur le débit. XXXXXXXXXXXXXXXXXXX

2.1 Implantation d'un FMA

Le FMA est le seul opérateur flottant prennant 3 opérandes en entrées. C'est pour cette raison que nous allons décrire les différentes étapes pour l'évaluation de $A + B \times C$. Nous avons pris pour model le FMA utilisé dans le PowerPC 604e [7] sans nous soucier du découpage pour pipeliner cet opérateur.

Tout d'abord, les données A, B et C sont chargées depuis leurs registres. Les exposants des trois opérandes sont ensuites utilisées pour calculer l'alignement de l'opérande A.

Les mantisses de B et C sont préparées (encodage de booth en base 4) pour être envoyées dans le multiplieur (53×53 bits). Ensuite, les produits partiels sont additionnés à travers un arbre de Wallace

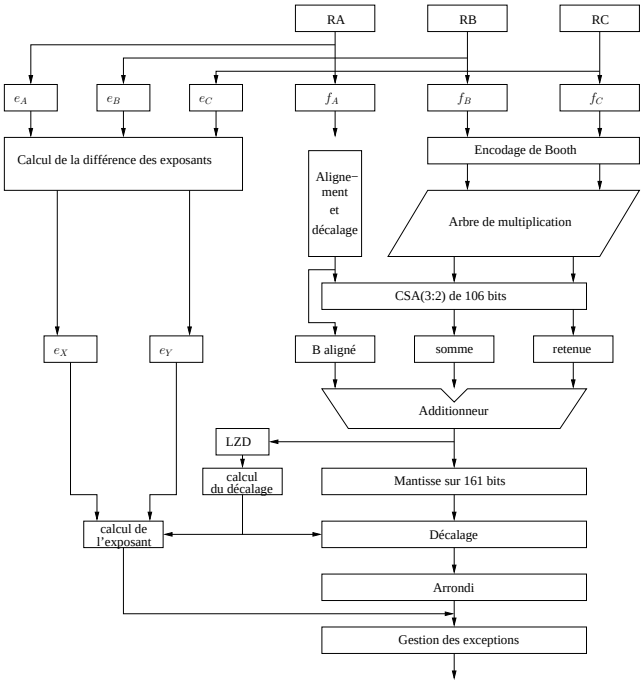


Fig. 1 – Le FMA du PowerPC 604e

composé de cellule CSA. Le résultat final est stoké à l'aide d'une somme et d'une retenue sur 106 bits.

En même temps que l'addition des produits partiels, les exposants des opérandes sont comparées pour calculer le décalage à effectuer sur A. Le résultat du décalage permet d'obtenir une mantisse A' sur 161 bits. Les 106 bits de poids faible de A' sont additonnés aux résultats codés sur 106 bits de la multiplication à l'aide d'un étage de CSA à 3 entrées, 2 sorties.

Le résultat de l'addition sur 106 bits au format Carry Save est ensuite envoyé dans un additionneur 161 bits en complément à 1 bits. Les 55 bits de poids fort de A' sont additionnés à cette étape pour obtenir le résultat final. Cet additionneur admet une particularité, puisque les 55 bits de poids fort de A' sont additionnés avec 0. Par conséquent, les 55 bits de poids fort de cet additionneur sont réalisée à l'aide de cellule d'incrémentation.

Le résultat sur 161 bits de l'addition précédente est utilisée pour compter le nombre de zéro de tête. Le nombre de zéro est utilisé pour décaler la mantisse du résultat et mettre à jour l'exposant du résultat final. Enfin le résultat est arrondi et la gestion des exceptions est effectuées.

Les différentes étapes que nous venons de décrire sont représentées dans la figure 1.

3 Nombres de port

Dans une unité flottante de processeur généraliste, le multiplieur est l'élément le plus couteux en surface suivit de près par le banc de registre [7]. La taille du banc de registre est fonction de la taille d'un registre et cette taille varie approximativement avec le carré du nombre de ses ports.

Les opérateurs à trois opérandes, comme le FMA ou le ADD2, nécessitent trois ports de lecture et un port d'écriture. Ils ont donc besoin d'un port de lecture supplémentaire par rapport aux opérateurs classiques. La surface du banc de registre s'en trouve donc augmentée. Cependant si l'on considère la surface globale de l'unité flottante (registres + unités), il est plus intéressant d'avoir un FMA au lieu d'une unité d'addition et de multiplication séparé [8]. De façon à réduire la taille du banc de registre, nous avons étudier si il était pertinent de partager certains ports sur les opérateurs fusionnés pour diminuer la taille de l'unité flottante totale.

Nous nous sommes basés sur le comportement des programmes qui composent les SPEC2000fp. Ces programmes ont été compilés avec le compilateur Compaq Alpha avec les optimisations de l'option (-

fast). Pour mener notre étude, nous avons développé des outils capables d'analyser de façon statique le code assembleur généré pour en extraire des propriétés sur les intructions et les registres utilisées.

Un premier outil détecte les instructions flottantes candidates pour être fusionnées en ADD2, FMA ou tout autre combinaison. En particulier, il marque les endroits où aparaissent deux additions flottantes consécutives dont le résultat de la première n'est utilisée que par la deuxième. Ces endroits correspondent à des instructions ADD2. À partir de ces résultats, un deuxième outil analyse si parmi les trois registres d'entrées, deux sont identiques. Les résultats de cette étude sont contenus dans le tableau 1. De façon à pouvoir comparer les résultats pour l'opérateur ADD2, nous avons également inclu les résultats d'autre opérateur.

À la lecture de la première colonne du tableau 1, il apparait clairement qu'il n'est pas interessant pour l'opérateur ADD2 de partager deux de ces ports. En effet, en moyenne seul 15% des opérateurs ADD2 ont deux opérandes identiques parmi trois. Toutefois cette remarque n'est pas valable pour le FMA puisque la moitiés des opérateurs ont deux opérandes identiques. Il serait donc pertinent pour l'opérateur FMA d'enviseager un accès au banc de registre ne nécessitant que deux ports de lecture.

Programme	$a+b+c$	$a+b\times c$	$a\times(b+c)$	$a\times b\times c$
168.wupwise	5.4	53.1	7.3	4.8
171.swim	37.8	38.8	41.3	23.3
172.mgrid	22.8	36.8	36.3	0.0
173.applu	20.1	50.2	31.2	23.6
177.mesa	4.4	38.4	9.6	16.3
178.galgel	3.4	70.4	4.5	7.1
179.art	16.5	82.6	5.1	7.4
183.equake	16.4	56.3	14.1	28.8
188.ammp	14	36.1	22.2	27.2
189.lucas	16.3	28.0	33.7	11.6
191.fma3d	14.8	42.2	14.4	12.9
301.apsi	11.2	25.2	26.0	13.1
Moyenne	15.3	46.5	20.5	14.7

TAB. 1 – Proportion d'opérateurs fusionnés, dont aux moins deux des trois opérandes utilisent la même donnée.

4 Mathématique de l'additionneur

Dans cette section nous décrivons l'algorithme que nous proposons pour l'opérateur ADD2. Nous utiliserons les notations que celles utilisées par Seidel et Even dans [13]. En revanche nous ne décrirons pas la façon de gérer les cas où l'une des entrées représente un dénormalisé, un 'NaN' ou un infini.

4.1 Présentation générale

Soit (s_a, e_a, f_a), (s_b, e_b, f_b) et (s_c, e_c, f_c) la représentation des trois opérandes flottantes comprenant respectivement leurs signes, leurs exposants et leurs mantisses. Soit les valeurs SOP_1 et SOP_2 qui déterminent si la première et la deuxième opération sont des additions (0) ou des soustractions (1). L'opération que nous souhaitons réaliser est donc la suivante :

$$\begin{aligned} ADD2(a,b,c) &= A(-1)^{SOP_1}B(-1)^{SOP_2}C \\ &= rnd((-1)^{s_a}\cdot 2^{e_a}\cdot f_a + (-1)^{SOP_1+s_b}\cdot 2^{e_b}\cdot f_b + (-1)^{SOP_2+s_c}\cdot 2^{e_c}\cdot f_c) \end{aligned}$$

Soit $\delta_{ab} = e_a - e_b$, $\delta_{ac} = e_a - e_c$ et $\delta_{bc} = e_b - e_c$ les différences d'exposants. Ces différences d'exposant permettent de réordonner les opérandes en grande ($G = (s_G, e_G, f_G)$), moyenne ($M = (s_M, e_M, f_M)$) et petite ($P = (s_P, e_P, f_P)$) opérandes de la façon suivante :

Conditions			G	M	P	δ_1	δ_2
$\delta_{ab} \geq 0$	$\delta_{bc} \geq 0$	$\delta_{ac} \geq 0$					
1	1	X	A	B	C	δ_{ab}	δ_{bc}
1	0	X	A	C	B	δ_{ac}	δ_{bc}
1	X	0	C	A	B	δ_{ac}	δ_{ab}
0	X	0	C	B	A	δ_{bc}	δ_{ab}
0	X	1	B	A	C	δ_{ab}	δ_{ac}
0	0	X	B	C	A	δ_{bc}	δ_{ac}

Pour tenir compte de l'opération (addition ou soustraction), lors du réordonnancement les signes des opérandes B et C sont modifiés de la façon suivante : $s_b \oplus SOP_1$ et $s_c \oplus SOP_2$. δ_1 représente le décalage à effectuer sur la mantisse de l'opérande M et δ_2 le décalage sur P. Le résultat de l'opération ADD2 est égal à :

$$ADD2(a,b,c) = rnd\left((-1)^{s_G} \cdot 2^{e_G} \cdot ((f_G + (-1)^{SGN_M} \cdot (f_M \cdot 2^{-|\delta_1|})) + (-1)^{SGN_P} \cdot (f_P \cdot 2^{-|\delta_2|}))\right)$$

avec $SGN_M =$

Les différentes étapes du calcul de la somme des trois opérandes A, B et C sont les suivantes :

1. Calcul de la différence des exposants δ_{ab}, δ_{bc} et δ_{ac}
2. Réordonnancement des opérandes en G, M et P
3. Calcul du décalage à effectuer :
   - Limitation du décalage de M par $\delta_{limM} = min(\alpha_M, |\delta_1|)$ avec $\alpha_M \geq 56$
   - Limitation du décalage de P par $\delta_{limP} = min(\alpha_P, |\delta_2|)$ avec $\alpha_P \geq 109$
4. Alignement des mantisses de M et P :
   - $f_0 = f_M \cdot 2^{-|\delta_{limM}|}$
   - $f'_0 = f_P \cdot 2^{-|\delta_{limP}|}$
5. Négation éventuelle des mantisses M et P alignées :
   - $f_1 = (-1)^{SGN_M} \cdot f_0$
   - $f'_1 = (-1)^{SGN_P} \cdot f'_0$
6. Calcul de la première addition $f_2 = f_G + f_1$
7. Calcul de la deuxième addition $f'_2 = f_2 + f'_1$
8. Normalisation et calcul de l'arrondi

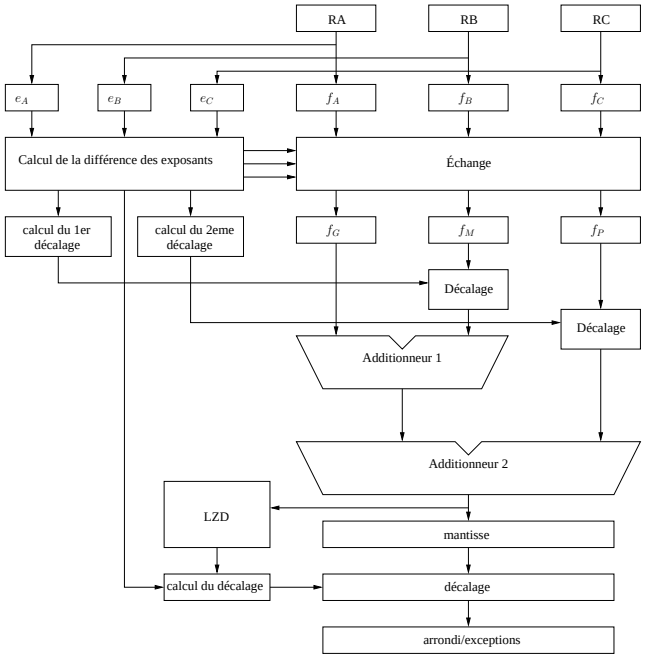
Dans cette implémentation le temps nécessaires pour les étapes 4, 6, 7 et 8 est logarithmique en la taille des entrées.

4.1.1 Calcul de la différence entre les exposants

Les valeurs δ_1 et de δ_2 représente les différences entre les opérandes. Dans un premier temps, le signe de ces valeurs est utilisé pour déterminer l'ordre des opérandes (G, M et P).

Dans un second temps, ces valeurs sont utilisées pour évaluer $\delta_{limM} = min(\alpha_M, |\delta_1|)$ et $\delta_{limP} = min(\alpha_P, |\delta_2|)$. La valeur α_M est définit comme supérieure où égale aux 53 bits de poids fort de la mantisse plus 1 bit de garde, 1 bit d'arrondi et 1 bit pour prévenir l'éventuelle propagation de retenue provenant de l'addition de M et P, ce qui correspond à 56. De façon identique, la valeur α_P est définit comme supérieure où égale aux 56 bits précédents plus les 53 bits pour la deuxième mantisse, ce qui représente 109 bits.

Les valeurs absolue de δ_{limM} et de δ_{limP} sont utilisées pour déterminer le décalage à effectuer sur les opérandes M et P. Dans le cas où δ_{limM} est égale à zéro et que l'opération entre G et M, ou entre M et P, corresponde à une soustraction ($(-1)^{SGN_M} = -1$ ou $(-1)^{SGN_P} = -1$), le signe du résultat la deuxième addition est utilisée pour mettre à jour le signe du résultat final.



4.1.2 Les additionneurs

Dans l'implémentation proposée, les 3 mantisses correctement alignées sont additionnées deux à deux. Les opérandes f_G et f_1 sont additionnées avec l'équivalent d'un additionneur à propagation de retenue sur 109 bits pour produire f_2. Les opérandes f_2 et f_1' sont additionnées avec l'équivalent d'un additionneur à propagation de retenue sur 162 bits pour produire f_2'. Ces additionneurs peuvent être implémentés à l'aide d'additionneur en complément à 1.

La position relative de la mantisse f_1 alignée sur 110 bits par rapport à f_G implique que les 56 bits de poids faibles de f_1 sont mathématiquement ajoutés à 0. Il est donc possible de réaliser le premier additionneur à l'aide d'un simple additionneur sur 53 bits, les 56 bits de poids faible étant directment envoyés vers les 56 bits de poids faible du résultat. Cette propriété est représentée par le schéma suivant :

$$\begin{aligned} f_G &= \overbrace{\text{x.xxx ... xxx}}^{53\ bits}\overbrace{\text{000 ... 000}}^{53+3\ bits} \\ f_1 &= \text{x.xxx ... xxxxxx ... xxx} \end{aligned}$$

$$\begin{aligned} f_2 &= \overbrace{\text{x.xxx ... xxxxxx ... xxx}}^{109\ bits}\overbrace{\text{000 ... 000}}^{53\ bits} \\ f_2 &= \text{x.xxx ... xxxxxx ... xxxxxx ... xxx} \end{aligned}$$

Nous noterons qu'un raisonnement similaire peut être appliqué pour le deuxième additionneur sur 162 bits qui peut être réalisé à l'aide d'un additionneur sur 109 bits. Le détail de ces deux additionneurs est présenté dans la figure 2

4.1.3 Prénormalisation

Le résultat intermédiaire sur 161 bits peut représenter un nombre entre $[0, 6)$, ce qui correspond à l'addition de trois mantisses comprisent entre $[0, 2)$. Par conséquent, un éventuel décalage à droite d'un bit devra être effectué sur le résultat intermédiaire. Cependant ce décalage peut être éffectué en parallèle avec le comptage du nombre de zéro de tête.

4.1.4 Arrondis

La norme IEEE-754 définit quatre modes d'arrondi : vers $+\infty$, vers $-\infty$, vers 0 et au plus près. Ils existe plusieurs solutons pour implémenter le calculde l'arrondi en matériel [14]. Par exemple, il est

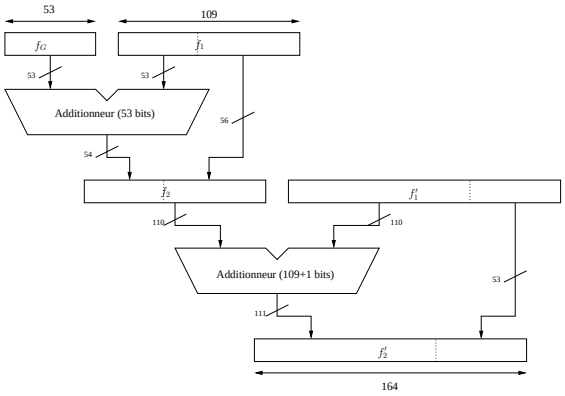


FIG. 2 – Fonctionnement des deux additionneurs

possible d'arrondir le résultat en utilisant la troncature. Cette méthode est basée sur l'addition d'une constante dépendant du mode d'arrondi, au résultat final avant la troncature. Cette constante est égale à 0 pour l'arrondi vers 0, à $2^{-52} - 2^{-53}$ pour l'arrondi vers l'infini, et à 2^{-53} pour l'arrondi au plus près.

5 Conclusion et perspectives

L'opérateur ADD2 a la propriété de réduire d'une facon similaire au FMA, la charge matérielle. Cette réduction s'applique sur les registres ou le cache d'instruction. Il améliore également la qualité et l'efficacité des logicielles numériques où la précision est un critère important. Nous avons donc proposé les grandes lignes d'une implémentation de l'opérateur ADD2 effectuant deux opérations flottantes consécutives. Pour cela, nous avons tout d'abord montré que trois port de lecture était nécessaire pour cet opérateur. Nous avons ensuite donné les éléments mathématiques nécessaire à son implémentation. Enfin nous avons terminé sur la description de certaine subtilité simplifiant cet opérateur, notemment au niveau des deux additionnneurs. Mais il nous maintenant à analyser les performances de cet opérateur en terme de délai et à étudier le gain en surface par rapport à deux additionneurs flottants séparés.

Cet opérateur présente une structure similaire aux implémentations d'opérateur FMA. Il serait donc intéressant d'étudier comment modifier les FMA actuels pour les tranformer en opérateurs polyvalents.

Références

[1] C : ISO/IEC 9899 revision of the standardization of the C language, known as C99, 2002.

[2] D. Defour. Collapsing dependent floating point operations. Research Report 02, DALI Research Team, LP2A, University of Perpignan, France, 52 avenue Paul Alduy, 66860 Perpignan cedex, France, december 2004. Submitted to IMACS 2005.

[3] G. Gerwig, E. M. Schwarz, J. Haess, C. A. Krygowski, B. M. Fleischer, and M. Kroener. The ibm eserver z990 floating-point unit. *IBM J. Res. Develop.*, 48(3/4) :311–322, May/July 2004.

[4] C. Heikes and G. Colon-Bonet. A dual floating-point coprocessor with fmac architecture. In *ISSCC Dig. Tech papers*, pages 354–355, February 1995.

[5] P.Y.T. Hsu. Design of the tfp microprocessor. *IEEE Micro*, 14(2) :22–23, April 1994.

[6] IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Standard, Std 754-1985,* New York, 1985.

[7] R.M. Jessani and M. Putrino. Comparison of single- and dual-pass multiply-add fused floating-point units. *IEEE Trans. on Computers*, 47(9) :927–937, September 1998.

[8] D. Lopez, J. Llosa, E. Ayguade, and M. Valero. Impact on performance of fused multiply-add units in aggressive vliw architectures. In *Proceedings of the 1999 International Conference on Parallel Processing*, pages 22–29. IEEE Computer Society, 1999.

[9] R. K. Montoye, E. Hokenek, and S. L. Runyon. Design of the ibm risc system/6000 floating-point execution unit. *IBM J. Res. Develop.*, 34(1) :59–70, 1990.

[10] A. Naini, A. Dhablania, W. James, and D. Das Sarma. 1-ghz hal sparc64 dual floating point unit with ras features. In *Proceedings of the 15th IEEE symposium on computer arithmetic (ARITH'01)*, 2001.

[11] T. Ogita, S. M. Rump, and S. Oishi. Accurate sum and dot product. *SIAM J. Sci. Comput.* to appear.

[12] E. M. Schwarz and C. A. Krygowski. The s/390 g5 floating-point unit. *IBM J. Res. Develop.*, 43(5/6) :707–721, September/November 1999.

[13] Peter-M Seidel and Guy Even. On the design of fast ieee floating-point adders. In *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, Los Alamitos, CA, 2001. IEEE Computer Society Press.

[14] P.M. Seidel. *On the design of IEEE compliant floating point units and their quantitative analysis.* PhD thesis, University of the Saarland, Germany, december 1999.

[15] K.C. Yeager. The mips r10000 superscalar processor. *IEEE Micro*, 16(2) :28–40, March 1996.