JOURNAL OF VLSI SIGNAL PROCESSING, ??, 1–13 (??)

A New Euclidean Division Algorithm for Residue Number Systems

JEAN-CLAUDE BAJARD, LAURENT-STÉPHANE DIDIER
Laboratoire d'Informatique de Marseille
CMI, *Université de Provence,*
39 rue Joliot-Curie, 13453 Marseille Cedex - FRANCE

JEAN-MICHEL MULLER
CNRS, *Laboratoire de l'Informatique du Parallélisme,*
46 Allée d'Italie, 69364 Lyon Cedex 07 - FRANCE

Received ??; Revised ??

Editors: ??

Abstract. We propose a new algorithm and architecture for performing divisions in residue number systems. Our algorithm is suitable for residue number systems with large moduli, with the aim of manipulating very large integers on a parallel computer or a special-purpose architecture. The two basic features of our algorithm are the use of a high-radix division method, and the use of a floating-point arithmetic that should run in parallel with the modular arithmetic.

1. Introduction

The context of this work is the manipulation of huge numbers (say, more than one thousand bits). This topic concerns various fields, e.g., computer algebra, cryptography, and algorithmic geometry. Residue number systems (abbreviated as RNS) are good number representations for these domains [1, 2]. We propose a new division algorithm for Residue Number Systems. Our algorithm has been designed for applications where manipulation of *huge numbers* is required. Conventional RNS VLSI architectures [3, 4] can be easily adapted in order to use this algorithm.

A Residue Number System (RNS) uses a set of *moduli* $(m_1, ..., m_n)$ that are relatively prime integers (i.e., $\mathrm{GCD}(m_i, m_j) = 1$ if $i \neq j$). A number X is represented by the residues

$$X \equiv x_i \ (\mathrm{modulo}\, m_i), \quad 0 \leq x_i < m_i.$$

The very origin of such systems is quite old: it goes back to the well-known *Chinese Remainder Theorem* (CRT) [5]. In an RNS, additions, subtractions and multiplications are straightforwardly performed in parallel (provided that no overflow occurs), using separate ALU's: this allows division-free RNS computations to be quickly completed [6, 7, 8]. Unfortunately, divisions and comparisons look difficult to perform in a residue number system [9, 10].

In the following, the set $(m_1, ..., m_n)$ of relatively prime integers is called the *RNS base*, and to simplify we assume that the m_i's are *prime numbers*.

The CRT shows that for any n-tuple $(x_1, ..., x_n)$, $0 \leq x_i < m_i$, there exists an unique integer X, $0 \leq X < M = \prod_{i=1}^{n} m_i$, such that $(x_1, ..., x_n)$ represents X. The CRT also gives an algorithm that computes X.

In 1989, D. Gamberger [11, 12] proposed a new algorithm for performing divisions in an RNS. Then, Lu and Chiang proposed an RNS division algorithm based on the combination of a classical division method and a parity checking method [13, 14]. More recently Hitz and Kaltofen proposed a division algorithm based on Newton's method [15].

In Section 2, we introduce a "weighted representation", that makes it possible to know the first significant digits and the order of magnitude of a number represented in an RNS. We also give an architecture for computing this representation from the RNS representation.

Using this "weighted representation", we present a new algorithm for RNS division in section 3. This algorithm gives the quotient and the remainder of the Euclidean division of two RNS numbers.

In Section 4, we analyze this algorithm and its performances.

In the last section, we compare our method to recently published algorithms.

2. Weighted RNS arithmetic

As outlined above, our algorithm uses a "weighted arithmetic", based on the combination of an RNS arithmetic and a floating-point-like arithmetic. The basic idea behind this — keeping the order of magnitude of RNS numbers — is not new. Similar ideas have been previously suggested [3, 10], in order to detect overflows. The new aspect is that we use this for comparisons and quotient-digit estimations.

We introduce a *Floating-Point Like* (FPL) notation which is associated with the residue notation and that makes it possible to know the order of magnitude of a number from its RNS representation.

2.1. The FPL representation

Our Floating-Point Like representation is deduced from the following theorem.

Theorem 1. *Let $(m_1, ..., m_n)$ be an RNS base. Let $\beta \geq 2$ and $\mu \geq 1$ be integers such that*

$$\beta^{\mu-1} \leq m_i < \beta^{\mu}, i = 1 \ldots n.$$

We assume that $M = \prod_{i=1}^{n} m_i$ satisfies

$$0 \leq X < M \times \beta^{-1}.$$

Let X be an integer, whose RNS representation is $(x_1, ..., x_n)$ in the RNS-base $(m_1, ..., m_n)$. We assume that the x_i's are represented in radix β. If we know X_{exp} such that

$$\beta^{-\mu+\ell+1} \leq \frac{X \times \beta^{X_{exp}}}{M} < \beta^{-1} \qquad (1)$$

with $\ell = \lceil \log_\beta n \rceil + 1$, then we can compute a $\mu - \ell$-digit approximation X_{frac} of $\frac{X \times \beta^{X_{exp}}}{M} \times \beta^{\mu}$, using 2μ digit integers only. This approximation X_{frac} satisfies:

$$X_{frac} \leq \frac{X \times \beta^{X_{exp}+\mu}}{M} < X_{frac} + \beta^{\ell} \qquad (2)$$

■

In Theorem 1, X_{exp} is a rather small number ($X_{\text{exp}} < n \times \mu$), such that $X \times \beta^{X_{\text{exp}}}$ is close to M. The integer X_{exp} indicates the order of magnitude of X relatively to M: the larger is X, the smaller is X_{exp}. Remember that X_{frac} is a μ-digit number. We have:

$$X_{\text{frac}} \times \beta^{-X_{\text{exp}}} \simeq X \left(\frac{\beta^{\mu}}{M}\right) \qquad (3)$$

Therefore $(X_{\text{exp}}, X_{\text{frac}})$ can be viewed as a (weighted) "floating-point representation" of X.

Definition 1. The couple $(X_{\text{exp}}, X_{\text{frac}})$ is called the *Floating-Point Like* (FPL) representation of the RNS number X (X_{exp} is the "exponent" and X_{frac} is the "fraction" of this representation).

There may be several values of $(X_{\text{exp}}, X_{\text{frac}})$ that satisfy the requirement of the above definition. The most accurate representation is the one with the largest value of the fraction – exactly as in conventional floating point arithmetic – and consequently with the largest value of X_{exp}.

2.2. Computation with FPL arithmetic

The FPL representation is manipulated in parallel with the RNS representation, using

the usual floating-point algorithm for addition, and a slightly modified one for multiplication. In order to compute $X \times Y$, $\left(X_{\text{frac}} \times \beta^{-X\text{exp}}\right)\left(Y_{\text{frac}} \times \beta^{-Y\text{exp}}\right)$ is first computed using the usual floating-point algorithm (that is, multiplication of the fractions and addition of the exponents), then the result is multiplied by a floating-point approximation of $\frac{M}{\beta^{\mu}}$ to take into account the "weight" $\frac{\beta^{\mu}}{M}$ that appears in (3)).

Generally, the FPL representations suffice for comparing two numbers. However, if the numbers that are being compared are very close, or if, after many computations, the FPL representations becomes too inaccurate. Then it is necessary to "refresh" them (i.e., to re-compute them from the RNS representations). This reconstruction is performed when the error on the FPL representations becomes so large that a comparison is impossible. A way to detect this is to implement an interval arithmetic with the FPL representations. In this case this reconstruction can be performed in two steps. Firstly, a "good" X_{exp} is determined. Secondly, the corresponding X_{frac} is computed. In the following, we show how to construct and how to "refresh" the FPL representation of an RNS number.

2.3. Construction of the FPL representation

In order to compute or to refresh the FPL representation of an RNS number X, the Mixed-Radix System associated with the RNS-base $(m_1, ..., m_n)$ can be used [5], but this solution seems intrinsically sequential. The formula that appears in the proof of the CRT looks much more parallelizable. This relation is:

$$X \equiv \sum_{i=1}^{n} x_i \times (M_i)_{m_i}^{-1} \times M_i \pmod{M} \qquad (4)$$

with $M_i = \frac{M}{m_i}$ and $(M_i)_{m_i}^{-1} \times M_i \equiv 1 \pmod{m_i}$.

We can notice that,

$$\sum_{i=1}^{n} x_i \times (M_i)_{m_i}^{-1} \times M_i = \left(\sum_{i=1}^{n} x_i \times \frac{(M_i)_{m_i}^{-1}}{m_i}\right) \times M$$

So, $\frac{X}{M}$ is the fractional part of:

$$\left(\sum_{i=1}^{n} x_i \times \frac{(M_i)_{m_i}^{-1}}{m_i}\right)$$

This construction is characterized by the fact that all the computations are performed with integers. Since the evaluation is done modulo M, we just have to manipulate the *fractional part* of each term $x_i \times \frac{(M_i)_{m_i}^{-1}}{m_i}$, and the fractional part of the sum $\sum_{i=1}^{n} x_i \times \frac{(M_i)_{m_i}^{-1}}{m_i}$ in order to compute X_{frac}.

2.3.1. Proof of Theorem 1: construction of X_{frac}

We want to perform this computation with μ-digit integers. During this computation only the first μ digits of the fractional part are taken into account.

According to the conditions of Theorem 1, we know X_{exp} such that,

$$\beta^{-\mu+\ell+1} \leq \frac{X \times \beta^{X\text{exp}}}{M} < \beta^{-1}$$

Then there is at least one significant digit in the first $\mu - \ell$ digits of the fractional part of $(X \times \beta^{X\text{exp}})/M$. We note $\tilde{X} = X \times \beta^{X\text{exp}}$ and $(\tilde{x}_1, ..., \tilde{x}_n)$ the RNS representation of $\tilde{X}$.

We can first remark that for each integer i, $1 \leq i \leq n$, we have

$$\beta^{-\mu} < \frac{(M_i)_{m_i}^{-1}}{m_i} < 1$$

Define Ω_i as the integer constituted by *the first 2μ digits of the fractional part of* $\frac{(M_i)_{m_i}^{-1}}{m_i}$:

$$\Omega_i = \left\lfloor \frac{(M_i)_{m_i}^{-1}}{m_i} \times \beta^{2\mu} \right\rfloor$$

$$\frac{(M_i)_{m_i}^{-1}}{m_i} = 0 \bullet \underbrace{\boxed{\mu\text{-digit number}}\boxed{\mu\text{-digit number}}}_{\Omega_i \text{ is a } 2\mu\text{-digit number}} \ldots$$

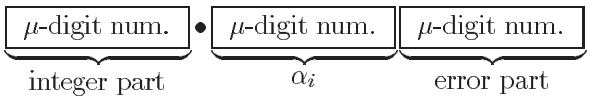
We have,

$$\frac{(M_i)_{m_i}^{-1}}{m_i} \times \beta^{2\mu} - 1 < \Omega_i \leq \frac{(M_i)_{m_i}^{-1}}{m_i} \times \beta^{2\mu} \qquad (5)$$

The terms Ω_i are pre-computed and stored constants. Now, let us multiply the μ-digit integer $\tilde{x}_i$ by Ω_i. The result is a 3μ-digit integer. We note α_i the integer whose digits are the first μ digits of the fractional part of $\tilde{x}_i \times \Omega_i \times \beta^{-2\mu}$:

$$\alpha_i = \left\lfloor \tilde{x}_i \times \Omega_i \times \beta^{-\mu} \right\rfloor - \left\lfloor \tilde{x}_i \times \Omega_i \times \beta^{-2\mu} \right\rfloor \times \beta^{\mu}$$

$\tilde{x}_i \times \Omega_i \times \beta^{-2\mu} =$



We have,

$$\tilde{x}_i \times \Omega_i \times \beta^{-\mu} - 1 < \alpha_i + \left\lfloor \tilde{x}_i \times \Omega_i \times \beta^{-2\mu} \right\rfloor \times \beta^{\mu} \leq \tilde{x}_i \times \Omega_i \times \beta^{-\mu}$$

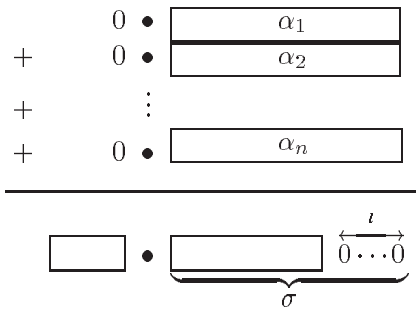
From (5), we deduce,

$$\begin{aligned} \tilde{x}_i \times \frac{(M_i)^{-1}_{m_i}}{m_i} \times \beta^{\mu} - \tilde{x}_i \times \beta^{-\mu} - 1 \\ < \alpha_i + \left\lfloor \tilde{x}_i \times \Omega_i \times \beta^{-2\mu} \right\rfloor \times \beta^{\mu} \\ \leq \tilde{x}_i \times \frac{(M_i)^{-1}_{m_i}}{m_i} \times \beta^{\mu} \end{aligned} \tag{6}$$

Define σ as the integer whose digits are the first $\mu - \ell$ digits of the fractional part of $\sum_{i=1}^{n} \alpha_i$ followed by ℓ zeros, that is:

$$\sigma = \left\lfloor \sum_{i=1}^{n} \alpha_i \times \beta^{-\ell} \right\rfloor \times \beta^{\ell} - \left\lfloor \sum_{i=1}^{n} \alpha_i \times \beta^{-\mu} \right\rfloor \times \beta^{\mu}$$

The least ℓ significant digits of σ are set to 0 in order to compensate for the error committed when computing $\sum_{i=1}^{n} \alpha_i$. This truncation suffices to satisfy (2).



Thus,

$$\sum_{i=1}^{n} \alpha_i - \beta^{\ell} < \sigma + \left\lfloor \sum_{i=1}^{n} \alpha_i \times \beta^{-\mu} \right\rfloor \times \beta^{\mu} \leq \sum_{i=1}^{n} \alpha_i \tag{7}$$

So, we deduce from (6), (7) and (4) that:

$$X \times \beta^{\mu} - \sum_{i=1}^{n} \tilde{x}_i \times \beta^{-\mu} \times M - n \times M < \sigma \times M \leq X \times \beta^{\mu}$$

Thus,

$$X \times \beta^{\mu} - 2 \times n \times M < \sigma \times M \leq X \times \beta^{\mu}$$

Using the definition of ℓ we can conclude that,

$$X \times \beta^{\mu} - \beta^{\ell} \times M < \sigma \times M \leq X \times \beta^{\mu}$$

This gives:

$$\sigma \leq \frac{X \beta^{\mu}}{M} < \sigma + \beta^{\ell} \tag{8}$$

Remark With regard to equations (2) and (8), if $\sigma \neq 0$ then we can take $X_{\text{frac}} = \sigma$. But if $\sigma = 0$, this means that X is small compared to M ($X \leq M\beta^{-\mu+\ell+1}$). In such a case, we have to start the process again with an increased value of X_{exp} that makes $\tilde{X}$ closer to M.

2.3.2. Construction of X_{exp} If X is small compared to M, then σ is too small to be computed accurately, and we need to make X larger. Since the order of magnitude of X is unknown, we have to proceed in a stepwise fashion: we multiply it (in the RNS system) by $\beta^{\mu-\ell}$ and try to convert it from the RNS to the FPL system at each step, until σ becomes significant. Of course $\mu - \ell$ is added to X_{exp} at each step. After this, σ is significant and the order of magnitude of X is known.

As a matter of fact, even if $\sigma \neq 0$, if σ is small compared to β^{μ}, it may be advantageous to multiply (in the RNS system) X by β^{i}, where i is such that $\beta^{\mu-1} \leq \sigma \times \beta^{i} < \beta^{\mu}$, and to start the process again, to get a much more accurate FPL representation. The worst case takes $\left\lceil \frac{\log_{\beta} M}{\nu - \ell} \right\rceil$ steps to compute the correct X_{exp}, where

$$\beta^{\nu} \leq M < \beta^{\nu+1}$$

Example 1.

We consider the base:
$\{65437, 65447, 65449, 65479, 65497\}$

$M = 1202097201816031191837293$ and $\mu = 16$ in radix 2.

$$M_i = \begin{cases} M_0 &= 1837029817711739 8289 \\ M_1 &= 1836749128021194 5419 \\ M_2 &= 1836693000375912 8357 \\ M_3 &= 1835851497145697 3867 \\ M_4 &= 1835346965228989 4069 \end{cases}$$

$$\Omega_i = \begin{cases} \Omega_0 &= 3062536088 \\ \Omega_1 &= 2923664766 \\ \Omega_2 &= 2335264193 \\ \Omega_3 &= 3837390181 \\ \Omega_4 &= 726046657 \end{cases}$$

Assume $X = 12822058820000 84821$. We have

$$X_{RNS} = \{40715, 64825, 36443, 60989, 5521\}$$

First step $X_{\text{exp}} = 0$. This gives

	Integer part	α_i	Err.
$x_0 \times \Omega_0$:	1110001011001111	11101100000110111 1	...
$x_1 \times \Omega_1$:	10101100010111111	10010111110010011 1	...
$x_2 \times \Omega_2$:	1001101011001100	11010011101001111 0	...
$x_3 \times \Omega_3$:	1101010011011011	01011011000000010 1	...
$x_4 \times \Omega_4$:	1110100101	01001101011011101 1	...
Sum :	10	11111111111110000	
		σ	Err.

Finally, according to the previous remark, we get $\sigma = 0$ with $X_{\text{exp}} = 0$. This is an insignificant value because X is small compared to M. X_{exp} should be larger. Note that ($X_{\text{frac}} = 65520$,$X_{\text{exp}} = 0$) corresponds to the FPL representation of the number 1201803721053869074847098.

We will add $\mu - \ell = 12$ to X_{exp} at each step until σ is significant.

Second step $X_{\text{exp}} = 12$. This gives

$\tilde{X} = X \times 2^{X\text{exp}} = 5251915292672347426816$
$\tilde{X}_{RNS} = \{35164, 4721, 46808, 8559, 17551\}$

	Integer part	α_i	Err.
$\tilde{x_0} \times \Omega_0$:	1100001111100011	11000100111011011	...
$\tilde{x_1} \times \Omega_1$:	1100100011011	01011000101011111	...
$\tilde{x_2} \times \Omega_2$:	1100011011010100	01111110111000001 1	...
$\tilde{x_3} \times \Omega_3$:	1110111011111 0	00100100001101111 1	...
$\tilde{x_4} \times \Omega_4$:	1011100101101	11101100101111100 0	...
Sum :	11	00000001000110000	
		σ	Err.

$\sigma = 272$

Even if this value is significant, it is not the most accurate. The final step is performed with a more accurate exponent $X_{\text{exp}} = 18$.

Third step

$\tilde{X} = X \times 2^{X\text{exp}} = 336122578731030235316224$
$X_{RNS} = \{25638, 40356, 50507, 23944, 9815\}$

	Integer part	α_i	Err.
$\tilde{x_0} \times \Omega_0$:	1000111011010 01	00111011110010011 1	...
$\tilde{x_1} \times \Omega_1$:	1101011010011111	00010101111110100 0	...
$\tilde{x_2} \times \Omega_2$:	1101011010001 01	10111000010001110 1	...
$\tilde{x_3} \times \Omega_3$:	1010011100100 01	00001110000000111 1	...
$\tilde{x_4} \times \Omega_4$:	11001111011	00101111100001000 0	...
Sum :	1	01000111100 10000	
		σ	Err.

We finally obtain the FPL representation of X: $X_{\text{frac}} = 18320$ and $X_{\text{exp}} = 18$. We can verify according to Theorem 1 that:

$$X_{\text{frac}} \leq \frac{X \times \beta^{X\text{exp}+\mu}}{M} < X_{\text{frac}} + \beta^{\ell}$$

with

$$\frac{X \times \beta^{X\text{exp}+\mu}}{M} = 18324 \text{ and } X_{\text{frac}} + \beta^{\ell} = 18336$$

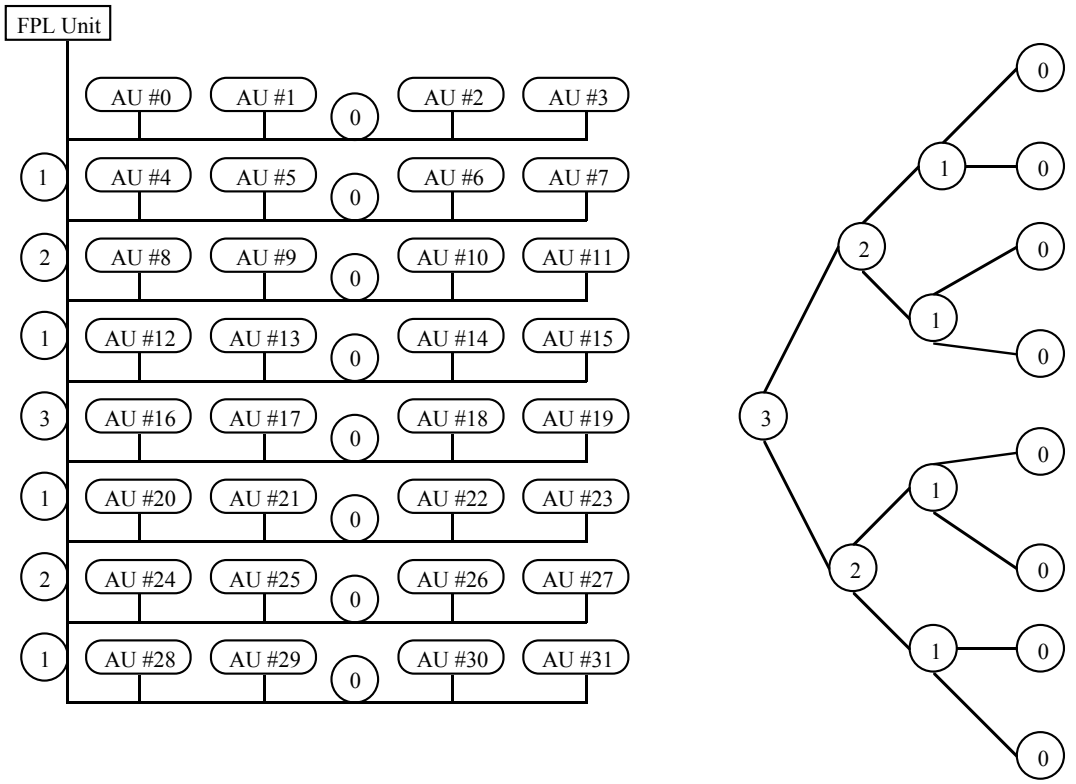


Fig. 1. Architecture of the implementation

2.4. An architecture for implementing the refreshment

The implementation uses n μ-digit Arithmetic Units (AU's) and an FPL unit, which are connected to each other with a μ-digit bus composed of sub-buses connected by gates, to realize of a binary tree for the evaluation of σ and X_{frac} in an $O(\log_2 n)$ computation time (figure 1).

Each AU has its own registers and memory that contain m_i, Ω_i and variables like x_i, α_i or the partial value of σ. Such a unit is a μ-digit adder with a control part that can perform specific additions to compute α_i and σ (additions with no overflow control: the overflows correspond to the integer part), and addition and multiplication modulo(m_i). The control part of each unit can be commanded by a single sequencer with instructions in a ROM for the different operations.

So the minimal functionalities of an AU-unit are:

- 7 μ-digit Registers for the variables m_i, Ω_i, x_i, α_i and buffers.
- One $\mu \times \mu$ multiplier, which computes the higher and the lower parts of the product in two μ-digit registers.
- One μ-digit adder

So we can propose an architecture similar to the normalizer part of the VLSI architecture presented in [3].

All the manipulated numbers are positive, so it is easy to consider only the least significant part knowing that the first significant digit must be positive. To have an idea of the size (which is $O(n \times \mu)$), a Borrow-Save implementation (radix $\beta = 2$ and digits in $\{-1, 0, 1\}$) with $\mu = 16$ and $n = 64$ (which gives $M \approx 2^{1000}$) uses around $50,000$ transistors for the arithmetic part. So it is possible to implement other operations in hardware, for example a modular multiplication [17].

The FPL-unit controls the bus routing and schedules the computations performed by all the units. With regard to the iteration step the FPL-unit commands the communications between the sub-buses (figure 1).

3. An RNS division algorithm

Let X_{RNS} be the RNS representation of X. From two numbers X_{RNS} and Y_{RNS}, we wish to compute to numbers Q_{RNS} and R_{RNS} satisfying:

$$X_{RNS} = Q_{RNS} \times Y_{RNS} + R_{RNS}$$

with $0 \leq R_{RNS} < Y_{RNS}$.

Our algorithm is derived from a classic high radix division algorithm, where Q and R are respectively initialized to 0 and X. The main iteration of this algorithm consists in determining $D = D_m \times \beta^{D_e}$, (where D_m can be viewed as a high-radix quotient digit), so that $X = (D+Q) \times Y + R$. After each step D is added to the quotient Q, and $D \times Y$ is subtracted from R. This iteration is performed until the remainder is less than the divisor. All the computations in the division process are performed in RNS.

The determination D and the comparison between R and Y are performed using the FPL representation of the operands. The FPL-unit broadcasts D_m and D_e to the AU-units that construct D_{RNS}. Each residue d_i is computed in the corresponding AU-unit. Next, the RNS computation of Q_{RNS} and R_{RNS} is performed. Each AU-unit determines the corresponding residues q_i and r_i.

The last comparison is performed in the mixed-radix system. We denote $X_{MRS} = (a_1, ..., a_n)$ the representation of X in the Mixed-Radix system associated with the RNS-base $(m_1, ..., m_n)$. That is to say,

$$X = a_1 + a_2 m_1 + a_3 m_1 m_2 + \cdots + a_n m_1 m_2 m_3 \ldots m_{n-1}$$

with $0 \leq a_i \leq m_i - 1$.

We first give the algorithm, and we will give comments, examples and proofs afterwards.

1. **Pre computing:**
2. Construction of $(X_{\text{exp}}, X_{\text{frac}})$ and $(Y_{\text{exp}}, Y_{\text{frac}})$
3. Construction of Y_{MRS}
4. **Initialization:**
5. $R_{RNS} \leftarrow X_{RNS}$
6. $(R_{\text{exp}}, R_{\text{frac}}) \leftarrow (X_{\text{exp}}, X_{\text{frac}})$
7. $Y_{ff} \leftarrow Y_{\text{frac}} + \beta^{\ell}$
8. $Q_{RNS} \leftarrow 0$
9. **loop:**
10. **while** $R_{\text{exp}} < Y_{\text{exp}}$ or $(R_{\text{frac}} \geq Y_{ff}$ and $R_{\text{exp}} = Y_{\text{exp}})$ (in FPL-unit)
11. **if** $R_{\text{frac}} \neq 0$ **then** (in FPL-unit)
12. (D_m, D_e) is computed in the FPL-unit
13. The FPL-unit broadcasts (D_m, D_e) to the AU-units
14. The AU-units compute D_{RNS}
15. $R_{RNS} \leftarrow R_{RNS} - Y_{RNS} \times D_{RNS}$ (in AU-units)
16. $Q_{RNS} \leftarrow Q_{RNS} + D_{RNS}$ (in AU-units)

17. **if** $Y_{\exp} - R_{\exp} \geq \mu - \ell - 2$ **then** (in FPL-units)
18. $R_{\exp} \leftarrow R_{\exp} + \mu - \ell - 2$ (in FPL-units)
19. **else** $R_{\exp} \leftarrow Y_{\exp}$ (in FPL-units)
20. **Reconstruct** R_{frac}
21. **end:**
22. **if** $R_{MRS} \geq Y_{MRS}$ **then**
23. $R_{RNS} \leftarrow R_{RNS} - Y_{RNS}$ (in AU-units)
24. $Q_{RNS} \leftarrow Q_{RNS} + 1_{RNS}$ (in AU-units)

Remarks

- All the RNS operations are performed in the AU-units.
- All the tests and operations with FPL notations are performed in the FPL-unit.
- During the computation, the FPL-unit controls the system.
- In order to get an upper bound on D, we use Y_{ff}, which is an upper bound of Y_{frac} according to (2). Thus R remains positive.
- The reconstruction of R_{frac} is performed according to the method depicted in the previous section.
- The test $R_{MRS} \geq Y_{MRS}$ is performed using a conventional mixed-radix conversion of R_{RNS}, and a mixed-radix comparison with the precomputed value of Y_{MRS} [5].

Computation of D_m and D_e in the FPL unit The evaluation of two numbers D_m and D_e such that $D = D_m \times \beta^{D_e}$ is performed in the FPL-unit using the following algorithm, that requires integer operations only.

1. Compute R_n such that $\beta^{-1} < \dfrac{R_{\text{frac}} \times \beta^{R_n}}{Y_{ff}} < \beta$ (Normalization)
2. **If** $Y_{\exp} - R_{\exp} - R_n \geq 0$ **then**
3. $R_s \leftarrow \inf(Y_{\exp} - R_{\exp} - R_n, \mu - 1)$
4. $D_m \leftarrow \left\lfloor \dfrac{R_{\text{frac}} \times \beta^{R_n + R_s}}{Y_{ff}} \right\rfloor$ (Division of the normalized fraction)
5. $D_e \leftarrow Y_{\exp} - R_{\exp} - R_n - R_s$ (Computation of the exponent)
6. **else**
7. $D_m \leftarrow 1$
8. $D_e \leftarrow 0$

Remarks Firstly, D_m is a $(\mu - 1)$-digit number and then $D_m \equiv D_m$ modulo (m_i) for each $1 \leq i \leq n$. Secondly, D is at least equal to 1 since $R > Y$. This remark is useful for the last iteration.

Computation of D modulo (m_i) in the i-th Arithmetic Unit (AU) The FPL unit broadcasts (D_m, D_e) to each arithmetic unit. The ith unit computes d_i such that $D \equiv d_i$ modulo(m_i). D_e is less than $n \times \mu$. The residues d_i can be computed in different ways.

If $n \times \mu < \beta^{\mu}$ then D_e can be represented by a μ-digit number. The latter assumption is often satisfied: for example, if $\beta = 2$, $\mu = 16$ that corresponds to $n < 2^{12}$. The residue of β^{D_e} modulo m_i can be read in a local table if μ is not too large. For example, if $\beta = 2$ and $\mu = 16$ then the size of the look-up tables is 2Kb. The computation of d_i can be performed in one step.

Otherwise, D_e can be decomposed in radix β^θ. In this case we can use $\log_{\beta^\theta}(n \times \mu)$ look-up tables of size $\beta^\theta \times \mu$. Thus the residue of β^{D_e} modulo m_i is computed with at most $\log_{\beta^\theta}(n \times \mu)$ look-up table accesses and $\log_{\beta^\theta}(n \times \mu) - 1$ modular multiplications. If β^θ is close to μ, then d_i is computed with at most $\log_\mu n + 1$ look-up table accesses and $\log_\mu n + 1$ modular multiplications.

Example 2.

We consider the base:

$$\{65437, 65447, 65449, 65479, 65497\}$$

which gives $\mu = 16$, in radix 2. In this example the Euclidean division of X by Y is performed in RNS:

$$\begin{array}{lcr} X & = & 12822058820000084821 \\ X_{\text{frac}} & = & 18320 \\ X_{\text{exp}} & = & 18 \\ Y & = & 135 \\ Y_{\text{frac}} & = & 17376 \\ Y_{\text{exp}} & = & 71 \\ Y_{ff} & = & 17392 \end{array}$$

$(X_{\text{frac}}, X_{\text{exp}})$ and $(Y_{\text{frac}}, Y_{\text{exp}})$ are the representations of X and Y in the FPL system. Note that the high-radix division is performed with Y_{ff} — which is an upper bound of the divisor — instead of Y_{frac} in order to obtain a positive remainder.

Step 1

$Y_{\text{exp}} - R_{\text{exp}} = 53$ and $FD = 1$

$\frac{R_{\text{frac}}}{Y_{ff}} = 9487804182776850.0$

$D = 9487685836079104$
$Q = 9487685836079104$
$R = 1368294129405781$
$Rk = 28$
$R_{\text{frac}} = 20016$

Step 2

$Y_{\text{exp}} - R_{\text{exp}} = 43$ and $FD = 1$

$\frac{R_{\text{frac}}}{Y_{ff}} = 10123194453341.4980$

$D = 10122701045760$
$Q = 9497808537124864$
$R = 1729488228181$
$Rk = 38$
$R_{\text{frac}} = 25904$

Step 3

$Y_{\text{exp}} - R_{\text{exp}} = 33$ and $FD = 1$

$\frac{R_{\text{frac}}}{Y_{ff}} = 12794024015.131556$

$D = 12793675776$
$Q = 9497821330800640$
$R = 2341998421$
$Rk = 48$
$R_{\text{frac}} = 35936$

Step 4

$Y_{\text{exp}} - R_{\text{exp}} = 23$ and $FD = 1$

$\frac{R_{\text{frac}}}{Y_{ff}} = 17332855.168353$

$D = 17332224$
$Q = 9497821348132864$
$R = 2148181$
$Rk = 58$
$R_{\text{frac}} = 33744$

Step 5

$Y_{\text{exp}} - R_{\text{exp}} = 13$ and $FD = 1$

$\frac{R_{\text{frac}}}{Y_{ff}} = 15894.137994$

$D = 15894$
$Q = 9497821348148758$
$R = 2491$
$Rk = 68$
$R_{\text{frac}} = 40064$

Step 6

$Y_{\text{exp}} - R_{\text{exp}} = 3$ and $FD = 0$

$\frac{R_{\text{frac}}}{Y_{ff}} = 18.428703$

$D = 18$
$Q = 9497821348148776$

$R = 61$
$Rk = 71$
$R_{\text{frac}} = 40064$

No extra correcting step is required, because R is less than Y, and

$$X = Q \times Y + R$$

with

$$\begin{array}{rclrcl} X & = & 1282205882000084821 & Y & = & 135 \\ Q & = & 9497821348148776 & R & = & 61 \end{array}$$

4. Analysis of the RNS division algorithm

This algorithm is very similar to high-radix, SRT-like, division algorithms. At each step, a new radix-$\beta^{\mu-\ell-2}$ digit of the quotient is computed. The difference lies in the fact that the remainders are computed in the Residue Number System and converted to the FPL system.

Theorem 2. *If $\mu > \ell + 2$ then from two numbers X_{RNS} and Y_{RNS}, Algorithm 0 gives Q_{RNS} and R_{RNS} such that:*

$$X_{RNS} = Q_{RNS} \times Y_{RNS} + R_{RNS}$$

with $0 \leq R_{RNS} < Y_{RNS}$

Proof of the "while" loop: R_{exp} decreases to Y_{exp}

Let us remind the notations:

$$\begin{array}{c} \beta^{X_e} \leq X < \beta^{X_e+1} \\ \beta^{Y_e} \leq Y < \beta^{Y_e+1} \\ \beta^{\nu} \leq M < \beta^{\nu+1} \\ \ell = \lceil \log_\beta n \rceil + 1 \end{array} \tag{9}$$

We have,

$$\frac{R}{M}\beta^{\mu+R_{\text{exp}}} - \beta^{\ell} < R_{\text{frac}} \leq \frac{R}{M}\beta^{\mu+R_{\text{exp}}}$$

and,

$$\frac{Y}{M}\beta^{\mu+Y_{\text{exp}}} \leq Y_{ff} < \frac{Y}{M}\beta^{\mu+Y_{\text{exp}}} + \beta^{\ell}$$

Thus,

$$\frac{R\beta^{\mu+R_{\text{exp}}} - M\beta^{\ell}}{Y\beta^{\mu+Y_{\text{exp}}} + M\beta^{\ell}} < \frac{R_{\text{frac}}}{Y_{ff}} \leq \frac{R\beta^{R_{\text{exp}}}}{Y\beta^{Y_{\text{exp}}}}$$

$$\frac{R - M\beta^{\ell-\mu-R_{\text{exp}}}}{Y + M\beta^{\ell-\mu-Y_{\text{exp}}}} < \frac{R_{\text{frac}}}{Y_{ff}}\beta^{Y_{\text{exp}}-R_{\text{exp}}} \leq \frac{R}{Y} \tag{10}$$

From $\beta^{\nu} \leq M < \beta^{\nu+1}$ we obtain,

$$\frac{R - \beta^{\nu+\ell-\mu-R_{\text{exp}}}}{Y + \beta^{\nu+\ell-\mu-Y_{\text{exp}}}} < \frac{R_{\text{frac}}}{Y_{ff}}\beta^{Y_{\text{exp}}-R_{\text{exp}}} \leq \frac{R}{Y}$$

and,

$$\frac{R\beta^{\nu-Y_e-1+\ell-\mu-Y_{\text{exp}}} + \beta^{\nu+\ell-\mu-R_{\text{exp}}}}{1 + \beta^{\nu-Y_{\text{exp}}-1+\ell-\mu-Y_e}} > R - \frac{R_{\text{frac}}}{Y_{ff}}\beta^{Y_{\text{exp}}-R_{\text{exp}}} \times Y \geq 0$$

$$\frac{\beta^{R_e+\nu-Y_e-2+\ell-\mu-Y_{\text{exp}}} + \beta^{\nu+\ell-\mu-R_{\text{exp}}}}{1 + \beta^{\nu-Y_{\text{exp}}-1+\ell-\mu-Y_e}} > R - \frac{R_{\text{frac}}}{Y_{ff}}\beta^{Y_{\text{exp}}-R_{\text{exp}}} \times Y \geq 0$$

From $\nu \geq R_e + R_{\text{exp}} + 1$ and if

$$Y_e + Y_{\text{exp}} = \nu - 1 \tag{11}$$

(that is possible with two FPL reconstructions) then,

$$\frac{\beta^{\nu+\ell-\mu-R_{\text{exp}}} + \beta^{\nu+\ell-\mu-R_{\text{exp}}}}{1 + \beta^{\ell-\mu}} > R - \frac{R_{\text{frac}}}{Y_{ff}}\beta^{Y_{\text{exp}}-R_{\text{exp}}} \times Y \geq 0$$

Thus

$$\beta^{\nu+\ell-\mu-R_{\text{exp}}+1} > R - \frac{R_{\text{frac}}}{Y_{ff}}\beta^{Y_{\text{exp}}-R_{\text{exp}}} \times Y \geq 0 \tag{12}$$

As we have,

$$\frac{R_{\text{frac}}}{Y_{ff}}\beta^{Y_{\text{exp}}-R_{\text{exp}}} \times (1 - \beta^{-\mu+1}) < D \leq \frac{R_{\text{frac}}}{Y_{ff}}\beta^{Y_{\text{exp}}-R_{\text{exp}}}$$

Eqn. (12) becomes,

$$\beta^{\nu+\ell-\mu-R_{\exp}+1} + \frac{R_{\text{frac}}}{Y_{ff}}\beta^{Y_{\exp}-R_{\exp}-\mu+1} \times Y > R - D \times Y \geq 0$$

$$\beta^{\nu+\ell-\mu-R_{\exp}+1} + R \times \beta^{-\mu+1} > R - D \times Y \geq 0$$

$$\beta^{\nu+\ell-\mu-R_{\exp}+1} + \beta^{R_e-\mu+2} > R - D \times Y \geq 0$$

we therefore obtain,

$$\beta^{\nu+\ell-\mu-R_{\exp}+2} > R - D \times Y \geq 0 \qquad (13)$$

Thus we have no overflow problems in $R_{RNS} \leftarrow R_{RNS} - Y_{RNS} \times D_{RNS}$ because $R - D \times Y \geq 0$. And, if $R_{\exp}+\mu-\ell-2 < Y_{\exp}$ then the next value of $R_{\exp}$ can be $R_{\exp}+\mu-\ell-2$, else $R_{\exp} = Y_{\exp}$. This confirms that $R_{\exp}$ decreases to $Y_{\exp}$.

When $Y_{\exp} = R_{\exp}$

As we have seen in (13), and considering (11) we deduce that just before the last iteration we have $R < \beta \times Y$ and $\frac{R}{Y} - 1 < D \leq \frac{R}{Y}$. Now the last iteration gives from (12),

$$\beta^{\nu+\ell-\mu=R_{\exp}+1} > R - \frac{R}{Y} \geq 0$$

and (13) becomes,

$$\beta^{R_e+\ell-\mu+2} + Y > R - D \times Y \geq 0$$

in other words for $\mu > \ell + 1$,

$$(\beta^{\ell-\mu+1} + 1) \times Y > R - D \times Y \geq 0$$

$$2 \times Y > R - D \times Y \geq 0 \qquad (14)$$

This requires at most another iteration with an exact comparison using a Mixed Radix number system.

4.1. Performance

To summarize, each iteration of the division algorithm requires:

- Communications:
  - The computation of $(R_{\text{frac}}, R_{\exp})$ needs $\log_2 n$ AU-to-AU communication steps for the construction of σ.
  - 2 broadcasts for the transmission of (D_m, D_e) from FPL to AUs
- Computations:
  - at most four μ-digit comparisons in the FPL-unit.
  - at most μ comparisons and shifts for the evaluation of R_n in the FPL-unit
  - four μ-digit additions, one μ-digit multiplication and one μ-digit integer division.
  - at most $\log_\mu n + 1$ μ-digit modular multiplications and $\log_\mu n + 1$ μ-digit table look-ups for the computation of the residues d_i in the AU-units.
  - two μ-digit modular additions and one μ-digit modular multiplication for the evaluation of R_{RNS} and Q_{RNS} in the AU-units.
  - one μ-digit comparison and one μ-digit addition to construct $R_{\exp}$ in the FPL-unit.

We can therefore assume that the complexity of one iteration is $O(\log n)$.

The number of steps is at most equal to $\frac{\nu}{\mu-\ell+2}$. Thus the cost of this loop is $O(n \log_2 n)$ with a small hidden constant. Finally the last step of our algorithm is done in time $O(n)$ [5]. This gives the following result.

Theorem 3. *The proposed RNS-division algorithm is a $O(n \log_2 n)$-time algorithm, with $O(n)$ space.*

5. Other recent algorithms

Comparison and division are difficult problems in RNS arithmetic. Many algorithms have been proposed in the literature to cope with these problems. In 1991 D. Gamberger presented an original RNS division algorithm without comparisons, whose execution time only depends on the value of the divisor [11]. Each iteration requires $2 \times n + 3$ modular steps, and often more than $\log_2(n)$ iterations. For example with moduli 43, 47, 53, 59 and 61, with a divisor close to 10^6 the maximum number of iterations is equal to 18. The execution time of our algorithm does not depend on the

value of the divisor, it only depends on the size of the difference between the dividend and the divisor. The worst case of our algorithm has a better execution time than the mean case of Gamberger's algorithm.

M. Lu and J.S. Chiang proposed an RNS division based on a classical high-radix division algorithm with comparisons performed using a parity checking [13, 14]. To have an efficient $O(n\log(n))$-time algorithm they use n tables. The size of each table is proportional to $m_i \times (\log_2(n \times M)$. For example, if $\nu = 1024 = 2^{10}$ then the size of the tables is close to 2^{32} bits. At each iteration, Lu and Chiang's algorithm computes a binary digit of the quotient. Thus, as our algorithm computes a radix-$2^{\mu-\ell-2}$-digit at each iteration, we must compare one iteration of our method to $\mu - \ell - 2$ iterations of M. Lu and J.S. Chiang's algorithm. In other words if the μ-digit integer division is performed faster than μ additions and $\mu\log(n)$ table look-ups, then we can assume that our algorithm is better. Moreover, our algorithm is not limited by the size of a table.

More recently, Hitz and Kaltofen have proposed an integer division in residue number systems, based on Newton iteration [15]. This algorithm uses an extended base and performs many conversions in a Mixed Radix System (MRS) and base transfers. The MRS conversion uses relation (4) of the CRT proof, that enables a $O(\log n)$ conversion time rather than the more classical linear algorithm [5]. The computation of (4) is done in MRS with $O(n^2)$ modular adders and an $O(n^2)$-size tree of modular multipliers. This algorithm divides two RNS numbers with an $O(n\log n)$ time complexity. The size complexity of our algorithm is $O(n)$ and enables the division of huge numbers (say, more than 1000 bits) in time $O(n\log n)$.

6. Conclusion

We have proposed an efficient algorithm for RNS division, the implementation of which is realistic. This algorithm is suited for the manipulation of huge numbers. The execution time of this algorithm is better than that of previously published algorithms, and it does not require large tables. We do not claim that our algorithm is attractive for applications such as computer cards — e.g. credit cards or phone cards — for such applications, other algorithms (see for instance [18]) looks more promising. The use of our algorithm to perform modular multiplications of huge numbers (i.e. a RNS multiplication followed by a RNS division) would be comparable to the use of modular multiplication algorithms proposed by N. Takagi [19, 18] or P. Kornerup [20] as soon as $\mu > \log_\beta n$.

References

1. D. Naccache, D. M'rairhi, W. Wolfowicz, and A. di Porto. Are crypto-acceleratos really inevitable? In *Advances in Cryptology, Eurocrypt'95*, pages 404–409, 1995.
2. H. Brönniman, I.Z. Emeris, V. Pan, and S. Pion. Computing exact geometric predicates using modular arithmetic with single precision. In *Proceedings of ACM Symposium on Computational Geometry*, Nice, 1997.
3. F. Barsi and E. Martinelli. A VLSI architecture for RNS with mi adders. *Integration, The VLSI Journal*, 11:67–83, 1991.
4. F. Barsi. Mod m arithmetic in binary systems. *Information Processing Letters*, 40:303–309, 1991.
5. D. Knuth. *The art of computer programming*, volume 2. Addison Wesley, 1973.
6. H.L. Garner. The residue number system. *IRE Transactions on Electronic Computers*, EC-8:140–147, 1959.
7. N.S Szabo and R.I. Tanaka. *Residue Arithmetic and its applications to computer Technology*. McGraw-Hill, 1967.
8. F.J. Taylor. A more efficient residue arithmetic implementation of the FFT. In *5th Symposium on Computer Arithmetic*. IEEE Computer Society Press, 1985.
9. G. Dimauro, S. Impedovo, and G. Pirlo. A new technique for fast number comparison in the residue number system. *IEEE Transactions on Computers*, 42(5):608–612, May 1993.
10. S. Kaushik. Sign detection in non-redundant residue number system with reduced information. In *6th Symposium on Computer Architecture*. IEEE Computer Society Press, 1983.
11. D. Gamberger. New Approach to Integer Division in Residue Number System. In P. Kornerup and D. Matula, editors, *proceedings of the 10th Symposium on Computer Arithmetic*, pages 84–91. IEEE Computer Society Press, 1991.
12. D. Gamberger. Incomplete specified numbers in residue number system - Defintion and applications. In M. D. Ercegovac and E. Swartzlander, editors, *proceedings of the 9th Symposium on Computer Arithmetic*, pages 210–215. IEEE Computer Society Press, 1989.
13. J.S. Chiang and M. Lu. A general division algorithm for residue number systems. In P. Kornerup and D. Matula, editors, *proceedings of the 10th IEEE*

Symposium on Computer Arithmetic, pages 76–83. IEEE Computer Society Press, June 1991.
14. M. Lu and J.S. Chiang. A novel division algorithm for the residue number system. *IEEE Transactions on Computers*, 41(8):1026–1032, August 1992.
15. M. A. Hitz and E. Kaltofen. Integer division in residue number systems. *IEEE Transactions on Computer*, 44(8):993, 1995.
16. D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–47, March 1991.
17. G. Alia and E. Martinelli. A VLSI modulo m multiplier. *IEEE Transactions on Computers*, 40(7):873–878, July 1991.
18. N. Takagi. A modular multiplication algorithm with triangle additions. In M.J. Irwin E.E. Swartzlander and G. Jullien, editors, *11th Symposium on Computer Artithmetic*, pages 272–276, Los Alamitos, CA, June 1993. IEEE Computer Society Press.
19. N. Takagi. A Radix-4 Modular Multiplication Hardware Algorithm Efficient for Iterative Modular Multiplications. In P. Kornerup and D. Matula, editors, *proceedings of the 10th Symposium on Computer Arithmetic*, pages 35–41. IEEE Computer Society Press, 1991.
20. P. Kornerup. High-radix modular multiplication for cryptosystems. In G. Jullien M.J Irwin, E. Swartzlander, editor, *11st IEEE Symposium on Computer Arithmetic*, pages 277–283. IEEE Computer Society Press, 1993.

Jean-Claude Bajard Jean-Claude was born in Saint-Etienne, France, in 1957. He received the Ph.D. degree in computer science in 1993 from the Ecole Normale Supérieure de Lyon.

He thought mathematics in high school from 1979 to 1990, and served as an assistant professor at Ecole Normale Supérieure de Lyon in 1993. He joined the Lab. LIM, Université de Provence, Marseille, France in 1993.

Dr. Bajard's research interests include computer arithmetic and VLSI design.

Laurent-Stéphane Didier

Laurent-Stéphane Didier was born in Marseille, France, in 1970. He received the M.S. degree in computer science from the Ecole Normale Supérieure in 1994 and the Ph.D. degree in computer science in 1998 from the Université de Provence. He is serving as assistant professor at Université de Provence.

His research interests are in computer arithmetic and computer architecture.

Jean-Michel Muller

Jean-Michel Muller was born in Grenoble, France, in 1961. He received the Engineer degree in applied mathematics and computer science in 1983, and the PhD in computer science in 1985, both from the Institut National Polytechnique de Grenoble, France. In 1986, he joined the CNRS (french national center for scientific research). He has been posted from 1986 to 1989 to Tim3-Imag laboratory, Grenoble, and then to LIP laboratory, Lyon. He teaches computer arithmetic in the Ecole Normale Supérieure de Lyon. His research interests include computer arithmetic and computer architecture. Dr Muller served as General Chairman of the 10th Symposium on Computer Arithmetic (Grenoble, France, June 1991), and Co-program chair of the 13th Symposium on Computer Arithmetic (Asilomar, California, July 1997). He is an associate editor of the IEEE Transactions on Computers since 1996.