Integer Division in Residue Number Systems

Markus A. Hitz and Erich Kaltofen

Abstract—**This contribution to the ongoing discussion of division algorithms for residue number systems (RNS) is based on Newton iteration for computing the reciprocal. An extended RNS with twice the number of moduli provides the range required for multiplication and scaling. Separation of the algorithm description from its RNS implementation achieves a high level of modularity, and makes the complexity analysis more transparent. The number of iterations needed is logarithmic in the size of the quotient for a fixed start value. With preconditioning it becomes the logarithm of the input bit size. An implementation of the conversion to mixed radix representation is outlined in the appendix.**

Index Terms—**Integer division, reciprocal, Newton iteration, extended residue number system, mixed radix conversion, base extension.**

I. Introduction

Szabo and Tanaka [9] have already devised two algorithms for general division in residue number systems (RNS). But it is only since 1980 that the number of papers dedicated to RNS division has started to increase. Chren [3] summarizes some of the previous efforts and discusses an improved version of Banerji et al.'s algorithm [2]. Most RNS division algorithms use some form of binary expansion for the quotient or the reciprocal. They are usually closely tied to their respective hardware implementation, making the complexity analysis difficult. An exception to this category is the work of Davida and Litow [4]. They give the complete analysis for an almost uniform logdepth division circuit. Another more recent result by Lu and Chiang [8] is based on comparison and binary search.

In our approach, we use an *extended* RNS which provides roughly the square of a normal RNS range for intermediate results. We try to avoid the term "redundant," because the additional residues are significant for several computational steps. We compute the (integer) reciprocal of the divisor with respect to the original range of the RNS, and, after multiplication by the dividend, use scaling to get the quotient (and remainder). The algorithm for the reciprocal is in the spirit of Aho, Hopcroft, and Ullman [1]. Gamberger [5] also uses an extended RNS. His division algorithm tries to find common divisors of the nominator and denominator, removing them iteratively by scaling. Davida and Litow have to increase the number of moduli in their algorithm to $2n^2 + 1$ for intermediate values, which probably makes their result impractical.

We only discuss implementation for unsigned RNS numbers. Adaption to symmetric RNS is not difficult, but destroys some of the clarity in the descriptions.

Manuscript received June 18, 1993; revised Apr. 5, 1994.

The authors are with the Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12189-3590; e-mail: {hitzm,kaltofen}@cs.rpi.edu.

IEEECS Log Number C95076.

II. Definitions

Let $m'_1, \ldots, m'_{2n} \in \mathbb{Z}$, pairwise relatively prime, such that

$$1 < m'_1 < m'_2 < \cdots < m'_{2n-1} < m'_{2n},$$

where $n \in \mathbb{Z}$, $n \geq 1$. We group these $2n$ *moduli* into two vectors of size n:

$$m_1 := m'_1,\ m_2 := m'_3,\ \ldots,\ m_n := m'_{2n-1},$$

and

$$m_{n+1} := m'_2,\ m_{n+2} := m'_4,\ \ldots,\ m_{2n} := m'_{2n}.$$

We call the RNS defined by those moduli an *extended* RNS, the RNS defined by the first n moduli *base* RNS, and the RNS defined by the moduli with indices $n + 1$ to $2n$ *extension* RNS. The respective *range* for the base and the extension RNS is:

$$M = \prod_{i=1}^{n} m_i, \qquad \overline{M} = \prod_{i=n+1}^{2n} m_i,$$

where the range for the entire extended RNS is $M\overline{M}$.

An integer X, $0 \leq X < M\overline{M}$ with residues $x_1 = X \bmod m_1$, $\ldots$, $x_{2n} = X \bmod m_{2n}$ will be represented in the extended RNS by:

$$[x_1, \ldots, x_n; x_{n+1}, \ldots, x_{2n}].$$

The representation in the associated *mixed radix system* (MRS) will be denoted by:

$$\langle v_1, \ldots, v_n; v_{n+1}, \ldots, v_{2n} \rangle,$$

where

$$X = \sum_{i=1}^{2n} v_i P_{i-1};$$

$$P_0 = 1, \quad P_i = m_1 m_2 \cdots m_i, \quad \text{for} \quad 1 \leq i < 2n,$$

$$\text{and} \quad 0 \leq v_i < m_i, \quad \text{for} \quad 1 \leq i \leq 2n.$$

We note the following properties:

1) M and $\overline{M}$ are relatively prime and $M < \overline{M}$, thus the mutliplicative inverse $M^{-1} \bmod \overline{M}$ exists. Furthermore the base part (left of the ';') of an extended RNS number represents the remainder modulo M.
2) For sufficiently large n and moduli of similar magnitude, the difference between M and $\overline{M}$ will be small. Otherwise the selection of the m_i from the m'_i can be adapted in order to balance the relative size without changing the first property.
3) Multiplication of two base RNS numbers (in the range M) performed in the extended RNS will never overflow because $M^2 < M\overline{M}$. The result can then be reduced to the base range using scaling by M.

III. The Division Algorithm

In this section we give a high level description of our division algorithm; in the next section we shall explain how to use basic RNS operations to implement it. "DIVREM" takes two integers X, $0 \le X < M$ and Y, $1 \le Y < M$, as input and returns the quotient $\lfloor X/Y \rfloor$, and the remainder X mod Y.

Algorithm DIVREM
Input: (X, Y)
Output: $(\lfloor X/Y \rfloor, X \bmod Y)$
begin
$Q \leftarrow \lfloor X * \text{RECIP}(Y)/M \rfloor$
$R \leftarrow X - Q * Y$
if $R < Y$ **then return** (Q, R)
else return $(Q + 1, R - Y)$
end.

One might notice that other then being smaller than M, we do not imply any restrictions on X and Y. DIVREM makes a call to RECIP (given below), which returns the reciprocal $\lfloor M/Y \rfloor$ of Y with respect to M. The following lemma explains the necessity for the correction step at the end of the algorithm.

LEMMA 1. *Algorithm DIVREM is correct.*

PROOF. Let $Z := \lfloor M/Y \rfloor$, i.e., $M = YZ + S$, or $Z = (M - S)/Y$, where $0 \le S < Y$. Then $\frac{XZ}{M} = \frac{X}{Y} - \frac{X}{M}\frac{S}{Y}$. We note that $X < M$ and $S < Y$, therefore $\frac{X}{M}\frac{S}{Y} < 1$. Thus either $\lfloor \frac{XZ}{M} \rfloor = \lfloor \frac{X}{Y} \rfloor$ or $\lfloor \frac{XZ}{M} \rfloor = \lfloor \frac{X}{Y} \rfloor - 1$. □

In "RECIP" we use Newton iteration to compute the reciprocal $\lfloor M/Y \rfloor$. This algorithm is similar to the one in Aho, Hopcroft, and Ullman [1]. However, their divide and conquer approach, where the number of valid bits doubles in each recursive step, is not suitable for RNS, because here we cannot access the high-order bit of the intermediate value. Still, the advantage of quadratic convergence is preserved in our algorithm. Applying the Newton iteration scheme

$$Z_{i+1} = Z_i - \frac{f(Z_i)}{f'(Z_i)},$$

to

$$f(Z_i) = \frac{M}{Z_i} - Y$$

with

$$f'(Z_i) = -\frac{M}{Z_i^2},$$

yields the recursion:

$$Z_{i+1} = Z_i + \frac{MZ_i - YZ_i^2}{M} = \frac{Z_i(2M - YZ_i)}{M}.$$

In RNS we have only integer division, so the final version becomes:

$$Z_{i+1} = \left\lfloor \frac{Z_i(2M - YZ_i)}{M} \right\rfloor = 2Z_i - \left\lceil \frac{YZ_i^2}{M} \right\rceil.$$

The algorithm for the reciprocal takes an integer Y, with $1 \le Y < M$, as input and returns $\lfloor M/Y \rfloor$:

Algorithm RECIP
Input: Y
Output: $\lfloor M/Y \rfloor$
begin
$Z_1 \leftarrow 0$
$Z_2 \leftarrow 2$
while $Z_1 \neq Z_2$ **do**
$Z_1 \leftarrow Z_2$
$Z_2 \leftarrow \lfloor Z_1 * (2M - Y * Z_1)/M \rfloor$
if $M - Y * Z_2 < Y$ **then return** Z_2
else return $Z_2 + 1$
end.

The necessity for the correction step at the end of the algorithm can be illustrated by the following example: for $M = 10$ and $Y = 3$, $\lfloor M/Y \rfloor = 3$, but $Z_2 = Z_1 = 2$. Lemmas 3 to 5 show the correctness of RECIP and indicate the number of iterations needed. The proofs in the remainder of this section may be skipped without loss of understanding.

LEMMA 2. *Subsequent approximations satisfy the following inequalities:*

$$\frac{Y}{M}\left(\frac{M}{Y} - Z_i\right)^2 \le \frac{M}{Y} - Z_{i+1} < \frac{Y}{M}\left(\frac{M}{Y} - Z_i\right)^2 + 1 \quad (1)$$

$$1 - \frac{Z_{i+1} + 2^i - 1}{M/Y} < \left(1 - \frac{Z_i + 2^{i-1} - 1}{M/Y}\right)^2 \quad (2)$$

PROOF. Equation (1) follows from the basic inequalities for the ceiling operator ($X \le \lceil X \rceil < X + 1$):

$$\begin{aligned} \frac{M}{Y} - Z_{i+1} &= \frac{M}{Y} - 2Z_i + \left\lceil \frac{YZ_i^2}{M} \right\rceil \\ &< \frac{M}{Y} - 2Z_i + \frac{YZ_i^2}{M} + 1 \\ &= \frac{Y}{M}\left(\frac{M}{Y} - Z_i\right)^2 + 1 \end{aligned}$$

The same argument is applied to prove the left side of (1). Also the inductive proof for inequality (2) makes use of the same properties of the ceiling operator:

$$\begin{aligned} 1 - \frac{Z_{i+1} + 2^i - 1}{M/Y} &= 1 - \frac{2Z_i - \left\lceil \frac{YZ_i^2}{M} \right\rceil + 2^i - 1}{M/Y} \\ &< 1 - \frac{2Z_i + 2^i - 1}{M/Y} + \frac{Z_i^2}{(M/Y)^2} + \frac{1}{M/Y} \\ &= 1 - \frac{2\left(Z_i + 2^{i-1} - 1\right)}{M/Y} + \frac{Z_i^2}{(M/Y)^2} \end{aligned}$$

$$\le 1-\frac{2\left(Z_i+2^{i-1}-1\right)}{M/Y}+\frac{\left(Z_i+2^{i-1}-1\right)^2}{(M/Y)^2}$$

$$=\left(1-\frac{Z_i+2^{i-1}-1}{M/Y}\right)^2 \qquad \square$$

LEMMA 3. *RECIP halts for any Y with* $1 \le Y < M$ *and is correct.*

PROOF. For $Z_1 = 2$ and $Y > \left\lfloor \frac{3M}{4} \right\rfloor$ the iteration stops with $Z_2 = Z_3 = 0$; for $\left\lfloor \frac{M}{2} \right\rfloor < Y \le \left\lfloor \frac{3M}{4} \right\rfloor$, it stops at $Z_2 = Z_3 = 1$. For the rest of the proof we suppose that $1 \le Y \le \left\lfloor \frac{M}{2} \right\rfloor \le \frac{M}{2}$. We can show by induction on i that $\frac{M}{Y} - Z_i \ge 0$, for $i = 1$:

$$\frac{M}{Y} - Z_1 \ge \frac{M}{M/2} - 2 = 0,$$

whereas

$$\frac{M}{Y} - Z_{i+1} \ge \frac{Y}{M}\left(\frac{M}{Y} - Z_i\right)^2 \ge 0$$

for $i > 1$, by the induction hypothesis and the left side of (1) in Lemma 2. Therefore $YZ_i \le M$ and

$$Z_{i+1} = \left\lfloor 2Z_i - \frac{YZ_i^2}{M} \right\rfloor = \left\lfloor Z_i\left(2 - \frac{YZ_i}{M}\right) \right\rfloor \ge Z_i.$$

Hence the sequence is monotonically increasing, and the halting condition $Z_{i+1} = Z_i = \left\lceil YZ_i^2/M \right\rceil$ will eventually be reached. Let $\rho = M/Y$, then we have the following quadratic inequality for Z_i at termination:

$$Z_i = \left\lceil \frac{Z_i^2}{\rho} \right\rceil < \frac{Z_i^2}{\rho} + 1, \quad \text{or} \quad Z_i^2 - \rho Z_i + \rho > 0,$$

which yields

$$\left(Z_i - \frac{\rho}{2}\right)^2 > \frac{1}{4}\left(\rho^2 - 4\rho\right).$$

For $\rho \ge 4$:

$$\rho^2 - 4\rho \ge \rho^2 - 8\rho + 16 = (\rho - 4)^2.$$

Therefore, the two solutions for Z_i are:

$$Z_i > \frac{\rho}{2} + \frac{1}{2}(\rho - 4) = \rho - 2,$$

and

$$Z_i < \frac{\rho}{2} - \frac{1}{2}(\rho - 4) = 2.$$

Being below the start value $Z_1 = 2$, the second solution is impossible. The first inequality states that upon reaching the halting condition, Z_i is equal either to $\lfloor M/Y \rfloor$ or $\lfloor M/Y \rfloor - 1$. It can easily be verified that RECIP halts for all $\rho < 4$ after at most two iterations (for $1 < \rho < 4/3$ it stops with $Z_2 = Z_3 = 0$, for $4/3 \le \rho < 2$ with $Z_2 = Z_3 = 1$, and for $2 \le \rho < 4$ with $Z_1 = Z_2 = 2$). $\square$

Since RECIP only depends on the fraction M/Y, Lemma 3 implies that for constant M/Y the algorithm finishes in a constant number of iterations. For the time complexity when M/Y is unbounded, we first determine the number of iterations it takes to get within one quarter of the exact quotient M/Y (Lemma 4). In a second stage (Lemma 5) we will decrease the difference to below 2, which is sufficient for the correction step at the end of RECIP.

LEMMA 4. Z_i *will be within one quarter of* M/Y *after* $O(\log(M/Y))$ *iterations.*

PROOF. For ease of argumentation, we choose $Z_1 \ge 30$ and assume M and Y such that $30 < M/Y$. The number of steps needed to reach a $Z_i \ge 30$ starting with $Z_1 = 2$ is always smaller than 10. For all cases $M/Y \le 30$ the iteration will stop earlier. Expanding (2) of Lemma 2 iteratively down to Z_1 yields:

$$1 - \frac{Z_i + 2^{i-1} - 1}{M/Y} < \left(1 - \frac{Z_1}{M/Y}\right)^{2^{i-1}} \tag{3}$$

Now let i such that

$$2^{i-1} \le \frac{6}{Z_1}\frac{M}{Y} + 1 < 2^i,$$

i.e.,

$$i > \log_2\left(\frac{6}{Z_1}\frac{M}{Y} + 1\right).$$

Then

$$\left(1 - \frac{Z_1}{M/Y}\right)^{2^{i-1}} = \underbrace{\left[\underbrace{\left(1 - \frac{Z_1}{M/Y}\right)^{\frac{6}{Z_1}\frac{M}{Y}+1}}_{< e^{-6}}\right]^{\frac{2^{i-1}}{\frac{6}{Z_1}\frac{M}{Y}+1}}}_{< e^{-3}}.$$

Note that

$$\frac{1}{2} < \frac{2^{i-1}}{\frac{6}{Z_1}\frac{M}{Y} + 1} \le 1,$$

and

$$\left(1 - \frac{1}{x}\right)^x < \frac{1}{e}$$

for all $x > 0$, where e is Euler's base of the natural logarithms. Finally, we multiply (3) by M/Y:

$$\frac{1}{e^3}\frac{M}{Y} > \frac{M}{Y}\left(1 - \frac{Z_1}{M/Y}\right)^{2^{i-1}} > \frac{M}{Y} - \left(Z_i + 2^{i-1} - 1\right),$$

$$> \frac{M}{Y} - \left(Z_i + \frac{6}{Z_1}\frac{M}{Y}\right),$$

thus

$$Z_i > \frac{M}{Y}\underbrace{\left(1 - \frac{1}{e^3} - \frac{6}{Z_1}\right)}_{\ge 0.7502\ldots} > \frac{3}{4}\frac{M}{Y}$$

or

$$\frac{M}{Y} - Z_i < \frac{1}{4}\frac{M}{Y}. \qquad \square$$

Let K such that $2^K \le M/Y < 2^{K+1}$. Using the right side of (1) from Lemma 2 once more, we now can enforce the starting conditions of Lemma 5:

$$\frac{M}{Y} - Z_{i+1} < \frac{Y}{M}\left(\frac{M}{Y} - Z_i\right)^2 + 1 < \frac{M}{Y}\left(\frac{1}{4}\right)^2 + 1$$
$$= \frac{1}{16}\frac{M}{Y} + 1 < 2^{K-3} + 2.$$

LEMMA 5. *If* $0 < M/Y - Z_{i+1} < 2^{K-3} + 2$, *it takes* $O(\text{loglog}(M/Y))$ *(additional) steps to get within a difference of* 2 *to the exact quotient* M/Y.

PROOF. We prove by induction on j that $M/Y - Z_{i+j} < 2^{K-3*2^{j-1}} + 2$, where the given condition for Z_{i+1} represents the base of the induction. By the induction hypothesis and the right side of (1) from Lemma 2, we get:

$$\frac{M}{Y} - Z_{i+j+1} < \frac{Y}{M}\left(\frac{M}{Y} - Z_{i+j}\right)^2 + 1$$
$$< 2^{-K}\left(2^{K-3*2^{j-1}} + 2\right)^2 + 1$$
$$= 2^{K-3*2^j} + 4*2^{-3*2^{j-1}} + 4*2^{-K} + 1$$
$$< 2^{K-3*2^j} + 2$$

Hence eventually $M/Y - Z_{i+j} < 3$ and $M/Y - Z_{i+j+1} < 2$, by applying (1) once more and observing that $Y/M < 1/30 < 1/9$. From the condition $2^{K-3*2^j} + 2 < 3$, we can deduce the number of iterations needed at most:

$$2^{K-3*2^j} < 1, \quad 2^j > K/3, \quad \text{or} \quad j > \log_2(K/3),$$

where

$$K \le \log_2(M/Y) < K + 1,$$

hence

$$K = \lfloor \log_2(M/Y) \rfloor,$$

and

$$j > \log_2(\lfloor \log_2(M/Y) \rfloor / 3). \qquad \square$$

IV. RNS Implementation

Here, we show how to implement DIVREM and RECIP using RNS operations. However, it is not the scope of this paper to discuss hardware realizations. Various models of computation were used so far to derive the complexity of basic RNS operations. More recently, a thorough analysis for $\mathcal{NC}^1$ circuits was carried out by Davida and Litow [4]. Unfortunately conversions were left out. We will show in the Appendix that conversion from RNS to mixed radix representation can be implemented in depth $O(\log n)$ using $O(n^2)$ RNS processor elements; by "RNS processor element" we mean a circuit for arithmetic or boolean operations, such as addition, multiplication or test for equality, modulo any of the m_i.

For the following, we assume that all numbers are already in extended RNS format. Otherwise, we have to do "base extension" from base RNS to extended RNS.

A. Base Extension

The conversion from base RNS to the extension RNS (or vice versa) amounts to a conversion to the associated MRS representation $\langle v_1, \ldots, v_n \rangle$ for

$$X = \sum_{i=1}^{n} v_i P_{i-1}$$

given by $[x_1, \ldots, x_n]$, with subsequent evaluation of the sum

$$\left(\sum_{i=1}^{n} v_i P_{i-1}\right) \bmod m_j$$
$$= \left(\sum_{i=1}^{n} (v_i \bmod m_j)(P_{i-1} \bmod m_j)\right) \bmod m_j,$$

for every modulus m_j with $n + 1 \le j \le 2n$. The constants $P_{i-1} \bmod m_j$ can be kept in a lookup table, while the products $v_i P_{i-1}$ are computed in parallel on $n \times n$ RNS processor elements. Finally the summation mod m_j is done in binary trees with $O(\log n)$ depth.

B. Test for "=" and "≠"

In RECIP, the halting condition for the Newton iteration is "$Z_2 = Z_1$." Two integers X and Y in RNS representation are equal if and only if they agree in each component: $x_i = y_i$. The overall result can be obtained from the individual tests by performing AND operations on a single bit in a binary tree with depth $O(\log n)$. If necessary, the result can be propagated back to the processor elements using the tree in the inverse direction. In the test for "not equal," AND is replaced by OR.

C. Division by M (Scaling)

In both DIVREM and RECIP the major operation is integer division by M. Let U be the intermediate result in extended RNS, before division by M ($U = X * \text{RECIP}(Y)$ or $U = Z_i * (2M - Y * Z_i)$), with residues $[u_1, \ldots, u_n; u_{n+1}, \ldots, u_{2n}]$. The left part of this vector represents the remainder $R = U \bmod M$. By performing base extension it can be converted into extension RNS representation. After subtracting $U \bmod M$ from U, we multiply by $M^{-1} \bmod \overline{M}$ to get the quotient Q in the extension RNS. Q has to be extended to base RNS by another base extension. The residues of $M^{-1} \bmod \overline{M}$ (denoted by $\overline{m}_{n+1}, \ldots, \overline{m}_{2n}$) can be precomputed. Thus, the sequence of operations is as follows:

$$
\begin{array}{rl}
\left[u_1, \ldots, u_n ; u_{n+1}, \ldots, u_{2n}\right] & \\
U \bmod M \rightarrow \text{base extension} & \\
; u'_{n+1}, \ldots, u'_{2n}\right] & \\
\hline
; t_{n+1}, \ldots, t_{2n}\right] & \ominus \\
M^{-1} \bmod \overline{M} ; \overline{m}_{n+1}, \ldots, \overline{m}_{2n}\right] & \\
\hline
Q ; q_{n+1}, \ldots, q_{2n}\right] & \otimes \\
\leftarrow \text{base extension} & \\
\left[q_1, \ldots, q_n ; q_{n+1}, \ldots, q_{2n}\right] &
\end{array}
$$

Although U is greater than M, the range of the extension RNS is sufficient because the result Q will always be smaller than M.

An alternate method works directly in MRS, avoiding the subtraction and multiplication in the extension RNS. However, because the mixed radix conversion uses $O(n^2)$ processor elements, the hardware requirement increases almost by a factor of 4, when $2n$ residues have to be converted instead of n. Let U have the *extended* mixed radix representation:

$$\langle v_1, \ldots, v_n; v_{n+1}, \ldots, v_{2n}\rangle, \qquad U = \sum_{i=1}^{2n} v_i P_{i-1}.$$

By definition, the products P_i with $n \le i < 2n$ are multiples of $M : P_i = MS_i$, where $S_n = 1$ and $S_i = m_{n+1} \cdots m_i$ for $n < i < 2n$. We now split the sum into

$$U = \sum_{i=1}^{n} v_i P_{i-1} + M \sum_{i=n+1}^{2n} v_i S_{i-1} = R + MQ,$$

where $R = U \bmod M$ and $Q = \lfloor U/M \rfloor$. Q is the desired result for the quotient, and has to be converted back to extended RNS by evaluating the sum for each modulus m_j $(1 \le j \le 2n)$, in the same way as described for base extension.

D. Comparison

For the correction step at the end of both algorithms, one comparison is necessary. Davida and Litow [4] describe an $\mathcal{NC}^1$ circuit for comparison of two RNS integers with size $O(b^2)$ and depth $O(\log b)$, where b is the input size in bits $(\lceil \log_2 M \rceil)$. Here, we want to show that some of the functional units for division by M can also be used for comparison (and overflow detection).

First we note that the numbers we want to compare are strictly smaller than M. If we subtract two such (nonnegative) integers X and Y the result Z will "underflow" whenever $X > Y$, which is equivalent to $Z = Y - X > M$ in extended RNS representation. After performing a base extension from base RNS to extension RNS, we test the result for equality with the residues $z_{n+1}, \ldots, z_{2n}$. If all components agree, we return "$X \le Y$," otherwise "$X > Y$."

While using the second method for division by M, we convert Z to (extended) mixed radix representation, and test wether every component $v_i = 0$ for $n < i \le 2n$.

From Lemma 4, Lemma 5, and the discussion of this section we conclude that:

THEOREM 1. *The division algorithm can be implemented in depth $O(\log n \log(M/Y))$ with $O(n^2)$ RNS processor elements.*

E. Speed up

Replacing $Z_1 = 2$ in RECIP by a start value which is "closer" to the result, the number of iterations can be reduced to $O(\text{loglog}(M/Y))$. In order to achieve this, we choose the nearest power of two by comparing Y in parallel against $\lfloor M/2^K \rfloor$ for $K = 1, \ldots, N$, where $2^N \le M < 2^{N+1}$. K is obtained by adding up the results of these comparisons (0 or 1) in a binary tree of height $\log_2 N = \log_2 \lfloor \log_2 M \rfloor$. The constants $\lfloor M/2^K \rfloor$ can be precomputed.

LEMMA 6. *With start value $Z_1 = 2^K$, where $\lfloor M/2^{K+1} \rfloor < Y \le \lfloor M/2^K \rfloor$, RECIP terminates after $O(\text{loglog}(M/Y))$ iterations.*

PROOF. If $\lfloor M/2 \rfloor < Y$ then $\lfloor M/Y \rfloor = 1$, and we are done. For $1 \le Y \le \lfloor M/2 \rfloor$, we first note that (for the chosen K):

$$\left\lfloor \frac{M}{2^{K+1}} \right\rfloor \le \frac{M}{2^{K+1}} < \left\lfloor \frac{M}{2^{K+1}} \right\rfloor + 1 \le Y \le \left\lfloor \frac{M}{2^K} \right\rfloor \le \frac{M}{2^K},$$

thus

$$Z_1 = 2^K \le \frac{M}{Y} < 2^{K+1}.$$

Hence

$$\frac{M}{Y} - Z_1 < \frac{1}{2}\frac{M}{Y},$$

and by Lemma 2:

$$\frac{M}{Y} - Z_2 < \frac{1}{4}\frac{M}{Y} + 1.$$

Finally

$$\frac{M}{Y} - Z_3 < \frac{1}{16}\frac{M}{Y} + \frac{1}{2} + \frac{Y}{M} + 1 < 2^{K-3} + 2$$

using $Y/M \le 1/2$). According to Lemma 4, we obtain the result in $O(\text{loglog}(M/Y))$ steps from here. □

With this modification, we can state the overall complexity as:

THEOREM 2. *The better start value reduces the depth of the division algorithm to $O(\log n \ \text{loglog}(M/Y) + \text{loglog}M)$, increasing the number of processor elements to $O(n \log M)$.*

PROOF. Lemma 6 and the scaling operation contribute $\log n \ \text{loglog}(M/Y)$ to the depth. The comparison step at the beginning can be implemented in MRS with $n \times N$ processor elements and depth $O(\log n + \log N)$. The conversion from RNS to MRS uses $O(n^2)$ processors and has depth $O(\log n)$. Because $N = \lfloor \log_2 M \rfloor > n$, $\log M$ will dominate n. □

V. CONCLUDING REMARKS

- Our RNS division algorithm is simple, and robust for a wide range of start values.
- Although not being the optimum regarding circuit depth, it offers a balance between space and time complexity.
- The high level description and analysis of the algorithm gives the advantage of modular implementation. Any improvements in basic RNS operations can easily be incorporated.

• Future investigations should concentrate on reducing the depth of the scaling operation (division by M).

APPENDIX
MIXED RADIX CONVERSION

One of the frequently cited results for conversion from RNS to mixed radix representation is Huang [6]. Unfortunately, with the kind of carry-pipelining outlined there, it is not possible to reach the claimed depth of $O(\log n)$ in terms of RNS processor elements. In this section we show that, with an additional carry look ahead correction step in the mixed radix system (MRS), this complexity can actually be achieved. Also, we show how to incorporate multipliers instead of lookup tables (which will be rather huge for large moduli). We restrict ourselves here to conversion from either base or extension RNS to the corresponding MRS. We will discuss the necessary adjustments for extended RNS at the end of the section.

Given X in the base RNS (or extension RNS, respectively) by its residues $[x_1, \ldots, x_n]$, we want to find its representation in the associated MRS: $\langle v_1, \ldots, v_n \rangle$. By virtue of the Chinese remainder theorem, we have:

$$X = \left(\sum_{i=1}^{n} \left(x_i M_i^{-1} \bmod m_i \right) M_i \right) \bmod M = \sum_{i=1}^{n} v_i P_{i-1}$$

where $M_i = M/m_i$ and $P_0 = 1$, $P_i = m_1 \cdots m_i$ for $1 \le i \le n$. The constants $M_i^{-1} \bmod m_i$ can be precomputed and stored directly in the RNS processor elements, whereas the computation of $w_i := (x_i M_i^{-1}) \bmod m_i$ requires one RNS multiplication.[1]

Finally we have to evaluate the whole sum in MRS. First we look at the MRS representation of the constants M_i. It can be easily verified that all MRS coefficients with index $j < i$ are zero because M_i is a product of moduli. We denote the nonzero coefficients for $i \le j \le n$ by $\mu_{i,j}$. Therefore the MRS representations become:

$$\begin{aligned}
M_1 &\triangleq \langle \mu_{1,1}, \mu_{1,2}, \ldots, \mu_{1,n-1}, \mu_{1,n} \rangle \\
M_2 &\triangleq \langle 0, \mu_{2,2}, \ldots, \mu_{2,n-1}, \mu_{2,n} \rangle \\
M_3 &\triangleq \langle 0, 0, \ldots, \mu_{3,n-1}, \mu_{3,n} \rangle \\
&\vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots \\
M_n &\triangleq \langle 0, 0, \ldots, 0, \mu_{n,n} \rangle
\end{aligned}$$

Now we use $n(n+1)/2$ modular multipliers to compute the products $w_i \mu_{i,j}$ in parallel. In Figure 1, modulo m_i multipliers are represented by "$*_i$." Each processor element generates a remainder $w_i \mu_{i,j} \bmod m_i$ as well as a carry $\lfloor w_i \mu_{i,j} / m_i \rfloor$. Both values are passed in registers (R) to the next stage. These intermediate values have to be added up, first in each column modulo m_i, and then all results across in one carry-lookahead MRS addition. The modulo m_i adders are arranged in binary trees of height $\lceil \log_2 i \rceil$.

1. By using the second form of the Chinese remainder theorem: $X = (\sum_{i=1}^{n} x_i \hat{M}_i) \bmod M$, with constants $\hat{M}_i = (M_i^{-1} \bmod m_i) M_i$, this multiplication could be saved. However, the overall carry would become considerably larger in this case.

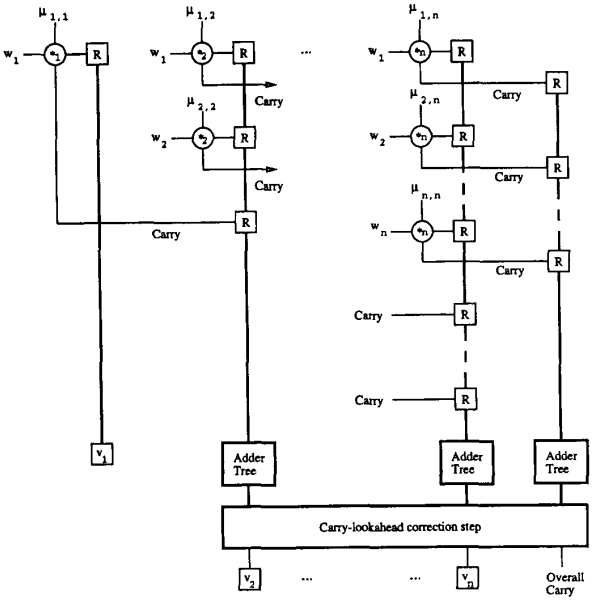


Fig. 1. The multiplication step.

Each processor element in an adder tree computes the sum modulo m_i of its input lines and updates the accumulated carry. Finally, we get the sum S_i and the accumulated carry C_i. The last column of Fig. 1 is only needed if we are to compute the overall carry (e.g., for conversion from extended RNS to MRS); normal integer addition is used in this case. We assume that the moduli are in order: $m_1 < m_2 < \cdots < m_n$. In the ith column, we have to add up $2i - 1$ numbers modulo m_i. Each of the $2i - 2$ additions can result in a carry of at most 1. It can easily be verified that m_{i+1} is always greater than $2i - 2$ (in the "worst" case, we have: $m_1 = 2$ and $m_2 = 3 > 2 \times 1 - 2 = 0$). Therefore, the accumulated carry from each adder tree is always smaller than the modulus of the next column, so we need only one carry-lookahead MRS addition to get the result:

		|	C_2	C_3	...	C_{n-1}	|	C_n
+	S_2	|	S_3	S_4	...	S_n	|	S_{n+1}
	v_2	|	v_3	v_4	...	v_n	|	carry

S_{n+1} is the sum of carries from the nth column. v_1 was already computed in the multiplication step, and v_2 is actually equal to S_2. Carry-lookahead addition in MRS uses the same method as in the binary system. We follow the notation of Leighton [7]. Addition in MRS spans over $i = 3, \ldots, n$ (indicated by the '|' delimiters). For the last column it is normal integer addition corrected by the carry from the MRS addition. For $i = 3, \ldots, n$ we compute the sum $C_{i-1} + S_i$ first, and compare it against $m_i - 1$:

$$\text{if} \quad C_{i-1} + S_i \begin{Bmatrix} < \\ = \\ > \end{Bmatrix} m_i - 1 \quad \text{then}$$

$$\text{a carry is} \begin{cases} \textit{stopped} & (s) \\ \textit{propagated} & (p) \\ \textit{generated} & (g) \end{cases} .$$

Parallel prefix applied to the s, p, and g values gives us the correction for each component, as well as the carry forwarded to the last column. $C_{i-1} + S_i$ and the correction value (0 or 1) are added together modulo m_i. The parallel prefix operation has depth $O(\log n)$.

Conversion from extended RNS to its associated MRS needs special attention. Because the smallest modulus m_{n+1} of the extension RNS is in general smaller than the largest modulus m_n of the base RNS, the requirement $m_i > 2i - 1$ (in particular $m_{n+1} > 2n + 1, \ldots, m_{2n} > 4n - 1$) is more restrictive. If some of the moduli are too small to satisfy all inequalities, the summation of the products and the carries modulo m_i, with $n + 1 \leq i \leq 2n$, can be done separately, requiring additional carry-lookahead correction step(s) at the end.

ACKNOWLEDGMENTS

This material is based on work supported in part by the National Science Foundation under Grant No. CCR-90-06077 and under Grant No. CDA-88-05910.

REFERENCES

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, The Design and Analysis of Computer Algorithms. Reading, Mass.: Addison and Wesley, 1974.

[2] D.K. Banerji, T.Y. Cheung, and V. Ganesan, "A high-speed division method in residue arithmetic," *Proc. Fifth. Symp. Computer Arithmetic*, pp. 158–164, 1981.

[3] W.A. Chren Jr., "A new residue number system division algorithm," *Computers Math. Appl.*, vol. 19, pp. 13–29, 1990.

[4] G.I. Davida and B. Litow, "Fast parallel arithmetic via modular representation," *SIAM J. Computing*, vol.20, pp. 756–765, 1991.

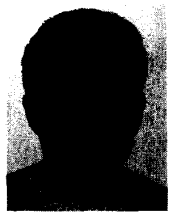
[5] D. Gamberger, "New approach to integer division in residue number systems," *Proc. 10th. Symp. Computer Arithmetic*, pp. 84–91, 1991.

[6] C.H. Huang, "A fully parallel mixed-radix conversion algorithm for residue number applications," *IEEE Trans. Computers*, vol. 32, pp. 398–402, 1983.

[7] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*. San Mateo, Calif.: Morgan Kaufmann, 1991.

[8] M. Lu and J.-S. Chiang, "A novel division algorithm for the residue number system," *IEEE Trans. Computers*, vol. 41, pp. 1,026-1,032, 1992.

[9] N.S. Szabo and R.I. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*. New York: McGraw-Hill, 1967.



Markus A. Hitz received his diploma in mathematics from the University of Zurich (Switzerland) in 1980, and the MS degree in computer science from Rensselaer Polytechnic Institute in 1988. He taught mathematics and physics on the high school level for several years, and worked as project engineer for a telecommunication company. He is currently pursuing the PhD degree at Rensselaer. His special fields of interest include residue number systems and algebraic computing. He is a member of Pi Mu Epsilon.

Erich Kaltofen received both his MS degree 1979 and his PhD degree in computer science in 1979 and 1982, respectively, from Rensselaer Polytechnic Institute. He was an assistant professor of computer science at the University of Toronto and an assistant and associate professor at Rensselaer Polytechnic Institute, where he is now a professor. His current interests are in computational algebra and number theory, design and analysis of sequential and parallel algorithms, and symbolic manipulation systems and languages.

Professor Kaltofen currently is the chair of ACM's Special Interest Group on Symbolic & Algebraic Manipulation and serves as associate editor on several journals on symbolic computation. From 1985–87 he held an IBM Faculty Development Award. From 1990–91 he was an ACM National Lecturer.