

An Approximate Sign Detection Method for Residue Numbers and its Application to RNS Division

C. Y. HUNG AND B. PARHAMI*

Department of Electrical and Computer Engineering

University of California

Santa Barbara, CA 93106-9560, U.S.A.

(Received July 1993; accepted August 1993)

Abstract—We present new division algorithms for Residue Number System (RNS). The algorithms are based on a sign estimation procedure that computes the sign of a residue number to be *positive*, *negative*, or *indeterminate*. In the last case, magnitude of the number is guaranteed to be in a limited interval whose size is related to the cost of the sign estimation process. Our division algorithms resemble SRT (Sweeney, Robertson, and Tocher) division; quotient digits in the set $\{-1, 0, 1\}$ are computed one by one. Assume that the RNS has n moduli, n residue processors, and b bits per modulus, and that each b -bit addition/subtraction takes unit time. Our sign estimation procedure uses relatively small lookup tables and takes $O(\log n)$ time. The first division algorithm based on the new sign estimation procedure requires $O(nb \log n)$ time. A second algorithm, which improves the time complexity to $O(nb)$, is the fastest algorithm proposed thus far. Intermediate between the two algorithms are a number of choices that offer speed/cost tradeoffs.

Keywords—Algorithms, Computer arithmetic, Residue number systems, Sign estimation, SRT division.

1. INTRODUCTION

Residue number systems (RNS) present the advantage of fast addition and multiplication over other number systems, and have thus received much attention for high-throughput computations, especially in digital signal processing. However, certain operations such as overflow detection, magnitude comparison, and division are quite difficult in RNS. By finding more efficient algorithm for division, many application areas for which RNS was previously infeasible can be explored.

We present two new division algorithms for residue numbers. The algorithms are based on a sign estimation procedure that when given a number in residue representation, computes its sign to be *positive*, *negative*, or *indeterminate*. In the last case, the magnitude of the input number is guaranteed to be small and within known bounds.

Assume that the RNS has n moduli, n residue processors, and b bits per modulus, and that each b -bit addition/subtraction takes unit time. Our sign estimation procedure uses relatively small lookup tables having a total size of $O(n2^b \log n)$ bits; in comparison, the mixed-radix conversion procedure requires tables of size $O(n^2 2^b)$ bits. Each sign estimation takes $O(\log n)$ time with n adders of widths $O(\log n)$ bits. The first division algorithm to be discussed requires $O(nb \log n)$ time. A second algorithm, which improves the time complexity to $O(nb)$, is the fastest algorithm proposed thus far. As in most works on residue arithmetic, we assume that

* Author to whom all correspondence should be addressed.

there are n residue processors capable of performing n parallel residue additions/multiplications in constant time. Any comparison-based division requires at least $O(\log Q)$ comparisons, where Q is the magnitude of the quotient, and thus must have worst case complexity of at least $O(nb)$. Our second algorithm is therefore asymptotically optimal within this class of algorithms.

Several algorithms for general residue division have been proposed in the past. Based on algorithm structure, there are two classes: comparison-based and subtractive. Comparison-based algorithms [1–3] determine the quotient by the iteration

$$A' = A - 2^i q_i D,$$

where A and A' are the current and next dividend, D is the divisor, and q_i is a quotient digit. A is compared with $2^i D$ to determine q_i . Typically the quotient is generated digit-sequentially as a radix-2 positional number. Of the three existing works, the digit set $\{0, 1\}$ is used in [1] and [3] and $\{-1, 0, 1\}$ is used in [2] (and in this paper). The methods to perform comparisons, or equivalently, sign detections, are all different: [1] uses mixed radix conversion, [2] uses parallel search, and [3] formulates the problem in terms of parity detection. Algorithms in this class have more predictable performance. Both [1] and [3] have the same time complexity of $O(n^2 b)$. The algorithm in [2] can theoretically achieve $O(nb)$ time complexity, but only with an impractical hardware cost of $O(2^{(n-1)b})$.

The second class of algorithms [4–7] determine the quotient by the iteration

$$A' = A - Q_i D.$$

The quotient Q_i generated in each iteration is typically a full-range residue number rather than belonging to a small set. The first three algorithms use mixed radix forms of A and D , take 1 or 2 leading digits, and access a large table for Q_i . Szabo and Tanaka [7, pp. 91–95] use a mixed radix form of D , approximate D as a product of moduli, and find Q_i by a scaling procedure. Chren [5] claims improvement over the three other algorithms in this class with experimental data. However, generally speaking, the performance of this class of algorithms is strongly data-dependent and difficult to analyze.

In this paper, a residue number system is specified by a list of n pairwise relatively prime moduli, m_1, m_2, \dots, m_n . A number X is represented by a list of residues (X_1, X_2, \dots, X_n) . Let $M = \prod m_i$ represent the product of all moduli. Conventionally, for unsigned numbers, the dynamic range of an RNS is $0 \leq X \leq M - 1$. For signed numbers, the dynamic range is¹

$$-\left\lfloor \frac{M}{2} \right\rfloor \leq X \leq \left\lfloor \frac{M-1}{2} \right\rfloor.$$

Let b be the number of bits needed to represent each residue. For efficiency of the algorithm and convenience of analyzing its complexity, we assume that the magnitudes of the moduli are more or less uniform. This assumption leads to

$$\log M \approx nb.$$

Range notations in the form $[x, y], [x, y), (x, y]$, and (x, y) are used, where parentheses stand for open boundaries and brackets stand for closed boundaries. For example, $(x, y) \equiv \{z \mid x < z \leq y\}$.

We use the expression $|x|_y$ as an extension to the notion of $x \bmod y$, where x and y are arbitrary positive real numbers. Formally, if $r = |x|_y$, then $0 \leq r < y$, and $r = x - ky$ for some integer k . The multiplicative inverse of x modulo y is written as $|x^{-1}|_y$. By definition, $|x|x^{-1}|_y = 1$.

In the lookup tables we store some truncated fractional numbers. Truncation of a number x below the $-t^{\text{th}}$ bit, or the 2^{-t} weighted bit, is represented as $[x]_{2^{-t}}$, and is related to the exact value by the inequality

$$[x]_{2^{-t}} \leq x < [x]_{2^{-t}} + 2^{-t}. \quad (1)$$

¹When M is even, generally an extra negative value is represented. This is consistent with our subsequent assumption of $|\frac{X}{M}|_1 \in [0, \frac{1}{2})$ being positive and $|\frac{X}{M}|_1 \in [\frac{1}{2}, 1)$ being negative.

2. SIGN ESTIMATION

Our sign estimation algorithm is based on the Chinese Remainder Theorem for converting a residue number to its magnitude. Given a signed number $X = (X_1, X_2, \dots, X_n)$, we have, according to the theorem,

$$|X|_M = \left| \sum_{i=1}^n \frac{M x_i}{m_i} \left| \left(\frac{M}{m_i} \right)^{-1} \right|_{m_i} \right|_M. \quad (2)$$

Dividing both sides of equation (2) by M , we obtain

$$\left| \frac{X}{M} \right|_1 = \left| \sum_{i=1}^n \frac{x_i}{m_i} \left| \left(\frac{M}{m_i} \right)^{-1} \right|_{m_i} \right|_1. \quad (3)$$

The quantity $F(X) = \left| \frac{X}{M} \right|_1 \in [0, 1)$ contains both magnitude and sign information. If $F(X) \in [0, 1/2)$, X is positive and $F(X)$ is the magnitude of X relative to M . Otherwise X is negative and $1 - F(X)$ is the relative magnitude of X .

The RNS division problem is solved if $F(X)$ can be computed economically. In comparison-based division, a k -bit quotient can be computed by k comparisons, and would take $O(k)$ time if $F(X)$ could be computed in constant time. Unfortunately, computing $F(X)$ involves addition of n fractional numbers, each of which has to be $\log M + \log n \approx nb + \log n$ bits long to guarantee a correct sign result. Note that computing $F(X)$ is as expensive as a full residue-to-binary conversion.

However, a very close estimate of the quotient, in fact off by at most one, can be obtained by using a proper estimate of $F(X)$. The estimate of $F(X)$, which we call $EF_\alpha(X)$, requires only $\lceil \log n \rceil$ steps, where each step involves a small number of single-precision additions and subtractions. An exact $F(X)$ always gives us the correct sign of X , whereas $EF_\alpha(X)$ leads to three possible conclusions: The sign of X is positive ($X \geq 0$), negative ($X < 0$), or indeterminate. In the last case $F(X)$ is too close to the critical boundaries 0, 1/2, or 1 for the estimate to accurately determine the sign. The procedure to obtain one of the three answers will be called “*sign estimation*”, with its outcome denoted by $ES_\alpha(X)$. Thus, $ES_\alpha(X) \in \{+, -, \pm\}$.

The parameter α for $EF_\alpha(X)$ and $ES_\alpha(X)$ controls the accuracy of sign estimation: a number X with $ES_\alpha(X) = \pm$ is guaranteed to be in the range $[-2^{-\alpha}M, 2^{-\alpha}M)$ if the input number to the procedure is known to be in the range

$$-\left(\frac{1}{2} - 2^{-\alpha}\right)M \leq X \leq \left(\frac{1}{2} - 2^{-\alpha}\right)M. \quad (4)$$

Intuitively, this limitation precludes $F(X)$ being close to 1/2, so that the indeterminate sign can imply that $F(X)$ is close to 0 or 1, which implies that the magnitude of X is small. It will become apparent in the analysis to follow why the range of the input is thus limited. This limitation somewhat reduces the dynamic range of computation, but the reduction is negligible if $2^{-\alpha}$ is made small.

The procedure for computing $EF_\alpha(X)$ is as follows. A set of n lookup tables is constructed, one for each modulus. For each modulus m_i , and for $j = 0, 1, \dots, m_i - 1$, the entry $EF_\alpha[i][j]$ is computed as

$$EF_\alpha[i][j] = \left[\left| \frac{j}{m_i} \left| \left(\frac{M}{m_i} \right)^{-1} \right|_{m_i} \right|_1 \right]_{2^{-\beta}}, \quad (5)$$

i.e., we truncate each term in the summation of equation (3) below the $-\beta^{\text{th}}$ bit, where

$$\beta = \alpha + \lceil \log n \rceil. \quad (6)$$

To compute $EF_\alpha(X)$, the tables are looked up using residues X_i as indices and outputs are summed modulo 1:

$$EF_\alpha(X) = \left| \sum_{i=1}^n EF_\alpha[i][X_i] \right|_1. \quad (7)$$

The estimated sign $ES_\alpha(X)$ is determined by testing $EF_\alpha(X)$ against some fixed bounds.

$$\text{If } 0 \leq EF_\alpha(X) < \frac{1}{2}, \quad ES_\alpha(X) = +, \quad \text{and } X \geq 0. \quad (8)$$

$$\text{If } \frac{1}{2} \leq EF_\alpha(X) < 1 - 2^{-\alpha}, \quad ES_\alpha(X) = -, \quad \text{and } X < 0. \quad (9)$$

$$\text{Otherwise,} \quad ES_\alpha(X) = \pm, \quad \text{and } -2^{-\alpha}M \leq X < 2^{-\alpha}M. \quad (10)$$

Correctness of the sign estimation procedure is proved as follows. We define $F[i][j]$, for $1 \leq i \leq n$, $0 \leq j \leq m_i - 1$, as

$$F[i][j] = \left| \frac{j}{m_i} \left| \left(\frac{M}{m_i} \right)^{-1} \right|_{m_i} \right|_1. \quad (11)$$

Note that

$$F(X) = \left| \sum_{i=1}^n F[i][X_i] \right|_1,$$

and therefore $F[i][j]$ and $F(X)$ are the exact counterparts of $EF_\alpha[i][j]$ and $EF_\alpha(X)$. Applying equation (1), we have

$$EF_\alpha[i][j] \leq F[i][j] < EF_\alpha[i][j] + 2^{-\beta}. \quad (12)$$

Summing n terms of both $EF_\alpha[i][j]$ and $F[i][j]$ and considering the fact that by equation (6) $n \cdot 2^{-\beta} \leq 2^{-\alpha}$, we obtain

$$\sum_{i=1}^n EF_\alpha[i][X_i] \leq \sum_{i=1}^n F[i][X_i] < \sum_{i=1}^n EF_\alpha[i][X_i] + 2^{-\alpha}. \quad (13)$$

Taking modulo 1 over both summations leads to

$$EF_\alpha(X) \leq F(X) < EF_\alpha(X) + 2^{-\alpha} \text{ if } EF_\alpha(X) < 1 - 2^{-\alpha}. \quad (14)$$

Otherwise,

$$EF_\alpha(X) \leq F(X) < 1 \text{ or } 0 \leq F(X) < EF_\alpha(X) + 2^{-\alpha} - 1. \quad (15)$$

Limiting the input range as in equation (4) implies

$$0 \leq F(X) \leq \frac{1}{2} - 2^{-\alpha} \text{ or } \frac{1}{2} + 2^{-\alpha} \leq F(X) < 1. \quad (16)$$

For the positive sign, we assert $0 \leq EF_\alpha(X) < 1/2$, which together with equation (14) imply

$$0 \leq F(X) < \frac{1}{2} + 2^{-\alpha}. \quad (17)$$

Intersection of (16) with the above to leads to $0 \leq F(X) < 1/2 - 2^{-\alpha}$, thus guaranteeing a positive sign for X . If the input X were allowed to be in the range $((1/2 - 2^{-\alpha})M, M/2) \cup [-M/2, -(1/2 - 2^{-\alpha})M)$, equation (16) would not be true, and the conclusion that $X \geq 0$ could not be reached.

For the negative sign, we assert $1/2 \leq EF_\alpha(X) < 1 - 2^{-\alpha}$, which together with equation (17) give us $1/2 \leq F(X) < 1$, thus guaranteeing a negative sign for X . The remaining interval, $1 - 2^{-\alpha} \leq EF_\alpha(X) < 1$, with equation (15) indicate

$$1 - 2^{-\alpha} \leq F(X) < 1 \text{ or } 0 \leq F(X) < 2^{-\alpha}, \quad (18)$$

which in turn implies $-2^{-\alpha}M \leq X < 2^{-\alpha}M$.

The time required to perform sign estimation is $O(\log n)$, assuming table lookup and comparisons take constant time. With the $EF_\alpha[i][j]$ tables precomputed, we need to perform n table lookups in parallel, sum up the outputs of tables modulo 1 to form $EF_\alpha(X)$, then compare $EF_\alpha(X)$ against the 2 constants $1/2$ and $1 - 2^{-\alpha^2}$ to determine $ES_\alpha(X)$. The width of additions is $\beta = \alpha + \lceil \log n \rceil = 4 + \lceil \log n \rceil$ for the first division algorithm and $5 + 2\lceil \log n \rceil$ for the second one. The $O(\log n)$ time complexity is obvious if each β -bit addition can be completed in constant time. Even if addition time is linear in operand length, we may use a carry-save adder tree to obtain in $O(\log n)$ time two operands of length β . These two operands are then added in time $\beta = O(\log n)$ using a simple ripple-carry adder or in less time using any fast adder design.

3. A DIVISION ALGORITHM

We present a division algorithm in this section, and then an improved version in the next section. The division algorithm is as follows. Given A and D , it computes Q and R such that $A = QD + R$ and $0 \leq R < D$.

1. Set $j = 0, Q = 0$
2. While $ES_\alpha(\lfloor M/8 \rfloor - 2D) \neq -$ do $D = 2D, j = j + 1$
3. While $ES_\alpha(A - D) \neq -$ do $A = A - 2D, Q = Q + 2$
4. For $i = 1, 2, 3, \dots, j$ do begin
 5. Case $ES_\alpha(A)$ of
 6. $+$: $A = 2(A - D), Q = 2(Q + 1)$
 7. $-$: $A = 2(A + D), Q = 2(Q - 1)$
 8. \pm : $A = 2A, Q = 2Q$
 9. end
10. end
11. Case $ES_\alpha(A)$ of
 12. $+$: $A = A - D, Q = Q + 1$
 13. $-$: $A = A + D, Q = Q - 1$
14. end
15. If $ES_\alpha(A) = -$ or ($ES_\alpha(A) = \pm$ and $S(A) = -$) then $A = A + D, Q = Q - 1$
16. $R = 2^{-j}A$.

For ease of description and analysis, we assume that both the dividend A and the divisor D are positive. The algorithm can be easily modified to deal with negative dividend or divisors. It is also not difficult to deal with operands of unknown sign. We just repeatedly double the operand until its sign can be detected by the sign estimation procedure. We also require $D \leq 3M/16$. This condition appears to be rather restrictive. However, for large divisors not satisfying this constraint, we can find the quotient easily by at most a few subtractions of D from A . The dividend A can have the full range for positive numbers $[0, M/2)$.

The sign estimation procedure is used early in the algorithm to normalize A and D and in the main loop to determine changes to the quotient. The precision parameter α is set to 4 for all instances of sign estimation in the algorithm. Thus, $2^{-\alpha}M = M/16$.

The overall structure of the algorithm is similar to the well-known SRT division (Sweeney, Robertson, and Tocher, see, e.g., [8, pp. 226–229]). The divisor D is first scaled up to make the most use of the precision of sign estimation. Then in the for loop, the quotient Q is adjusted by $-1, 0$, or 1 and repeatedly doubled. Essentially we are generating a radix-2 quotient digit in the digit set $\{-1, 0, 1\}$ in each iteration, with the most significant digit coming out first and the least significant digit last. Finally a possible correction of -1 on the quotient is made, and the remainder R is produced by scaling A back. Detailed description of the algorithm follows.

Line 1 sets the counter j and quotient Q to zero. On line 2, D is repeatedly doubled until $2D > \lfloor M/8 \rfloor$ is guaranteed. The number of doublings required is registered in j and is used

²Should not count zero since $EF_\alpha(X) \geq 0$ is always true.

for the loop count. If D^{in} denotes the input divisor, we have $D = 2^j D^{\text{in}}$ after the while loop terminates. Moreover, we must have $\lfloor M/8 \rfloor - 2D < 0$ because the last ES_α yields a negative sign and $\lfloor M/8 \rfloor - D \geq -M/16$ because the next to the last ES_α must have yielded a positive or indeterminate sign. Since $2D > \lfloor M/8 \rfloor$ implies $2D \geq \lfloor M/8 \rfloor + 1$, we have

$$2D \geq \left\lfloor \frac{M}{8} \right\rfloor + 1 > \frac{M}{8},$$

or $D > M/16$. For the upper bound of D , we have

$$D \leq \left\lfloor \frac{M}{8} \right\rfloor + \frac{M}{16} \leq \frac{3M}{16}.$$

Thus, D is normalized to

$$\frac{M}{16} < D \leq \frac{3M}{16}. \quad (19)$$

On line 3, A is repeatedly reduced by $2D$ until $ES_\alpha(A - D) = -$, and meanwhile, Q is repeatedly incremented by 2. Upon termination of the while loop, we have $A - D < 0$ in view of the last sign estimation and $A + D \geq -M/16$ in view of the next to the last ES_α . It follows that

$$-D - M/16 \leq A < D. \quad (20)$$

The number of times A is reduced is at most 4, since $A < M/2$ and $D > M/16$.

Lines 4 to 10 comprise the main loop of the algorithm, with i counting from 1 to j . In each iteration, $ES_\alpha(A)$ is computed, and then based on the returned sign, one of the lines 6, 7, or 8 is executed. We assert that during the execution of the for loop, A is always in the range

$$-2D \leq A < 2D. \quad (21)$$

This is proved by induction on the loop count. When the loop is entered, we know $-2D < A < D$ from (19) and (20), therefore equation (21) is true. Let A^i stand for the value of A in the beginning of i^{th} iteration. Suppose $A^i \in [-2D, 2D)$, we need to show that $A^{i+1} \in [-2D, 2D)$. The sign estimation procedure may declare the sign of A^i as positive, negative, or indeterminate. We have 3 cases:

- A^i is positive. In this case $A^i \in [0, 2D)$. Subtracting D from A and doubling bring the range back to $[-2D, 2D)$.
- A^i is negative. In this case $A^i \in [-2D, 0)$. Adding D to A and doubling again bring the range back to $[-2D, 2D)$.
- Sign of A^i is indeterminate. In this case $A^i \in [-M/16, M/16) \subseteq [-D, D)$ based on equation (19). Doubling of A brings the range back to $[-2D, 2D)$.

Lines 11 to 14 contain a case block similar to the one on lines 5 to 9, only A and Q are not doubled. Thus we know after exiting this case block that $A \in [-D, D)$. Since the desired remainder is in the range $[0, D)$, A needs to be increased by D if $A < 0$. On line 15, sign estimation is tried first to detect a negative sign, and if the returned sign is indeterminate the exact sign $S(A)$ is computed. Line 16 scales A back by 2^{-j} to obtain the remainder R , because while $A \in [0, D)$, D is normalized to $2^j D^{\text{in}}$ on line 2.

Let A^* and Q^* denote the final values of A and Q at completion of the algorithm and let A^{in} denote the input dividend. We define $q_i \in \{-1, 0, 1\}$ to be the quotient digit (change to Q) before doubling in iteration i . We count the second case block as iteration $j + 1$ and absorb the initial and final adjustments to Q in q_1 and q_{j+1} . Unfolding the changes in Q and A , we have

$$\begin{aligned} Q^* &= (\cdots ((q_1 \cdot 2 + q_2) \cdot 2 + q_3) \cdots) \cdot 2 + q_{j+1} \\ A^* &= (\cdots (((A^{\text{in}} - q_1 D) \cdot 2 - q_2 D) \cdot 2 - q_3 D) \cdots) \cdot 2 - q_{j+1} D \\ &= 2^j A^{\text{in}} - D((\cdots ((q_1 \cdot 2 + q_2) \cdot 2 + q_3) \cdots) \cdot 2 + q_{j+1}) \\ &= 2^j A^{\text{in}} - D Q^* \\ &= 2^j A^{\text{in}} - 2^j D^{\text{in}} Q^*. \end{aligned}$$

Note that A^* is divisible by 2^j . Since $A^* \in [0, 2^j D^{\text{in}})$, we know that A^* is the remainder of $2^j A^{\text{in}}$ divided by $2^j D^{\text{in}}$. Therefore, $R = 2^{-j} A^*$ is the remainder of A^{in} divided by D^{in} . Furthermore, Q^* is the correct quotient for both divisions.

For all the instances of sign estimation used in the algorithm, the input range must comply with equation (4). With $\alpha = 4$, we must have $-7M/16 \leq X \leq 7M/16$. The instances on line 2 have input

$$\left\lfloor \frac{M}{8} \right\rfloor - 2D \in \left[\left\lfloor \frac{M}{8} \right\rfloor - \frac{3M}{8}, \left\lfloor \frac{M}{8} \right\rfloor \right) \subseteq \left(-\frac{3M}{8}, \frac{M}{8} \right).$$

The instances on line 3 have input

$$A - D \in \left[-\frac{M}{16} - 2D, \frac{M}{2} - D \right) \subseteq \left[-\frac{M}{16} - \frac{3M}{8}, \frac{M}{2} - \frac{M}{16} \right) \subseteq \left[-\frac{7M}{16}, \frac{7M}{16} \right).$$

The instances on lines 5 and 11 have input

$$A \in [-2D, 2D) \subseteq \left[-\frac{3M}{8}, \frac{3M}{8} \right).$$

Input range of the ES_α instance on line 15 is $[-D, D)$, so is safe as well. Therefore, the input range limitation is satisfied for all the instances.

The exact sign computation on line 15, if required, is most efficiently carried out by a residue-to-mixed radix conversion and then a comparison against a precomputed bound in its mixed radix representation. If we call the bound B , then B can be anywhere in the range $[3M/16 + 1, 13M/16 - 1]$. It is usually possible to choose a B so that its mixed radix form has a leading nonzero digit, and zeros for all other digits. With n single-precision residue processors, the conversion takes $O(n)$ time [7, pp. 43–45] and the comparison takes constant time.

To ease the task of dividing A by 2^j to obtain R , all moduli should be made odd.³ In this case we may precompute and store the residues of 2^j for all possible j . The scaling takes only constant time. In case M is even, i.e., one of the moduli is even, scaling can still be performed in $O(n)$ time using the *base extension* method [7, pp. 47–50]. Although scaling is more expensive when M is even, the $O(n)$ time required does not affect the asymptotic time complexity of the division algorithm.

The computations required by the algorithm, with the exception of sign estimation and possibly scaling by 2^{-j} , can all be carried out with residue arithmetic. With the rather weak assumption that a residue addition/subtraction takes constant time, the overall time complexity of the division algorithm is $O(j \log n) + O(n) = O(nb \log n)$, since in the worst case j is close to nb . Residue multiplication, which is required in residue-to-mixed radix conversion in exact sign computation and in scaling by 2^{-j} , may take $O(b)$ time and still doesn't affect the asymptotic time complexity.

Any comparison-based division algorithm must take at least $O(\log Q) = O(nb)$ time. The algorithm in this section is a factor of $\log n$ over this bound. We observe that the algorithm can be made optimal by removing a $\log n$ factor from the time complexity. The improved version, presented in Section 5, achieves this goal.

4. EXAMPLE FOR THE FIRST ALGORITHM

In this section we present an example for our first division algorithm. The moduli are 5, 7, 9, 11. For the first division algorithm, $\alpha = 4$, $\beta = \alpha + \lceil \log n \rceil = 6$. The $EF_\alpha[i][j]$ values are truncated at $1/64$. The entries in Table 1 are $64 \cdot EF_\alpha[i][j]$. To compute $ES_\alpha(X)$, we sum up $EF_\alpha[i][X_i]$ modulo 1 to find $EF_\alpha(X)$, and compare $EF_\alpha(X)$ with $1/2 = 32/64$ and $1 - 2^{-\alpha} = 60/64$. For example,

$$EF_\alpha(2) = EF_\alpha((2, 2, 2, 2)) = \left\lfloor \frac{51 + 54 + 56 + 29}{64} \right\rfloor_1 = \frac{62}{64} \geq \frac{60}{64}.$$

³This same condition is required by the method of sign detection through the determination of parity [3].

Therefore, $ES_\alpha((2, 2, 2, 2)) = \pm$ is returned.

$$EF_\alpha(100) = EF_\alpha((0, 2, 1, 1)) = \left| \frac{0 + 54 + 28 + 46}{64} \right|_1 = 0.$$

Therefore, $ES_\alpha((0, 2, 1, 1)) = +$ is returned.

As a sample division, we try $A = 125 = (0, 6, 8, 4)$ and $D = 14 = (4, 0, 5, 3)$. The correct quotient $Q = 8 = (3, 1, 8, 8)$ and remainder $R = 13 = (3, 6, 4, 2)$ are produced. Table 2 shows the intermediate values during execution of the algorithm.

Table 1. $64EF_\alpha[i][j]$ for the first division algorithm.

i	m_i	j											
		0	1	2	3	4	5	6	7	8	9	10	
1	5	0	25	51	12	38							
2	7	0	27	54	18	45	9	36					
3	9	0	28	56	21	49	14	42	7	35			
4	11	0	46	29	11	58	40	23	5	52	34	17	

5. AN IMPROVED DIVISION ALGORITHM

The following division algorithm uses fewer n -operand summations to reduce the time complexity.

1. Set $j = 0, Q = 0$
- 2.1 Compute $D' = EF_\alpha(D)$.
- 2.2 While $D' \leq 1/16$ or $D' > 1/2$ do begin
- 2.3 $D = 2D, D' = 2D', j = j + 1$
- 2.4 If $j \bmod (\lceil \log n \rceil + 1) = 0$ then compute $D' = EF_\alpha(D)$
- 2.5 end
- 2.6 Compute $D' = EF_\alpha(D)$
3. While $ES_\alpha(A - D) \neq -$ do $A = A - 2D, Q = Q + 2$
4. For $i = 1, 2, 3, \dots, j$ do begin
- 5.1 If $i \bmod \lceil \log n \rceil = 1$ then compute $A' = EF_\alpha(A)$
- 5.2 Compute $ES_\alpha(A)$ using A' .
- 5.3 Case $ES_\alpha(A)$ of
6. $+$: $A = 2(A - D), A' = |2(A' - D' - 2^{-\alpha})|_1, Q = 2(Q + 1)$
7. $-$: $A = 2(A + D), A' = |2(A' + D')|_1, Q = 2(Q - 1)$
8. \pm : $A = 2A, A' = |2A'|_1, Q = 2Q$
9. end
10. end
11. Case $ES_\alpha(A)$ of
12. $+$: $A = A - D, Q = Q + 1$
13. $-$: $A = A + D, Q = Q - 1$
14. end
15. If $ES_\alpha(A) = -$ or ($ES_\alpha(A) = \pm$ and $S(A) = -$) then $A = A + D, Q = Q - 1$
16. $R = 2^{-j} A$.

The overall structure of this algorithm is the same as the previous one. There are some local modifications, and the line numbers correspond to the lines in the previous algorithm. For example, lines 2.1 through 2.6 are spawned from line 2 of the previous algorithm. The modifications are on lines 2.1 through 2.6 and lines 5.1 through 8. In essence, only some instances

Table 2. Computing 125/14 with the first division algorithm.

Iteration	Variable	Value	Residues mod (5,7,9,11)	Notes
$j = 0$	D	14	(4,0,5,3)	$M = 3465$
	$\lfloor M/8 \rfloor$	433	(3,6,1,4)	
	$\lfloor M/8 \rfloor - 2D$	405	(0,6,0,9)	
$j = 1$	$D = 2D$	28	(3,0,1,6)	$EF_\alpha = 6/64, ES_\alpha = +$
	$\lfloor M/8 \rfloor - 2D$	377	(2,6,8,3)	
$j = 2$	$D = 2D$	56	(1,0,2,1)	$EF_\alpha = 5/64, ES_\alpha = +$
	$\lfloor M/8 \rfloor - 2D$	321	(1,6,6,2)	
$j = 3$	$D = 2D$	112	(2,0,4,2)	$EF_\alpha = 4/64, ES_\alpha = +$
	$\lfloor M/8 \rfloor - 2D$	209	(4,6,2,0)	
$j = 4$	$D = 2D$	224	(4,0,8,4)	$EF_\alpha = 2/64, ES_\alpha = +$
	$\lfloor M/8 \rfloor - 2D$	-15	(0,6,3,7)	
$j = 5$	$D = 2D$	448	(3,0,7,8)	$EF_\alpha = 62/64, ES_\alpha = \pm$
	$\lfloor M/8 \rfloor - 2D$	-463	(2,6,5,10)	
line 3	A	125	(0,6,8,4)	$EF_\alpha = 56/64, ES_\alpha = -$
	$A - D$	-323	(2,6,1,7)	
$i = 1$	A	125	(0,6,8,4)	$EF_\alpha = 1/64, ES_\alpha = +$
	$A = 2(A - D)$	-646	(4,5,2,3)	
	$Q = 2(Q + 1)$	2	(2,2,2,2)	
$i = 2$	$A = 2(A + D)$	-396	(4,3,0,0)	$EF_\alpha = 56/64, ES_\alpha = -$
	$Q = 2(Q - 1)$	2	(2,2,2,2)	
$i = 3$	$A = 2(A + D)$	104	(4,6,5,5)	$EF_\alpha = 0/64, ES_\alpha = +$
	$Q = 2(Q - 1)$	2	(2,2,2,2)	
$i = 4$	$A = 2(A - D)$	-688	(2,5,5,5)	$EF_\alpha = 50/64, ES_\alpha = -$
	$Q = 2(Q + 1)$	6	(1,6,6,6)	
$i = 5$	$A = 2(A + D)$	-480	(0,3,6,4)	$EF_\alpha = 54/64, ES_\alpha = -$
	$Q = 2(Q - 1)$	10	(0,3,1,10)	
line 11	$A = A + D$	-32	(3,3,4,1)	$EF_\alpha = 61/64, ES_\alpha = \pm, S = -$
	$Q = Q - 1$	9	(4,2,0,9)	
line 15	$A = A + D$	416	(1,3,2,9)	Quotient = 8
	$Q = Q - 1$	8	(3,1,8,8)	
line 16	2^j	32	(2,4,5,10)	Remainder = 13
	2^{-j}	758	(3,2,2,10)	
	$R = 2^{-j}A$	13	(3,6,4,2)	

of the sign estimation procedure are changed. One implicit yet important change in the algorithm is $\alpha = 5 + \lfloor \log n \rfloor$.

On line 2 of the previous algorithm, the variable D is normalized by repeated doublings while $ES_\alpha(\lfloor M/8 \rfloor - 2D)$ is tested. Lines 2.1 through 2.6 perform the same normalization with slightly more complicated operations. The ES_α function is not used, instead D' , an estimate of $F(D)$, is compared with $1/16$. The comparison of D' with $1/2$ is to guard against cases where D' is just slightly less than one for a small $F(D)$. D' is renewed by computing $D' = EF_\alpha(D)$ once every $\lfloor \log n \rfloor + 1$ iterations and is repeatedly doubled with the doublings of D . Recall that there is a cost of $O(\log n)$ time for each instances of sign estimation, and the $O(\log n)$ time complexity is due to the n -operand summation in computing the EF_α function. By computing $EF_\alpha(D)$ once every $O(\log n)$ times D is doubled, instead of every time, we cut down the time complexity of the normalization step by a factor of $\log n$.

Upon renewal, D' has an error of $2^{-\alpha}$ with respect to $F(D)$. With each doubling of D' the error is doubled as well. The extra precision of the sign estimation procedure, as reflected in the

larger α , ensures that the maximal error is bounded by

$$2^{\lfloor \log n \rfloor} 2^{-\alpha} = \frac{1}{32}.$$

It follows that

$$D' M \leq D < D' M + \frac{M}{32}.$$

After the while loop terminates, we have $D' > 1/16$, and therefore $D > M/16$. For the next to the last test, we have $(D/2)' \leq 1/16$, so

$$\frac{D}{2} < \frac{M}{16} + \frac{M}{32} = \frac{3M}{32}.$$

Therefore we have $M/16 < D < 3M/16$. Note that the range of the normalized D is almost identical to equation (19) for the previous algorithm.

A similar strategy is used in the sign estimation performed in the main loop. A new variable A' is used to keep track of an estimate of $F(A)$. A' instead of $EF_\alpha(A)$ is used in determining $ES_\alpha(A)$. On line 5.1, we renew A' from $EF_\alpha(A)$ once every $\lfloor \log n \rfloor$ iterations. For the other iterations, A' is obtained from updates that reflect the changes in A , as shown on lines 6,7, and 8 of the algorithm. On line 6, where D is subtracted from A , $D' + 2^{-\alpha}$ instead of D' is subtracted from A' to ensure that A' always *underestimates* $F(A)$. We are interested in the growth of error of A' with respect to $F(A)$.⁴ For convenience, we use δ to denote $2^{-\alpha}$ and define ΔA_i as the upper bound of error in the i^{th} iteration. We start out with $A' = EF_\alpha(A)$ and update with $A' = 2(A' \pm D')$ in each iteration. Thus, we have the initial condition

$$\Delta A_1 = \delta, \tag{22}$$

and the recurrence

$$\Delta A_i = 2(\Delta A_{i-1} + \delta), 2 \leq i \leq \lfloor \log n \rfloor. \tag{23}$$

For $i \geq \lfloor \log n \rfloor + 1$, the process repeats itself. The solution of this recurrence is

$$\Delta A_i = (2^i + 2^{i-1} - 2) \delta. \tag{24}$$

Therefore, the worst case occurs at $i = \lfloor \log n \rfloor$:

$$\Delta A_{\max} = (2^{\lfloor \log n \rfloor} + 2^{\lfloor \log n \rfloor - 1} - 2) \delta < 2^{\lfloor \log n \rfloor + 1} \delta.$$

With $\alpha = 5 + \lfloor \log n \rfloor$ and $\delta = 2^{-\alpha}$, we have

$$\Delta A < \frac{1}{16}, \tag{25}$$

and indeed the upper bound error is no greater than the upper bound error of $EF_\alpha(A)$ with respect to $F(A)$ in the previous algorithm (with $\alpha = 4$).

Compared with the previous algorithm, D is in an almost identical range and the upper bound error of $ES_\alpha(A)$ is no greater. Proof of correctness for this algorithm is almost identical to that for the previous algorithm, and is therefore omitted.

The asymptotic time complexity is improved to $O(nb)$. In the worst case, we still have $O(nb)$ iterations on the normalization while loop and the main loop. The n -operand summation is performed once every $O(\log n)$ iterations instead of every iteration. Therefore, the time complexity

⁴The error of D' with respect to $F(D)$ is just their difference, $F(D) - D'$, because D is in a tight range ($M/16, 3M/16$). Strictly speaking, the error of A' with respect to $F(A)$ is how much A' falls behind in a modulo one ring, i.e., the minimal $d \geq 0$ and $A' + d_1 = F(A)$.

is cut down by a factor of $\log n$. We keep the sign estimation on lines 3, 11, and 15, since it is performed only fixed number of times there.

In this algorithm we use longer additions, having widths of $5 + 2\lceil\log n\rceil$ bits as opposed to $4 + \lceil\log n\rceil$ bits in the first algorithm. However, in most practical systems such additions can be completed in a constant number of clock cycles. As an example, 11-bit additions will be required for $n = 8$. Besides, carry-save addition can be used in much the same way as in the previous algorithm to guarantee $O(\log n)$ time complexity for the sign estimation procedure even under the assumption of linear-time addition.

Obviously there is room for trade-offs in the spacing of the EF_α computations and the width of additions. We may for example reduce the two spacings on lines 2.4 and 5.1 by 1 to $\lfloor\log n\rfloor$ and $\lfloor\log n\rfloor - 1$, respectively, and reduce the additions width to $4 + 2\lceil\log n\rceil$.

6. EXAMPLE FOR THE SECOND ALGORITHM

In this section we present an example for our second division algorithm. The moduli are again 5, 7, 9, 11, and the dividend and the divisor are 125 and 4 as in the first example. We require $\alpha = 5 + \lceil\log n\rceil = 7$ for the second algorithm, leading to $\beta = \alpha + \lceil\log n\rceil = 9$. The $EF_\alpha[i][j]$ values are truncated at $1/512$. Table 3 shows the entries $512 \cdot EF_\alpha[i][j]$. A number X is declared positive if $EF_\alpha(X) \in [0, 255/512]$, negative if $EF_\alpha(X) \in [256/512, 479/512]$, and indeterminate if $EF_\alpha(X) \in [480/512, 511/512]$. Table 4 shows the intermediate values during execution of the division algorithm.

Table 3. $512EF_\alpha[i][j]$ for the second division algorithm.

i	m_i	j											
		0	1	2	3	4	5	6	7	8	9	10	
1	5	0	204	409	102	307							
2	7	0	219	438	146	365	73	292					
3	9	0	227	455	170	398	113	341	56	284			
4	11	0	372	232	93	465	325	186	46	418	279	139	

7. CONCLUSIONS

A sign estimation procedure has been proposed. The procedure either returns the correct sign or indicates that the magnitude of the input is in a limited range. Two division algorithms based on this sign estimation procedure were presented. With n moduli, b bits per modulus, and the assumption that there are n residue processors capable of n parallel residue operations in constant time, the first algorithm achieves $O(nb \log n)$ time complexity. The second algorithm modifies the sign estimation process of the first algorithm, essentially performing most of the sign estimations incrementally, and improves the time complexity to $O(nb)$. It is the fastest general RNS division algorithm proposed thus far.

Several obvious variations of the division algorithms can be devised to make the algorithms more efficient for specific applications. For example, the normalizations of A and D can be simplified or even skipped if the ranges of A and D are known in advance. The computation of the quotient Q can be omitted if we are only interested in the remainder, as in a modulo operation. The single exact sign computation can be avoided if an error of 1 in the quotient (or error of D in the remainder) can be tolerated.

In addition to the formal proofs of correctness presented in this paper, the algorithms have been experimentally evaluated for several moduli sets with 2 to 5 moduli. For each moduli set, all permissible A, D pairs were tested. In all these cases, correct results were obtained as expected.

We are currently investigating efficient residue division algorithms for fixed divisors [9], assuming that some preprocessing based on D is allowed. Other potential research topics include:

Table 4. Computing 125/14 with the second division algorithm.

Iteration	Variable	Value	Residues mod (5,7,9,11)	Notes
$j = 0$	D	14	(4,0,5,3)	$D' = E F_{\alpha}(D) = 1/512 < 1/16$
$j = 1$	$D = 2D$	28	(3,0,1,6)	$D' = 2/512 < 1/16$
$j = 2$	$D = 2D$	56	(1,0,2,1)	$D' = 4/512 < 1/16$
$j = 3$	$D = 2D$	112	(2,0,4,2)	$D' = E F_{\alpha}(D) = 15/512 < 1/16$
$j = 4$	$D = 2D$	224	(4,0,8,4)	$D' = 30/512 < 1/16$
$j = 5$	$D = 2D$	448	(3,0,7,8)	$D' = 60/512 \geq 1/16$
line 2.6	D	448	(3,0,7,8)	$D' = E F_{\alpha}(D) = 64/512$
line 3	A	125	(0,6,8,4)	
	$A - D$	-323	(2,6,1,7)	$E F_{\alpha} = 462/512, E S_{\alpha} = -$
$i = 1$	A	125	(0,6,8,4)	$A' = E F_{\alpha}(A) = 17/512, E S_{\alpha} = +$
	$A = 2(A - D)$	-646	(4,5,2,3)	$A' = 418/512, E S_{\alpha} = -$
	$Q = 2(Q + 1)$	2	(2,2,2,2)	
$i = 2$	$A = 2(A + D)$	-396	(4,3,0,0)	
	$Q = 2(Q - 1)$	2	(2,2,2,2)	
$i = 3$	A	-396	(4,3,0,0)	$A' = E F_{\alpha}(A) = 453/512, E S_{\alpha} = -$
	$A = 2(A + D)$	104	(4,6,5,5)	$A' = 10/512, E S_{\alpha} = +$
	$Q = 2(Q - 1)$	2	(2,2,2,2)	
$i = 4$	$A = 2(A - D)$	-688	(2,5,5,5)	
	$Q = 2(Q + 1)$	6	(1,6,6,6)	
$i = 5$	A	-688	(2,5,5,5)	$A' = E F_{\alpha}(A) = 408/512, E S_{\alpha} = -$
	$A = 2(A + D)$	-480	(0,3,6,4)	
	$Q = 2(Q - 1)$	10	(0,3,1,10)	
line 11	A	-480	(0,3,6,4)	$E F_{\alpha}(A) = 440/512, E S_{\alpha} = -$
	$A = A + D$	-32	(3,3,4,1)	
	$Q = Q - 1$	9	(4,2,0,9)	
line 15	A	-32	(3,3,4,1)	$E F_{\alpha}(A) = 506/512, E S_{\alpha} = -$
	$A = A + D$	416	(1,3,2,9)	
	$Q = Q - 1$	8	(3,1,8,8)	Quotient = 8
line 16	2^j	32	(2,4,5,10)	
	2^{-j}	758	(3,2,2,10)	
	$R = 2^{-j} A$	13	(3,6,4,2)	Remainder = 13

adapting the algorithms for special classes of moduli (e.g., $2^k \pm 1$), adapting the algorithms for specific applications, study of the fault tolerance aspect of the algorithms, and applications of the sign estimation process to other residue operations such as square rooting and base extension.

REFERENCES

1. Y.A. Keir, P.W. Cheney and M. Tannenbaum, Division and overflow detection in residue number systems, *IRE Transactions on Electronic Computers* **EC-11** (4), 501-507 (August 1962).
2. M.-L. Lin, E. Leiss and B. McInnis, Division and sign detection algorithms for residue number systems, *Computers Math. Applic.* **10** (4/5), 331-342 (1984).
3. 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).
4. D.K. Banerji, T.Y. Cheung and V. Ganesan, A high speed division method in residue arithmetic, In *Proceedings of the Fifth Symposium on Computer Arithmetic*, pp. 158-164, IEEE Press, (May 1981).
5. W.A. Chren Jr., A new residue number system division algorithm, *Computers Math. Applic.* **19** (7), 13-29 (1990).
6. E. Kinoshita, H. Kosako and Y. Kojima, General division in the symmetric residue number system, *IEEE Transactions on Computers* **C-22** (2), 134-142 (February 1973).

7. N. Szabo and R.I. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*, McGraw-Hill, (1967).
8. K. Hwang, *Computer Arithmetic, Principles, Architecture, and Design*, John Wiley & Sons, Inc., New York, (1979).
9. C.Y. Hung and B. Parhami, Fast RNS division algorithms for fixed divisors with application to RSA encryption, Manuscript. University of California, Department of Electrical and Computer Engineering, (March 1993).