A Novel Division Algorithm for the Residue Number System

Mi Lu, *Member, IEEE*, and Jen-Shiun Chiang

Abstract—**We present in this paper a novel general algorithm for signed number division in Residue Number Systems (RNS). A parity checking technique is used to accomplish the sign and overflow detection in this algorithm. Compared with conventional methods of sign and overflow detection, the parity checking method is more efficient and practical. Sign magnitude arithmetic division is implemented using binary search. There is no restriction to the dividend and the divisor (except zero divisor), and no quotient estimation is necessary before the division is executed. Only simple operations are needed to accomplish this RNS division. All these characteristics have made our algorithm simple, efficient, and practical to be implemented on a real RNS divider.**

Index Terms—**Binary search, core function, division algorithm, fractional representation, number comparison, overflow detection, parity checking, residue number system.**

I. Introduction

RESIDUE Number Systems (abbreviated as RNS) are attractive to many people. An RNS is composed of moduli that are independent of each other. A number in the RNS is represented by the residue of each modulus, and arithmetic operations are accomplished based on each modulus. Since the moduli are independent of each other, there is no carry propagation among them, and it is easy to implement RNS computations on a multi-ALU system. The operation based on each modulus can be performed by a separate ALU, and all the ALU's can work concurrently. These characteristics allow RNS computations to be completed more quickly—an attractive feature for people who need high speed arithmetic operations [1], [2].

Overflow detection, sign detection, number comparison, and division in RNS are very difficult and time consuming [3], [4]. These shortcomings limited most of the previous RNS applications to addition, subtraction, and multiplication.

The general division algorithms can be classified into two groups [5]: multiplicative algorithms and subtractive algorithms. There are several RNS division algorithms that are classified as multiplicative algorithms [4], [6], [7]. These multiplicative algorithms use mixed radix number conversion to find the reciprocal of the divisor and to compare numbers. Iteratively, the approximate quotient is made closer to the accurate one. Due to the involvement of the mixed radix number conversion, the arithmetic calculation is very complicated and needs a lot of stored tables. Among these multiplicative algorithms, Kinoshita's algorithm [7] uses mixed radix numbers to approximate the quotient, and requires either a decimal divider or the storage of a very large table. Banerji's algorithm [6] also uses the mixed radix number approach and requires a lot of storage. Chren [4] criticizes that the standard deviation of the mean of the execution time needed in this algorithm is high. Chren's algorithm [4] is modified from Banerji's. Chren made some effort to reduce the storage and to improve the standard deviation of the mean execution time, but the storage and the computation time needed by the mixed radix number conversion are still expensive.

On the other hand, there are several algorithms classified as subtractive algorithms [3], [8], [9]. These subtractive algorithms use the conventional division approach, and no mixed radix number conversion is necessary. Therefore, the arithmetic calculations are not complicated. However, its number comparison and sign detection consume a lot of time and hardware. Szabo's algorithm [3] is not a general division algorithm but a scaling algorithm. Keir *et al.* [8] present two algorithms, both of which involve the binary expansion of the quotient. The speed of Keir's first algorithm is not desirable. His second algorithm uses look-up tables so that the hardware requirements are huge. Lin's algorithm [9] is a modification of the well known CORDIC division algorithm, but it needs a lot of comparators, which is not practical for general computing applications.

In this paper we use parity checking for sign and overflow detection. Compared to conventional methods, the parity checking method is more efficient and practical. Based on the extension of overflow and sign detection techniques, a new signed RNS division algorithm is presented. Basically, this is a subtractive division algorithm using an efficient method to detect overflows and compare numbers. We use sign magnitude arithmetic for RNS division. In this division algorithm, binary search is used. There are no restrictions to dividends and divisors (except zero divisor), and no quotient estimation is necessary before the division is executed. In a hardware implementation, only a few small tables are required. All these characteristics have made our algorithm simple, efficient, and practical.

Manuscript received October 12, 1991; revised February 29, 1992. This work was supported in part by the National Science Foundation under Grant MIP-8809328. Based on "A General Division Algorithm for Residue Number Systems" by Jen-Shiun Chiang and Mi Lu which first appeared in Proceedings of 10th IEEE Symposium on Computer Arithmetic, Grenoble, France, June 26–28, 1991, pp. 76–83. © 1991 IEEE.

The authors are with the Department of Electrical Engineering, Texas A&M University, College Station, TX 77843-3128.

IEEE Log Number 9201905.

II. Residue Codes

A. Residue Numbers and Arithmetics of Residue Numbers

The RNS representation of an integer is defined as follows. Let $\{m_1, m_2, \cdots, m_n\}$ be a set of positive numbers all greater than 1. The m_i's are called moduli and the n-tuple set $\{m_1, m_2, \cdots, m_n\}$ is called the modulus set. Consider an integer number X. For each modulus in set $\{m_1, m_2, \cdots, m_n\}$, we have $x_i = X \bmod m_i$ (denoted as $|X|_{m_i}$). Thus a number X in RNS can be represented as $X = (x_1, x_2, \cdots, x_n)$, given a specific modulus set $\{m_1, m_2, \cdots, m_n\}$. In order to avoid redundancy, the moduli of a residue number system must be pair-wise relatively prime.

Let $M = \prod_{i=1}^{n} m_i$. It has been proved, in [3], that if $0 \leq X < M$, the number X is one by one corresponding to the RNS representation. If the result of a calculation exceeds M, we say that overflow occurs. All the numbers should be within the dynamic range M (i.e., $0 \leq X < M$). Then, the RNS arithmetic can be performed.

Suppose that two numbers, X and Y, are represented as $X = (x_1, x_2, \cdots, x_n)$ and $Y = (y_1, y_2, \cdots, y_n)$ in RNS. We use $\otimes$ to represent the operator of additions, subtractions, and multiplications. The arithmetic in RNS can be expressed as $X \otimes Y = (z_1, z_2, \cdots, z_n)$, where $z_i = |x_i \otimes y_i|_{m_i}$. From the definition of the mod operation, all moduli are positive. x_i may be less than y_i, which yields $x_i - y_i < 0$. In the mod operation, if $x_i - y_i < 0$, then z_i is defined as

$$z_i = m_i + (x_i - y_i). \tag{1}$$

B. Number Comparison for Unsigned Numbers

As we know, number comparison and overflow detection in RNS are very difficult. It is necessary to find methods that are efficient, practical, and easy to implement.

Let parity indicate whether an integer number is even or odd. We say two numbers are of the same parity if they are both even or both odd. Otherwise the two numbers are said to be of different parities. In residue number systems, we can use a redundant modulus, modulus 2, to find the parity of a number. If a number modulo 2 equals 0, it is indicated as an even number. On the other hand, an odd number modulo 2 is equal to 1. We will apply the properties of the parities of numbers to accomplish the number comparison.

In a survey of research in the former Soviet Union on residue number systems [10], Miller *et al.* defined a function called the *core* function and explored its properties as follows:

Let $m_1, m_2, \cdots, m_n$ be the relatively prime moduli of a residue number system with product M. For fixed integers $w_1, w_2, \cdots, w_n$, the core R_N of an integer is defined as follows:

$$R_N = \sum_{i=1}^{n} w_i \left\lfloor \frac{N}{m_i} \right\rfloor$$

where $\lfloor X \rfloor$ denotes the largest integer not greater than X. The coefficients w_i's are fixed for the moduli set and do not depend on the integer N.

Theorem 1: Let the moduli m_i and the core R_M be odd. Let $(a_1, a_2, \cdots, a_n)$ and $(b_1, b_2, \cdots, b_n)$ be the residue representations of integers A, $B \in [0, M)$. Then $A + B$ causes an overflow if

i) $(a_1 + b_1, \cdots, a_n + b_n)$ is odd, and A and B have the same parity; or
ii) $(a_1 + b_1, \cdots, a_n + b_n)$ is even, and A and B have different parities.

Let the interval $[0, M/2]$ represent positive numbers and the interval $(M/2, M)$ represent negative numbers.

Theorem 2: If the moduli m_i and the core R_M are odd, and $(a_1, a_2, \cdots, a_n)$ is the residue representation of a nonzero integer $A \in [0, M)$, then A is positive if and only if $(|2a_1|_{m_1}, \cdots, |2a_n|_{m_n})$ is even.

According to the theory of core functions, if the core function of an RNS number is known, it is easy to detect overflows and the signs of the numbers. However, it is very difficult to find the core function in RNS by the method given in [10]. Discarding the core function and revising the theorems mentioned by Miller *et al.*, the following theorems express the properties we need for the comparison of unsigned numbers. Consider the whole dynamic range, $[0, M)$, of positive numbers from 0 to $(M - 1)$. Let all m_i's in the modulus set $\{m_1, m_2, \cdots, m_n\}$ be odd numbers, and $X = (x_1, x_2, \cdots, x_n)$ and $Y = (y_1, y_2, \cdots, y_n)$ be two RNS numbers. Suppose $Z = X - Y = (z_1, z_2, \cdots, z_n)$, then we have the following theorem.

Theorem 3: Let X and Y have the same parity and $Z = (X - Y)$. $X \geq Y$, iff Z is an even number. $X < Y$, iff Z is an odd number.

Proof: If $X \geq Y$, then $X - Y \geq 0$ and Z equals $X - Y$. We know from the mathematical axioms that the two numbers are with the same parity, and the result of the subtraction should be an even number. Therefore, $X \geq Y$ implies that Z is an even number.

On the other hand, suppose that Z is an even number and X and Y are with the same parity. If $X < Y$, then $X - Y < 0$. From (1) we have $Z = X - Y + M$. Since M is an odd number and $X - Y$ is even, Z must be an odd number. This contradicts the assumption that Z is even. Therefore, if Z is an even number and X and Y are with the same parity, then $X \geq Y$.

If $X < Y$, then $X - Y < 0$. From (1) we have $Z = X - Y + M$. Since m_i's are all odd numbers, M should be an odd number. In addition, $(X - Y)$ is an even number and this implies that Z is an odd number. Therefore, it is obvious that if $X < Y$, Z is an odd number.

On the other hand, suppose that Z is an odd number and X and Y are with the same parity. If $X \geq Y$, then $X - Y \geq 0$. Since X and Y are with the same parity, Z must be an even number. This contradicts the assumption that Z is an odd number. Therefore, if Z is an odd number and X and Y are with the same parity, then $X < Y$.

Theorem 3 shows us a method to compare two numbers if the parities of these two numbers are the same. Similarly, if the parities of two numbers are different, then the following theorem can tell us which one is bigger.

Theorem 4: Let X and Y have different parities and $Z = X - Y$. $X \geq Y$, iff Z is an odd number. $X < Y$, iff Z is an even number.

The proof of Theorem 4 is similar to that of Theorem 3 and is hence omitted.

C. Signed Numbers and the Properties

The method used to represent negative numbers in RNS is similar to that used in conventional radix number systems. Letting the dynamic range be M, we can define the positive and negative numbers as follows [3].

Definition 1: Given m_i's in the modulus set all odd, and the dynamic range $M = \prod_{i=1}^{n} m_i$, the range of a positive number X is defined as $0 \leq X \leq \lfloor M/2 \rfloor$, and the range of a negative number Y is defined as $\lfloor M/2 \rfloor < Y < M$. For any positive number $X \neq 0$, the additive inverse of X is represented by $M - X$.

The following definition is to define overflows in the RNS addition.

Definition 2: Given m_i's in the modulus set all odd, and two numbers in RNS such as $X = (x_1, x_2, \cdots, x_n)$ and $Y = (y_1, y_2, , , , y_n)$, overflow exists if $|X+Y| > (M-1)/2$.

Note that the following cases need to be considered.

1) X and Y are with the same sign. The absolute value of the sum should be no greater than $\lfloor M/2 \rfloor$.
2) X and Y have different signs. No overflow will occur.

Corollary 1: The overflow detection theory in Definition 2 applies to the addition of only two numbers.

When comparing two signed numbers, three cases need to be considered. If X and Y are with different signs, the positive number is greater than the negative number. If X and Y are both positive numbers, Theorems 3 and 4 can be applied to compare X and Y. If X and Y are both negative numbers, then find the absolute values for X and Y, and compare them applying Theorems 3 and 4. The number with a greater absolute value is smaller.

Consider the number $|b|_m$, the multiplicative inverse of it is defined as follows.

Definition 3: If $0 \leq a < m$ and $|ab|_m = 1$, a is called the multiplicative inverse of $(b \bmod m)$, and is denoted as $|b^{-1}|_m$ or $|1/b|_m$.

D. The Parity of an RNS Number

We use the parity checking technique to compare numbers and detect overflows in the addition of two numbers. For the parity checking, a redundant modulus 2 is required. The parity of a number, 0 if it is even or 1 if odd, can be obtained by looking up a table. The entries of the table contain the residue representations of the numbers, and all the residues of those numbers modulo 2. The size of the table is proportional to the dynamic range, M. If M is not big, the size of the redundant modulus 2 table is reasonable. Otherwise, such a table is not practical, and the following alternative method should be used.

Given a modulus set, $\{m_1, m_2, \cdots, m_n\}$, whose dynamic range is equal to M, an RNS number X, $(x_1, x_2, \cdots, x_n)$, is corresponding to the modulus set. By the Chinese remainder theorem, X can be converted from its residue number representation by the weighted sum such as

$$X = \left| \sum_{j=1}^{n} \hat{m}_j \left| \frac{x_j}{\hat{m}_j} \right|_{m_j} \right|_M \tag{2}$$

with $\hat{m}_j = M/m_j$.

Since $|A|_M$ denotes the least positive residue of A modulo M, (2) can be rewritten as

$$X = \sum_{j=1}^{n} \hat{m}_j \left| \frac{x_j}{\hat{m}_j} \right|_{m_j} - rM \tag{3}$$

$$= \frac{M}{m_1} \left| \frac{x_1}{\hat{m}_1} \right|_{m_1} + \frac{M}{m_2} \left| \frac{x_2}{\hat{m}_2} \right|_{m_2} + \ldots + \frac{M}{m_n} \left| \frac{x_n}{\hat{m}_n} \right|_{m_n} - rM \tag{4}$$

with r being an integer.

All the moduli, $m_1, m_2, \cdots m_n$, are odd numbers, therefore, $\hat{m}_1, \hat{m}_2, \cdots, \hat{m}_n$, and M are all odd numbers in (4). Under this situation, $|x_1/\hat{m}_1|_{m_1}, |x_2/\hat{m}_2|_{m_2}, \cdots, |x_n/\hat{m}_n|_{m_n}$, and r in (4) will determine the parity of X. To find the parities of $|x_i/\hat{m}_i|_{m_i}$ and r, we can extract the least significant bit (LSB) of $|x_i/\hat{m}_i|_{m_i}$'s and r, and "Exclusive OR," $\oplus$, them together, i.e.,

$$P = LSB(\left| \frac{x_1}{\hat{m}_1} \right|_{m_1}) \oplus LSB(\left| \frac{x_2}{\hat{m}_2} \right|_{m_2}) \oplus \cdots LSB(\left| \frac{x_n}{\hat{m}_n} \right|_{m_n}) \oplus LSB(r). \tag{5}$$

Here $P = 0$ means that X is an even number, and $P = 1$ means that X is an odd number.

The numbers $|x_i/\hat{m}_i|_{m_i}$'s can be precalculated, and their LSB's can be stored in a table. Considering that the table storing the parity of $|x_i/\hat{m}_i|_{m_i}$'s is much smaller than the table storing the parity of X, we have reduced the size of the table needed for the parity checking. The next problem is how to find r.

In 1985, Van Vu [11] developed a fractional representation in Chinese remainder theorem to detect the sign of an RNS number. His approach can be revised to find the integer number r.

Let $|x_i/\hat{m}_i|_{m_i} = S_i$, with $i = 1, 2, 3, \cdots, n$, and divide (4) by M on both sides. We have

$$r = \frac{S_1}{m_1} + \frac{S_2}{m_2} + \cdots + \frac{S_n}{m_n} - \frac{X}{M}. \tag{6}$$

As we know, a number modulo m_i is less than m_i. Therefore, $S_i < m_i$ and $S_i/m_i < 1$. A number X is always less than M, and we can find $X/M < 1$. Obviously r is equal to the integer part of $\sum_i S_i/m_i$, and (6) can be rewritten as

$$r = \left\lfloor \frac{S_1}{m_1} + \frac{S_2}{m_2} + \cdots + \frac{S_n}{m_n} \right\rfloor. \tag{7}$$

We discuss below the number of the bits needed in the corresponding implementation. Ideally, the binary representation for the fractional part of S_i/m_i has infinite length. However, in the physical electronic system it can have only finite length. Suppose that t bits are used to represent the fractional part of

S_i/m_i. For simplicity, we denote S_i/m_i as u_i. Let the rounded value $\hat{u_i}$ be equal to $\lceil 2^t u_i \rceil 2^{-t}$, where $\lceil Y \rceil$ is the smallest number that is not smaller than Y. Since $\hat{u_i}$ is a rounded number of u_i, an error e_i is involved such that $\hat{u_i} = u_i + e_i$. Taking the summation over i from both sides of the previous equation, we have

$$\sum_i \hat{u_i} = \sum_i u_i + \sum_i e_i. \tag{8}$$

Denote $\sum_i \hat{u_i}$, $\sum_i u_i$, and $\sum_i e_i$ as $\hat{U}, U$, and e, respectively. Equation (8) can be rewritten as $U = \hat{U} - e$. Substituting U into (6), and rearranging the equation, we have

$$\hat{U} = r + e + \frac{X}{M}. \tag{9}$$

Here, we hope that $e + X/M < 1$. In other words, $e < 1 - (X/M)$.

Since $X/M < 1$, and the smallest difference between 1 and X/M is $1/M$, the integer part of $\hat{U}$ will not be bothered if $e < 1/M$ in (9). From

$$e = \sum_{i=1}^{n} e_i < \frac{1}{M} \quad \Longrightarrow \quad ne_i < \frac{1}{M} \quad \Longrightarrow \quad e_i < \frac{1}{nM},$$

we can choose t (the number of bits in the fractional part of S_i) as $t > \log_2(nM)$. In that case, the integer r can be represented, by the rounded value $\hat{u_i}$, as $r = \lfloor \sum_{i=1}^{n} \hat{u_i} \rfloor$.

For the calculation of the parity, all we need is the LSB of r. Therefore, $t + 1$ bits are needed for $\hat{u_i}$, with 1 bit for the integer part. The value of $\hat{u_i}$ can be precalculated and stored in a table. Since each modulus, m_i, in RNS is not a large number, the table for storing $\hat{u_i}$ is small. The summation of all $\hat{u_i}$'s can be accomplished by using fast multioperand binary adders proposed by Daniel *et al.* [12].

III. Division Algorithm

A. Description of the Algorithm

Given two numbers, dividend X and divisor Y, the division in RNS is to find the quotient $Z = \lfloor X/Y \rfloor$. The absolute value of the dividend and the divisor are used when performing the division calculation, and the parity checking technique described in the previous section is applied for number comparison and overflow detection. Given modulus set $\{m_1, m_2, \cdots, m_n\}$ with dividend $X = (x_1, x_2, \cdots, x_n)$ and divisor $Y = (y_1, y_2, \cdots, y_n)$, we are to find the quotient Z, where $Z = \lfloor X/Y \rfloor$. The dynamic range, M, of the RNS is $M = \prod_{i=1}^{n} m_i$. Corollary 1 tells us that the overflow detection can be applied only to the addition of two numbers, a special case of which is the addition of two equal numbers. In other words, multiplying a number by 2 is allowed, and our algorithm is developed on this basis (see Part II below).

This algorithm can be divided into five parts. Part I detects the signs of the dividend and the divisor and converts them to positive numbers. Part II finds 2^k, such that $(Y \cdot 2^k) \leq X < (Y \cdot 2^{k+1})$. Part III finds the difference between 2^k and the quotient. Part IV deals with the case $(Y \cdot 2^k) \leq X \leq (M-1)/2 < (Y \cdot 2^{k+1})$ and then goes to Part III to find out the difference between 2^k and the quotient. Part V transforms the quotient to the proper representation in RNS (positive or negative).

Part I: Take the absolute value of X and Y, and record the signs of them. If the signs of the dividend and divisor are different, then the quotient is negative, and we have to set the sign variable, SIGN, to 1. SIGN will be used to convert the quotient to a proper form in Part V.

Part II: We find the proper 2^k such that $(Y \cdot 2^k) \leq X < (Y \cdot 2^{k+1})$ in the following way. Two variables, LB (Lower-Bound) and UB (Upper-Bound), are set to record the range in which the value of the quotient is to be found. The LB and the UB will dynamically change, as the algorithm is executed. In iteration i, $LB = 2^i$ and $UB = 2^{i+1}$. We repeatedly compare $(2^i \cdot Y)$ with X and detect whether $(2^{i+1} \cdot Y)$ is greater than $(M-1)/2$ (denoted as M_p), until we find some i, denoted as k, such that $(Y \cdot 2^k) \leq X < (Y \cdot 2^{k+1})$. Then we make the record by setting $\mathcal{LB}_0 = 2^k$ and $\mathcal{UB}_0 = 2^{k+1}$. In each iteration, the LB_{i+1} is updated by doubling LB_i, and UB_{i+1} is equal to the twice of LB_{i+1}. The following are the equations for finding the Upper-Bound and the Lower-Bound.

$$LB_{i+1} = \gamma_{i+1} \tag{10}$$

$$UB_{i+1} = 2 \cdot LB_{i+1} \tag{11}$$

with

$$\gamma_{i+1} = \begin{cases} 2 \cdot LB_i, & \text{if } X > Y \cdot LB_i \\ LB_i, & \text{if } X \leq Y \cdot LB_i \end{cases}$$

and $LB_0 = 2^0$. Suppose that the procedure halts in iteration $i+1$ when $X \leq Y \cdot LB_i$ is tested. According to (10) and (11), $LB_{i+1} = LB_i$ and $UB_{i+1} = UB_i$, respectively. Let us define this i to be k, then $\mathcal{LB}_0 = LB_i = 2^k$ and $\mathcal{UB}_0 = 2 \cdot LB_i = 2^{k+1}$.

Two cases may occur when the above procedure halts. In one case, $(UB_k \cdot Y)$ is smaller than M_p. Then a binary search starts in Part III. Otherwise go to Part IV.

Part III: We have found the Upper-Bound ($\mathcal{UB}_0$) and the Lower-Bound ($\mathcal{LB}_0$) from the previous part such that $Y \cdot \mathcal{LB}_0 \leq X < Y \cdot \mathcal{UB}_0$. In this part we perform a binary search to find the difference between $\mathcal{LB}_0$ and the quotient, denoted as QE. k steps are needed to finish this part, since 2^k integers exist in the range $[2^k, 2^{k+1})$. Before the binary search, we set the initial value QE_0 to be 0. In each step of the binary search, we have to compare X with $(Y \cdot (\mathcal{UB}_{j+1} + \mathcal{LB}_{j+1})/2)$ (for convenience, we use another variable "Bounding," B_{j+1}, to denote $(\mathcal{UB}_{j+1} + \mathcal{LB}_{j+1})/2$). If $(X - Y \cdot B_{j+1}) < 0$, then set $\mathcal{UB}_{j+1} = B_{j+1}$ and $QE_{j+1} = 2 \cdot QE_j$. Otherwise set $\mathcal{LB}_{j+1} = B_{j+1}$ and $QE_{j+1} = (2QE_j + 1)$. When this procedure is finished, we can find the quotient, Z, to be $Z = (\mathcal{LB}_0 + QE_k)$. The following are the equations for the binary search.

$$B_{j+1} = \frac{\mathcal{UB}_j + \mathcal{LB}_j}{2} \tag{12}$$

$$R_{j+1} = X - Y \cdot B_{j+1} \tag{13}$$

$$QE_{j+1} = 2 \cdot QE_j + \delta_{j+1} \tag{14}$$

$$\mathcal{UB}_{j+1} = \sigma_{j+1} \tag{15}$$

$$\mathcal{LB}_{j+1} = \theta_{j+1} \tag{16}$$

where

$$\delta_{j+1} = \begin{cases} 1, & \text{if } R_{j+1} \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\sigma_{j+1} = \begin{cases} \mathcal{UB}_j, & \text{if } R_{j+1} \geq 0 \\ B_{j+1}, & \text{otherwise} \end{cases}$$

$$\theta_{j+1} = \begin{cases} \mathcal{LB}_j, & \text{if } R_{j+1} < 0 \\ B_{j+1}, & \text{otherwise} \end{cases}$$

$\delta_0 = 0$, $QE_0 = 0$, $\mathcal{LB}_0 = 2^k$, and $\mathcal{UB}_0 = 2^{k+1}$. The procedure halts when $j + 1 = k$. To decide the sign of the quotient, we have to go to Part V.

Part IV: If $(Y \cdot 2^k) \leq X \leq M_p < (Y \cdot 2^{k+1})$, we have to update $\mathcal{UB}_1$ as $(\mathcal{UB}_0 + \mathcal{LB}_0)/2 = (2^k + 2^{k+1})/2$, and $\mathcal{LB}_1$ as 2^k. QE_1 is updated as $QE_1 = 2 \cdot QE_0$. Repeatedly, in the $(j + 1)$th iteration, update $B_{j+1} = (\mathcal{UB}_j + \mathcal{LB}_j)/2$ and examine whether $(Y \cdot B_{j+1})$ overflows again. If there is an overflow, set $\mathcal{UB}_{j+1} = B_{j+1}$ and $QE_{j+1} = 2QE_j$. Continue this procedure until $(Y \cdot B_{j+1})$ does not overflow. If $(Y \cdot B_{j+1})$ does not overflow and $(X - Y \cdot B_{j+1}) \geq 0$, set $\mathcal{LB}_{j+1} = B_{j+1}$ and $QE_{j+1} = (2 \cdot QE_j + 1)$, and detect overflow again. If $(Y \cdot B_{j+1})$ does not overflow and $(X - Y \cdot B_{j+1}) < 0$, set $\mathcal{UB}_{j+1} = B_{j+1}$ and $QE_{j+1} = 2 \cdot QE_j$, and perform the similar operations as defined in the binary search of Part III. After taking k steps in total, then go to Part V.

The following equations are applied in the above operations.

$$B_{j+1} = \frac{\mathcal{UB}_j + \mathcal{LB}_j}{2} \tag{17}$$

$$R_{j+1} = Y \cdot B_{j+1} - M_p \tag{18}$$

$$R'_{j+1} = X - Y \cdot B_{j+1} \tag{19}$$

$$QE_{j+1} = 2 \cdot QE_j + \delta_{j+1} \tag{20}$$

$$\mathcal{UB}_{j+1} = \sigma_{j+1} \tag{21}$$

$$\mathcal{LB}_{j+1} = \theta_{j+1} \tag{22}$$

where

$$\delta_{j+1} = \begin{cases} 0, & \text{if } (R_{j+1} > 0) \text{ OR } [(R_{j+1} \leq 0) \\ & \quad \text{AND } (R'_{j+1} < 0)] \\ 1, & \text{otherwise} \end{cases}$$

$$\sigma_{j+1} = \begin{cases} B_{j+1}, & \text{if } (R_{j+1} > 0) \text{ OR } [(R_{j+1} \leq 0) \\ & \quad \text{AND } (R'_{j+1} < 0)] \\ \mathcal{UB}_j, & \text{otherwise} \end{cases}$$

$$\theta_{j+1} = \begin{cases} B_{j+1}, & \text{if } [(R_{j+1} \leq 0) \text{ AND } (R'_{j+1} \geq 0)] \\ \mathcal{LB}_j, & \text{otherwise} \end{cases}$$

$\delta_0 = 0, QE_0 = 0, \mathcal{LB}_0 = 2^k$, and $\mathcal{UB}_0 = 2^{k+1}$ [$\mathcal{LB}_0$ and $\mathcal{UB}_0$ are from (13)]. If $[(Y \cdot B_{j+1} - M_p \leq 0) \text{ AND } (X - Y \cdot B_{j+1} < 0)]$, go to Part III, and continue the search procedures in Part III. If $|\mathcal{UB}_{j+1} - \mathcal{LB}_{j+1}| = 1$, the search stops. Let quotient $Z = \mathcal{LB}_{j+1}$, and go to Part V to get the proper form for Z (as a positive number or a negative number).

Part V: From Part I, the exact quotient may be negative. Therefore, if SIGN=1, the absolute value of the found quotient should be complemented.

B. The Correctness of the Algorithm

The absolute value of a quotient is equal to the quotient of the absolute value of the dividend and the absolute value of the divisor. The sign of the quotient depends on the signs of the dividend and divisor. If the signs of the dividend and divisor are different, then the quotient is negative. Otherwise, the quotient is positive. To prove the correctness of the algorithm, we concentrate on Part II and Part III only. The proofs of Part IV and Part III are similar and are hence omitted.

In Part II, by replacing the initial value and the iterative values, we can find LB_{i+1} and UB_{i+1} as follows:

$$LB_{i+1} = 2^i, \quad \text{and} \quad UB_{i+1} = 2^{i+1}.$$

When the procedure halts, i.e., when $X \geq Y \cdot LB_i = Y \cdot 2^i$, we set the value i to be k, hence

$$LB_k = \mathcal{LB}_0 = 2^k \leq Z < 2^{k+1} = UB_k = \mathcal{UB}_0. \tag{23}$$

The values decided for $\mathcal{LB}_0$ and $\mathcal{UB}_0$ are required by Part III.

In Part III, substituting the initial values of δ_0, $\mathcal{UB}_0$, $\mathcal{LB}_0$ into (13)–(17), we can find B_1, R_1, QE_1, $\mathcal{UB}_1$, and $\mathcal{LB}_1$. Iteratively we can find B_2, R_2, QE_2, $\mathcal{UB}_2$, and $\mathcal{LB}_2$. In the jth iteration, the following equations hold.

$$B_{j+1} = \mathcal{LB}_0 + \sum_{m=1}^{j} \delta_m \frac{\mathcal{LB}_0}{2^m} + \frac{\mathcal{LB}_0}{2^{j+1}} \tag{24}$$

$$R_{j+1} = X - Y \cdot B_{j+1} \tag{25}$$

$$QE_{j+1} = \sum_{m=1}^{j+1} 2^{j+1-m} \delta_m \tag{26}$$

$$\mathcal{UB}_{j+1} = \mathcal{LB}_0 + \sum_{m=1}^{j+1} \delta_m \frac{\mathcal{LB}_0}{2^m} + \frac{\mathcal{LB}_0}{2^{j+1}} \tag{27}$$

$$\mathcal{LB}_{j+1} = \mathcal{LB}_0 + \sum_{m=1}^{j+1} \delta_m \frac{\mathcal{LB}_0}{2^m}. \tag{28}$$

When $j + 1 = k$,

$$\begin{aligned} R_k &= X - Y \cdot B_k \\ &= X - Y\left(\mathcal{LB}_0 + \sum_{m=1}^{k-1} \delta_m \frac{\mathcal{LB}_0}{2^m} + \frac{\mathcal{LB}_0}{2^k}\right). \end{aligned} \tag{29}$$

Since $\mathcal{LB}_0 = 2^k$, therefore,

$$R_k = X - Y\left(\mathcal{LB}_0 + \sum_{m=1}^{k-1} \delta_m 2^{k-m} + 1\right). \tag{30}$$

Letting both sides of (30) be divided by Y, we can have

$$\frac{X}{Y} = \frac{R_k}{Y} + \mathcal{LB}_0 + \sum_{m=1}^{k-1} \delta_m 2^{k-m} + 1. \tag{31}$$

After substituting $k = j + 1$ into (26) to find QE_k, (31) can be rewritten as follows,

$$\frac{X}{Y} = \frac{R_k}{Y} + \mathcal{LB}_0 + QE_k - \delta_k + 1. \tag{32}$$

In the above equation, δ_k is decided by R_k/Y [from (13) and (14)]. There are two cases:

1) $R_k/Y \geq 0$, then we find $\delta_k = 1$, and (32) becomes

$$\frac{X}{Y} = \frac{R_k}{Y} + \mathcal{LB}_0 + QE_k.$$

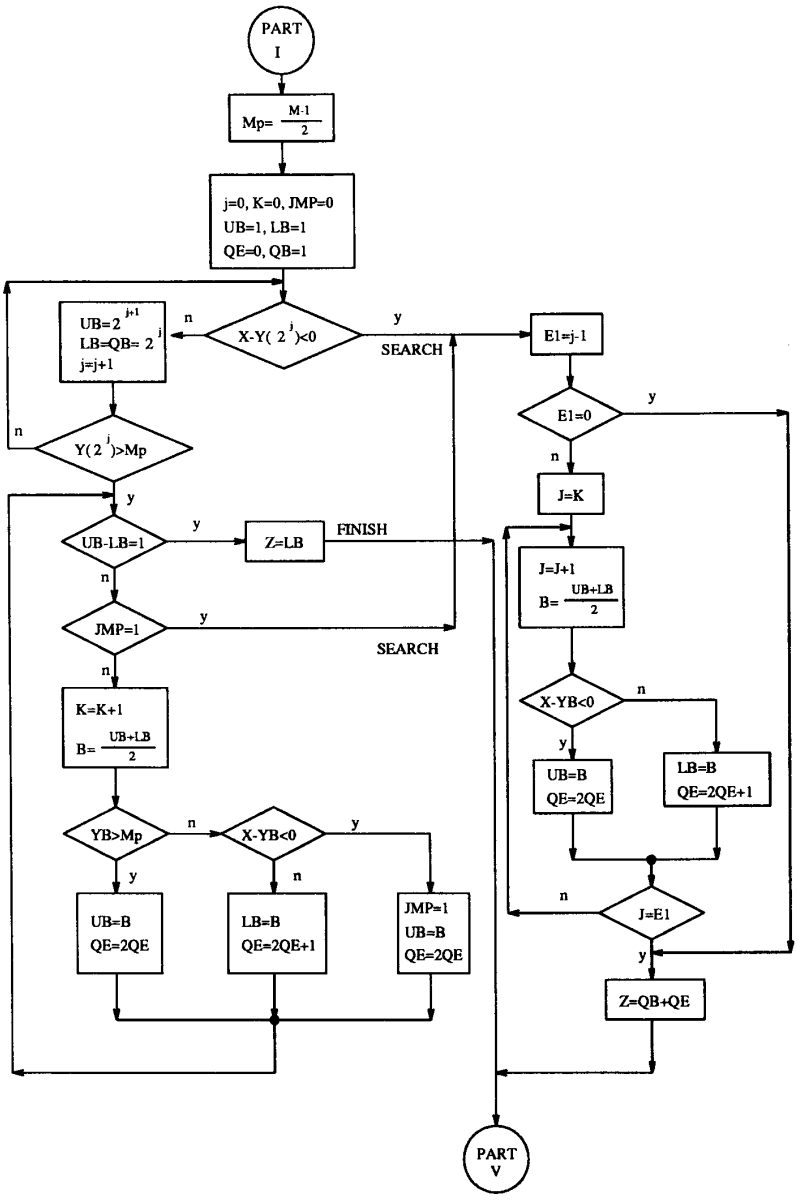

Fig. 1. Flowchart of the division algorithm (Part II to Part IV).

Since $R_k/Y < 1$, taking the floor values of both sides in the above equation, we can find the desired quotient value, Z, by

$$\left\lfloor \frac{X}{Y} \right\rfloor = \left\lfloor \frac{R_k}{Y} + \mathcal{LB}_0 + QE_k \right\rfloor = \mathcal{LB}_0 + QE_k = Z.$$

2) $R_k/Y < 0$, then we find $\delta_k = 0$, and (32) becomes

$$\frac{X}{Y} = \frac{R_k}{Y} + \mathcal{LB}_0 + QE_k + 1.$$

Since $R_k/Y < 0$ and $|R_k/Y| < 1$, therefore, $R_k/Y + 1 < 1$. Taking the floor values of both sides in the above equation, we can find the desired quotient value, Z, by

$$\left\lfloor \frac{X}{Y} \right\rfloor = \left\lfloor 1 + \frac{R_k}{Y} + \mathcal{LB}_0 + QE_k \right\rfloor = \mathcal{LB}_0 + QE_k = Z.$$

C. Division Algorithm

The flowchart of Part II to Part IV of the algorithm is shown in Fig. 1. Since LB and UB are used as program variables, they can represent previously described LB_i and UB_i, or $\mathcal{LB}_j$ and $\mathcal{UB}_j$.

D. Discussions

This algorithm requires five parts of computations. Constant time is needed in Part I to find the absolute values of the dividend and the divisor, and in Part V to transfer the absolute value of the quotient, $|Z|$, to the proper form. In Parts II, III, and IV, our algorithm needs $(2 \cdot \log_2 Z)$ steps to finish the division operation. The first $\log_2 Z$ steps find the range which the quotient falls in, and the second $\log_2 Z$ steps find the difference between QB and the quotient. Each step needs several RNS additions and subtractions, one RNS multiplication, a table look-up for the parity of S_i's, a table look-up for the $\hat{u}_i$'s, a multioperand binary addition over the $\hat{u}_i$'s and an Exclusive OR for finding the parity of a number.

We have implemented the presented algorithm on a divider with modulus set being $\{17, 13, 11, 7, 5\}$ and $M = 17 \times 13 \times 11 \times 7 \times 5 = 85085$. The Verilog Hardware Description Language (VHDL, Cadence simulation package) has been used to simulate the design in the logic gate level. The result has proved that our algorithm is correct and efficient as expected.

IV. Conclusion

We have presented a division algorithm which needs only simple RNS arithmetic operations, and which can be easily implemented. This is a general division algorithm, with no restrictions to either dividend or divisor. No estimation of the quotient is required before the division is executed. These characteristics make the calculation less complicated, more efficient, and speedier.

We also presented a very good and easy technique for overflow detections and number comparisons. In the traditional way of detecting overflow and comparing numbers in RNS, mixed radix numbers have to be used. This is time consuming and requires complex hardware. Our method is more efficient and less complicated than the existing algorithms.

A parity-checking technique is presented in this paper for number comparisons and overflow detections. We use the fractional number approach to find the parity of an RNS number. Several small tables and a multioperand binary adder are required. Some other small tables needed are for storing the values of the multiplicative inverse of 2, $|2^{-1}|_{m_i}$. Except the tables mentioned above, no other table is required, and all we need is simple arithmetic calculations. This algorithm can be easily implemented on hardware and can achieve good time performance which is logarithmic to the size of the quotient.

Acknowledgment

The authors would like to thank the anonymous reviewer for his helpful comments.

References

[1] W. K. Jenkins and B. J. Leon, "The use of residue number systems in the design of finite impulse response digital filters," *IEEE Trans. Circuits Syst.*, vol. CAS-24, no. 4, pp. 199–201, 1973.
[2] F. J. Taylor, "A VLSI residue arithmetic multiplier," *IEEE Trans. Comput.*, vol. C-31, pp. 540–546, June 1982.
[3] N. S. Szabo and R. I. Tanaka, *Residue Arithmetic and Its Application to Computer Technology*. New York: McGraw-Hill, 1967.
[4] W. A. Chren, Jr., "A new residue number system division algorithm," *Comput. Math. Appl.*, vol. 19, no. 7, pp. 13–29, 1990.
[5] S. Waser and M. J. Flynn, *Introduction to Arithmetic for Digital System Designers*. New York: Holt, Rinehart & Winston, 1982, ch. 5, p. 172.
[6] D. K. Banerji, T. Y. Cheung, and V. Ganesan, "A high-speed division method in residue arithmetic," in *Proc. 5th IEEE Symp. Comput. Arithmetic*, 1981, pp. 158–164.
[7] E. Kinoshita, H. Kosako, and Y. Kojima, "General division in the symmetric residue number system," *IEEE Trans. Comput.*, vol. C-22, pp. 134–142, Feb. 1973.
[8] Y. A. Keir, P. W. Cheney, and M. Tannenbaum, "Division and overflow detection in residue number systems," *IRE Trans. Electron. Comput.*, vol. EC-11, pp. 501–507, Aug. 1962.
[9] M.-L. Lin, E. Leiss, and B. McInnis, "Division and sign detection algorithm for residue number systems," *Comput. Math. Appl.*, vol. 10, no. 4/5, pp. 331–342, 1984.
[10] D. D. Miller, J. N. Polky, and J. R. King, "A survey of Soviet developments in residue number theory applied to digital filtering," in *Proc. 26th Midwest Symp. Circuits Syst.*, Aug. 1983.
[11] T. Van Vu, "Efficient implementations of Chinese remainder theorem for sign detection and residue decoding," *IEEE Trans. Comput.*, vol. C-34, pp. 646–651, July 1985.
[12] D. E. Atkins and S.-C. Ong, "Time-component complexity of two approaches to multioperand binary addition," *IEEE Trans. Comput.*, vol. C-28, pp. 918–926, Dec. 1979.

Mi Lu (S'84–M'86) received the B.S.E.E. degree from the Shanghai Institute of Mechanical Engineering, China, in 1981, and the M.S. and Ph.D. degrees in electrical and computer engineering from Rice University in 1984 and 1987, respectively.

She joined the Department of Electrical Engineering, Texas A&M University, College Station, as an Assistant Professor in 1987. Her research interests include parallel processing, computer arithmetic, parallel computer architectures and applications, computational geometry, and VLSI algorithms. She has published over 40 technical papers. Her research was funded by NSF and the Texas Advanced Technology Program.

Dr. Lu is a member of the IEEE Computer Society.

Jen-Shiun Chiang was born in Taichung, Taiwan, China, in 1960. He received the B.S. degree in electronic engineering from Tamkang University, Taipei, Taiwan, in 1983, and the M.S. degree in electrical engineering from University of Idaho, Moscow, ID, in 1988.

At present, he is pursuing the Ph.D. degree in electrical engineering at Texas A&M University. His research interests include computer arithmetic, ALU design, computer architecture, VLSI design, and parallel processing.

Mr. Chiang is a member of the IEEE Computer Society.