Computers Math. Applic. Vol. 19, No. 7, pp. 13–29, 1990
Printed in Great Britain. All rights reserved

0097-4943/90 $3.00 + 0.00

A NEW RESIDUE NUMBER SYSTEM DIVISION ALGORITHM

W. A. CHREN JR
Department of Electrical and Computer Engineering, The University of Kentucky, Lexington, KY 40506-0046, U.S.A.

(*Received* 7 *April* 1988; *in revised form* 23 *March* 1989)

Abstract—A new division algorithm is presented for the residue number system (RNS). It is 5% faster and has an 8% smaller standard deviation of running time than the best previously published algorithm. Furthermore, it uses storage proportional to the sum of the moduli, instead of the square of the largest modulus.

1. INTRODUCTION

1.1. The residue division problem

A residue number system (henceforth abbreviated as RNS) is a set of representations (known as "residue numbers" or "residue codes") of integers, together with rules for adding, subtracting and multiplying them. An RNS allows hardware implementations in which an integer is represented simultaneously in independent, small arithmetic units operating in parallel. Addition, subtraction and multiplication are performed on each arithmetic unit concurrently and independently, without the need for carries, borrows or partial products. These operations can be done quickly, in an amount of time that is essentially independent of the operation and the operands themselves. As a result, the RNS has received considerable attention in the literature [1–7].

The division operation, however, is relatively slow, and its execution time depends to a large extent on the operands. The fastest known division algorithm for arbitrary modulus sets has an average execution time that is almost 50 times longer than that required for addition (assuming randomly distributed numerators and denominators). Also, the division time standard deviation about the mean, a figure of merit previously not considered, can be as large as 40% for typical modulus sets. Furthermore, this algorithm uses a large stored table.

The problem considered here (henceforth called the "division problem") is to develop an RNS division algorithm that possesses the following attributes: (1) it works for all modulus sets so long as the moduli are pairwise relatively prime; (2) it does not require an inordinate amount of hardware or storage to implement; (3) it works for all possible divisors in the range of the RNS defined by the moduli. Furthermore, we would like the algorithm to be fast and have a small running time standard deviation. Such an algorithm would enhance immensely the appeal of the RNS for use as a general purpose computer number system.

1.2. Previous work

Several RNS division algorithms have been published. Each can be classified in one of two groups, as is the case with binary division algorithms [8]. These groups are referred to as the subtractive algorithms and the multiplicative algorithms. Members of the former employ subtraction of multiples of the denominator (called "quotient estimates" or "estimates" for short) from the numerator until the difference becomes less than the denominator. The multiplicative algorithms, however, compute a reciprocal of the divisor, and the quotient by multiplication of this reciprocal by the numerator.

Of the eight RNS division algorithms published to date, seven fall into the subtractive class. The eight will be referred to as the binary expansion Algorithms I, II and III (Refs [9–11], respectively), pure scaling algorithm [10, pp. 91–94], one sided rounding Algorithms I and II [12], Kinoshita algorithm [13], and large-divisor reciprocal algorithm [1].

The binary expansion Algorithm I computes, in residue, the terms in the binary expansion of the quotient. Binary expansion Algorithm II is the same as I, except that it uses a stored table lookup to replace a lengthy calculation. Binary expansion Algorithm III is an adaptation to the RNS of the well-known CORDIC algorithm. The pure scaling algorithm uses divisor approximations to compute quotient estimates. The approximate divisor is specially chosen so that the quotient estimates can be found exclusively by means of a fundamental residue procedure called "scaling". The one-sided rounding Algorithms I and II, as well as the Kinoshita algorithm, use mixed radix information about approximations to the denominator and numerator to access a stored table of quotient estimates. The large-divisor reciprocal algorithm uses a first order iterative process to compute a suitable defined reciprocal of the divisor, accurate to within an error of 1, for divisors which are not less than the largest mixed radix coefficient.

Five of these algorithms have drawbacks which make them less suitable as solutions of the RNS division problem. The binary expansion Algorithm III requires an amount of hardware that increases linearly with the dynamic range of the RNS, and is thus impractical for general purpose computer implementation. The large-divisor reciprocal algorithm does not work for all divisors in the range of the RNS. Furthermore, it is not clear how the quotient is to be found once the reciprocal is computed. The pure scaling and one sided rounding Algorithm II can be used only for specially chosen modulus sets. The Kinoshita algorithm requires either a decimal divider or a very large amount of storage.

Of the three remaining algorithms (binary expansion Algorithms I and II and the one sided rounding algorithms I), the one sided rounding Algorithm I is the fastest [12, p. 162]. For this reason, the one sided rounding Algorithm I will be considred the "best" RNS division algorithm published to date, and will be the algorithm with which our new algorithm is compared. However, this algorithm (henceforth abbreviated OSRA) has two shortcomings. The first is that its storage, although reasonable, is quite large. It requires an amount proportional to the square of the maximum modulus. This large amount is needed because the majority of the quotient estimates are stored rather than computed. The second drawback of the OSRA is that the standard deviation about the mean of its execution time is large, being as high as 40% for typical modulus sets.

1.3. Purpose of this research and the results achieved

The purpose of this research was to develop an RNS division algorithm which solves the division problem previously stated, and (a) uses less storage than the OSRA, and (b) has a smaller mean execution time and better standard deviation about the mean than the OSRA. This goal was originally motivated by the need for an RNS division algorithm for use in optical computing [7].

The result of this research is an algorithm called the reciprocal algorithm (RA). It solves the division problem stated previously, uses less storage than the OSRA, and has a smaller mean execution time and better standard deviation about the mean.

The remainder of this paper is organized as follows. Section 2 is a review of the RNS and various operations performed in it. It also contains a review of the OSRA. Section 3 presents the RA, and Section 4 compares its required storage and statistics of performance with those of the OSRA. Section 5 contains conclusions, and the Appendix contains proofs of the lemmas and theorems.

2. THE RNS AND THE OSRA

In this section we present the RNS, and review the basic operations used when calculating in it. The section also contains a detailed discussion of the OSRA, which as explained in the previous section, is considered to be the "best" RNS division algorithm so far published.

2.1. The residue representation

The RNS representation of an integer X is defined as follows. Let $\{m_1, m_2, \ldots, m_n\}$ be a set of positive integers greater than 1, called the "modulus" set (or "set of moduli"). Also, let m_i be called the ith modulus. Then the representation of X in the RNS corresponding to the modulus set $\{m_1, m_2, \ldots, m_n\}$ is the n-tuple of least nonnegative residues of X mod m_i, written $(x_1, x_2, \ldots, x_n)$, where $x_i = X \bmod m_i$. The phrase "X has the residue code $(x_1, x_2, \ldots, x_n)$" will sometimes be written

$$X \leftrightarrow (x_1, x_2, \ldots, x_n), \tag{1}$$

and x_i will sometimes be called the "ith residue digit of X." If the moduli are chosen relatively prime in pairs, then the Chinese remainder theorem guarantees that the residue representation of X is unique, provided that we choose X from a set of consecutive integers containing no more than M elements, where M is the product of the moduli. In the literature this set is chosen to be $[0, M-1]$ (where $[\alpha, \beta]$ denotes the set of integers between α and β, inclusive) if positive arithmetic alone is desired. If operands of both signs are needed, then

$$\left[\frac{-(M-1)}{2}, \frac{(M-1)}{2}\right]$$

is used if M is odd, or

$$\left[\frac{-M}{2}, \frac{M}{2} - 1\right]$$

is used if M is even. In either case, the chosen interval is called the "interval of definition" (ID) for the RNS corresponding to $\{m_1, m_2, \ldots, m_n\}$.

2.2. *Addition, subtraction and multiplication*

Residue addition, subtraction and multiplication are done as follows. Let " · " denote any one of these operations, and let X and Y be the operands, where

$$X \leftrightarrow (x_1, x_2, \ldots, x_n) \tag{2}$$

and

$$Y \leftrightarrow (y_1, y_2, \ldots, y_n). \tag{3}$$

Then

$$X \cdot Y \leftrightarrow (z_1, z_2, \ldots, z_n), \tag{4}$$

where z_i is defined as

$$z_i = x_i y_i \bmod m_i. \tag{5}$$

From this definition we see that the ith residue digit of the result depends solely on the ith residue of the operands.

2.3. *Integer division*

Integer division refers to the calculation of a quotient X/Y when it is known *a priori* that the quotient is an integer and the ith residue digit of Y is relatively prime to m_i for all i. It has limited application in itself but is used in a more general division method called the scaling procedure.

The integer division procedure is as follows. Let X and $Y \neq 0$ be in the ID, and let

$$X \leftrightarrow (x_1, x_2, \ldots, x_n) \tag{6}$$

and

$$Y \leftrightarrow (y_1, y_2, \ldots, y_n). \tag{7}$$

Then Z, the integral quotient, is

$$Z \leftrightarrow (z_1, z_2, \ldots, z_n), \tag{8}$$

where

$$z_i = x_i y_i^{-1} \bmod m_i, \tag{9}$$

and y_i^{-1} is the unique integer such that

$$1 \leqslant y_i^{-1} \leqslant m_i - 1 \tag{10}$$

and

$$y_i y_i^{-1} = 1 \bmod m_i. \tag{11}$$

The integer y_i^{-1} is called the multiplicative inverse of $y_i \bmod m_i$. Clearly, integer division consists of a single residue multiplication of the numerator by the inverse of the denominator.

2.4. *The scaling procedure*

If the quotient is not known *a priori* to be an integer, then rounding of the result must be performed. The predominant form of rounding in the literature is "truncation" rounding, which converts the quotient X/Y to the greatest integer less than or equal to it, written $\lfloor X/Y \rfloor$. If Y is an element of a set of "special" divisors, the truncated quotient $\lfloor X/Y \rfloor$ can be found in a relatively simple manner, if we can accept the answer specified on a proper subset of the moduli. This is done by a procedure called the "scaling" procedure, which will now be described.

The scaling procedure is used only when the divisor Y is a product of first powers of moduli. Suppose that Y is the single modulus m_i. Then the truncated quotient

$$\left\lfloor \frac{X}{Y} \right\rfloor = \frac{X - X \bmod Y}{Y} = \frac{X - x_i}{m_i}, \tag{12}$$

where, as usual, x_i is the ith residue digit of X. Since $\lfloor X/Y \rfloor$ is an integer, it can be found in this case by using the integer division method described above. Since the moduli are relatively prime in pairs, $m_i^{-1} \bmod m_j$ exists for all $j \neq i$. Therefore, the jth residue digit of the truncated quotient, written $\lfloor X/Y \rfloor_j$, is

$$\lfloor X/Y \rfloor_j = (x_j - x_i) y_j^{-1} \bmod m_j, \quad j \neq i. \tag{13}$$

A method for finding the ith residue digit, called the base extension procedure, will be discussed later. If the divisor Y is a product of $k > 1$ single powers of moduli, then the truncated quotient can be found by successive applications of the preceding method, once for each modulus in the divisor. The interested reader is referred to Ref. [10] for a complete description of the scaling algorithm for this case.

2.5. *The mixed radix representation*

The mixed radix representation is a weighted (i.e. "positional") representation of an integer which is well-suited for the RNS. It allows the magnitude of the integer to be determined, and will now be described.

The mixed radix representation of a nonnegative operand X in the ID is defined to be

$$X = x_n \prod_{i=1}^{n-1} m_i + x_{n-1} \prod_{i=1}^{n-2} m_i + \cdots + x_2 m_1 + x_1, \tag{14}$$

where $0 \leqslant x_j < m_j$, for all j. The mixed radix representation of a negative X in the ID is defined to be that of the positive integer $M + X$. In this case,

$$m + X = x_n \prod_{i=1}^{n-1} m_i + x_{n-1} \prod_{i=1}^{n-2} m_i + \cdots + x_2 m_1 + x_1, \tag{15}$$

where, as in equation (14), $0 \leqslant x_j < m_j$, for all j. In either case, x_j is called the jth mixed radix digit of X. The mixed radix digits are found by successive scaling by the moduli, in a process called mixed radix conversion [10]. Throughout the remainder of this paper, we will denote the mixed radix digits of X by

$$X \leftrightarrow \langle x_n, x_{n-1}, \ldots, x_1 \rangle. \tag{16}$$

Furthermore, we will refer to x_n and x_i as the most significant and least significant mixed radix digits, respectively.

2.6. *The base extension procedure*

The base extension procedure is a method for computing unknown residue digits of an operand. These digits can be those which have been erased by the scaling procedure, or can be those corresponding to extra moduli. The base extension procedure is essentially a mixed radix conversion, and is described in full generality in Ref. [10]. The division algorithm presented in this paper uses a special case of the procedure. This special type of base extension, henceforth called a "broadcast", will now be discussed.

A broadcast is defined as a base extension in which the operand to be extended, X, is nonnegative and less than some modulus m_i for which x_i is known. Here, it is assumed that all residue digits of X are found by a stored table which contains the full length residue representations for all nonnegative integers less than the maximum modulus. Broadcasts are used in the scaling and mixed radix conversion procedures.

2.7. *The OSRA*

This subsection contains a detailed discussion of the OSRA, presented for later comparison. Experimentally derived statistics of its average running time will be given in Section 4. We begin with a short discussion of notation.

Let X and Y denote the numerator and denominator, respectively. The OSRA uses an iterative procedure, so we will use X_i to denote the numerator at iteration i. Also, we will denote the mixed radix digits of X_i and Y by

$$X_i \leftrightarrow \langle x_n, x_{n-1}, \ldots, x_1 \rangle \tag{17}$$

and

$$Y \leftrightarrow \langle y_n, y_{n-1}, \ldots, y_1 \rangle, \tag{18}$$

with x_n and y_n the most significant. Furthermore, we will let x_k and y_l be the most significant nonzero mixed radix digits of X_i and Y, respectively. Also, we will denote by E_{i+1} an estimate of the quantity X_i/Y. This notation is the same as that used by Banerji *et al.* [2], except that E_{i+1} is used instead of Z_{i+1}.

2.7.1. Overview of the method. The OSRA finds the truncated quotient $\lfloor X/Y \rfloor$ for any X and $Y \neq 0$ in the ID $\lfloor 0, m-1 \rfloor$. The moduli can be any positive pairwise relatively prime integers, and they are assumed to be ordered as $m_n > \cdots > m_1$.

The OSRA proceeds as follows. In the ith iteration, it makes an estimate E_{i+1} of the quantity X_i/Y, and tests to see if this estimate equals zero. If it does, then the algorithm stops, and $\lfloor X/Y \rfloor$ is the sum of the estimates plus an error term. Otherwise, X_{i+1} is computed and a new estimate is made.

2.7.2. Formal statement of the OSRA. The OSRA uses the iteration

$$X_{i+1} = X_i - E_{i+1} Y, \tag{19}$$

where $X_0 = X$, with stopping conditions $X_r = 0$ or $E_{r+1} = 0$. When one of the stopping conditions is satisfied, the truncated quotient is given by

$$\left\lfloor \frac{X}{Y} \right\rfloor = \sum_{i=0}^{r-1} E_{i+1} + E', \tag{20}$$

where

$$E' = \begin{cases} 0, & \text{if } X_r = 0 \text{ or } E_{i+1} = 0 \text{ and } X_r < Y, \\ 1, & \text{if } E_{i+1} = 0 \text{ and } X_r \geqslant Y. \end{cases} \tag{21}$$

The estimate E_{i+1} is found from mixed radix information about rounded approximations to X_i and Y. Y is approximated as

$$Y \approx (y_l + 1) m_{l-1} \cdots m_1, \tag{22}$$

and X_i is approximated as

$$X_i \approx x_k m_{k-1} \cdots m_1. \tag{23}$$

Using these approximations, the quotient estimate E_{i+1} is computed as

$$E_{i+1} = \begin{cases} 0, & \text{if } k < l, \\ \left\lfloor \dfrac{k}{y_l + 1} \right\rfloor, & \text{if } k = l, \\ x_k \left\lfloor \dfrac{m_l}{y_l + 1} \right\rfloor, & \text{if } k = l + 1, \\ x_k \left\lfloor \dfrac{m_l}{y_l + 1} \right\rfloor, & \text{if } k > l. \end{cases} \tag{24}$$

The residue codes for $\lfloor x_k/(y_l + 1) \rfloor$ and $\lfloor m_l/(y_l + 1) \rfloor$ are found by accessing a table which stores the residue representation of all possible quotients $\lfloor \alpha/(y_l + 1) \rfloor$, for $1 \leqslant \alpha \leqslant m_n - 1$. This table is indexed by the quantities x_k (or equivalently m_l) and y_l, and the residue code for the estimate is read out. The residue codes for all possible products $m_{l+1} \ldots m_{k-1}$, for $k > l + 1$ are stored in another table, indexed by k and l. The residue representation of x_k is found by a broadcast of x_k to all other moduli.

A Pascal version of the OSRA is given in Fig. 1.

3. THE RECIPROCAL ALGORITHM

The reciprocal algorithm (RA) is the major result of this research. It computes the truncated quotient $\lfloor X/Y \rfloor$ in smaller mean execution time, with less variability about the mean, and while using much less storage than the OSRA. The reduction in storage is a result of the fact that the quotient estimates are computed, rather than stored. The computation involves a newly defined reciprocal of the divisor, and is performed (for most modulus sets) in an extra "scratch pad" modulus. This extra modulus is not necessary, however, if the moduli can be chosen arbitrarily.

The notation used in Section 3 is the same as that used in Sections 1 and 2. Furthermore, we define

$$P_\alpha = \prod_{i=1}^{\alpha} m_i \tag{25}$$

and $P_0 = 1$.

3.1. *Overview of the RA*

The RA computes the truncated quotient $\lfloor X/Y \rfloor$ for any X and $Y \neq 0$ in the ID $[0, M - 1]$ defined by the mod $m_n > m_{n-1} > \cdots > m_1$. These moduli may be any pairwise relatively prime integers. In the ith iteration, the algorithm makes a quotient estimate E_{i+1} of the quantity X_i/Y, and tests to see if a stopping condition is satisfied. If it is, the truncated quotient is the sum of the estimates plus an error term. Otherwise, a new numerator X_{i+1} is calculated, and this process is repeated.

The RA has a smaller average execution time than the OSRA, and slightly smaller standard deviation. Most importantly, the RA uses much less storage.

3.2. *Formal statement of the RA*

The RA uses the iteration

$$X_{i+1} = X_i - E_{i+1} Y, \tag{26}$$

where $X_0 = X$, with stopping conditions $X_r < Y$ or $E_{r+1} = 0$. When the stopping condition is satisfied, the truncated quotient

```
PROGRAM OSRA (INPUT, OUTPUT);

(* This program uses the One Sided Rounding Algorithm to compute the truncated
quotient, QUOT, of numerator NUMER and denominator DENOM. Its purpose is to
illustrate the OSRA, and therefore the residue representation is not used since its use
would obscure the essentials of the OSRA*)

VAR NUMER, DENOM, QUOT: INTEGER;
    MSLOCN,MSLOCD,MSDIGITN,ESTIMATE : INTEGER;
    MODULUS  : ARRAY[1..10] OF INTEGERS;
    MRDIGITSN: ARRAY [1..10] OF INTEGERS;
    MRDIGITSD: ARRAY[1..10] OF INTEGERS;
    LABEL 5;
BEGIN
    QUOT:=0;
    CONVERT(DENOM,MRDIGITSD,MSLOCD);
    MSDIGITD := MRDIGITSD(MSLOCD);
    WHILE NUMER < > 0 DO
       BEGIN
          CONVERT(NUMER,MRDIGITSN,MSLOCN);
          MSDIGIT:= MRDIGITSN(MSLOCN);
          IF MSLOCN < MSLOCD
          THEN ESTIMATE :=0
          ELSE IF MSLOCN = MSLOCD
                THEN ESTIMATE:= FINDQUOT(MSDIGITN,MSDIGITD)
                ELSE IF MSLOCN=MSLOCD + 1
                   THEN ESTIMATE:= MSDIGITN
                                   *FINDQUOT(MODULUS(MSLOCD),MSDIGITD)
                   ELSE ESTIMATE:= MSDIGITN
                                   * FINDQUOT(MODULUS(MSLOCD),MSDIGITD)
                                   * FINDPROD(MSLOCN,MSLOCD);
          IF ESTIMATE = 0
          THEN BEGIN
               IF NUMER > = DENOM
               THEN QUOT:= QUOT +1;
               GO TO 5
               END
          ELSE BEGIN
               QUOT: = QUOT + ESTIMATE;
               NUMER: = NUMER - ESTIMATE * DENOM;
               END;
       END;
    5: WRITELN(QUOT);
END.
```

Procedure CONVERT (X,Y,Z) returns in Y the mixed radix digits of X, and returns in Z the location of the most significant nonzero digit. Function FINDQUOT(X,Y) returns the quantity $\left\lfloor \frac{X}{Y+1} \right\rfloor$ and represents the quotient table used by OSRA. Function FINDPROD (X,Y) returns the product $m_{l+1} \ldots m_{k-1}$ when passed k,l in X,Y, respectively. It represents the modulus products table used by OSRA)

Fig. 1

$$\left\lfloor \frac{X}{Y} \right\rfloor = \sum_{i=0}^{r-1} E_{i+1} + E', \tag{27}$$

where

$$E' = \begin{cases} 0, & \text{if } X_r < Y, \\ 1, & \text{if } E_{r+1} = 0. \end{cases} \tag{28}$$

The $(i+1)$st quotient estimate is

$$E_{i+1} = \begin{cases} \left\lfloor \dfrac{x_k \left\lfloor \dfrac{P_l}{Y} \right\rfloor}{m_k} \right\rfloor, & \text{if } k = l, \\ x_k \left\lfloor \dfrac{P_l}{Y} \right\rfloor, & \text{if } k = l+1, \\ x_k \left\lfloor \dfrac{P_l}{Y} \right\rfloor m_{l+1} \cdots m_{k-1}, & \text{if } k > l+1. \end{cases} \tag{29}$$

This estimate is an approximation of the quantity

$$E = \left\lfloor \frac{X_i \left\lfloor \dfrac{M}{Y} \right\rfloor}{M} \right\rfloor, \tag{30}$$

which can be shown to be a low estimate of X_i/Y, in error by at most 1. The approximation is accomplished by using approximations of X_i and $\lfloor m/Y \rfloor$, as follows. X_i is approximated by its most significant mixed radix term, viz.

$$X_i \approx x_k P_{k-1}. \tag{31}$$

The quantity $\lfloor M/Y \rfloor$ is approximated as

$$\left\lfloor \frac{M}{Y} \right\rfloor \approx m_n m_{n-1} \cdots m_{l+1} \left\lfloor \frac{P_l}{Y} \right\rfloor, \tag{32}$$

where as defined earlier, the most significant nonzero mixed radix digit of Y is the lth.

3.2.1. Calculation of the estimate E_{i+l}. The estimate E_{i+1} is calculated using either the scaling procedure or stored tables, depending on k and 1.

If $K > l$, two stored tables are used; one for $\lfloor P_l/Y \rfloor$ and the other for the product $m_{l+1} \ldots m_{k-1}$. The product table is identical to the table used in the OSRA. The divisor reciprocal table storing residue encoded values of $\lfloor P_l/Y \rfloor$ is new and will now be discussed.

The residue encoded value of $\lfloor P_l/Y \rfloor$ is found by a search of a table T_l. In order to discuss the entries in T_l, we introduce the following notation. We will denote by $j_1 < j_2 < \ldots < j_s$ the unique values that $\lfloor P_l/Y \rfloor$ can assume as Y ranges throughout $[P_{l-1}, P_l - 1]$. Furthermore, we will let Y_{j_i} denote the largest Y in $[P_{l-1}, P_l - 1]$ such that $\lfloor P_l/Y \rfloor = j_i$. The values Y_{j_i} can be thought of as the values of Y for which $\lfloor P_l/Y \rfloor$ "changes" as Y assumes values starting at $P_l - 1$ and decreasing to P_{l-1}. The values j_i are then equal to $\lfloor P_l/(Y_{j_i}) \rfloor$.

Divisor reciprocal table T_l stores the ordered pairs of integers (Y_{j_i}, j_i) for $i = 1, 2, \ldots, s$ in increasing order on j_i. The integers Y_{j_i} are mixed radix encoded, while the integers j_i are residue coded.

The residue coded value of $\lfloor P_l/Y \rfloor$ is found by using the mixed radix digits of Y to search the entries Y_{j_i} in T_l. The search continues until the pair (Y_{j_a}, j_a) such that

$$Y_{j_a+1} < Y \leqslant Y_{j_a} \tag{33}$$

is found. At such a time, the residue encoded value of $\lfloor P_l/Y \rfloor = j_a$ is read out. This process needs to be done only once, at the beginning of a division problem.

The size of T_l is bounded by m_l, because $\lfloor P_l/Y \rfloor \leqslant m_l$. This makes the storage used by the RA proportional to the sum of the moduli, which is an improvement of the OSRA storage.

If $k = l$, the scaling procedure is used to compute the estimate. The residue code for the divisor reciprocal $\lfloor P_l/Y \rfloor$ is found by stored table as discussed above. Its product with x_k is scaled by mod m_k. As discussed previously, the scaling procedure erases those residue digits which correspond to the scaling moduli. The kth digit of the estimate is restored by a broadcast from the nth (if $k < n$), because in this case it can be shown that $E_{i+1} < m_n$. However, in the case $k = n$, the nth digit cannot be restored from a single original modulus, in general, because E_{i+1} could exceed m_{n-1} for general modulus sets. Consequently, in general the RA requires the use of a "scratch pad" mod $m_{n+1} = m_n - 1$. This modulus is used to restore (by broadcast) the erased nth digit of E_{i+1} when $k = n$, because in this case it can be shown that $E_{i+1} < m_n - 1$. Note that for modulus sets such that $m_{n-1} = m_n - 1$, the extra modulus is not needed. In this case, the nth digit of E_{i+1} can be restored from the $(n - 1)$st digit. Consequently, it is recommended that the modulus set be chosen such that $m_{n-1} = m_n - 1$ if the RA is to be used.

3.3. Proof of the validity of the RA

The RA will be shown correct in four steps. The first, Lemma 1, shows what range of values E_{i+1} must have if the algorithm is to stop eventually. The second step, Lemma 2, shows that if the stopping condition $E_{r+1} = 0$ is satisfied, then $X_r < 2Y$. The third step, Lemma 3, shows that the estimate E_{i+1} either equals zero, or lies in the "convergence" range established by Lemma 1. Finally, Theorem 1 shows that the RA halts and computes the truncated quotient $\lfloor X/Y \rfloor$. All proofs are in the Appendix.

Lemma 1

For the iteration

$$X_{i+1} = X_i - E_{i+1} Y, \tag{34}$$

we have

$$0 < E_{i+1} \leqslant \frac{X_i}{Y} \Rightarrow 0 \leqslant X_{i+1} < X_i.$$

Lemma 2

For the RA estimate, viz.

$$E_{i+1} = \begin{cases} \left\lfloor \dfrac{x_k \left\lfloor \dfrac{P_l}{Y} \right\rfloor}{m_k} \right\rfloor, & \text{if } k = l, \\ x_k \left\lfloor \dfrac{P_l}{Y} \right\rfloor, & \text{if } k + l + 1, \\ x_k \left\lfloor \dfrac{P_l}{Y} \right\rfloor m_{l+1} \cdots m_{k-1}, & \text{if } k > l + 1, \end{cases} \tag{35}$$

we have $E_{i+1} = 0 \Rightarrow X_i < 2Y$.

Lemma 3

For the RA estimate (repeated in Lemma 2 above), we have

$$0 \leqslant E_{i+1} \leqslant \frac{X_i}{Y},$$

for all X_i, $Y \neq 0$.

Theorem 1

The RA eventually halts, at which time

$$\left\lfloor \frac{X}{Y} \right\rfloor = \sum_{i=0}^{r-1} E_{i+1} + E', \tag{36}$$

where

$$E' = \begin{cases} 0, & \text{if } X_r < Y, \\ 1, & \text{if } E_{r+1} = 0. \end{cases} \tag{37}$$

A Pascal version of the RA is given in Fig. 2.

3.4. Examples of use of the RA

The first example shows the calculations when the scratch pad modulus is not used, and the second shows how it is used. In these examples, angle brackets "$\langle a_3, a_2, a_1 \rangle$" will denote mixed radix digits, with the least significant digit on the right. Parentheses "(r_3, r_2, r_1)" will denote residue digits. The symbol "d" will denote a "don't care" residue digit. The operation count is the sum of the number of additions, subtractions and multiplications needed.

```
PROGRAM RA (INPUT, OUTPUT);

(*This program uses the Reciprocal Algorithm to compute the truncated quotient, QUOT
of numerator NUMER and denominator, DENOM. Its purpose is to illustrate the RA, and
therefore the residue representation is not used, since its use would obscure the essentials of
the RA *)

VAR NUMER, DENOM, QUOT: INTEGER;
    INVERSE: INTEGER;
    MSLOCN, MSLOCD, MSDIGITN, ESTIMATE: INTEGER;
    MODULUS: ARRAY[1..10] OF INTEGERS;
    MRDIGITSN: ARRAY[1..10] OF INTEGERS;
    MRDIGITSD: ARRAY[1..10] OF INTEGERS;
    LABEL 5;
BEGIN
    QUOT:=0;
    CONVERT (DENOM,MRDIGITSD,MSLOCD);
    RECIP (MRDIGITSD, INVERSE);
    CONVERT (NUMER,MRDIGITSN,MSLOCN);
    WHILE NUMER > = DENOM DO
       BEGIN
          MSDIGITN:= MRDIGITSN (MSLOCN);
          IF MSLOCN = MSLOCD
          THEN BEGIN
                ESTIMATE := (MSDIGITN * INVERSE) DIV MODULUS(MSLOCN);
                IF ESTIMATE = 0
                THEN BEGIN
                           QUOT := QUOT + 1;
                           GO TO 5
                        END;
                END;
          ELSE IF MSLOCN = MSLOCD +1
                THEN ESTIMATE := INVERSE * MSDIGITN
                ELSE ESTIMATE := INVERSE * MSDIGITN* FINDPROD(MSLOCN,MSLOCD);
          QUOT:= QUOT + ESTIMATE;
          NUMER:= NUMER - ESTIMATE * DENOM;
          CONVERT (NUMER, MRDIGITSN, MSLOCN);
       END;
    5: WRITELN (QUOT);
END.

(* Procedure CONVERT (X,Y,Z) returns in Y the mixed radix digits of X, and returns in
Z the location of the most significant nonzero digit. RECIP (X,Y) searches the divisor
reciprocal table using the mixed radix digits in X. It returns the divisor reciprocal in Y.
Function FINDPROD (X,Y) returns product m_{l+1}...m_{k-1} when passed k,l, in X,Y,
respectively. It represents the modulus product table *)
```

Fig. 2

Example 1

For the modulus set $\{m_3, m_2, m_1\} = \{17, 13, 11\}$, compute $\lfloor 2200/20 \rfloor$.

Solution. Extra modulus $m_4 = 16$ is needed for some division problems because $m_2 \neq m_3 - 1$ However, for this division problem, m_4 is not needed.

Step 1. Mixed radix conversion of Y gives $Y \leftrightarrow \langle 0, 1, 9 \rangle$ and $l = 2$. The conversion requires 4 operations.

Step 2. Find $\lfloor P_2/Y \rfloor$ by searching the table T_2 with $Y \leftrightarrow \langle 0, 1, 9 \rangle$. T_2 contains the entries

$(\langle 0, 12, 10\rangle,$	$(1, 1, 1))$
$(\langle 0, 6, 5\rangle,$	$(2, 2, 2))$
$(\langle 0, 4, 3\rangle,$	$(3, 3, 3))$
$(\langle 0, 3, 2\rangle,$	$(4, 4, 4))$
$(\langle 0, 2, 6\rangle,$	$(5, 5, 5))$
$(\langle 0, 2, 1\rangle,$	$(6, 6, 6))$
$(\langle 0, 1, 9\rangle,$	$(7, 7, 7))$
$(\langle 0, 1, 6\rangle,$	$(8, 8, 8))$
$(\langle 0, 1, 4\rangle,$	$(9, 9, 9))$
$(\langle 0, 1, 3\rangle,$	$(10, 10, 10))$
$(\langle 0, 1, 2\rangle,$	$(11, 11, 0))$
$(\langle 0, 1, 0\rangle,$	$(13, 0, 2)).$

We find $\langle 0, 1, 6\rangle < Y \leqslant \langle 0, 1, 9\rangle$, so that $\lfloor P_2/Y \rfloor = (7, 7, 7)$.

Step 3. Mixed radix conversion on X_0 gives $X_0 \leftrightarrow \langle 15, 5, 0\rangle$. Since $X_0 \geqslant Y$, we proceed with $k = 3$. Add 4 to the operation count.

Step 4. Since $k = l + 1$, we compute

	17	13	11
x_3:	15	2	4
$\lfloor P_2/Y \rfloor$:	$x7$	7	7
E_1:	3	1	6
Y:	$x3$	7	9
$E_1 Y$:	9	7	10
X_0:	7	3	0
X_1:	15	9	1

Add 3 to the operation counter.

Step 5. Mixed radix conversion on X_1 gives $X_1 \leftrightarrow \langle 0, 9, 1\rangle$.
Add 4 to operation counter. We have $X_1 \geqslant Y$ and $k = 2$.

Step 6. We calculate

	17	13	11
x_2:	9	9	9
$\lfloor P_2/Y \rfloor$:	$x7$	7	7
	12	11	8
Scale by m_2:	-11	11	0
	1	0	8
	$x4$		6
E_2:	4		4
restore using m_3:	4	4	4

Add 3 to the operation counter.

Step 7. Since $E_2 \neq 0$, we compute $X_2 = X_1 - E_2 Y \leftrightarrow (3, 7, 9)$. Add 2 to the operation counter.

Step 8. Mixed radix conversion on X_2 gives $X_2 \leftrightarrow \langle 0, 1, 9\rangle$. Add 4 to operation counter. We have $X_2 \geqslant Y$ and $k = 2$.

Step 9. $k = l$, so we compute

	17	13	11
x_2:	1	1	1
$\lfloor P_2/Y \rfloor$:	$x7$	7	7
	7	7	7
Scale by m_2:	-7	7	7
	0	0	0
	$x4$		6
E_2:	0		0
restore using m_3:	0	0	0

Add 3 to the operation counter.

Step 10. Since $E_3 = 0$, we stop and compute the quotient $Q \leftrightarrow (3, 1, 6) + (4, 4, 4) + (1, 1, 1) = (8, 6, 0) \leftrightarrow 110$, answer. Add 1 to the operation counter because of the addition of the error term. The operation count for the problem is 30.

Example 2

For the same modulus set, find $\lfloor 2043/171 \rfloor$.

Solution. Scratch pad mod $m_4 = 16$ is needed for this example.

Step 1. Mixed radix conversion on Y gives $Y \leftrightarrow \langle 1, 2, 6 \rangle$. Add 4 to the operation counter.

Step 2. $\lfloor P_3/Y \rfloor$ is found by search of table T_3 using $\langle 1, 2, 6 \rangle$. The ordered pairs found in the search are $(\langle 1, 2, 8 \rangle, (14, 1, 3, 14))$ and $(\langle 1, 1, 8 \rangle, (15, 2, 4, 15))$, because $\langle 1, 1, 8 \rangle < Y \leqslant \langle 1, 2, 8 \rangle$. So $\lfloor P_3/Y \rfloor \leftrightarrow (14, 1, 3, 14)$. Note that mod 16 information has been included and that in general, T_n will be the only table which will include information for the $(n + 1)$st residue digit.

Step 3. Mixed radix conversion on X_0 gives $X_0 \leftrightarrow \langle 14, 3, 8 \rangle$. Add 4 to the operation counter.

Step 4. Since $k = 1$, we compute

	17	13	11	16
x_3:	14	1	3	14
$\lfloor P_3/Y \rfloor$:	$\times 14$	1	3	14
	9	1	9	4
scale by m_3:	-9	9	9	9
	0	5	0	11
	$\times 10$		2	1
E_1:	11		0	11
restore using m_4:	11	11	0	11

Add 3 to operation counter.

Step 5. X_1 is calculated as

E_1:	11	11	0	11
Y:	$\times 1$	2	6	d
$E_1 Y$:	11	9	0	d
X_1:	9	6	8	d

Add 2 to operation counter.

Step 6. Mixed radix conversion on X_1 gives $X_1 \leftrightarrow \langle 1, 1, 8 \rangle$. Add 4 to operation counter. Since $X_1 < Y$, stop with

$$\left\lfloor \frac{2043}{171} \right\rfloor = Q \leftrightarrow (11, 11, 0) \leftrightarrow 11.$$

The operation count for the problem is 17.

4. COMPARISON OF THE RA WITH THE OSRA

This section is a comparison of the RA with the OSRA. The comparison is made for three figures of merit. These are stored table size, mean execution time and standard deviation of execution time. Execution time is measured by the number of residue additions, subtractions and multiplications required by a division problem.

4.1. Comparison of storage requirements of RA with OSRA

The total storage used by the OSRA is given by the expression

$$\frac{(m_n-1)(m_n-2)}{2}+\frac{(n-2)(n-1)}{2}. \tag{38}$$

The first term in the storage required by the $\lfloor \alpha/y_l+1 \rfloor$ table, assuming that table entries for $\alpha < y_l+1$ are not stored (since they equal zero). Such a table is of size $(m_n-1)(m_n-2)/2$, since $1 \leqslant \alpha \leqslant m_n-1$ and $1 \leqslant y_l \leqslant m_n-1$. The second term is the total storage required by the table of products $m_{l+1} \ldots m_{k-1}$, and is $(n-2)(n-1)/2$.

The total storage used by the RA is

$$S_{\mathrm{RA}} < \sum_{l=1}^{n} (m_l-1)+\frac{(n-2)(n-1)}{2}. \tag{39}$$

This will now be shown.

The RA uses $n+1$ stored tables. One of these tables stores products $m_{l+1} \cdots m_{k-1}$ for $k > l+1$, and its size is $(n-2)(n-1)/2$ as explained for the OSRA. Tables T_l, for $l = 1, 2, \ldots, n$ are used to determine $\lfloor P_l/Y \rfloor$ when $P_{l-1} \leqslant Y \leqslant P_l-1$. Table T_l stores ordered pairs (Y_{j_i}, j_i), where Y_{j_i} is the mixed radix form of the largest Y such that $\lfloor P_l/Y \rfloor = j_i$, and j_i is one of the s unique volumes that $\lfloor P_l/Y \rfloor$ can have for $P_{l-1} \leqslant Y \leqslant P_l-1$. The size of T_l is s, and it can be shown that

$$s = \begin{cases} m_l-1, & \text{if } m_l \leqslant P_{l-1}, \\ 2\lfloor\sqrt{P_l}\rfloor - P_{l-1}, & \text{if } P_{l-1} > l-1 \text{ and } \left\lfloor \dfrac{P_l}{\lfloor\sqrt{P_l}\rfloor} \right\rfloor \neq \lfloor\sqrt{P_l}\rfloor, \\ 2\lfloor\sqrt{P_l}\rfloor - P_{l-1}-1, & \text{if } m_l > P_{l-1} \text{ and } \left\lfloor \dfrac{P_l}{\lfloor\sqrt{P_l}\rfloor} \right\rfloor = \lfloor\sqrt{P_l}\rfloor. \end{cases} \tag{40}$$

However, a simple upper bound for s is found as follows. Conservatively, $s \leqslant m_l$, because $1 \leqslant \lfloor P_l/Y \rfloor \leqslant m_l$. But since we always have $Y_{j_1} = P_l - 1$, the first ordered pair need not be stored. Therefore, a simple upper bound for s is $m_l - 1$.

From these formulas, it can be seen that the RA storage is proportional to the sum of the moduli, while the OSRA storage is proportional to the square of the maximum modulus.

4.2. Statistics of execution time for the RA and OSRA

The RA and OSRA were simulated on the following five modulus sets:

$$M1 = \{3, 5, 7, 11, 13, 17, 19, 23, 29, 31\};$$
$$M2 = \{31, 37, 41, 43, 47, 53, 55, 59, 61, 63\};$$
$$M3 = \{23, 29, 31, 37, 41, 43, 47, 53, 59, 61\};$$
$$M4 = \{37, 41, 43, 47, 53, 55, 59, 61, 63, 64\};$$
$$M5 = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29\}.$$

Each set was chosen to have the same number of moduli because the execution time of the algorithms is dependent on the number of moduli. Each modulus was chosen so that its residues are representable by at most six binary bits. This enables arithmetic to be performed by commercially available 4K × 8 PROMS in the large moduli.

Subject to the above two restrictions, modulus set $M1$ is the set of smallest odd relatively prime moduli. Set $M2$ is the set of largest odd relatively prime moduli. Set $M3$ is the set of largest odd prime moduli. Set $M4$ is the set of largest relatively prime moduli. Set $M5$ is the set of smallest relatively prime moduli.

Simulating programs for the RA and OSRA were written. These programs and their flowcharts are given in Ref. [13].

Table 1. Time statistics for the RA and OSRA

	Execution time and difference (%)			Standard deviation and difference (%)		
Modulus sets	RA	OSRA	%	RA	OSRA	%
{3, 5, 7, 11, 13, 17, 19, 23, 29, 31}	47.6	50.35	5	18.0	20.2	11
{37, 41, 43, 47, 53, 55, 59, 61, 63, 64}	47.3	49.4	4	17.0	18.2	7
{23, 29, 31, 37, 41, 43, 47, 53, 59, 61}	47.0	49.2	4	16.7	17.9	7
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29}	47.7	50.4	5	18.3	20.7	12
{31, 37, 41, 43, 47, 53, 55, 59, 61, 63}	47.25	49.2	4	17.0	17.9	5
Average:	47.4	49.7	5	17.5	19.0	8

For each of $M1$–$M5$, the RA and OSRA were simulated on a sample set of 40,000 division problems, randomly selected over the ID $[0, M - 1]$ defined by each modulus set. The sample average execution times and sample standard deviations of execution time are listed in Table 4.1. The sample average execution time is measured in units of number of elementary residue operations (viz. additions, subtractions and multiplications). Table accesses and compares were not counted, following the literature prevalent method of measuring execution time for residue arithmetic algorithms [10, 12].

From Table 1 we see that the RA uses 2.3 (5%) fewer residue operations per problem than the OSRA, for the modulus sets chosen. Furthermore, the standard deviation of the RA is 1.5 operations (8%) less than that of the OSRA, for the modulus sets chosen.

5. CONCLUSION

The purpose of this study was to develop an RNS division algorithm which uses less storage than the (OSRA) and has a smaller mean execution time and standard deviation about the mean than the OSRA.

The result achieved is an algorithm called the (RA). It is shown to be correct, examples are given and its statistics of performance are presented. It solves the division problem using a reciprocal approach, which is heretofore considered only in the Soviet literature [1]. For typical modulus sets, it is an improvement of the best previously published algorithm (OSRA). Specifically,

(1) it uses significantly less storage than the OSRA;
(2) it has a smaller mean execution time and smaller standard deviation about the mean than the OSRA.

Acknowledgement—The author is grateful to Professor Stuart A. Collins Jr for suggesting the topic.

REFERENCES

1. V. A. Vyshyskyy and V. D. Petushchak, Algorithm for determination of the reciprocal of a number in a residue class system. *Soviet autom. Control* **6**(3), 58–61 (1973).
2. 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 Systems* **CAS-24**(4), 199–201 (1977).
3. G. A. Jullien, Residue number scaling and other operations using ROM arrays. Report IEEE T-C, pp. 325–336, April (1978).
4. W. K. Jenkins, Recent advances in residue number techniques for recursive digital filtering. Report IEEE T-ASSP-27, No. 1, pp. 19–30, Feb. (1979).
5. F. J. Taylor, Residue arithmetic: a tutorial with examples. *Computer* **May,** 50–62 (1984).
6. D. D. Miller, J. N. Polky and J. R. King, A survey of Soviet developments in residue number theory applied to digital filtering. *Proc. 26th Midwest Symp. Circuits Systems*, Aug. (1983).
7. Optical computing: a field in flux. *IEEE Spectrum Mag.*, pp. 34–57, Aug. (1986).
8. S. Waser and M. J. Flynn, *Introduction to Arithmetic for Digital Systems Designers*, p. 172. Holt, Rinehart & Winston, New York (1982).
9. Y. A. Keir, P. W. Cheney and M. Tannenbaum, Division and overflow detection in residue number systems. Report IRE EC-11, No. 4, pp. 501–507, Aug. (1962).
10. N. Szabo and R. I. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*, pp. 88–89. McGraw-Hill, New York (1967).
11. 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).
12. D. K. Banerji, T. Y. Cheung and V. Ganesan, A high speed division method in residue arithmetic. *IEEE Proc. Fifth Symp. Comput. Arith.*, pp. 158–164, May (1981).

13. W. A. Chren Jr, New and improved residue number system division algorithms. Ph.D. Dissertation, Dept. of Electrical Engineering, Ohio State University, Columbus (1987).
14. E. Kinoshita, H. Kosako and Y. Kojima, General division in the symmetric residue number system. Report IEEE T-C, No. 22, pp. 134–142, Feb. (1973).

APPENDIX

Lemma 1

For the iteration $X_{i+1} = X_i - E_{i+1} Y$, we have

$$0 < E_{i+1} \leqslant \frac{X_i}{Y} \Rightarrow 0 \leqslant X_{i+1} < X_i.$$

Proof. We have

$$0 < E_{i+1} \leqslant \frac{X_i}{Y} \Rightarrow X_i - \left(\frac{X_i}{Y}\right) Y \leqslant X_{i+1} < X_i - (0)Y \Leftrightarrow 0 \leqslant X_{i+1} < X_i.$$

Lemma 2

For the RA estimate, viz.

$$E_{i+1} = \begin{cases} \left\lfloor \dfrac{x_k \left\lfloor \dfrac{P_l}{Y} \right\rfloor}{m_k} \right\rfloor, & \text{if } k = l, \\ x_k \left\lfloor \dfrac{P_l}{Y} \right\rfloor, & \text{if } k = l + 1, \\ x_k \left\lfloor \dfrac{P_l}{Y} \right\rfloor m_{l+1} \cdots m_{k-1}, & \text{if } k > l + 1, \end{cases}$$

we have

$$E_{i+1} = 0 \Rightarrow X_i < 2Y.$$

Proof. We have $E_{i+1} = 0 \Rightarrow k = l$, obviously. Therefore, we must show that

$$\left\lfloor \frac{x_k \left\lfloor \dfrac{P_l}{Y} \right\rfloor}{m_k} \right\rfloor = 0 \Rightarrow X_i < 2Y.$$

By assumption, we have

$$X_i = x_k P_{k-1} + x_{k-1} P_{k-2} + \cdots + x_1$$

and

$$Y = y_l P_{l-1} + y_{l-1} P_{l-2} + \cdots + y_1 = y_k P_{k-1} + y_{k-1} P_{k-2} + \cdots + y_1,$$

since $k = l$. Furthermore, since $P_{k-1} \leqslant Y \leqslant P_k - 1$, we have $1 \leqslant \lfloor P_k / Y \rfloor \leqslant m_k$.

We will first show that $\lfloor P_k / Y \rfloor = j \Rightarrow \lfloor P_k / (j+1) \rfloor + 1 \leqslant Y$, for $j \in [1, m_k]$. This is because

$$\left\lfloor \frac{P_k}{Y} \right\rfloor = j \Rightarrow jY \leqslant P_k \leqslant (j+1)Y - 1 \Rightarrow \frac{P_k}{j+1} \leqslant Y - \frac{1}{j+1}$$
$$\Rightarrow \frac{P_k}{j+1} + \frac{1}{j+1} \leqslant Y \Rightarrow \left\lfloor \frac{P_k}{j+1} \right\rfloor + 1 \leqslant Y,$$

since Y is an integer. Now, let $\lfloor P_k / Y \rfloor = j$, for $j \in [1, m_k]$.

We have $E_{i+1} = 0 \Rightarrow \lfloor x_k j / m_k \rfloor = 0 \Rightarrow x_k j < m_k \Rightarrow j \neq m_k$. Also, for $j \in [1, m_k - 1]$,

$$x_k < \frac{m_k}{j} \Rightarrow \begin{cases} x_k \leqslant \left\lfloor \dfrac{m_k}{j} \right\rfloor, & \text{if } \dfrac{m_k}{j} \text{ not an integer,} \\ x_k \leqslant \left\lfloor \dfrac{m_k}{j} \right\rfloor - 1, & \text{if } \dfrac{m_k}{j} \text{ is an integer,} \end{cases}$$

$$\Rightarrow \begin{cases} X_i \leqslant \left\lfloor \dfrac{m_k}{j} \right\rfloor P_{k-1} + P_{k-1} - 1, & \text{if } \dfrac{m_k}{j} \text{ not an integer,} \\ X_i \leqslant \left(\left\lfloor \dfrac{m_k}{j} \right\rfloor - 1 \right) P_{k-1} + P_{k-1} - 1, & \text{if } \dfrac{m_k}{j} \text{ is an integer,} \end{cases}$$

$$\Rightarrow\begin{cases} X_i \leqslant \left(\left\lfloor \dfrac{m_k}{j} \right\rfloor + 1\right) P_{k-1} - 1, & \text{if } \dfrac{m_k}{j} \text{ not an integer,} \\ X_i \leqslant \left\lfloor \dfrac{m_k}{j} \right\rfloor P_{k-1} - 1, & \text{if } \dfrac{m_k}{j} \text{ is an integer,} \end{cases}$$

$$\Rightarrow\left\{ X_i \leqslant \left\lceil \frac{m_k}{j} \right\rceil P_{k-1} - 1, \right.$$

where "$\lceil \alpha \rceil$" denotes the least integer greater than or equal to α. We will show that $X_i < 2Y$ by showing that $jX_i < 2jY$ for $j \in [1, m_k - 1]$.

We want to determine if

$$jX_i \leqslant j \left\lceil \frac{m_k}{j} \right\rceil P_{k-1} - j < 2j \left\lfloor \frac{P_k}{j+1} \right\rfloor + 2j \leqslant 2jY.$$

But $j\lceil m_k/j \rceil P_{k-1} - j = (m_k + |-m_k|_j)P_{k-1} - j = P_k + P_{k-1}|-m_k|_j - j$, where $|\alpha|_j$ denotes the quantity α mod j. Furthermore

$$2j \left\lfloor \frac{P_k}{j+1} \right\rfloor + 2j = (j+1) \left\lfloor \frac{P_k}{j+1} \right\rfloor + (j-1) \left\lfloor \frac{P_k}{j+1} \right\rfloor + 2j$$
$$= P_k - |P_k|_{j+1} + (j-1) \left\lfloor \frac{P_k}{j+1} \right\rfloor + 2j.$$

So we have

$$P_k + P_{k-1}|-m_k|_j - j < P_k - |P_k|_{j+1} + (j-1) \left\lfloor \frac{P_k}{j+1} \right\rfloor + 2j$$
$$\Leftrightarrow P_{k-1}|-m_k|_j < (j-1) \left\lfloor \frac{P_k}{j+1} \right\rfloor + 3j - |P_k|_{j+1}.$$

But $3j - |P_k|_{j+1} > 0$ for $j \in [1, m_k - 1]$, and $|-m_k|_j \leqslant j - 1$, for $j \in [1, m_k - 1]$. So, we must show

$$P_{k-1} \leqslant \left\lfloor \frac{P_k}{j+1} \right\rfloor, \quad \text{for } j \in [1, m_k - 1].$$

But

$$\left\lfloor \frac{P_k}{j+1} \right\rfloor = \left\lfloor \frac{m_k}{j+1} P_{k-1} \right\rfloor \geqslant P_{k-1}, \quad \text{for } j \in [1, m_k - 1].$$

Therefore, $X_i < 2Y$.

Lemma 3

For the RA estimate

$$E_{i+1} = \begin{cases} \left\lfloor \dfrac{x_k \left\lfloor \dfrac{P_l}{Y} \right\rfloor}{m_k} \right\rfloor, & \text{if } k = l, \\ x_k \left\lfloor \dfrac{P_l}{Y} \right\rfloor, & \text{if } k = l + 1, \\ x_k \left\lfloor \dfrac{P_l}{Y} \right\rfloor m_{l+1} \cdots m_{k-1}, & \text{if } k > l + 1, \end{cases}$$

we have $0 \leqslant E_{i+1} \leqslant X_i/Y$, for all X_i, $Y \neq 0$.

Proof. Clearly, $E_{i+1} \geqslant 0$ in each case $k = l$, $k = l + 1$ and $k > l + 1$. Now, by assumption,

$$X_i = x_k P_{k-1} + x_{k-1} P_{k-2} + \cdots + x_1$$

and

$$Y = y_l P_{l-1} + y_{l-1} P_{l-2} + \cdots + y_1.$$

We will show $E_{i+1} \leqslant X_i/Y$ for each case $k = l$, $k = l + 1$ and $k > l + 1$. For $k = 1$:

$$E_{i+1} = \left\lfloor \frac{x_k \left\lfloor \dfrac{P_k}{Y} \right\rfloor}{m_k} \right\rfloor = \left\lfloor \frac{x_k P_{k-1} \left\lfloor \dfrac{P_k}{Y} \right\rfloor}{P_k} \right\rfloor.$$

Let $x_k P_{k-1} = X_i - \epsilon$, for $0 \leqslant \epsilon < P_{k-1}$. Also, let $|P_k|_Y$ denote P_k mod Y. We have

$$E_{i+1} = \left\lfloor \frac{(X_i - \epsilon)\left\lfloor \frac{P_k}{Y} \right\rfloor}{P_k} \right\rfloor = \left\lfloor \frac{(X_i - \epsilon)Y\left\lfloor \frac{P_k}{Y} \right\rfloor}{P_k Y} \right\rfloor = \left\lfloor \frac{(X_i - \epsilon)(P_k - |P_k|_Y)}{P_k Y} \right\rfloor$$

$$= \left\lfloor \frac{X_i}{Y} - \frac{\epsilon}{Y} - \frac{(X_i - \epsilon)|P_k|_Y}{P_k Y} \right\rfloor.$$

Now $Y \geqslant P_{k-1}$, so $0 \leqslant \epsilon/Y < 1$. Also,

$$0 \leqslant \frac{|P_k|_Y}{Y} < 1, \quad \text{and} \quad \frac{X_i - \epsilon}{P_k} = \frac{x_k}{m_k} < 1.$$

Therefore,

$$0 \leqslant \frac{(X_i - \epsilon)|P_k|_Y}{P_k Y} < 1, \quad \text{and so} \quad E_{i+1} \leqslant \left\lfloor \frac{X_i}{Y} \right\rfloor \leqslant \frac{X_i}{Y}.$$

For $k = l + 1$:

$$E_{i+1} = x_k \left\lfloor \frac{P_l}{Y} \right\rfloor = x_k \left\lfloor \frac{P_{k-1}}{Y} \right\rfloor \leqslant \frac{x_k P_{k-1}}{Y} \leqslant \frac{X_i}{Y}.$$

For $k > l + 1$:

$$E_{i+1} = x_k \left\lfloor \frac{P_l}{Y} \right\rfloor m_{l+1} \cdots m_{k-1} \leqslant x_k \frac{P_l}{Y} m_{l+1} \cdots m_{k-1} = \frac{x_k P_{k-1}}{Y} \leqslant \frac{X_i}{Y}.$$

Therefore, in each case $k = l$, $k = l + 1$, and $k > l + 1$ we have $0 \leqslant E_{i+1} \leqslant X_i/Y$.

Theorem 1

The RA eventually halts, at which time

$$\left\lfloor \frac{X}{Y} \right\rfloor = \sum_{i=0}^{r-1} E_{i+1} + E',$$

where

$$E' = \begin{cases} 0, & \text{if } X_r < Y, \\ 1, & \text{if } E_{r+1} = 0. \end{cases}$$

Proof. Halting will be proven first.

Let $\{X_i\}$ and $\{E_{i+1}\}$, for $i = 0, 1, \ldots$, denote the sequence of numerators and estimates, respectively. By Lemma 3, we have $0 \leqslant E_{i+1} \leqslant X_i/Y$, for all i. If $E_{i+1} = 0$, the RA halts. If $E_{i+1} > 0$, then $X_{i+1} < X_i$ by Lemma 1. Therefore $\{X_i\}$ is a decreasing sequence of positive integers, and so $\{E_{i+1}\}$ is also. Therefore, eventually for some r, $X_r < Y$ or $E_{i+1} = 0$, and the RA halts.

Now, to show that RA computes $\lfloor X/Y \rfloor$ we have

$$\begin{aligned} X_1 &= X - E_1 Y, \\ X_2 &= X_1 - E_2 Y, \\ &\vdots \\ X_r &= X_{r-1} - E_r Y, \end{aligned}$$

and either $X_r < Y$ or $E_{i+1} = 0$. We have

$$\begin{aligned} X_r &= X_{r-1} - E_r Y, \\ &= X_{r-2} - E_{r-1} Y - E_r Y, \\ &= X_{r-3} - E_{r-2} Y - E_{r-1} Y - E_r Y, \\ &= \cdots = X - E_1 Y - E_2 Y - \cdots - E_{r-1} Y - E_r Y. \end{aligned}$$

Therefore,

$$X = (E_1 + E_2 + \cdots + E_r)Y + X_r,$$

and so

$$\left\lfloor \frac{X}{Y} \right\rfloor = \sum_{i=0}^{r-1} E_{i+1} + \left\lfloor \frac{X_r}{Y} \right\rfloor.$$

Let $E' = \lfloor X_r/Y \rfloor$. If $X_r < Y$, then $E' = 0$. If $E_{r+1} = 0$, then by Lemma 2, $X_r < 2Y$. But $X_r \geqslant Y$ because otherwise the RA would have stopped before calculating E_{r+1}. Therefore, $Y \leqslant X_r < 2Y$, and so $E' = 1$. Therefore,

$$E' = \begin{cases} 0, & \text{if } X_r < Y, \\ 1, & \text{if } E_{r+1} = 0. \end{cases}$$