ELSEVIER

Information Processing Letters 51 (1994) 163–169

Information Processing Letters

# Fast RNS division algorithms for fixed divisors with application to RSA encryption

Ching Yu Hung [1], Behrooz Parhami

*Department of Electrical and Computer Engineering, University of California, Santa Barbara, Santa Barbara, CA 93106-9560, USA*

Communicated by G.R. Andrews; received 8 July 1993; revised 29 April 1994

*Keywords:* Algorithm complexity; Computer arithmetic; Cryptography; Division; Modular multiplication; Residue number system; Sign detection

## 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. Digit-parallel, carry-free, and constant-time multiplication and addition is a unique feature of RNS. However, certain operations such as overflow detection, magnitude comparison, and division are quite difficult in RNS. Thus, RNS is in general limited to applications that do not require extensive use of those difficult operations; for example, filtering in Digital Signal Processing. By improving RNS division, many application areas for which RNS was previously infeasible, such as RSA encryption, can use the fast RNS multiplication without being penalized too much by the slow RNS division.

In this paper we consider the problem of division by fixed divisors in RNS. Ordinary integer division is performed; i.e., given the dividend $X$ and the divisor $D$, we wish to find the quotient $Q = \lfloor X/D \rfloor$ and the remainder $R = X - QD$. The idea is to perform some preprocessing based on the divisor to improve the on-line speed of divisions. Our algorithms do not require inordinately expensive preprocessing; the cost of preprocessing is negligible if $O(\log_2 D)$ divisions are performed with the same $D$, and that is true in the context of RSA encryption. In $m$-bit encryption, up to $2m$ modular multiplications are performed with the same $m$-bit modulus.

Several algorithms for general residue division have been proposed in the past [1,3,5,8]. Under the assumption that $D$ is fixed and $X$ is uniformly distributed, the fastest of them [5] has a worst-case time complexity of $O(\log_2 X) = O(nb)$, where $n$ is the number of moduli and $b$ is the number of bits in the largest modulus.

In this paper we present two division algorithms for fixed divisors that achieve time

* Corresponding author. Email: parhami@ece.ucsb.edu.
[1] Email: ching@simd.ece.ucsb.edu.


*SSDI* 0020-0190(94)00099-K

complexity of $O(n)$ for each division. The first algorithm is based on the well-known division method of multiplying by the divisor reciprocal. The second algorithm is based on the Chinese Remainder Theorem (CRT) decoding and table lookup, and requires that the divisor $D$ be relatively prime to all moduli. The second algorithm requires more storage but is faster. The computation time analysis is based on the usual parallel residue processor assumption: There are $n$ residue processors for an $n$ modulus system, each being capable of one modulus-size modular addition or multiplication in one time step. Additionally, we assume that the divisor $D$ is relatively prime to all moduli, even for the first algorithm that does not require it but becomes somewhat more efficient if it holds. Unless otherwise noted, the computation times given in the paper are parallel times. For software implementation of RNS, we also provide the sequential time complexities.

We also present adaptation of both algorithms for RSA encryption. The first algorithm leads to $4n + b$ time steps per modular multiplication, while the second algorithm requires $2n$ time steps per modular multiplication. The second algorithm is found to be very competitive with previously proposed RSA implementations.

## 2. Basic operations in RNS

### 2.1. Notation

The expression $|x|_y$ denotes the remainder of $x$ divided by $y$, where $x$ and $y$ are real with $y > 0$. When $x$ and $y$ are relatively prime integers, with $y > 1$, the multiplicative inverse of $x$ modulo $y$, $|x^{-1}|_y$, is defined.

A residue number system is specified by a list of $n$ pairwise relatively prime moduli, $m_1, m_2, \ldots, m_n$. A number $X$ is represented in RNS by a list of residues $(x_1, x_2, \ldots, x_n)$, where $x_i = |X|_{m_i}$. Let $M = \prod m_i$ represent the product of all moduli. Let $M_i = M/m_i$ and $\alpha_i = |M_i^{-1}|_{m_i}$. We also define $M[a,b]$, $1 \leqslant a \leqslant b \leqslant n$, as the product of a sequence of moduli: $M[a,b] = \prod_{i=a}^{b} m_i$. We let $M[a,b] = 1$ for $b < a$. Signed numbers in the range $-\lfloor M/2 \rfloor \leqslant X \leqslant \lfloor (M-1)/2 \rfloor$ are represented. These RNS parameters are used throughout the rest of the paper without being explicitly noted in each case.

Let $b$ be the number of bits needed to represent each residue. For algorithm efficiency and convenience in analyzing complexities, we assume that the magnitudes of the moduli are more or less uniform. This assumption leads to $m_i \leqslant 2^b$, $m_i/m_j \approx 1$, and $M \approx 2^{nb}$. We say a number $X$ is $k$ digits long when $X \approx 2^{kb}$.

### 2.2. Base extension and division by a product of moduli

Let a number $X$ be representable by $k$ residues, $k < n$, in an $n$ modulus RNS. Base extension refers to the procedure of finding the $n - k$ unknown residues. Base extension is usually implemented with mixed-radix conversion (see, e.g., [9, pp. 41–47]) and takes exactly $2k - 1$ time steps with $n$ residue processors.

Base extension can be used to divide a number by a product of first powers of moduli [9, pp. 47–50]. Let $X$ be the dividend and $M[1,k]$ be the divisor. The first $k$ residues of the remainder $R = X \mod M[1,k]$ are simply the first $k$ residues of $X$. The $n - k$ remaining residues are found by a base extension from the front $k$ residues toward the back $n - k$ residues, taking $2k - 1$ steps. With all residues of the remainder known, the quotient $Q$ can be found by evaluating $(X - R)M[1,k]^{-1}$ in one step. However, $M[1,k]^{-1}$ is defined only for the back $n - k$ residues, so only the corresponding residues of $Q$ are known. Another base extension, from the back $n - k$ residues to the front $k$ residues, is applied, taking $2(n - k) - 1$ steps. Totally the division takes $2n - 2$ steps.

### 2.3. Sign detection

In [5], Hung and Parhami propose a sign estimation procedure that in $\lceil \log_2 n \rceil$ steps indicates whether a residue number is positive, negative, or too small in magnitude to tell. The procedure outlined below uses a parameter $u$, $u > 1$,

to specify input range and output precision: The procedure requires input number $X$ in the range $[-(1/2-2^{-u})M, (1/2-2^{-u})M]$; i.e., a fraction of the dynamic range is excluded. When the output $ES(X)$ is indeterminate, $X$ is guaranteed to be in the range $[-2^{-u}M, 2^{-u}M]$.

**Preprocessing**

1. $EF[i][j]$ = Truncate $|j\alpha_i/m_i|_1$ to the $(-t)$th bit, for $1 \leqslant i \leqslant n$, $0 \leqslant j < m_i$, where $t = u + \lceil \log_2 n \rceil$

**Sign estimation of an input $X$**

2. $EF(X) = |\sum_{i=1}^{n} EF[i][x_i]|_1$
3. $ES(X) = +$, if $EF(X) < 1/2$
4. $-$, if $1/2 \leqslant EF(X) < 1 - 2^{-u}$
5. $\pm$, otherwise

From this sign estimation procedure we construct a sign *detection* procedure as follows. The relatively inexpensive sign estimation is tried first. In case it fails (sign being indeterminate), we compute the sign by mixed-radix conversion. Since the chance of having to use the mixed-radix conversion is low, the sign detection requires $\lceil \log_2 n \rceil$ steps on the average and $2n + \lceil \log_2 n \rceil$ steps in the worst case (We assume uniform distribution of $X$ in the allowed range, and ignore the time spent on communication.)

### 2.4. Chinese Remainder Theorem and $B(X)$

The Chinese Remainder Theorem states that

$$|X|_M = \left| \sum_{i=1}^{n} |\alpha_i x_i|_{m_i} M_i \right|_M. \tag{1}$$

We define $B(X)$ [9, p. 30] as the number of times the modular summation in Eq. (1) overflows $M$:

$$|X|_M = \sum_{i=1}^{n} |\alpha_i x_i|_{m_i} M_i - B(X)M. \tag{2}$$

The sign estimation procedure can be adapted to efficiently compute $B(X)$ when input $X$ is nonnegative. The same $EF[i][j]$ table is used, with parameter $u$ satisfying $u > 1$. The preprocessing stage is the same as in sign estimation, followed by:

2. $Y = \sum_{i=1}^{n} EF[i][x_i]$
3. $EF(X) = |Y|_1$, $B(X) = \mathrm{Int}(Y)$
4. If $EF(X) \leqslant 1/2$ return $B(X)$
5. Otherwise, return $B(X) = B(X) + 1$

### 2.5. General division

Reference [5] also contains an algorithm to perform division in RNS without preprocessing. We shall call this algorithm general division since it does not require prior knowledge of the divisor.

The general division algorithm is based on the well-known binary SRT division. After proper normalization of the dividend $X$ and the divisor $D$, in each iteration we perform $X = 2X$, $X = 2(X - D)$, or $X = 2(X + D)$, based on the estimated sign of $X$. To optimize for hardware implementation, the $n$ operand summation (line 2) of the sign estimation procedure is performed once every $\lceil \log_2 n \rceil$ iterations so that the average cost per iteration is constant. The algorithm presented in [5] takes $O(\log_2(M/D) + \log_2 Q)$ steps, where $Q$ is the quotient computed by the algorithm. The controlled way in which we use general division in our fixed-divisor algorithms renders some of the computations unnecessary. Specifically, when $\lceil \log_2 D \rceil$ is known or is guaranteed to be within a small range, normalization of $D$ can be simplified. In this case, the general division takes $3\lceil \log_2 Q \rceil$ time steps on the average and $3\lceil \log_2 Q \rceil + 2n$ time steps in the worst case. The extra $2n$ time is due to a possible final sign detection by mixed-radix conversion. For software implementation, the sequential time is roughly $2n \log_2 Q$.

## 3. Multiplying by the divisor reciprocal

Our first algorithm for fixed divisor RNS division precomputes the reciprocal of the divisor and uses it to compute the approximate quotient ($X$ is the dividend and $D$ the divisor):

**Preprocessing**

1. Compute $C = \lfloor M/D \rfloor$, choose $k$ such that $1 \leqslant k \leqslant n$ and $M[1, k-1] \leqslant D < M[1, k]$

**Each subsequent division**
2. $X' = \lfloor X/M[k,n] \rfloor$
3. $Q = \lfloor X'C/M[1,k-1] \rfloor$
4. $X'' = X - QD$
5. Call general division to obtain $Q$ and $R$ with $0 \leqslant R < D$ such that $X'' = Q'D + R$
6. Return $Q'' = Q + Q'$ and $R$

In the preprocessing stage, $C$ is computed as the quotient of $M$ divided by $D$, using the general division algorithm. For each subsequent division, first we scale down $X$ by a factor of $1/M[k,n]$ to obtain $X'$. Then $X'$ is multiplied by $C$ and scaled down by $1/M[1,k-1]$ to find an approximate quotient $Q$. The remainder $X''$, found on line 4, can thus be off by a multiple of $D$. We next use general division to divide $X''$ by $D$, thereby correcting the error of the approximate quotient. We call lines 2–4 the coarse modulo stage and lines 5–6 the correction stage.

The approximate quotient satisfies

$$\left\lfloor \frac{X}{D} \right\rfloor - \left\lceil \frac{M[k,n]}{D} \right\rceil - 1 \leqslant Q \leqslant \left\lfloor \frac{X}{D} \right\rfloor,$$

i.e., there is an upper bound error of $\lceil M[k,n]/D \rceil + 1$ with respect to the correct quotient. Intuitively, $C$ is close to the fraction $M/D = M[1,n]/D$, $X'$ is close to $X/M[k,n]$, and so $Q$ is close to $X'C/M[1,k-1] = X/D$. Hence, $Q$ is close to the correct quotient $\lfloor X/D \rfloor$. Any error is due to truncations in the three integer divisions.

The general division needed in the preprocessing stage takes $3(n-k)b$ steps on the average and $2n$ steps more in the worst case [5]. The coarse modulo stage in each subsequent division requires two divisions by products of moduli, $M[k,n]$ and $M[1,k-1]$, besides a few multiplications and additions. Dividing by a product of moduli is by base extension and each takes $2n-2$ steps. With the largest error in the quotient being $\lceil M[k,n]/D \rceil + 1$, or $(n-k+1) - (k-1) = n - 2k + 2$ digits long, the correction stage takes about $3b(n-2k)$ steps on the average and $3b(n-2k) + 2n$ steps in the worst case. (When $n < 2k$, it takes constant steps on the average and $2n$ steps in the worst case.) Each subsequent division using this algorithm thus takes about $4n + 3b(n-2k)$ steps on the average and $6n + 3b(n-2k)$ steps in the worst case. In a software implementation, a base extension for $k$ unknown residues takes $n^2 - k^2$ steps, and so a division by a product of moduli takes $n^2 - k^2 + n^2 - (n-k)^2 = n^2 + 2nk - 2k^2$ steps. Our algorithm takes $2n(n-k)b$ steps for preprocessing and $2n^2 + 4nk - 4k^2 + 2nb(n-2k)$ steps for each division.

## 4. CRT decoding and table lookup

The second algorithm for fixed divisor division achieves faster computation with a larger lookup table ($n+1$ entries rather than 1). An outline of the algorithm follows.

**Preprocessing**
1. $Z = |M|_D$
2. For $i = 1, 2, \ldots, n$ do
3. $k_i = |-ZD^{-1}|_{m_i}$
4. Compute $Z_i = (Z + k_i D)/m_i$

**Each subsequent division**
5. Compute $B(X)$
6. $Y = \sum_{i=1}^{n} |\alpha_i x_i|_{m_i} Z_i + B(X)(D - Z)$
7. Call general division to obtain $Q$ and $R$ with $0 \leqslant R < D$ such that $Y = QD + R$
8. Return $Q' = |(X-R)D^{-1}|_M$ and $R$

The algorithm is based on the Chinese Remainder Theorem. When $X$ is nonnegative, Eq. (2) becomes

$$X = \sum_{i=1}^{n} |\alpha_i x_i|_{m_i} M_i - B(X)M. \tag{3}$$

We view Eq. (3) as a linear decomposition of $X$. To reduce $X$ modulo a fixed divisor $D$, we precompute $Z_i = M_i \bmod D$ and $Z = M \bmod D$. We then have

$$|X|_D \equiv \sum_{i=1}^{n} |\alpha_i x_i|_{m_i} Z_i + B(X)(D - Z) \pmod{D}. \tag{4}$$

Thus, for each division, the algorithm first computes the weighted sum of Eq. (4) in the coarse modulo stage. The sum, $Y$, is at most

$(\sum_{i=1}^{n}(m_i - 1) + n)D$, or $(\log_2 n)/b + 1$ digits longer than the divisor. Next, the correction stage utilizes general division to further reduce $Y$ to the correct remainder. In step 8, $X - R$ is divisible by $D$ and since $D$ is assumed to be relatively prime to each $m_i$, $D^{-1}$ exists.

The proof that the expression $Z_i = (Z + k_iD)/m_i$ computed by the algorithm is actually $M_i \bmod D$ follows from the following easily provable statements:

$$|Z + k_iD|_{m_i} = 0,$$

$$\left|\frac{Z + k_iD}{m_i}\right|_D = |Zm_i^{-1}|_D = |M_i|_D,$$

$$\text{and} \quad 0 \leqslant \frac{Z + k_iD}{m_i} < D.$$

It takes $3b(n - k)$ steps to compute $Z$ with general division, constant time to compute all the $k_i$'s, and $2(n - k)$ time to compute each $Z_i$ with base extension, where $k$ is the number of residues required to represent the divisor $D$, and so is roughly the same as the $k$ defined in Section 3. Total time for the preprocessing stage is $3b(n - k) + 2n(n - k)$. Note that it would take $3b(n - k)(n + 1)$ steps if the preprocessing is performed as $n + 1$ instances of general division. Sequential time is $2nb(n - k) + n(n - k)^2$.

Computation time for each division is analyzed as follows. Computing $B(X)$ takes $\lceil \log_2 n \rceil$ time steps. Computing $|\alpha_i x_i|_{m_i}$ takes one step. The weighted summation takes about $2n$ steps assuming that the time needed to broadcast the $|\alpha_i x_i|_{m_i}$ values to all processors is negligible (each processor does $n + 1$ multiplications for taking the weights into account and $n$ additions). If it takes one time step to send a residue-sized number to an adjacent processor, the broadcast operation takes $n$ steps on a ring. The general division on line 7 takes $3(b + \log_2 n)$ steps on the average and $3(b + \lceil \log_2 n \rceil) + 2n$ steps in the worst case. Total time for each subsequent division is thus $\log_2 n + 2n + 3(b + \log_2 n) \approx 2n + 3b$ steps on the average and $4n + 3b$ steps in the worst case. Sequential time is $2n^2 + 2nb$.

## 5. Application to RSA cryptography

Encryption and decryption in RSA cryptography are modular exponentiation operations of the form $Z = X^Y \bmod D$. For encryption, $X$ is the plain text, $Y$ and $D$ together comprise the encryption key, and $Z$ is the ciphered text. For decryption, $X$ is the ciphered text, $Y$ and $D$ the decryption key, and $Z$ is the deciphered text. All operands, $X$, $Y$, $D$ are potentially very large integers, perhaps 1000 bits long. Let $m$ be the number of bits in $D$, and let $\log_2 Y \approx m$. A modular exponentiation requires up to $2m$ modular multiplications in a simple square-and-multiply scheme (see, e.g., [10]). It is possible to use only $(1 + \varepsilon)m$ modular multiplications with exponent recoding [7], and, in decryption only, perform shorter modular operations with respect to the two (secret) factors of $D$. We shall compare our algorithm with existing ones in terms of computation time for each modulo-$D$ multiplication.

Our fixed-divisor algorithms apply to the modulo-$D$ reduction step that follows a regular multiplication. The dynamic range of RNS thus needs to be at least the square of the modulus $D$. The $2m$ instances of modular reduction in a modular exponentiation are viewed as a sequence of divisions with the same divisor $D$. The preprocessing based on $D$ is therefore good for $2m$ divisions, and can be good for many times more when a long message is broken into several modular exponentiations with the same modulus $D$. The preprocessing times are $\frac{3}{2}nb$ and $n^2 + \frac{3}{2}nb$, respectively for the first and the second algorithm, when $k \approx n/2$. With $n = 2m/b$, the preprocessing times become $3m$ and $4m^2/b^2 + 3m$, both of which are negligible compared to the $O(mn) = O(m^2/b)$ time taken by $2m$ instances of $O(n)$-time division. The conversions between RNS and binary take $O(n) = O(m/b)$ time, also negligible compared to $O(m^2/b)$.

The on-line portion of each algorithm is further divided into the coarse modulo stage and the correction stage. For modular exponentiation, it is not necessary to fully reduce intermediate results modulo $D$. In its coarse modulo stage, the

first algorithm reduces an $n$ digit dividend to an approximate remainder up to $n - k + 1$ digits long. This is not enough since we know that $n \geqslant 2k$ and that the modular reduction must at least reduce a dividend to half-length to accommodate the squaring in the modular exponentiation. It is necessary, therefore, for the first fixed-divisor algorithm to perform at least $b$ iterations of general division after the coarse modulo stage.

The second fixed-divisor algorithm works better. It produces, in the coarse modulo stage, an approximate remainder up to $(\log_2 n)/b + 1 + k$ digits long. If we make $n$ large enough such that $2[(\log_2 n)/b + 1 + k]$ is no more than $n$, we get sufficient reduction in the coarse modulo stage. The second algorithm is also more efficient in this stage, taking $2n$ steps versus $4n$ steps for the first algorithm. With the coarse modulo stage of the second algorithm and constant-time multiplication inherent in RNS, each modular multiplication takes about $2n \approx 4m/b$ steps. The sequential time is $2n^2 \approx 8m^2/b^2$ for each modular multiplication.

## 6. Conclusions

We have presented two new algorithms for RNS division with fixed divisors. Adaptation of our second algorithm leads to an efficient RSA implementation, with $4m/b$ steps per modular multiplication.

Existing implementations of RSA encryption can be roughly classified into word-level single processor, bit-level array processors, and word-level array processors. Because of the variety of special hardware involved in the designs, it is rather difficult to compare different designs in terms of time complexity; we almost always have to compare actual or estimated encryption rates of the designs.

We compare our proposed method with two classical sequential methods: one uses a binary version of multiplying by divisor reciprocal for modular reduction[2], the other uses a residue table for modular reduction[6]. With a $b$-bit processor, a modular multiplication takes $9(m/b)^2$ and $4(m/b)^2$ steps, respectively. Treating a $b$-bit processor as having hardware complexity of $b^2$, the conventional methods have hardware-time products of $9m^2$ and $4m^2$. Our method has a hardware complexity of $(2m/b)b^2 = 2mb$, and a hardware-time product of $8m^2$. On the basis of hardware and time complexity, our design competes well with sequential implementation of classical methods. Actual encryption speed depends on the hardware platform, and is still under investigation.

While we have analyzed the time complexity of our algorithms, there are many implementation details that must be considered. For example, the communication and storage requirements of the algorithm, integration of the binary-residue and residue-binary conversions into the algorithm, and the possibility of systolic implementation. Other cryptographic algorithms can benefit from our new techniques. Our choice of RSA to illustrate the efficiency of these techniques is merely a reflection of the fact that it is better known and more widely applied. We are also looking for other applications of our new residue division algorithms. The $EF$ function serves as an index function of residue numbers in our sign detection procedure. In a recent publication [4], Dimauro et al. propose another index function, called the Sum of Quotients, for comparison of residue numbers. While a straightforward implementation of their technique seems as expensive as residue-to-binary conversion with the Chinese Remainder Theorem, it remains to be investigated whether a truncated version leads to an efficient approximate comparison procedure.

## Acknowledgement

Careful reading of the manuscript by the referees has led to a significant improvement in our presentation. We thank them for their efforts.

## References

[1] D.K. Banerji, T.-Y. Cheung and V. Ganesan, A high-speed division method in residue arithmetic, in: *Proc.*

*5th Symp. on Computer Arithmetic* (IEEE Press, New York, 1981) 158–164.

[2] P. Barrett, Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor, in: A.M. Odlyzko, ed., *Advances in Cryptology, Proc. Crypto 86* (Springer, Berlin, 1986) 311–323.

[3] W.A. Chren Jr, A new residue number system division algorithm, *Comput. Math. Appl.* **19** (7) (1990) 13–29.

[4] G. Dimauro, S. Impedovo and G. Pirlo, A new technique for fast number comparison in the residue number system, *IEEE Trans. Comput.* **42** (5) (1993) 608–612.

[5] C.Y. Hung and B. Parhami, An approximate sign detection method for residue numbers and its application to RNS division, *Comput. Math. Appl.* **27** (4) (1994) 23–35.

[6] S. Kawamura and K. Hirano, A fast modular arithmetic algorithm using a residue table, in: C.G. Günther, ed., *Advances in Cryptology, Proc. Eurocrypt 88* (Springer, Berlin, 1988) 245–250.

[7] Ç.K. Koç and C.-Y. Hung, Adaptive $m$-ary segmentation and canonical recoding algorithms for multiplication of large binary numbers, *Comput. Math. Appl.* **24** (3) (1992) 3–12.

[8] M. Lu and J.-S. Chiang, A novel division algorithm for the residue number system, *IEEE Trans. Comput.* **41** (8) (1992) 1026–1032.

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

[10] N. Takagi, A radix-4 modular multiplication hardware algorithm for modular exponentiation, *IEEE Trans. Comput.* **41** (8) (1992) 949–956.