New Approach to Integer Division in Residue Number Systems

Dragan Gamberger
Rudjer Bošković Institute
41000 Zagreb, Yugoslavia

Abstract

A new division algorithm substantially different from the known ones, and especially appropriate for the Residue Number Systems (RNS), is presented. It makes use of the fact that multiplicative inverse element of a divisor, that is relatively prime to system moduli, can be easily determined in the RNS. The number of its iterations depends only on the magnitude of the divisor and the moduli of the system. The problems in the algorithm realization are analyzed in detail and a complete solution using the incompletely specified RNS is described.

1 Introduction

It is known that complexity of integer division in the Residue Number System (RNS) represents one of the main reasons for relatively seldom usage of this number system. Practically, the RNS is used only in the applications where it is possible to avoid integer division at all, like digital filtering or number theoretic transforms. Although two different algorithms for integer division have been suggested already 23 years ago [1], and although there have been some efforts to realize them [2], not a single successful implementation of integer division has been published yet.

One of the suggested integer division algorithms makes use of the method similar to the conventional binary division. The application of this algorithm and its modifications have the main disadvantage that each iteration requires magnitude comparison. This operation is very fast in weighted number systems but relatively complicated and slow in the RNS [1,3]. Apart from that, this algorithm requires a table of the residue representations of the integer powers of 2. This table is iteratively used at the beginning of the algorithm in the computation of an approximate quotient, and later in the iterative process of successive approximation in the computation of the correct quotient [1,2].

The second integer division method suggested (division by approximate divisor) does not have this disadvantages. It makes use of the fact that in the RNS the division by a product of some system moduli can be realized relatively simply [3]. The idea is to substitute the real divisor by an approximate one, i.e. by a product of some system moduli and to use this division iteratively until a correct quotient is found. Algorithm correctness is assured only if for any divisor Y there is an approximate divisor $\tilde{Y}$ ($Y \leq \tilde{Y} < 2Y$), what requires the RNS with specially chosen moduli. The main drawback of this algorithm is its need for special logic and look-up tables in the determination of the best approximate divisor and that the rate of convergence depends not only on the dividend and divisor magnitude but on the quality of divisor approximation, as well [1].

The objective of this paper is to present a novel division algorithm substantially different from the mentioned ones. In the section 2 of this paper the algorithm is described and the proof of its correctness is given. In the section 3 the problems of its realization are analyzed and a complete solution is presented. The unique algorithm characteristic, that the number of iterations depends only on the magnitude of divisor, is analyzed in section 4.

In the presentation the realization by read only memories (ROM's) as in [3] is supposed. Hardware and time complexity of the presented algorithms is measured by the number of necessary ROM's for its realization and the worst case number of successive memory accesses, called modular steps, respectively.

2 General Algorithm Principles

The goal of the algorithm is to compute the quotient Z

$$Z = \left\lfloor \frac{X}{Y} \right\rfloor$$

where both X and Y are positive integers and where $\lfloor a \rfloor$ denotes the greatest integer equal or less than a. The trivial case is for $Y = 1$ when it holds $Z = X$.

For the non-trivial case $Y > 1$, the idea of the algorithm is to compute a new dividend X' and a new divisor Y', both positive integers so that conditions (1) are satisfied

$$\begin{gathered} X' < X \\ 1 \leq Y' < Y \\ Z = \left\lfloor \frac{X}{Y} \right\rfloor = \left\lfloor \frac{X'}{Y'} \right\rfloor . \end{gathered} \tag{1}$$

It means that we are looking for new dividend and divisor that are smaller than the original ones but have the same quotient. Now if Y' equals 1, then Z equals X', and if $Y' > 1$ then X' and Y' can be used as the new starting dividend and divisor, respectively. The process can be iterated. After a finite number of steps, some Y' must be equal to 1 and the corresponding X' is the result of division.

The presented idea can be applied in any number system but the problem is how to efficiently compute X' and Y' which satisfy conditions (1). A general solution to this problem is not known. A solution applicable only for the RNS defined with N moduli $m_1, m_2, \ldots, m_N$, that are different prime numbers is presented in this paper. In such RNS a number X is represented by N-tuple $X = (x_1, x_2, \ldots, x_N)$, where $x_i = X \bmod m_i$ and there is a unique representation for each positive integer in the range $X < M$, where M is the product of all system moduli. We suppose that the moduli are selected so that this condition is satisfied both for the dividend and the divisor.

In the RNS thus defined the new dividend and the new divisor can be computed in each iteration as follows:

A iteration: If $\gcd(Y, M) = G > 1$, where gcd denotes greatest common divisor function

$$X' = \left\lfloor \frac{X}{G} \right\rfloor \tag{2}$$

$$Y' = \frac{Y}{G}. \tag{3}$$

It is obvious that relations (2) and (3) satisfy conditions (1) and that they can be relatively easily realized in the RNS because they both are divisions by some product of system moduli.

B iteration: If $\gcd(Y, M) = 1$, what means that the divisor is relatively prime with system moduli, then X' and Y' can be computed by

$$X' = \left\lfloor \frac{XD}{M} \right\rfloor \tag{4}$$

$$Y' = \left\lfloor \frac{YD}{M} \right\rfloor = \frac{YD-1}{M} \tag{5}$$

where D represents multiplicative inverse element of $Y \bmod M$

$$YD = 1 \bmod M \tag{6}$$

or in other words

$$YD = kM + 1 \quad (k \text{ is an integer}).$$

D satisfying inequality $1 < D < M$ always exists, and it is unique according to the theorem 2-2 in [1] and the fact that conditions $\gcd(Y, M) = 1$ and $1 < Y < M$ are both true in this case. The proof that the relations (4) and (5) thus defined satisfy conditions (1) is given in the appendix.

Computation of the multiplicative inverse element in the RNS is not a problem because its each modulus defines a separate finite field and the operation can be performed independently and completely in parallel for different moduli. Because weighted number systems are indefinite fields the necessity to compute the multiplicative inverse element in this algorithm is the main reason that it can not be implemented in them. Even if weighted number systems would be used for representing finite fields, the presented algorithm could not be implemented in them because multiplicative inverse element computation could be executed only by iterative repetition of integer division itself. In order to demonstrate only the ideas of the new algorithm, in Example 1 we suppose for a moment that decade number system presents a finite field with M elements and that we know how to compute multiplicative inverse elements in it.

Example 1 Divide 502 by 15 in a decade number system with $M = 1024$;

1. iteration $X = 502$, $Y = 15$, $\gcd(15, 1024) = 1$ $D = 751$ because of $751 * 15 = 11 * 1024 + 1$ $X' = \lfloor 502 * 751/1024 \rfloor$, $Y' = (15 * 751 - 1)/1024$
2. iteration $X = 368$, $Y = 11$, $\gcd(11, 1024) = 1$ $D = 931$ because of $931 * 11 = 10 * 1024 + 1$ $X' = \lfloor 368 * 931/1024 \rfloor$, $Y' = (11 * 931 - 1)/1024$
3. iteration $X = 334$, $Y = 10$, $\gcd(10, 1024) = 2$ $X' = \lfloor 334/2 \rfloor$, $Y' = 10/2$
4. iteration $X = 167$, $Y = 5$, $\gcd(5, 1024) = 1$ $D = 205$ because of $205 * 5 = 1024 + 1$ $X' = \lfloor 167 * 205/1024 \rfloor$, $Y' = (5 * 205 - 1)/1024$
5. iteration $X = 33$, $Y = 1$, $Z = 33$.

3 The Algorithm Implementation Analysis

Although at first glance it might seem that both A and B iterations can be easily realized in the RNS, practically it is not so. In the A iteration, for example, both relations (2) and (3) include division by a variable product of moduli. Realization of division by any defined product of moduli is not a great problem [3] but the complete solution should include hardware for all possible combinations of product of moduli and this number grows very fast with the number of moduli. For example, in the RNS with 5 moduli 30 different products of moduli are possible and according to realization suggested in [3], 395 look-up tables stored in ROM's are necessary only for this operation. The same problem exists in the integer division algorithm by an approximate divisor. In this paper the problem is solved so that the relations (2) and (3) are realized by the same hardware as relations (4) and (5) of the B iteration.

In the realization of the B iteration, as we have already noticed, the computation of the multiplicative inverse element D and the multiplication by it are not problems in the RNS. The division by M can be also easily realized because it is a constant product of moduli. The main problem is that the results of multiplication XD and YD should be presented in the RNS in which variables of the magnitude $(M-1)(M-2)$ can be uniquely represented. The problem might be solved so that we introduce the extended RNS that besides N moduli $m_1, m_2, \ldots, m_N$ includes additional R moduli $p_1, p_2, \ldots, p_R$, where P, product of all the additional moduli p_i, is at least $M-1$. In this case all the computations should be done in the so defined extended RNS except that the multiplicative inverse element D is computed only for the moduli m_i and then transformed by the standard base extension algorithm to the representation in the extended RNS. The drawback of this solution is its relative great hardware complexity and execution time. The execution of the B iteration in the extended RNS requires at least $2N+R+1$ modular steps: N for the base extension algorithm, $N+R$ for division and 1 for multiplication. Only division by M in an extended RNS system with 5 main moduli and 5 additional moduli would require 65 look-up tables in ROM's.

To make the problem easier we suggest the following modifications. The relations (4), (5), and (6) can be rewritten as

$$X' = \frac{XD - (XD) \bmod P}{P} \quad (7)$$

$$Y' = \frac{YD-1}{P} \quad (8)$$

$$YD = 1 \bmod P. \quad (9)$$

The first modification is that instead of the extended RNS we have introduced a new RNS, called auxiliary RNS defined with R moduli $p_1, p_2, \ldots, p_R$. Each modulus p_i of the auxiliary RNS should be a prime number different from all the moduli of the main system and selected so that their product P is greater than the product of all main moduli M. Multiplicative inverse element of the divisor is now computed in the auxiliary RNS and division is by the constant P. It can be easily verified in the Appendix that because of $P > M$, relations (7) and (8) satisfy conditions (1), as well.

The second modification refers to the changes in the relation (7) where we have introduced subtraction of the quantity $(XD) \bmod P$. This modification ensures that the difference is divisible by P without remainder. This fact is important because now we do not have to use division algorithm presented in [3] that requires N modular steps but we can use algorithm for division with zero remainder presented in [1,4] executable in only 1 modular step. The second condition for the application of this algorithm is also satisfied in this case because of P is always relatively prime to all the moduli of the main RNS.

We must also notice that relations (7) and (8), in contrast to the relations (4) and (5), do not have to be executed in the extended RNS although their numerators can be out of the range of uniquely defined numbers for the main RNS [4]. The reasons are that we know in advance that the final results are integers uniquely representable in the main RNS and that direct division as defined in [5] is used.

The main advantage of the suggested modification is that B iteration can be executed now in $N+R+2$ modular steps. The disadvantage is that magnitude $(XD) \bmod P = (X/Y) \bmod P = Z'$ must be computed and subtracted in relation (7). The complete B iteration based on relations (7), (8), and (9) is presented in Figure 1.

block	number of look-up tables
a	N(N-1)/2 + R(N-1)
b	N(N-1)/2 + R(N-1)
d	R
e	R(R-1)/2 + N(R-1)
f	R(R-1)/2 + N(R-1)
g	N
h	N
i	N

Table 1: Number of look-up tables in the blocks

The iteration execution starts with the conversion of magnitudes X and Y to the auxiliary RNS. A distinct hardware is used for each conversion (blocks '**a**' and '**b**') so that after N modular steps both variables are presented in the auxiliary system. Thereafter the multiplicative inverse element D in this system is computed and immediately transformed back to the main RNS in R modular steps by the block '**e**'. Multiplication Y by D in the main system is executed by the block '**h**'. In the same block subtraction of 1 and division by P is incorporated and it is possible according to [3] because they both are constants. In parallel to the transformation of D to the main system, X is directly divided by Y in the auxiliary RNS and the result Z' is transformed to the main system by the blocks '**d**' and '**f**', respectively. The block '**g**' waits the number D to be computed in the main RNS and then executes multiplication X by D. The final value of X' is computed by the '**i**' block.

Execution time in the form of the modular steps number is given in Figure 1 on the right-hand side of each block. One can easily verify that the total is $N+R+2$ for the computation of X' and $N+R+1$ for Y'. Necessary number of look-up tables for the block realizations as functions of the moduli number in the main and the auxiliary systems is given in Table 1.

The total is $N(N+2R)+R(2N+R-2)$ look-up tables in ROM's for the whole B iteration. In the system with 5 main and 5 auxiliary moduli, for example, it makes 140 look-up tables.

Example 2 Let us show how iteration B would look like in a concrete example. The task is to compute X' and Y' from $X = 502$ and $Y = 16$ in the RNS with moduli $7, 11, 13$ and $M = 1001$ using auxiliary RNS with moduli $5, 17, 19$ and $P = 1615$.

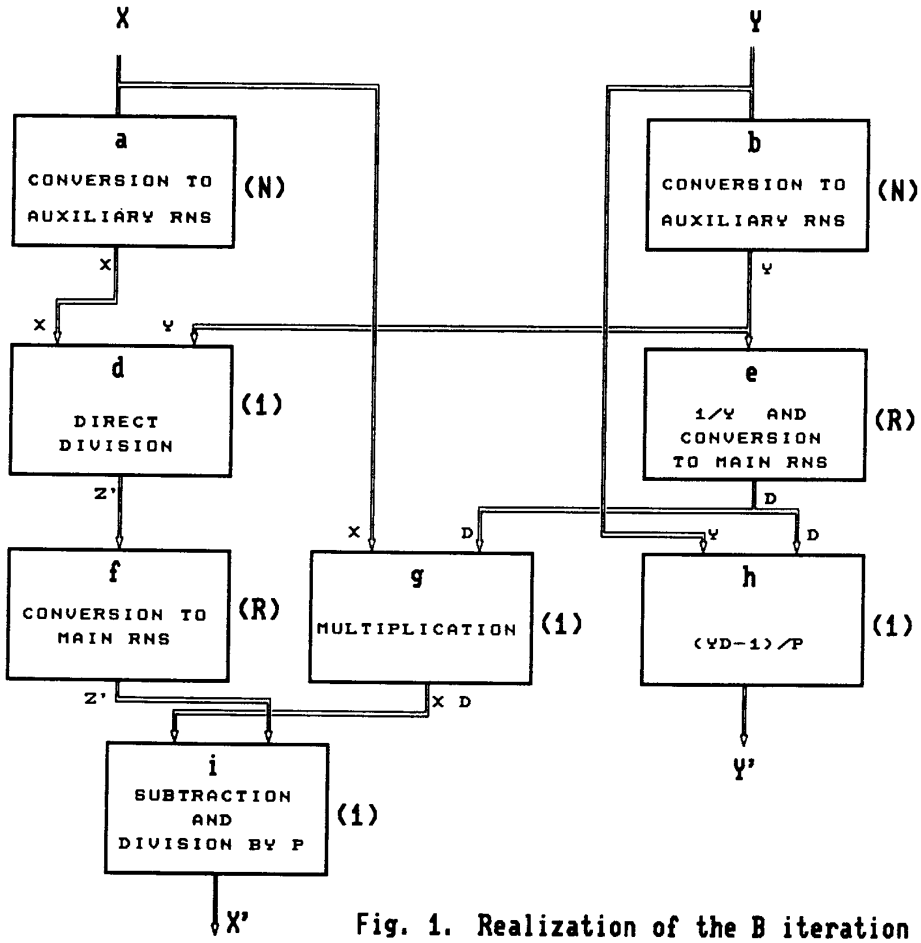

Fig. 1. Realization of the B iteration

main RNS $\quad$ aux. RNS

$m_1 = 7, m_2 = 11, m_3 = 13 \quad p_1 = 5, p_2 = 17, p_3 = 19$

input

$X = (5,7,8) = 502$

$Y = (2,5,3) = 16$

output of block

a: $\quad X = (2,9,8) = 502$

b: $\quad Y = (1,16,16) = 16$

d: $\quad Z' = (2,8,10) = 637$

e: $\quad D = (3,2,10) = 101$

f: $\quad Z' = (0,10,0) = 637$

g: $\quad XD = (1,3,2) = 50702$

h: $\quad Y' = \frac{(2,5,3)(3,2,10)-(1,1,1)}{(5,9,3)} = (1,1,1) = 1$

i: $\quad X' = \frac{(1,3,2)-(0,10,0)}{(5,9,3)} = (3,9,5) = 31$.

Now, we would like to show how the same hardware with slight modifications can be used for the A iteration, as well. We should firstly notice that, because of the fact that the multiplicative inverse element is by the suggested modification computed in the auxiliary RNS, the test whether the divisor is relative prime with the moduli, must be also executed in this system. If $\gcd(Y,P) = G > 1$ then the A iteration follows. Relations (2) and (3) can be rewritten as:

$$X' = \frac{X * P/G - (X \bmod G) * P/G}{P} \tag{10}$$

$$Y' = \frac{Y * P/G}{P}. \tag{11}$$

The differences are that the relation (2) is transformed so that division without remainder can be used and that division by a variable G is substituted by the multiplication by P/G and division by the constant P .

The computation problem of the variable P/G can be solved by application of the base extension algorithm for incompletely specified numbers presented in

[5]. This algorithm is qualified by the same time and hardware complexity as the normal base extension algorithm presented in [3] when the prime moduli greater than 2 are used. The operations of both algorithms are equal for completely specified numbers and the completely specified version in blocks '**e**' and '**f**' can be substituted by the more general incompletely specified version without any change in the B iteration execution. An additional characteristic of the algorithm presented in [5] is as follows: if the starting value is an incompletely specified number X in an RNS system, the result is a completely specified number XM_u presented in an other system, where M_u is the product of moduli of the starting system in which number X had unspecified value.

Figure 2 shows the realization of the complete algorithm including both A and B iterations. Each iteration starts with both values X and Y being transformed by blocks '**a**' and '**b**' to the auxiliary RNS. Thereafter, by a simple AND/OR logic, it is tested whether Y is relatively prime to P. The variable F is set if this is true. The value of variable F influences the operation of block '**h**' and the new introduced block '**c**'. If $F = 1$ then the block '**c**' output is $\tilde{Y} = Y$ and the same operations, as previously described, are performed by blocks '**d**'-'**h**'.

If Y is not relatively prime to the moduli of the auxiliary system, then $F = 0$ and Y is substituted by $\tilde{Y}$ in the '**c**' block. Number $\tilde{Y}$ has the value 1 for the moduli in which number Y had value equal to 0 and the unspecified value 'u' for the moduli in which number Y had a value different from 0. Thus the number $\tilde{Y}$ represents an incompletely specified value 1 with the product of unspecified moduli equal P/G.

In Figure 2 it is supposed that the '**c**' block is realized by a set of R look-up tables in ROM's and that its execution requires one modular step although it could be also realized by simple random logic circuits. The total time for the execution of one iteration has increased to $N + R + 3$ modular steps and the total hardware requirement is now $N(N+2R) + R(2N+R-1)$ look-up tables in ROM's.

The multiplicative inverse element of $\tilde{Y}$ is it itself. After conversion to the main system in the block '**e**' the result is $D = P/G$. The value Y' can be computed in the block '**h**' in a single modular step by the multiplication YD and division by the constant P. This function is slightly different from the one in the block '**h**' for the B iteration where subtraction of constant 1 was also included. Because of that in the block '**h**' both expressions should be realized and the appropriate one selected by the variable F.

Block '**d**' executes in both iterations the operation of direct division of X by $\tilde{Y}$. In the case when $\tilde{Y}$ has incompletely specified value 1 with the product of unspecified moduli equal P/G then the result of division is $Z' = (X) \bmod G$ with the same unspecified moduli. After conversion to the main system in block '**f**' the result is $Z' = (X \bmod G) * P/G$. The result of multiplication in block '**g**' is $XD = X * P/G$. This way we have prepared all the values according to (10) that the same operation in block '**i**' as in B iteration can calculate the value of X'.

Example 3 Let us show the iteration A appearing in a concrete example. The task is to compute X' and Y' from $X = 502$ and $Y = 15$ in the RNS with moduli $7, 11, 13$ and $M = 1001$ using auxiliary RNS with moduli $5, 17, 19$ and $P = 1615$.

	main RNS	aux. RNS
	$m_1 = 7, m_2 = 11, m_3 = 13$	$p_1 = 5, p_2 = 17, p_3 = 19$
input		
	$X = (5,7,8) = 502$	
	$Y = (1,4,2) = 15$	
output of block		
a:		$X = (2,9,8) = 502$
b:		$Y = (0,15,15) = 15$
		$F = 0$
c:		$\tilde{Y} = (1,u,u) = 1$
d:		$Z' = (2,u,u) = 2$
e:	$D = (1,4,11) = 323$	
f:	$Z' = (2,8,9) = 646$	
g:	$XD = (5,6,10) = 16214$	
h:	$Y' = \frac{(1,4,2)(1,4,11)}{(5,9,3)} = (3,3,3) = 3$	
i:	$X' = \frac{(5,6,10)-(2,8,9)}{(5,9,3)} = (2,1,9) = 100$.	

Figure 2 presents other necessary elements of the division hardware, as well: input registers X and Y, output register Z, control logic, and test logic for $Y = 1$. The division begins with the START signal which loads starting values into the input registers and triggers control logic. Afterwards this logic generates a series of L signals with a period of $N + R + 3$ modular steps. Their function is to load values X' and Y' into the input registers. At the end, when condition $Y = 1$ is detected, the S signal stops this series of pulses and generates STOP signal that loads the output register.

It follows a complete division example in the RNS.

Example 4 Divide 502 by 15 in the RNS with moduli $7, 11, 13$ and $M = 1001$ using auxiliary RNS with moduli $5, 17, 19$ and $P = 1615$.

1. iteration $X = (5,7,8) = 502$, $Y = (1,4,2) = 15$, $S = 0$

   in aux. RNS $X = (2,9,8)$, $Y = (0,15,15)$,
   $F = 0$, $\tilde{Y} = (1,u,u)$, $Z' = (2,u,u)$,
   $D = (1,u,u)$

   in main RNS $Z' = (2,8,9) = 646$,
   $D = (1,4,11) = 323$,
   $X' = \frac{(5,7,8)(1,4,11)-(2,8,9)}{(5,9,3)} = (2,1,9) = 100$,
   $Y' = \frac{(1,4,2)(1,4,11)}{(5,9,3)} = (3,3,3) = 3$

2. iteration $X = (2,1,9) = 100$, $Y = (3,3,3) = 3$, $S = 0$

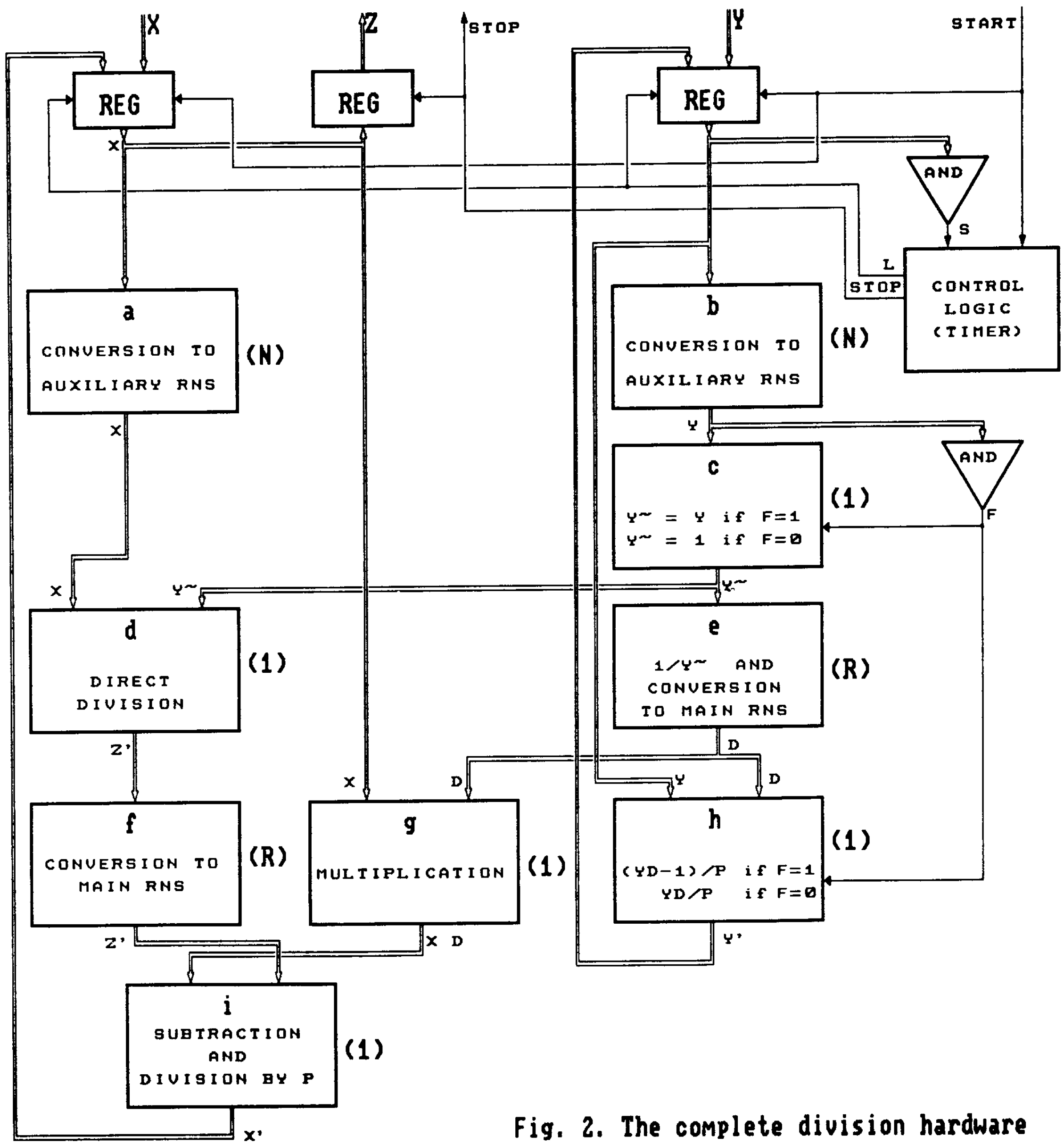

Fig. 2. The complete division hardware

in aux. RNS $X = (0,15,5), Y = (3,3,3),$
$F = 1, \tilde{Y} = (3,3,3), Z' = (0,5,8),$
$D = (2,6,13)$

in main RNS $Z' = (4,10,5) = 1110,$
$D = (6,10,11) = 1077,$
$X' = \frac{(2,1,9)(6,10,11)-(4,10,5)}{(5,9,3)} = (3,0,1) = 66,$
$Y' = \frac{(3,3,3)(6,10,11)-(1,1,1)}{5,9,3)} = (2,2,2) = 2$

3. iteration $X = (3,0,1) = 66$, $Y = (2,2,2) = 2$, $S = 0$

in aux. RNS $X = (1,15,9), Y = (2,2,2),$
$F = 1, \tilde{Y} = (2,2,2), Z' = (3,16,14),$
$D = (3,9,10)$

in main RNS $Z' = (5,0,7) = 33,$
$D = (3,5,2) = 808,$
$X' = \frac{(3,0,1)(3,5,2)-(5,0,7)}{(5,9,3)} = (5,0,7) = 33,$
$Y' = \frac{(2,2,2)(3,5,2)-(1,1,1)}{(5,9,3)} = (1,1,1) = 1$

4. iteration $X = (5,0,7) = 33$, $Y = (1,1,1) = 1$, $S = 1$, $Z = (5,0,7) = 33$.

4 The Unique Characteristic of the Algorithm

In all known division algorithms the number of iterations depends on the magnitude of the quotient Z . On the contrary, the number of iterations in the presented algorithm depends only on the divisor magnitude. It follows the fact that in relations (3) and (5) the value of the new divisor depends only on the value of the previous divisor.

Theoretically, the maximum number of iterations needed for the division by the divisor Y might be $Y-1$. The real number of iterations for the divisor Y depends on the magnitude of the number system used. Generally, the same divisor needs different numbers of iteration in various number systems. Although a great divisor might possibly need less iterations than a small one, it is normal that the number of iterations grows with the magnitude of the divisor. An exact method to determine the real number of iterations for a given system is not known. Table 2 presents results obtained by simulation in an auxiliary RNS with moduli $43, 47, 53, 59, 61$ and $P = 385499687$.

The described algorithm characteristic can lead to time and hardware savings in the cases when all divisors are of a small value or when more numbers should be divided by the same divisor.

The theory of the presented algorithm presents a base on which fast division by constant (scaling) can be realized in both, the RNS and weighted number systems. Actually, if the multiplicative inverse element of a number can be computed in advance and memorized, then division by this number reduces to one multiplication and one division. This division is in the RNS division by a product of system moduli and for a weighted systems it is division by a product of bases, both easily realizable.

divisor range	mean number of iterations	max. number of iterations
$10-19$	2.2	4
$100-109$	4.9	9
$1000-1009$	6.2	9
$10000-10009$	10.4	15
$100000-100009$	9.9	19
10^6-10^6+9	13.1	18
10^7-10^7+9	14.7	18

Table 2: Mean and maximum number of iterations in the RNS with $P = 385499687$ for divisors of different magnitude.

5 Conclusion

As it is true that some simple algorithms in weighted systems are hard to realize in RNS, it seems that sometimes the opposite is true, as well. Such an example is the computation of the multiplicative inverse element in the RNS of a number relatively prime with all the system moduli. It can be realized in only one step by small independent look-up tables, as fast and simple as addition, subtraction, and multiplication. A new approach to integer division in the RNS makes use of this fact.

The presented algorithm can be realized in a few different ways and among them the described one has some significant advantages: both iterations are realized with the same hardware, each iteration can be executed in the $N+R+3$ modular steps, and the necessary control logic is very simple. Although in the presented realization we succeeded to significantly reduce the number of necessary ROM's, a great hardware complexity is still the main problem of all the integer division algorithms in the RNS.

Regardless of the way the new algorithm is realized, its disadvantage is that, except by simulation, we do not know the way to estimate the mean and maximum number of necessary iterations. At the same time the fact that the number of iterations depends only on the divisor magnitude might be very interesting in some special purpose applications.

Appendix

Proof that the relations (4) and (5) satisfy conditions (1).

The conditions $X < X$ and $Y' < Y$ are satisfied because of the fact that $D < M$. The third condition $\lfloor X'/Y' = Z \rfloor$ is also satisfied if we can prove that in relation

$$X' = ZY' + Q'$$

condition $0 \leq Q' < Y'$ is true where X' and Y' are new dividend and divisor, respectively, and Z the correct result of division of the original dividend and divisor.

From

$$X' = \left\lfloor \frac{XD}{M} \right\rfloor$$

$$D = \frac{1 + Y'M}{Y}$$

$$X = ZY + Q \quad (0 \leq Q < Y)$$

follows

$$X' = \left\lfloor \frac{X + ZYY'M + QY'M}{YM} \right\rfloor$$

$$Q' = \left\lfloor \frac{X + QY'M}{YM} \right\rfloor .$$

Obviously, $Q' \geq 0$ and because of $X \leq M - 1$ and $Q \leq Y - 1$

$$\begin{aligned} Q' &\leq \left\lfloor \frac{M - 1 + YY'M - Y'M}{YM} \right\rfloor = \\ &= \left\lfloor Y' + \frac{M - 1 - Y'M}{YM} \right\rfloor . \end{aligned}$$

Because of $Y' \geq 1$ it follows

$$Q' \leq \left\lfloor Y' - \frac{1}{YM} \right\rfloor = Y' - 1.$$

So it is proved that condition $0 \leq Q' < Y'$ is satisfied and that the presented algorithm is correct.

References

[1] N. S. Szabo, R. I. Tanaka, *Residue Arithmetic and Its Applications to Computer Technology,* McGraw-Hill, 1967.

[2] E. Kinoshita, H. Kosako, Y. Kojima, "General Division in the Symmetric Residue Number System," *IEEE Trans. on Computers*, Vol. C-22, pp. 134-142, 1973.

[3] G. A. Jullien, "Residue Number Scaling and Other Operations Using ROM Arrays," *IEEE Trans. on Computers*, Vol. C-27, pp. 325-336, 1978.

[4] R. T. Gregory, E. V. Krishnamurthy, *Methods and Applications of Error-Free Computation,* Springer-Verlag, 1984.

[5] D. Gamberger, "Incompletely Specified Numbers in the Residue Number System - Definition and Applications" *Proceedings of the* 9^{th} *Symposium on Computer Arithmetic*, pp. 210-215, Santa Monica, U.S.A., 1989.