Binary Logic for Residue Arithmetic Using Magnitude Index

THAMMAVARAPU R. N. RAO, MEMBER, IEEE, AND AVTAR K. TREHAN

Abstract—We consider a residue number system using n pairwise relatively prime moduli $m_1, \cdots, m_n$ to represent any integer X in the range $M/2 \leq X > M/2$, when $M = \prod m_i$. The moduli m_i are chosen to be of the 2−1 type, in order that the residue arithmetic can be implemented by means of binary registers and binary logic. Further, for each residue number X, a magnitude index P_x is maintained for all arithmetic operations. We investigate the properties of such a system and derive the addition, subtraction, multiplication, sign determination, and overflow detection algorithms. The proposed organization is found to improve the operation times for sign detection and overflow detection operations, while rendering multiplication to be a difficult operation.

Index Terms—Base extension, index generation logic, magnitude index, modular adders, naturalized form, overflow detection, residue mult plication, residue number system, scale by 2, sign determination.

I. Introduction

A CONSIDERABLE amount of attention has recently been devoted to the use of residue number theory in computer system design. The first documented effort in this direction was by Svoboda [8] in 1954. Later Garner [3] and Aiken and Semon [1] studied and demonstrated the advantages of residue number systems for computer applications.

The principal advantage of doing arithmetic in the residue mode is in multiplication, addition, and complementation, the three basic arithmetic operations. In residue arithmetic, a digit of the result depends only on the corresponding digits of the operands. This property eliminates the propagation of carries between the digit positions, and consequently there is no need to form partial products in multiplication except for the individual moduli. Hence, the time required for multiplication is limited by the largest modulus used in the system.

The main disadvantage of the residue system is that the sign of any number in the residue representation is a function of all the residue digits, which is not so in the conventional system. This makes sign determination in a residue computer relatively difficult and time consuming. The detection of overflow in the addition operation and the magnitude comparison of two numbers involve sign determination and hence are equally difficult. However, the problems of sign determination and overflow detection usually reduce to the problem of magnitude comparison with $M/2$ and M, where M is the range of the number system and is equal to the product of the moduli when they are pairwise relatively prime, in which case the system is a nonredundant representation of numbers.

Manuscript received May 15, 1969; revised February 9, 1970. This work was supported by the National Science Foundation under Grant GK-1543.

The authors are with the University of Maryland, College Park, Md. 20742.

In this paper an attempt has been made to speed up the above-mentioned arithmetic operations, sign determination, and magnitude determination, using moduli of the type 2^k or 2^k-1 as suggested by Merrill [5]. Also with each residue number a magnitude index will be attached as suggested by Sasaki [6].

For the purpose of background a brief discussion of [5] and [6] will be provided. The reader's knowledge of the base extension [9] is assumed, as it will be used later in the multiplication algorithm.

A. Notations and Definitions

$|x|_m$ = least nonnegative residue of x modulo m

m_i = ith modulus $i = 1, 2, \cdots, n$. The moduli are pairwise relatively prime

M = range of the residue number system, i.e., the number of different integers which can be represented nonredundantly in the residue system where

$$M = \prod_{i=1}^{n} m_i$$

$M_i = M/m_i$

$n_i = |1/M_i|_{m_i}$, i.e., the multiplicative inverse of M_i mod m_i

$X_{ii} = |n_i X|_{m_i}$

$y_i = n_i M_i$, called the ith weight

$X^* = (X_1, X_2, \cdots, X_n)$ residue number representation of X with respect to moduli $m_1, m_2, \cdots, m_n$, $X_i = |X|_{m_i}$

$X^* = X_1, X_2, \cdots, X_n/P_x$, residue number representation with magnitude index of X; X_i here is any integer

$X^* = X_1, X_2, \cdots, X_n//P$, naturalized residue number with magnitude index of X, $0 \leq X_i < m_i$.

B. Moduli of the Type 2^k, 2^k-1, and Binary Coding of Residues

Merrill [5] and Rao [10] have suggested independently that residue arithmetic can be implemented using some modifications of a conventional binary arithmetic unit. Merrill demonstrated that in a nonredundant system using moduli which are of the form 2^k or 2^k-1, addition, subtraction, and multiplication can be carried out a great deal faster than in a conventional computer. Specifically, he considered a conventional binary system with a word length of 25 bits, and a residue system with moduli 128, 127, 63, 31. In this system addition and subtraction in the residue arithmetic mode are three times faster, and multiplication twelve times faster, than in the conventional system.

The basic system design approach of Merrill is best described as the modification of the arithmetic and control units of a parallel organized conventional computer, so tha

computation routines can be executed in either the con-
tional binary mode or the residue mode. The arithmetic
it is utilized to execute the residue system routines by the
rtitioning of the adder and shift register into segments
om a control standpoint. Each corresponding adder and
ift register segment have the same number of bit positions.
e number of bits in each segment is that required for
nary encoding of the residue digits of the particular
odulus. The adder shift register segments are controlled
parately during the residue computation mode, so that
rtain shift and carry bit transfer operations peculiar to
ch modulus can be accomplished. The control unit has
so to be modified to accept the residue add and shift in-
ructions.

By selecting moduli that are powers of 2 or one less
an a power of 2, residue addition, subtraction, and multi-
lication can be implemented very efficiently using con-
entional binary arithmetic circuitry, assuming, of course,
at the residue digits are binary encoded.

Addition modulo 2^k of two residue digits, each modulo 2^k,
equivalent to conventional k-bit binary addition, except
at the most significant bit carry is ignored, as $|2^j|_{2^k}=0$
or $j \geq k$. Since $|2^k|_{2^k-1}=1$, addition modulo (2^k-1) is
quivalent to conventional k-bit binary addition with end-
round carry. Hence addition modulo 2^k or 2^k-1 can be
ccomplished in one k-bit binary addition. Residue subtrac-
ion can be accomplished by adding mod m_i, the subtra-
hend's additive inverse to the minuend. The additive inverse
or $|X|_{2^k-1}$ is $2^k-1-|X|_{2^k-1}$, or the 1's complement of the
binary representation, and it is $2^k-|X|_{2^k}$, or the 2's com-
plement of the binary representation of $|X|_{2^k}$. By utilizing
a k-bit adder which accepts a carry input to the lowest bit
position, mod 2^k subtraction can be accomplished in one
k-bit addition interval by forming the 1's complement of the
binary representation for the subtrahend and adding the
result to the minuend while at the same time introducing a
1 into the lowest bit position carry input.

Multiplication in this system can be accomplished by
the usual shift and add technique. Multiplying by 2^j is
accomplished in the mod 2^k shift register exactly as in a
conventional k-bit shift register because $|2^j|_{2^k}=0$ for all
$j \geq k$.

However, one can show that $|2^j|_{2^k-1}=2^{|j|_k}$, and therefore
multiplying by a power of 2 in a mod (2^k-1) register is
accomplished as in a conventional k-bit shift register,
except that each bit that is shifted out is recirculated into the
register's lowest bit position by a cycle or rotate operation.
In general, multiplying by a power of 2 mod 2^k or 2^k-1
requires only shifting or cycling operations and no addi-
tions. Using a mixed radix conversion is discussed in [5].

C. *Residue Systems with Magnitude Index*

Following the notation used by Sasaki [6] with some minor modifications, an integer x of any sign and magnitude is represented by $x_1, x_2, \cdots, x_n/P_x$, if

$$x = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n - P_x M \qquad (1)$$

where M is the product of the moduli, and the y_i are the digit weights as defined before. P_x will be called the magnitude index of x. We denote this by

$$x \overset{*}{=} x_1, x_2, \cdots, x_n/P_x.$$

Note here that in the conventional residue representation which uses no magnitude index, the x_i are such that $0 \leq x_i < m_i$ and $0 \leq x < M$. In the magnitude index system the x_i and P_x can be any integers, and therefore x can be any integer. On the other hand, if x is represented in the *naturalized* or *natural* form with magnitude index, then we write

$$x \overset{*}{=} x_1, x_2, \cdots, x_n//P_x$$

and for $0 \leq x_i < m_i$. (The double slash is used here to emphasize the natural form.) As an example, consider the 7, 5, 3 system (that is, $m_1=7$, $m_2=5$, and $m_3=3$). Some of the expressions for $x=2$ are

$$2 \overset{*}{=} 2, 2, -1/0$$
$$2 \overset{*}{=} -5, -3, -1/-2$$
$$2 \overset{*}{=} 2, 2, 2/2.$$

The reader may check these expressions by substituting the values for $y_1=15$, $y_2=21$, $y_3=70$, and $M=105$ $(n_1=1, n_2=1, n_3=2)$ in (1).

Since the residues x_i are in the range $0 \leq x_i < m_i$ in the expression 2, 2, 2/2, this is already in naturalized form, and and hence can be written as $2 \overset{*}{=} 2, 2, 2//2$.

Representation of negative numbers poses no problem; (1) applies to negative numbers as well. For example, in the system with moduli 7, 5, 3, $x=-2 \overset{*}{=} 5, 3, 1//2$ as can be seen.

D. *Addition and Subtraction*

Consider two numbers x, y:

$$x = x_1, x_2, \cdots, x_n//P_x$$
$$y = y_1, y_2, \cdots, y_n//P_y$$
$$z = x \pm y = x_1 \pm y_1, x_2 \pm y_2, \cdots, x_n \pm y_n/P_x \pm P_y$$
$$z = z_1, z_2, \cdots, z_n//P_z.$$

The expression for sum (or difference) is to be naturalized by a generation of borrows (or carries) from each residue position and absorption of them with the magnitude index $P_x \pm P_y$. A carry (or borrow) from the ith residue to be absorbed has a value of n_i, since $m_i y_i = m_i M_i n_i = M n_i$.

Examples: (7, 5, 3 moduli) $M = 7 \times 5 \times 3 = 105$

$$M_1 = 15; \quad M_2 = 21, \quad M_3 = 35$$

$$n_1 = \left|\frac{1}{15}\right|_7 = 1 \quad n_2 = \left|\frac{1}{21}\right|_5 = 1 \quad n_3 = \left|\frac{1}{35}\right|_3 = 2$$

$$y_1 = M_1 n_1 = 15, \quad y_2 = M_2 n_2 = 21, \quad y_3 = M_3 n_3 = 70$$

$$\begin{array}{rl} 19 & \overset{*}{=} 5, 4, 1//2 \\ +32 & \overset{*}{=} 4, 2, 2//2 \\ \hline 51 & \overset{*}{=} 9, 6, 3/4 \\ & \overset{*}{=} 9-7, 6-5, 3-3//4-n_1-n_2-n_3 \\ & \overset{*}{=} 2, 1, 0//4-1-1-2 \\ & \overset{*}{=} 2, 1, 0//0. \end{array}$$

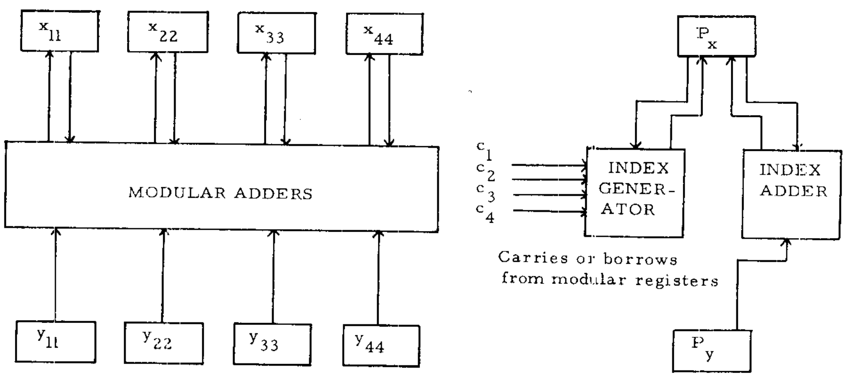

Fig. 1. Schematics for the addition registers and magnitude index generation logic.

Verifying (1), $51 = 2\times 15 + 21\times 1 + 0\times 70 - 0\times 105$.

$$\begin{aligned}
&19 \overset{*}{=} 5, 4, 1//2\\
&\underline{32 \overset{*}{=} 4, 2, 2//2}\\
19 - 32 = {}&-13 \overset{*}{=} 1, 2, -1/0\\
&\overset{*}{=} 1, 2, (-1 + 3)//0 + n_3\\
&\overset{*}{=} 1, 2, 2//2.
\end{aligned}$$

These examples demonstrate that the magnitude index reintroduces carries into the residue system, which obviously slows down operation and also increases the hardware costs. These carries or borrows from the residue digits to the magnitude index do not have the value 1, but rather n_i, which can be an awkwardly large number. Hence magnitude index formation, which involves addition of P_x and P_y and subtraction of the values n_i from the n residues, is a very formidable problem. One might say that magnitude index formation is the bottleneck in the process flow.

II. Residue System with Magnitude Index Using 2^k, 2^k-1-Type Moduli

A new residue system is proposed which uses the magnitude index in a modified form, as well as moduli of the type 2^k, 2^k-1. That means we combine the techniques suggested by Sasaki and Merrill. In this process, we introduce a somewhat modified representation scheme and a magnitude index generator logic.

In the proposed system, the moduli are of the 2^k, 2^k-1 type and are pairwise relatively prime. For any x, $-M/2 \leq x < M/2$, represented by the expression

$$x = x_{11}, x_{22}, \cdots, x_{nn}//P_x$$

where $x_{ii} = |x/M_i|_{mi} = |n_i x|_{mi}$ for all i, x satisfies

$$x = x_{11}M_1 + x_{22}M_2 + \cdots + x_{nn}M_n - P_xM. \qquad (2)$$

It is important to note the distinctions made in the proposed representation and the previously given representation as given by (1). In (2) we use x_{ii} and M_i as against x_i and y_i in (1). The digit weights in the proposed system are M_i instead of $y_i = n_iM_i$; consequently, the carries (or borrows) from each residue position (or modular register i) to the magnitude index register are going to be simply 1's, instead of being n_i, and hence provide easy absorption of these carries (and borrows). Further, from (2), using $-M/2 \leq$ $< M/2$, we get

$$P_x = \left(\sum_{i=1}^{n} x_{ii}M_i - x\right)/M. \qquad (3$$

Since each $x_{ii}M_i < M$, P_x can be no greater than n and n less than 0, giving $0 \leq P_x \leq n$. In the Sasaki system we ge

$$0 \leq P_x \leq \sum_{i=1}^{n} n_i.$$

The magnitude index register in the proposed system wil therefore be very much smaller.

A. System Organization

The system organization is essentially the same as used by Merrill [5], except that an extra shift register called a magnitude index register (MIR), a magnitude index adder, and an index generator (IG) as shown are required. (See Fig. 1.) The MIR size is dependent on the number of moduli and the index generation scheme employed. For a system using four moduli, such as 128, 127, 63, and 31, the MIR need only be of 5 bits.

The index adder forms $P_x + P_y$ and stores the result in MIR. The index generator receives borrows (or carries) from the residue adders and $P_x + P_y$ from the MIR and obtains P_z. Since $P_x + P_y$ is obtained in a parallel fashion with the residues $x_{11} + y_{11}$, the extra time required for the addition is only that needed for absorption of the possible carries in IG. The index generator, a combinational function of 9 inputs $(5+n)$ and 5 outputs, is a formidable network. But it is the most important functional block and its speed is of immense value for the success of this proposed system. Hence it is proposed that this function be generated in a few logic levels of delay, of the order of 4 levels or less. We will show that in such an event, shift or scale operation and sign detection can be obtained relatively fast.

B. Addition

For the moduli

$$127 \quad 63 \quad 31 \quad 128$$

we consider addition:

$$x \overset{*}{=} x_{11} \quad x_{22} \quad x_{33} \quad x_{44}//P_x$$
$$y \overset{*}{=} y_{11} \quad y_{22} \quad y_{33} \quad y_{44}//P_y$$
$$+ y \overset{*}{=} x_{11} + y_{11}, x_{22} + y_{22}, x_{33} + y_{33}, x_{44} + y_{44}//P_x + P_y - C$$

here

$$C = C_1 + C_2 + C_3 + C_4.$$

ere $x_{ii}+y_{ii}$ is addition modulo m_i, and C denotes the sum the carries from the modular registers. The operation me is the time required for the addition modulo 127, plus e time delay in the subtraction of C. (Note P_x+P_y is formed parallel with the $x_{ii}+y_{ii}$.) If the response of IG is Δ, the dition in the proposed system is Δ units longer than the dition without the magnitude index.

C. Complement

For any x, we denote $-x$ by $\bar{x}$.

$$x = x_{11} \quad x_{22} \cdots x_{nn}//P_x$$
$$\bar{x} = m_1 - x_{11} \quad m_2 - x_{22} \cdots m_n - x_{nn}//P_{\bar{x}}$$
$$x + \bar{x} = 0 \quad 0 \cdots 0//P_x + P_{\bar{x}} - n.$$

Therefore $P_x + P_{\bar{x}} - n = 0$, and hence $P_{\bar{x}} = n - P_x$.

While the complements of the residues are with respect to the moduli m_i (i.e., 2's and 1's complements), the complement of the magnitude index is with respect to n, the number of moduli. Subtraction of y from x can be carried out by adding $\bar{y}$ to x. The time required for subtraction is about the same as for addition.

D. Multiplication

Let the two operands x and y be defined as follows:

$$x = \sum_i x_{ii}M_i - P_xM, \quad -M/2 \le x < M/2.$$
$$y = \sum_i y_{ii}M_i - P_yM, \quad -M/2 \le y < M/2.$$

Let $z = |x \; y|_M = \sum Z_{ii}M_i - P_zM$ and we desire to obtain $z = z_{11}, z_{22}, \cdots, z_{nn}//P_z$. Each z_{ii} can be obtained from the corresponding values of x_{ii} and y_{ii}, but P_z is a function of all x_{ii}, all y_{ii}, P_x, and P_y. Realization of P_z involves a horrendous amount of computation and brings out the hidden lie about residue systems with magnitude index. No simple algorithm can be conceived to determine P_z from the two operand expressions. An algorithm is given here for the realization of z; it is by no means simple, but the most reasonable we can think of. This algorithm includes, as a first step, obtaining z_{ii}, and then computation of P_z from the z_{ii} by means of base extension. The z_{ii} determine uniquely the product z if z is constrained to the limits, such as $0 \le z < M$, and P_z contains only redundant information. Since we are using $n_1, n_2, \cdots, n_n//P_1$ to represent 1, z_{ii} is not equal to $|x_{ii} \; y_{ii}|_{m_i}$, but

$$z_{ii} = |x_{ii} \; y_{ii} \; n_i^{-1}|_{m_i}.$$

Therefore the algorithm involves

1) obtaining $r_i = |x_{ii} \; y_{ii}|_{m_i}$ for $i = 1, 2, \cdots, n$;
2) obtaining $z_{ii} = |r_i n_i^{-1}|_{m_i}$ for $i = 1, 2, \cdots, n$;
3) computing P_z from $(z_{11}, z_{22}, \cdots, z_{nn})$ by the use of the base extension method [9] and (3) derived below.

Given $(z_1, \cdots, z_n)$, where $z_i = |z|_{m_i}$, for some z $(0 \le z < M = \prod_i m_i)$, the method used to obtain $z_{n+1} = |z|_{m_{n+1}}$ for a new modulus m_{n+1} is called *base extension.* (Note that all moduli including m_{n+1} are pairwise relatively prime.) The base extension operation requires $n+1$ multiplications and n subtractions [9]. The procedure for obtaining P_z from the z_{ii} is as follows [2]: for $0 \le z < M$,

$$z = \sum_{i=1}^{n} z_{ii}M_i - MP_z$$

or

$$P_z = \sum_{i=1}^{n} z_{ii}\frac{M_i}{M} - \frac{z}{M}$$
$$|P_z|_{m_{n+1}} = \left| \sum z_{ii}\frac{1}{m_i} - \frac{1}{M}z \right|_{m_{n+1}}.$$

Since $\text{GCD}(M, m_{n+1}) = 1$, we get

$$|P_z|_{m_{n+1}} = \left| \sum z_{ii}\left|\frac{1}{m_i}\right|_{m_{n+1}} - \left|\frac{1}{M}\right|_{m_{n+1}} z_{n+1} \right|_{m_{n+1}}.$$

This equation may be simplified by denoting the multiplicative inverse of m_i and M mod m_{n+1} as follows:

$$\left|\frac{1}{m_i}\right|_{m_{n+1}} = \mu_i$$
$$\left|-\frac{1}{M}\right|_{m_{n+1}} = \mu_s.$$

Further, if $P_z < m_{n+1}$, we get

$$P_z = \left| \sum_{i=1}^{n} z_{ii}\mu_i + z_{n+1}\mu_s \right|_{m_{n+1}}. \tag{4}$$

Equation (4) gives P_z for the given values of $z(0 \le z < M)$ and $z_{11}, z_{22}, \cdots, z_{nn}$. This is because the base extension method derives z_{n+1} from $z_1, z_2, \cdots, z_n$ for $0 \le z < M$ but not for $-M/2 \le z < M/2$. Therefore conversion of P_z for the range $-M/2 \le z < M/2$ is required. This can be accomplished by adding 1 to P_z whenever z is greater than or equal to $M/2$. Fortunately, one can recognize this condition from the base extension procedure used. These ideas are illustrated by the examples below.

Example: Consider the 8, 7, 3 system. Here $M = 8 \cdot 7 \cdot 3 = 168$. Given $x = y = 9 \overset{*}{=} 5, 3, 0//1$, we intend to determine $z = xy \overset{*}{=} z_{11}, z_{22}, z_{33}//P_z$ as follows:

$$n_1^{-1} = |M_1|_8 = |21|_8 = 5$$
$$n_2^{-1} = |24|_7 = 3,$$
$$n_3^{-1} = |56|_3 = 2$$
$$z_{11} = |x_{11} \; y_{11} \; n_1^{-1}|_{m_1} = |5 \cdot 5 \cdot 5|_8 = 5$$
$$z_{22} = |x_{22} \; y_{22} \; n_2^{-1}|_{m_2} = |3 \cdot 3 \cdot 3|_7 = 6$$
$$z_{33} = |x_{33} \; y_{33} \; n_3^{-1}|_{m_3} = |0 \cdot 0 \cdot 2|_3 = 0.$$

Also $z_i = |z_{ii} \; n_i^{-1}|_{m_i}$. Hence $z_1 = 1$, $z_2 = 4$, $z_3 = 0$.

Extending to the base $m_4=5$, we get $z_4=|z|_5=1$ and that $z<M/2=84$. (The base extension is not shown here.) Further, we obtain

$$\mu_1 = \left|\frac{1}{8}\right|_5 = 2$$

$$\mu_2 = \left|\frac{1}{7}\right|_5 = 3$$

$$\mu_3 = \left|\frac{1}{3}\right|_5 = 2$$

$$\mu_s = \left|\frac{-1}{168}\right|_5 = 3.$$

Substituting these above values in (4), we get

$$P_z = P_{81} = |z_{11}\mu_1 + z_{22}\mu_2 + z_{33}\mu_3 + z_4\mu_s|_{m_s}$$
$$= |5 \times 2 + 6 \times 3 + 0 \times 2 + 3 \times 1|_5 = 1.$$

Since $z<M/2$, as provided by base extension, no correction is required. Hence

$$z = 81 \overset{*}{=} 5, 6, 0//1$$

is the required result.

As a verification of (2),

$$81 = 5 \times 21 + 6 \times 24 + 0 \times 56 - 1 \times 168.$$

E. Scaling by 2

We define here an elementary operation $O(x)$ on a number x expressed in the natural form. The operation is a multiplication by 2 with the result expressed in the natural form, as follows:

$$x \overset{*}{=} x_{11}, x_{22}, \cdots, x_{nn}//P_x$$

$$O(x) = 2x \overset{*}{=} 2x_{11}, 2x_{22}, \cdots, 2x_{nn}/2P_x$$

$$O(x) = y \overset{*}{=} y_{11}, y_{22}, \cdots y_{nn}//P_y.$$

This operation is very useful in the sign detection scheme to be described in the next section, and also in floating-point arithmetic. The multiplication by 2 can be accomplished by a left shift or left rotation of the residue digits to form y_i. At the same time, P_y is obtained from the output of the combinational circuit whose inputs are $2P_x$ and the one most significant bit from each of the n residues. We can use the same magnitude index generation logic as described in Section II-A. In this case the inputs to the index generator (IG) are the most significant bits of each of the n residues instead of carries from them (see Fig. 2).

This operation is much faster than addition, as here we do not have to wait for the carries, and the most significant bits are always available. The following example illustrates the operation $O(x)$.

Example: Let $m_1=8$, $m_2=7$, and $m_3=3$. We shall use binary notation to illustrate the formation of the magnitude index. We use a three-bit MIR. Let

$$x = 110 \quad 101 \quad 01//001$$

$$O(x) = 100 \quad 011 \quad 10/010 - (S_1 + S_2 + S_3)$$

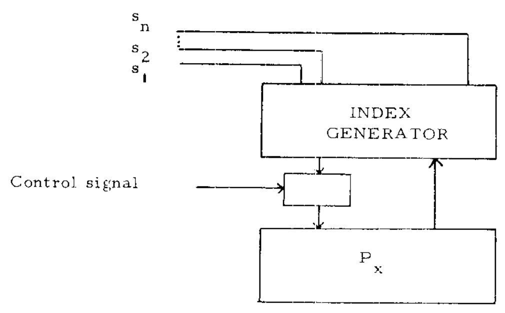

Fig. 2. Index generation logic.

$$O(x) = 100 \quad 011 \quad 10//010 - (1 + 1 + 0)$$
$$= 100 \quad 011 \quad 10//000.$$

(Note that S_i is the leftmost bit of x_i for $i=1, 2, 3$.)

F. Sign Detection

The sign function $S(x)$, such that $-M/2 \le x < M/2$, is defined as follows:

$$S(x) = \begin{cases} 0, & 0 \le x < M/2 \\ 1, & -M/2 \le x < 0. \end{cases}$$

To find the function $S(x)$ for any number x represented in the naturalized form we look at (2). (Also $0 \le P_x \le n$ for $-M/2 \le x < M/2$).

$$x = x_{11}M_1 + x_{22}M_2 + \cdots + x_{nn}M_n - P_xM.$$

By inspection of this equation it is seen that if $P_x \ge n$, then $S(x)=1$. Also since all the terms in the above equation are positive, $S(x)=0$ if $P_x \le 0$. Hence we get the following rules for determining the sign function $S(x)$.

1) If $x_{ii}=0$, for all $i=1, 2, \cdots, n$, then $S(x)=1$ when $P_x>0$, and $S(x)=0$ when $P_x \le 0$.
2) If $P_x \ge n$, then $S(x)=1$.
3) If $P_x \le 0$, then $S(x)=0$.

In order to determine the function $S(x)$, we perform the elementary operation $O(x)$, discussed in the last section, on x. $O(x)$ is checked for rules 1, 2, or 3, and if the function $S(x)$ is not obtainable from $O(x)$, we look at $O(O(x))$ denoted by $O^2(x)$, and if $S(x)$ is still not obtainable by a check of the above rules, we look at $O^3(x)$, $O^4(x)$ and so on, till we get the function $S(x)$. The following lemma puts an upper bound on the number of scaling operations for obtaining the sign function $S(x)$. Let $[K]$ denote the integer greater than or equal to K.

Lemma: U scaling operations are sufficient to determine the sign of x, where $U=[\log_2 nM]$.

Proof: Let $x \overset{*}{=} x_{11}, x_{22}, \cdots, x_{nn}//P_x$. From (2) we get

$$x = x_{11}M_1 + x_{22}M_2 + \cdots + x_{nn}M_n - P_xM.$$

If $x=0$, then all $x_{ii}=0$, and $S(x)$ is determined by P_x by the application of rule 1. Hence the case of interest will be for $x \ne 0$. Scaling U times implies multiplication by $2^U \ge nM$. Therefore, if $2^Ux=y=y_{11}, y_{22}, \cdots, y_{nn}/P_y$, then

$$2^Ux = \sum_{i=1}^{n} y_{ii}M_i - MP_y$$

or

$$P_y = \frac{1}{M}\sum_{i=1}^{n} y_{ii}M_i - \frac{2^U}{M}x.$$

Denoting

$$\frac{1}{M}\sum_{i=1}^{n} y_{ii}M_i = k$$

$$\frac{2^U}{M} = u,$$

we can write $P_y = k - ux$, where $k < n$, and $u > n$. We consider the following cases.

Case 1—$x > 0$: $P_y < 0$, which implies $S(x) = 0$.
Case 2—$x < 0$: $P_y = k + u|x|$, or $P_y > n$, which implies that $S(x) = 1$.

Hence scaling U times is sufficient to determine the sign of x. This is the number of operations needed for the worst case. Usually it is possible to determine the sign in much fewer than U operations. Since we check for rules 1, 2, or 3 after every scaling operation, the process may be terminated when $S(x)$ is obtained.

For a four-modulus system, with $2^{24} < M \leq 2^{25}$ we need $\log_2 (4 \cdot 2^{25}) = 27$ scaling operations. If a very fast logic for the scaling operation exists, we can continue scaling till $S(x)$ is available.

This scheme uses a large number of operations to extract $S(x)$. The simplicity and uniqueness of the operation give it a definite advantage over other methods [7]. Also the size of the magnitude index register is only 5 bits, which is of small size and allows a reasonably fast logic for the magnitude index generation. It may be noticed that this scheme is not the same as the one suggested by Sasaki [7], in which scaling is done by the moduli of the system successively, which in turn increases the size of the magnitude index register and slows down the process, not to mention the increase in hardware.

G. Overflow Detection in Addition

Let $z = x + y$, where x and y are in the range $[-M/2, M/2)$. The sum z is said to overflow this range if $z \geq M/2$ or $z < -M/2$. This could happen if some two positive (or two negative) operands x and y are added. Detection of overflow in conventional residue systems is by a check of the sign of the two operands against the sign of the sum, thus involving three sign-determination operations. The presence of the magnitude index makes overflow detection possible in a rather simplified manner. Since P_z is uniquely determined for the given n residues $z_{11}, z_{22}, \cdots, z_{nn}$ in the range $-M/2 \leq z < M/2$, any overflow (or underflow) to a value outside the range can be detected. Hence the overflow detection algorithm is as follows.

1) Add x and y to get the result z.
2) Compute P_z from $z_{11}, z_{22}, \cdots, z_{nn}$ as given in the multiplication algorithm.
3) Compare the P_z of step 2 with P_z obtained in step 1. If the two are different, an overflow is indicated.

Although the computation of P_z from the residues is not simple, this method should yield a favorable operation time when compared with the conventional overflow detection.

III. Conclusion

A new residue system has been proposed which uses n moduli of the types 2^k and $2^k - 1$, and also uses a magnitude index P_x for each integer x represented in the system. A 1 in the system is represented here as $(n_1, n_2, \cdots, n_n // P_1)$ rather than as $(1, 1, \cdots, 1, // P_1)$ thereby easing considerably the size and complexity of the magnitude index generation logic. Addition and subtraction (by complement and add) can be accomplished with very little loss of speed by the use of a high-speed index generator circuit, which absorbs the carries from the modular adders. Multiplication is found to be the most difficult operation in the proposed system, replacing the old nemesis, the sign detection operation. A new fundamental operation, called scaling by 2, is designed to accomplish multiplication by 2 in about the time required for a shift in conventional binary logic. This enables sign detection to be accomplished by a repetition of the scale operation as required. An upper bound U on the number of scale operations required to detect the sign is given by $U = [\log_2 Mn]$, where M is the range and n is the number of moduli. For a 128, 127, 63, 31 system, the sign detection requires fewer than $U = [\log_2 Mn] = 27$ scale operations (or 27 shifts), assuming a high-speed magnitude index generator logic is used. The presence of magnitude index also eases to some extent the problem of additive overflow; thus we could conclude that the magnitude index improves sign detection operation significantly, while the multiplication operation replaces the sign detection as the one operation to be dealt with in the future.

Acknowledgment

The authors thank the Guest Editor, Dr. R. Shively, and the IEEE referees for many helpful suggestions.

References

[1] H. Aiken and W. Semon, "Advanced digital computer logic," Wright Air Development Center, Tech. Rept. WADC TR-59-472, July 1959.
[2] I. Flores, *The Logic of Computer Arithmetic.* Englewood Cliffs, N. J.: Prentice-Hall, 1963.
[3] H. L. Garner, "The residue number system," *IRE Trans. Electronic Computers*, vol. EC-8, pp. 140–147, June 1959.
[4] O. L. MacSorley, "High-speed arithmetic in binary computers," *Proc. IRE*, vol. 49, pp. 67–91, January 1961.
[5] R. D. Merrill, Jr., "Improving digital computer performance using residue number theory," *IEEE Trans. Electronic Computers*, vol. EC-13, pp. 93–101, April 1964.
[6] A. Sasaki, "Addition and subtraction in the residue number system," *IEEE Trans. Electronic Computers*, vol. EC-16, pp. 157–164, April 1967.
[7] ——, "The basis for implementation of additive operations in [illegible] residue number system," *IEEE Trans. Computers*, vol. C-17, [illegible] 1066–1073, November 1968.
[8] A. Svoboda, "The numerical system of residual classes in [illegible] matical machines," *Information Processing, Proc. UNESCO* [illegible] (Paris, June 1959), 1960, pp. 419–422.
[9] N. S. Szabo and R. I. Tanaka, *Residue Arithmetic and Its Applications to Computer Technology.* New York: McGraw-Hill, 1967, pp. 47–50.
[10] T. R. N. Rao, "The general properties of finite, weighted number systems," Ph.D. dissertation, University of Michigan, Ann Arbor, December 1963.