SIGN DETECTION IN RESIDUE NUMBER SYSTEMS

by

Dilip K. Banerji, B. Tech.

Submitted in partial fulfillment of
the requirements for the degree
of Master of Science

Department of Electrical Engineering,
Faculty of Pure and Applied Science,
University of Ottawa
Ottawa, Ontario
Canada.

April, 1967

TO

PIA

ABSTRACT

This thesis is concerned with the sign detection problem in residue (modular) number systems. Residue codes are introduced and their important properties are discussed. It is shown that under rather general conditions an explicit, closed formula for the sign function can be obtained. In a special case, when one of the moduli is 2, the sign function becomes an EXCLUSIVE OR function. Using the above formula, a general sign detection algorithm is proposed and methods of implementing the algorithm are presented. For completeness of presentation, some system design concepts are presented including a simple method of input translation. The thesis concludes with a comparison of various sign detection methods.

ACKNOWLEDGMENTS

To Dr. Janusz A. Brzozowski go my very sincere thanks for his guidance, many stimulating comments and suggestions which made this thesis possible.

Thanks are also due to Dr. Wayne A. Davis for introducing the author to residue arithmetic, and to my colleagues, Vinod Batra, Godfrey Muehle and Shanker Singh, whose criticism has been very helpful. My particular thanks to Miss Dodie Kocis who so patiently typed the manuscript.

The financial support of the National Research Council of Canada, under Grant No. A - 1617, is gratefully acknowledged.

CONTENTS

1. INTRODUCTION

Residue number systems have interested mathematicians for a very long time[1]. However, the use of the system as a tool for machine computation has attracted attention quite recently. The first known work in this direction was done in Czechoslovakia by Valach and Svoboda [1]. Their work has been followed up mainly by Garner [2], Aiken and Semon [3], Cheney, [4], Szabo [5] and Watson and Hastings [8].

The most attractive feature of the residue number system is that in the operations of addition, subtraction and multiplication, any particular digit of the result is determined solely by the corresponding digits of the operands. As we shall see, this results in the elimination of carry propagation from one residue digit to another in all the three operations. Furthermore, it removes the need for partial products in multiplication. Therefore, the execution time is appreciably reduced and can be made approximately the same for all the three operations. One of the main drawbacks of the system is the fact that the algebraic sign of any number in an arbitrary residue code is a function of all the residue digits. As a result the sign detection process in this system is complicated, slow and expensive. Therefore, the operations requiring the sign information are relatively slow. Hence such problems as relative magnitude determination and overflow detection in the system present problems.

It is the purpose of this thesis to investigate the sign detection problem. A solution to this problem will automatically solve the closely related problems of relative magnitude determination and overflow detection.

[1] Sun-Tsu, a Chinese mathematician in first century A.D. stated the Chinese Remainder Theorem. Gauss, Fermat, Ramanüjan and several other mathematicians have also studied the system.

2. RESIDUE CODES AND THEIR PROPERTIES

In this section we introduce residue codes ; the discussion is based mainly on [2].

2.1 Notations and Definitions

The number theoretical concept of congruences is the basis for residue codes.

Definition 2.1 : If a, b and m are integers, $m > 0$, then a is congruent to b modulo m, written $a \equiv b \mod m$, if and only if, a and b have the same remainder when divided by m. It is known that $a \equiv b \mod m$ iff $m \mid (a - b)$ or $a - b = km$ or $a = b + km$, where $k \in J$, the set of all integers. b is called a residue of a, with respect to m and m the base or modulus.

Example 2.1 :

$3 \equiv 1 \mod 2$ since $3 = 1(2) + 1$

$5 \equiv 1 \mod 2$ since $5 = 2(2) + 1$

1, 3, and 5 belong to the same residue class modulo 2.

If, in the relation $a = b + km$, $0 \le b < m$, then b is called the least positive residue of a, modulo m, denoted by $[a]_m = b$. The least positive residues of a number with respect to different moduli are used to represent the number in the residue system based on the moduli.

2.2 Properties of Congruences

Some properties of congruences relevant to residue systems are discussed here.

(1) Congruence mod m is an equivalance relation :

(i) $a \equiv a \mod m$,

(ii) $a \equiv b \mod m$ implies $b \equiv a \mod m$,

since both a and b, when divided by m have the same remainder.

(iii) $a \equiv b \mod m$ and $b \equiv c \mod m$ implies $a \equiv c \mod m$, since a and b divided by m have the same remainder and b and c divided by m have the same remainder. Hence a and c have the same remainder.

(2) If $a_i \equiv b_i$ mod m for $i = 1, 2, \ldots, n$, then

$$\sum_{i=1}^{n} a_i \equiv \sum_{i=1}^{n} b_i \mod m$$

because

$$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} (b_i + k_i \ m) = \sum_{i=1}^{n} b_i + m \sum_{i=1}^{n} k_i = \sum_{i=1}^{n} b_i + mk$$

(3) If $a_i \equiv b_i$ mod m for $i = 1, 2, \ldots, n$, then

$$\prod_{i=1}^{n} a_i \equiv \prod_{i=1}^{n} b_i \mod m.$$

Consider $a_1 \equiv b_1 + k_1 m$, $a_2 = b_2 + k_2 m$.

Then $a_1 a_2 = b_1 b_2 + (b_1 k_2 + b_2 k_1 + k_1 k_2) m$,

or $a_1 a_2 \equiv b_1 b_2$ mod m.

Repeating this argument we obtain the desired result.

(4) If $a \equiv b$ mod m and $a \equiv b$ mod n, then $a \equiv b$ mod l.c.m. (m,n), where l.c.m. is the least common multiple.

This is easily proved. If p is a prime and p^c is the highest power of p which divides l.c.m. (m,n), then $p^c | m$ or $p^c | n$. Therefore $p^c | (a-b)$. This is true for every prime factor of l.c.m (m,n) and hence $a \equiv b$ mod l.c.m. (m,n). This generalizes to any number of moduli.

(5) If g.c.d. (k,m) = 1, where g.c.d. is the greatest common divisor, then $ka \equiv kb$ mod m iff $a \equiv b$ mod m.

Other properties of congruences can be found in references [6], [7]; for the purpose of this thesis the ones discussed above are sufficient.

2.3 Residue Codes

Consider an ordered set of moduli $m_1, m_2, \ldots, m_n$ instead of just one modulus m considered above. The corresponding ordered set of least positive residues of a natural number, with respect to the moduli, forms the residue representation for that number.

Example 2.2 : Consider moduli 2, 3, 5 and a natural number 14.

$$\begin{aligned} 14 &= 7\,(2) + 0 \\ &= 4\,(3) + 2 \\ &= 2\,(5) + 4 \end{aligned}$$

0, 2 and 4 are the least positive residues of 14 with respect to moduli 2, 3 and 5 respectively and form its residue representation in this system. Thus we represent 14 by (0, 2, 4). Residue representations for this set of moduli, for a few natural numbers are given in Table 2.1.

Number	Least Positive Residues		
	Mod 2	Mod 3	Mod 5
0	0	0	0
2	0	2	2
3	1	0	3
5	1	2	0
17	1	2	2
29	1	2	4

Table 2.1

In order to avoid redundancy (unless redundancy is desirable) the moduli of a residue number system must be pair-wise relative prime i.e., the greatest common divisor of each pair of moduli must be unity. If so, then the number of integers that can be coded uniquely in a system with moduli $m_1, m_2, \ldots, m_n$ equals the product $m_1 \cdot m_2 \cdot \ldots \cdot m_n$. This is a direct consequence of the Chinese Remainder Theorem [7]. In the above example, therefore, a total of 30 integers can be coded uniquely. These can correspond to the natural numbers 0 through 29.

The effect of redundancy is demonstrated in the following example:

No.	Mod 2	Mod 4	Mod 2	Mod 3
0	0	0	0	0
1	1	1	1	1
2	0	2	0	2
3	1	3	1	0
4	0	0	0	1
5	1	1	1	2
6	0	2	0	0
7	1	3	1	1

Table 2.2

When the moduli are not relatively prime (as in case of 2 and 4) the total number of unique representations is not equal to the product of the moduli. This happens because some combinations can not occur. For example, consider a number X with representation (0, 1) in mod 2 and mod 4. Then $[X]_2 = 0$ implies X is even and $[X]_4 = 1$ implies X is odd, which is impossible. For moduli 2 and 4 only 4 integers (0 through 3) can be coded uniquely, whereas with relatively prime moduli 2 and 3, 6 integers (0 through 5) have a unique residue code.

2.4 Residue Arithmetic

Addition and Multiplication : The operations of addition and multiplication in a residue number system are valid, provided the resulting sum or product has a proper representation. If this is not true then some sum or product may overflow the range of representation. In case of overflow, more than one sum or product of the natural number system can have the same residue representation. However, if no overflow occurs, there exists an isomorphic relation with respect to operations of addition and multiplication in the residue system and a finite real number system. Since each digit of a residue representation is obtained with respect to a different modulus, the rules of arithmetic for each digit are different.

Let $(a_1, a_2, \ldots, a_n)$ and $(b_1, b_2, \ldots, b_n)$ be two residue numbers in a system of relatively prime moduli $m_1, m_2, \ldots, m_n$. Addition is defined by :

$$s_i \equiv (a_i + b_i) \bmod m_i , \quad i = 1, 2, \ldots, n.$$

Each digit of the sum is obtained by adding the corresponding digits a_i and b_i modulo m_i. Addition of digits in different moduli is undefined and invalid. The dependence of each sum digit on the corresponding operands only, effectively implies the absence of any carries from one residue to another. It also means that addition with respect to different moduli can be performed simultaneously and the time required is equal to the addition time of the slowest unit. A block diagram of the addition mechanism is drawn below.

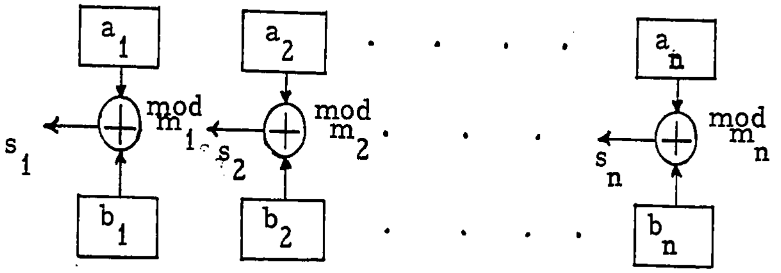


Fig. 2.1 Block Diagram of Adder

Modulo addition is denoted by $\oplus$.

The process of multiplication is also carried out on digit by digit basis :

$$p_i \equiv (a_i \cdot b_i) \bmod m_i , \; i = 1, 2, \ldots, n.$$

Again each digit of the product is obtained by multiplying the corresponding operands a_i and b_i modulo m_i. Multiplication of two digits in different moduli is not defined. There is no carry propagation from one residue position to another, and multiplication can be performed in time taken by the slowest unit. A block diagram of the multiplication mechanism is shown below.

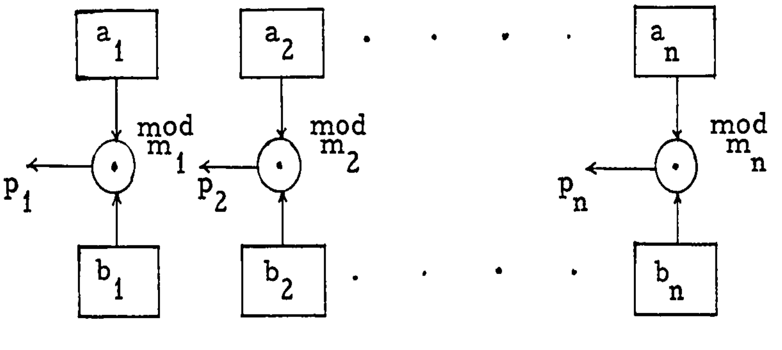


Fig. 2.2 Block Diagram of Multiplier

Modulo multiplication is denoted by $\odot$.

Example 2.3 : Consider the residue system with moduli 2,3 and 5. If we assume an isomorphism between this system and the system of real positive integers 0 through 29, then an isomorphism exists for the operations of addition and multiplication, provided there is no overflow.

$$\begin{array}{rrcl} & 13 & = & (1,\ 1,\ 3) \\ + & 9 & \overset{\oplus}{=} & (1,\ 0,\ 4) \\ \hline & 22 & = & (0,\ 1,\ 2) \end{array}$$

The sum (0, 1, 2) is the residue representation for 22, which is the correct result. Let us see what happens in case of overflow.

$$\begin{array}{rrcl} & 23 & = & (1,\ 2,\ 3) \\ + & 11 & \overset{\oplus}{=} & (1,\ 2,\ 1) \\ \hline & 34 & & (0,\ 1,\ 4) \end{array}$$

The residue representation (0, 1, 4) corresponds to 4. The correct result should be 34, which has overflowed the range of representation. We note that $34 \equiv 4 \mod 30$ i.e., the resulting sum is the correct sum modulo 30.

The process of multiplication in the residue system is shown below.

$$\begin{array}{rrcl} & 5 & = & (1,\ 2,\ 0) \\ \cdot & 5 & \overset{\odot}{=} & (1,\ 2,\ 0) \\ \hline & 25 & = & (1,\ 1,\ 0) \end{array}$$

(1, 1, 0) is the representation for 25 which is the correct result. Of course, if the product overflows, then the resulting product is the correct product module 30.

Subtraction :

The process of subtraction in residue systems is performed by complement addition. The complement of a residue number is obtained by finding the additive inverse of each residue digit with respect to the corresponding modulus. An additive inverse X' of a residue number X is defined by the relation

$$X \oplus X' \equiv 0 \bmod M, \text{ where } M = \prod_{i=1}^{n} m_i .$$

It follows that for any X there exists a unique least positive inverse. If

$$X = (x_1, x_2, \ldots, x_n), \text{ then}$$

$$X' = (x'_1, x'_2, \ldots, x'_n) \text{ where } x'_i = m_i - x_i \text{ for } i = 1, \ldots, n.$$

The additive inverse always exists since the elements of a residue representation belong to a finite ring of M integers.

The process of subtraction does not present any basic problems. The real problem, however, is to represent negative numbers. The problem becomes more clear after we discuss an example.

Example 2.4 : Consider the residue system with moduli 2, 3 and 5 arranged in that order.

Subtraction is defined by

$$s = X - Y = X \oplus Y'$$

Case 1 : $|X| > |Y|$, Let $X = 10$, $Y = 5$

$X = 10 = (0, 1, 0)$

$Y = 5 = (1, 2, 0)$

$Y' = (1, 1, 0)$. Therefore,

$$s = X \oplus Y' = \begin{array}{r} (0,\ 1,\ 0) \\ \oplus\ (1,\ 1,\ 0) \\ \hline (1,\ 2,\ 0) \end{array} .$$

The representation (1, 2, 0) corresponds to 5, which is the correct difference.

Case 2: Consider $s = Y - X = Y \oplus X'$

$X' = (0, 2, 0)$

$$s = \begin{array}{r} (1,\ 2,\ 0) \\ \oplus\ (0,\ 2,\ 0) \\ \hline (1,\ 1,\ 0) \end{array} ,$$

which is the additive inverse of $Y = (1, 2, 0)$. The problem now is that unless some more information is given, (1, 1, 0) may be interpreted to correspond to either -5 or +25.

One way to avoid this difficulty is to divide the residue number range in two parts. One part can correspond to positive integers and the other to negative integers. The negative integers are then represented in radix complement form, defined in terms of additive inverse. Thus, (-X) is represented by X', where X' is as defined before. Consider the system with moduli 2, 3 and 5 again. The total range of representation is divided into two parts : the residue representations corresponding to natural numbers 0 to 14 are considered positive, those corresponding to numbers 15 to 29 are considered to be complement representations and are assigned to negative integers from -15 to -1 . This type of representation is particularly advantageous in the subtraction process. In case 2 of the subtraction process discussed before, we now see that s =(1, 1, 0) is the complement of Y = (1, 2, 0), which corresponds to +5. Hence (1, 1, 0) must correspond to -5.

Division :

The main factor that makes the division process in residue systems difficult is the fact that the residue division and the normal division process for natural numbers are in one to one correspondence only if the resulting quotient is an integer. In the normal division process

$$X/Y = q \text{ iff } X = Yq$$

But in residue arithmetic it is sufficient that

$$X \equiv Yq \mod m$$

holds. In this case, q corresponds to the quotient only when it has an integral value.

Example 2.5 Consider a mod 7 system. Its elements are 0,1,2,3,4,5 and 6. Let

$$6/2 = q$$

Then $2q \equiv 6 \bmod 7$ or $q = 3$.

Now, let $q = 5/2$.

Then $2q \equiv 5 \bmod 7$,

or $q = 6$, since

$2.6 = 12 \equiv 5 \bmod 7$.

Obviously, $5/2 \neq 6$.

In a system consisting of several moduli, the same difficulty is encountered. Division of a number $X = (x_1, x_2, \ldots, x_n)$ by $Y = (y_1, y_2, \ldots, y_n)$ is expressed by a set of congruence relations;

$q_{\ddot{n}} \equiv (x_i/y_i) \bmod m_i$, $i = 1, \ldots, n$, where $Q = (q_1, \ldots, q_n)$ is the result of division. If the divisor $Y = (y_1, \ldots, y_n)$ has any zero digits, then its multiplicative inverse $\overline{Y}$ does not exist. Hence

$QY \not\equiv X \bmod M$.

However, for the special case $x_i = 0$, $y_i = 0$, a valid congruence of the form

$$\frac{QY}{m_i} \equiv \frac{X}{m_i} \bmod \frac{M}{m_i}$$

holds.

2.5 Conversion From Residue Code to Natural Numbers

Whatever the coding scheme used in a computer, it is desirable to have input and output in decimal, binary or bcd form. We have seen how the natural numbers are converted into a residue code. It is possible to mechanize this process. In this section we shall discuss methods of converting a residue representation to a natural number form.

Since a residue representation is not a polynomial type of representation and hence does not contain digit weights, the magnitude of a residue number is not readily available. The magnitude is determined by using the theorem [7] stated below.

Chinese Remainder Theorem : A system of congruences $X \equiv x_i \bmod m_i$, $i = 1, \ldots, n$ has a unique solution mod M if and only if m_i are pairwise relatively prime. The following equations define the process of getting the unique solution [2] :

$$x_1 X_1 \frac{M}{m_1} + \ldots + x_n X_n \frac{M}{m_n} \equiv X \bmod M,$$

where $X_i \frac{M}{m_i} \equiv 1 \bmod m_i$ and

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

Example 2.6: Consider a residue number system with moduli 6 and 7 and let us obtain the conversion equation :

$m_1 = 6$, $m_2 = 7$

$M = 6.7 = 42$

$X_1 \cdot \frac{42}{6} \equiv 1 \bmod 6$

Hence $X_1 = 1$

$X_2 \cdot \frac{42}{7} \equiv 1 \bmod 7$ or $6 X_2 \equiv 1 \bmod 7$.

Hence $X_2 = 6$, since $36 \equiv 1 \bmod 7$.

Therefore, the conversion equation is :

$7 x_1 + 36 x_2 \equiv X \bmod 42.$

Now consider a residue number (4, 3) in this system. We want to find the natural number corresponding to this.

$x_1 = 4$, $x_2 = 3$

$7.4 + 36.3 \equiv X \bmod 42$

or $136 \equiv X \bmod 42$

Hence, $X = 10$.

We can check that the residue code for 10 in this system is indeed (4,3).

There exists a corollary of the Remainder Theorem, which allows us to obtain the natural number from a residue code, when the moduli are composite (not relatively prime).

Corollary : A set of congruences $X \equiv x_i \bmod m_i$, $i = 1, \ldots, n$, has a unique solution mod l.c.m. of the m_i's, if the moduli are composite.

$$X \equiv x_1 \mod m_1 \tag{1}$$

$$X \equiv x_2 \mod m_2 \tag{2}$$

$$\vdots$$

$$X \equiv x_n \mod m_n \tag{n}$$

From (1), $X = m_1 y_1 + x_1$.

Substituting in (2), we obtain

$$m_1 y_1 + x_1 \equiv x_2 \mod m_2$$

or $$m_1 y_1 \equiv (x_2 - x_1) \mod m_2 \tag{1A}$$

The equation (1A) and consequently the pair of equations (1) and (2) is solvable iff $x_2 - x_1 \equiv 0 \mod d$, where $d = \text{g.c.d.}\ (m_1, m_2)$. Then if (1A) has more than one solution, the solutions will differ by multiples of $\frac{m_2}{d}$ [7]. The solutions of (1A) will then be of the form $y_1 + \frac{m_2}{d} t$.

Hence

$$X = m_1 \left(y_1 + \frac{m_2}{d} t\right) + x_1$$

$$= m_1 y_1 + x_1 + \frac{m_1 m_2}{d} t .$$

Thus, the values of X which satisfy both (1) and (2) differ by multiples of $\frac{m_1 m_2}{d}$, which is clearly the least common multiple of m_1 and m_2. Let us suppose X_1 is such a value of X. Then congruences (1) and (2) may be replaced by the single congruence

$$X \equiv X_1 \mod l_1 \tag{2A}$$

where, $l_1 = m_1 \ m_2/d$.

Now (2A) and (3) may be considered simultaneously and if they have a solution, we get another congruence,

$$X \equiv X_2 \mod l_2 \qquad (3A)$$

whose solutions satisfy (2A) and (3) and hence satisfy (1), (2) and (3). l_2 is the l.c.m. of l_1 and m_3 and, therefore, of m_1, m_2 and m_3. This process is continued and existing solutions of the set of congruences are found. Such solutions will differ by multiples of the l.c.m. of the moduli and, therefore, the solution modulo l.c.m. will be unique.

Example 2.7 : Consider moduli 2 and 6. This system can represent 6 integers uniquely, as shown in Table 2.3 .

No.	Mod 2	Mod 6
0	0	0
1	1	1
2	0	2
3	1	3
4	0	4
5	1	5

Table 2.3

Let us take the residue representation (1, 3) and find the integer corresponding to this using the corollary. Let the integer be X. Then

$$X \equiv 1 \mod 2 \qquad (1)$$

$$X \equiv 3 \mod 6 \qquad (2)$$

From (1)

$$X = 2y_1 + 1$$

Putting this in (2)

$$2y_1 + 1 \equiv 3 \mod 6 \qquad (3)$$

or $2y_1 \equiv (3-1) \mod 6$

$$d = \text{g.c.d.}\ (2, 6) = 2$$

and $3 - 1 = 2 \equiv 0 \mod 2$

Therefore, solutions of (3) are of the form $y_1 + \frac{6}{2}t$

$$2y_1 \equiv 2 \mod 6 \text{ implies } y_1 = 1$$

Solutions of (3) are then in the form $(1+3t)$ and

$$X = m_1 y_1 + x_1 + \frac{m_1 m_2}{d} t$$

$$= 2.1 + 1 + \frac{2.6}{2} t$$

$$= 2 + 1 + 6t$$

l.c.m. of 2 and 6 = 6

$$X \equiv 3 \mod 6$$

We can check from Table 2.3 that this is the correct value of X for the given residue representation.

Mixed Radix Conversion Process

A close look at the Chinese Remainder Theorem reveals that it is not a convenient method of obtaining a natural number from a residue number. In a residue computer, addition and multiplication will be performed modulo m_i and not modulo M as required by the theorem. This means additional hardware would be needed for mechanizing this conversion process. An alternative method is to use the Mixed Radix Conversion process which requires operations modulo m_i [2], [3], [5].

The mixed radix representation of interest is of the form

$$X = r_n \prod_{i=1}^{n-1} m_i + \ldots + r_3 m_1 m_2 + r_2 m_1 + r_1, \; 0 \le r_i < m_i ,$$

where r_i are the mixed radix integers which are to be determined. Any integer in the range

0 to $\prod_{i=1}^{n} (m_i) - 1$ may be represented in this form and hence this representation has the same range as a residue system of moduli $m_1, m_2, \ldots, m_n$.

The first digit to be determined is r_1. Clearly

$$r_1 = [X]_{m_1}$$

Hence the least significant mixed radix digit is the same as the first residue digit. To determine r_2, we note that

$$(X - r_1) / m_1 = r_n \prod_{i=2}^{n-1} m_i + \ldots + r_3 m_2 + r_2$$

Hence, $r_2 = [(X - r_1) / m_1]_{m_2}$

The division of $X - r_1$ by m_1 is actually multiplication by the multiplicative inverse I_1 of m_1 with respect to m_2, where

$$[I_1 m_1]_{m_2} = 1$$

By successively subtracting r_i and dividing by m_i all the r_i can be determined.

Example 2.8 :

Let $m_1 = 4$, $m_2 = 5$, $m_3 = 7$, and consider a residue number (3, 2, 5) to be converted.

Moduli	4	5	7	Mixed radix digits
Residue Number	3	2	5	$r_1 = 3$
Subtract $[3]_{m_i}$	3	3	3	
	0	4	2	
Multiply by $[\frac{1}{4}]_{m_i}$		$\odot$ 4	2	
		1	4	$r_2 = 1$
Subtract $[1]_{m_i}$		1	1	
		0	3	
Multiply by $[\frac{1}{5}]_{m_i}$			$\odot$ 3	
			2	$r_3 = 2$

X is given by

$$X = r_3 m_1 m_2 + r_2 m_1 + r_1$$

$$= 2 . 20 + 1 . 4 + 3$$

$$= 40 + 4 + 3 = 47.$$

We can check that in the given system, residue representation for 47 is (3,2,5).

This chapter has provided the fundamental properties of residue number systems. In the following chapter we will consider the problem of sign determination.

3. SIGN DETERMINATION

The problem of sign determination is one of the major problems encountered in the design of a computer based on residue arithmetic. Attaching a sign bit to a residue number does not help. This is because, as pointed out in Chapter 2, the magnitude of a residue number is not readily available and, therefore, after adding a positive and a negative number, the sign of the result is not immediately known. We have seen in Chapter 2 how the whole range of representation in a residue system is divided into two parts to represent positive and negative integers. One obvious way, therefore, is to convert a given residue number to its natural number form which will fall either in the positive or the negative region of the representation. This process will definitely establish the sign of the given residue number. However, this is not an attractive solution for the problem because it is slow and, therefore, offsets the advantage of speed in a residue computer. Thus, a faster method is required.

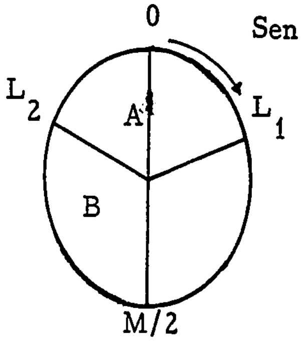
The sign determination problem deserves a thorough study also because it is closely related to the problems of overflow detection and relative magnitude determination.

We consider a sign function S which is 0 for positive integers and 1 for negative integers.

In considering the general sign determination problem, it might be expected that all the residue information is not needed just to determine sign (one bit of information), We might expect that for a particular set of moduli, we only need part of the information, such as the parity of each residue digit. Szabo [5] has proved that such a scheme is impossible, and in the general case no reduction of information from any residue digit is possible without loss of sign information. The statement of his theorem follows.

3.1 Coding Theorem

Let two sectors in the modular ring of M numbers be designated by two end points L_1 and L_2. Call the sector which is traversed when proceeding from L_2 in the sense of increasing numbers, Sector A, and the remaining sector, Sector B (see Fig. 3.1)



Sense of increasing numbers.

Fig. 3.1. Range of Selected Number System

Furthermore, let $L_1 \in B$ and $L_2 \in A$. Select a modulus m_p and construct a function $g([X]_{m_p})$ such that $g([X]_{m_p})$ may take on r values, $r < m_p$. Now consider a function:

$$f\left([X]_{m_1}, \ldots [X]_{m_{p-1}},\ g([X]_{m_p}), [X]_{m_{p+1}}, \ldots, [X]_{m_n}\right)$$

That is, f is a function of all $[X]_{m_i}$, $i \neq p$, and furthermore is a function of $g([X]_{m_p})$.

Then regardless of the choice of f and g, there exist at least two numbers X_1 and X_2, $0 \leq X_1 < M$, $0 \leq X_2 < M$, such that $f(X_1) = f(X_2)$ and $X_1 \in A$ and $X_2 \in B$, provided the following conditions are satisfied:

(i) $$[L_1 - L_2]_M > m_p$$
$$[L_2 - L_1]_M > m_p$$

(ii) $$\hat{m}_p > m_p, \text{ where } \hat{m}_p = \frac{M}{m_p}$$

<u>Explanation of Terminology :</u> The partitioning of numbers into positive and negative sets is accomplished by designating two end points L_1 and L_2. Condition (i) imposes very broad limitations on how this partitioning is achieved. The two sectors defined by the end points are labeled as A and B. Designating either one of them as positive does not alter the validity of the theorem. $g([X]_{m_p})$ is any arbitrary mapping which maps m_p points (values of $[X]_{m_p}$) into less than m_p points, thus reducing

the information of the p th residue digit. $f(X)$ is the sign function. It maps all positive numbers into one set of points and all negative numbers into a disjoint set of points. In the proof the author [5] shows that the restrictions imposed on functions f and g are incompatible.

In other words the theorem proves that all the information from a residue digit must be used in any sign determination process, provided the modulus m_p of the digit is smaller than $\sqrt{M}$. This is seen easily since it is required that

$$\hat{m}_p > m_p \quad \text{or} \quad \frac{M}{m_p} > m_p \quad \text{or} \quad m_p < \sqrt{M} .$$

The following corollary treats the case when $\hat{m}_p < m_p$. It states that sign detection is impossible if the p th residue digit is coded into less than $\hat{m}_p$ states.

Corollary : Let $f(X)$, $g(X)$, L_1, L_2, A and B be defined as in the previous theorem. Furthermore, let $g(X)$ take on r different values where r is now smaller than $\hat{m}_p$. Then regardless of the choice of f and g, there exist at least two numbers X_1 and X_2, $0 \le X_1 < M$, $0 \le X_2 < M$ such that $f(X_1) = f(X_2)$ and $X_1 \in A$ and $X_2 \in B$ provided the following conditions are satisfied :

(i) $$[L_1 - L_2]_M > \hat{m}_p$$
$$[L_2 - L_1]_M > \hat{m}_p$$

(ii) $$\hat{m}_p < m_p$$

The corollary yields a positive result. It shows that it is possible to reduce the information from a residue digit but only within a certain limit. This limit is fixed by the modulus whose information is to be reduced and by the other moduli of the system. For example, if we consider a system with moduli m_1 and m_2, $m_1 < m_2$, then m_1 is the lower limit on the reduction of information from m_2 since

$\hat{m}_2 = \frac{M}{m_2} = \frac{m_1 m_2}{m_2} = m_1$ and the corollary states that we cannot reduce the information from m_2 to less than $\hat{m}_2 = m_1$ states. In the approach presented in this thesis, we reduce the information from some residue digits but not beyond the limit imposed by the corollary.

3.2 The Two Moduli Case. We shall now consider the case of two moduli and propose a solution for the sign determination problem. Later we shall consider the case when the number of moduli is greater than two.

Theorem 3.1 : Given two moduli 2M and N, N odd and $N > 2M$, let $\ell = [N]_{2M}$, the least positive residue of N modulo 2M. Then the sign of a number X, $0 \leq X \leq 2MN - 1$, is established by a proposition P such that

$$S = 0 \text{ iff } P : \bigvee_{i=0}^{M-1} (X_o \equiv X_m + i\ell) \bmod 2M \text{ is true.}$$

Explanation of Terminology :

$X_o = [X]_{2M}$, the least positive residue of X modulo 2M.

$X_m = [[X]_N]_{2M}$, the least positive residue of $[X]_N$ modulo 2M, $[X]_N$ being the least positive residue of X modulo N.

The symbol "V" denotes a logical "OR". The theorem states that for a residue number $([X]_{2M}, [X]_N)$ in the above system of moduli, if $[X]_{2M}$ and $[[X]_N]_{2M}$ are same or differ by a multiple of ℓ, then the number is positive. The multiplying factor i of ℓ can vary from 0 to M-1. If the proposition P is not true for any value of i i.e., if X_o and $X_m + i\ell$ do not compare for all values of i then the number is negative.

Proof : The moduli are obviously relatively prime, since one modulus is even and the other one odd. Hence, they can represent 2 MN integers, 0 to 2 MN - 1, uniquely. Without any loss of generality, integers in the range 0 to MN - 1 will be considered to be positive and those in the range MN to 2MN - 1 to be negative.

Consider an integer $X = KN + Y$, $0 \leq Y \leq N - 1$, $0 \leq K \leq 2M - 1$.

By definition $[X]_{2M} = X_o = [KN+Y]_{2M}$

$\ell = [N]_{2M}$

or $N \equiv \ell \mod 2M$

or $KN \equiv K \cdot \ell \mod 2M$

or $KN = 2ML_1 + K\ell, \; L_1 = 1, 2, \ldots$

or $KN + Y = 2ML_1 + K\ell + Y$ (1)

Now $[KN+Y]_N = Y$

By definition, $[[X]_N]_{2M} = [[KN+Y]_N]_{2M} = X_m$

Therefore, $[KN+Y]_N = 2ML_2 + X_m, \; L_2 = 0, 1, 2, \ldots$

or $Y = 2ML_2 + X_m$

Substituting this value of Y in (1)

$$KN + Y = 2ML_1 + K\ell + 2ML_2 + X_m$$

or $[KN+Y]_{2M} = [X_m + K\ell]_{2M}$

or $X_o \equiv (X_m + K\ell) \mod 2M$, for any value of K.

For X to lie in the positive region, the maximum value K can take is (M - 1). This is shown as foldows.

Consider $X = (M-1)N + Y$. The maximum value Y can take is $N - 1$. For this value of Y, $X = (M-1)N + N - 1 = MN - 1$, which is the limit of the positive region. Therefore, the proposition P is true for all values of K between 0 and M - 1, and S = 0 iff P : $(X_o \equiv X_m) \vee (X_o \equiv X_m + \ell) \ldots \vee (X_o \equiv X_m + (M-1)\ell) \mod 2M$ is true. Or

$$P: \bigvee_{i=0}^{M-1} (X_o \equiv X_m + i\ell) \mod 2M \text{ is true.}$$

To complete the proof we have to show that P is false in the negative region. For values of K between M and (2M - 1), $X = KN + Y$ lies in the negative range. Therefore in this range, S = 1 iff proposition

$$P': \bigvee_{j=M}^{2M-1} (X_o \equiv X_m + j\ell) \bmod 2M \text{ is true or } P \text{ is false.}$$

This completes the proof.

Corollary 3.1 Given two moduli $(2, N)$, $N > 2$ and N odd, then for any X, $0 \le X \le 2N - 1$

$$X_o \equiv X_m \bmod 2 \text{ in the positive region}$$

$$X_o \equiv (X_m + 1) \bmod 2 \text{ in the negative region.}$$

where $X_o = [X]_2$, and $X_m = [[X]_N]_2$ is least significant bit of $[X]_N$ if binary coding is employed. This follows directly from Theorem 3.1. This means that in the positive region, X_o and X_m are equal. Therefore, their mod 2 or "EXCLUSIVE OR" sum must be zero. In the negative region, they differ by 1 or their mod 2 sum is always 1. Therefore, $S = X_o \oplus X_m$, where $\oplus$ here denotes the "EXCLUSIVE OR" sum. If the above sum is zero, the residue number $([X]_2, [X]_N)$ is positive and if the sum is 1, the number is negative.

This result is significant in the sense that if binary coding is employed to represent the residues in this system, we only need the least significant bits of the two residues to determine the sign.

The following two examples will illustrate the use of the theorem and its corollary for sign determination.

Example 3.1.a: Consider a system with moduli 4 and 5. Here $2M = 4$ or $M = 2$ and $N = 5$. $\ell = [5]_4 = 1$. This system can represent twenty integers uniquely, as shown in Table 3.1.a.

In using Theorem 3.1, here i varies from 0 to 1. Therefore, a residue number $([X]_4, [X]_5)$ is positive iff,

$$[X_o - X_m]_4 = 0 \text{ or } 1.$$

Again, $([X]_4, [X]_5)$ represents a negative integer iff $[X_o - X_m]_4 = 2$ or 3.

This is confirmed from Table 3.1.a.

	Integers X	$X_o = [X]_4$	$[X]_5$	$[[X]_5]_4 = X_m$	$(X_o - X_m)$ Mod 4	S
Positive Region	0	0	0	0	0	0
	1	1	1	1	0	0
	2	2	2	2	0	0
	3	3	3	3	0	0
	4	0	4	0	0	0
	5	1	0	0	1	0
	6	2	1	1	1	0
	7	3	2	2	1	0
	8	0	3	3	1	0
	9	1	4	0	1	0
Negative Region	-10, 10	2	0	0	2	1
	-9, 11	3	1	1	2	1
	-8, 12	0	2	2	2	1
	-7, 13	1	3	3	2	1
	-6, 14	2	4	0	2	1
	-5, 15	3	0	0	3	1
	-4, 16	0	1	1	3	1
	-3, 17	1	2	2	3	1
	-2, 18	2	3	3	3	1
	-1, 19	3	4	0	3	1

Table 3.1.a

Example 3.1.b : Consider a system with moduli 2 and 5. This can represent 10 integers, as shown in Table 3.1.b.

	Integer X	$X_o = [X]_2$	$[X]_5$	$X_m = [[X]_5]_2$	$[X_o \oplus X_m]_2$	S
Positive Region	0	0	0	0	0	0
	1	1	1	1	0	0
	2	0	2	0	0	0
	3	1	3	1	0	0
	4	0	4	0	0	0
Negative Region	-5, 5	1	0	0	1	1
	-4, 6	0	1	1	1	1
	-3, 7	1	2	0	1	1
	-2, 8	0	3	1	1	1
	-1, 9	1	4	0	1	1

Table 3.1.b

By corollary 3.1.b, a residue number $([X]_2, [X]_5)$ is positive iff, $[X_o \oplus X_m]_2 = 0$ and negative iff $[X_o \oplus X_m]_2 = 1$. Table 3.1.b confirms this.

If we use binary coding for the residues in this system, we can schematically represent the sign determination scheme as shown in Fig. 3.2.

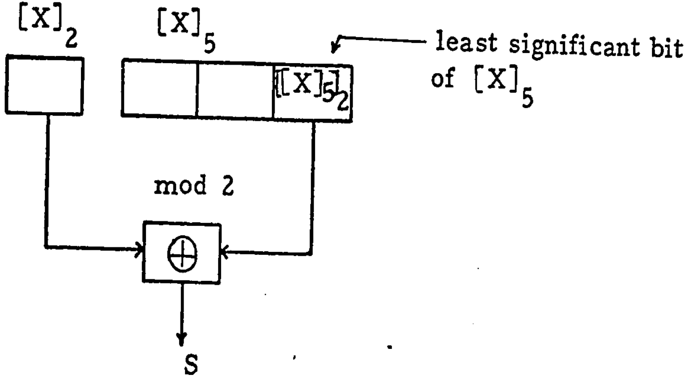


Fig. 3.2.

It should be clear that in using Theorem 3.1 we are mapping a set of N points 0,1,...., N-1 into a set of 2M points, 2M < N. Thus we are reducing the residue information for residues modulo N but not beyond the limit imposed by the corollary of the coding theorem. If we put $m_p = N$, then

$$\hat{m}_p \doteq \frac{2MN}{m_p} = \frac{2MN}{N} = 2M < N \quad \text{or} \quad \hat{m}_p < m_p.$$

Also we have divided the range of representation in such a way that $[L_1 - L_2]_{2MN} = [L_2 - L_1]_{2MN} = MN$.

We can show that $MN > \hat{m}_p$ or $MN > 2M$ since $N > 2M$ or $MN > 2M$. Therefore, Theorem 3.1 satisfies the conditions imposed by the corollary of the coding theorem.

3.3 Sign Determination ; A General Approach

The problem of sign determination becomes more complicated and time consuming when the number of moduli is large. The basic philosophy still remains unchanged and we shall make use of Theorem 3.1 and its corollary for determining sign. A given system of moduli is reduced to the 2-moduli form, and then the sign can be established.

Let us consider a system of n mutually prime moduli $m_1, m_2, \ldots m_n$. There is not much loss of generality if we assume one of the moduli to be even. Let $m_1 = 2m$. It is obvious that all other moduli must be odd. We shall partition the set of moduli into two groups $(m_1, \ldots, m_j)$ and $(m_{j+1}, \ldots m_n)$ where

$$\prod_{i=1}^{j} m_i = 2m \left(\prod_{i=2}^{j} m_i \right) < \prod_{i=j+1}^{n} m_i.$$

and $m_{j+1} < m_{j+2} < \ldots < m_n$.

There can be several such partitions available and some consequences of the choice will be discussed later.

Let $\prod_{i=1}^{n} m_i = 2K$ and let all X, $0 \le X \le K-1$ represent positive intgers, and all X', $K \leq X' \leq 2K-1$ represent the corresponding negative integers such that $X \oplus X' \equiv 0 \bmod 2K$.

Let $(x_1, x_2, \ldots, x_n)$ be the residue representation of an integer X, where $x_i = [X]_{m_i}$, $i = 1, 2, \ldots, n$.

Let $2m\left(\prod_{i=2}^{j} m_i\right) = 2M$ and $\prod_{i=j+1}^{n} m_i = N$. We shall find $[X]_{2M}$ and $[[X]_N]_{2M}$ and represent them in mod $m_1, m_2, \ldots, m_j$, since we do not have any mod 2M arithmetic unit i.e., we shall find $[[[X]_N]_{2M}]_{m_i}$ and $[[X]_{2M}]_{m_i}$.

In doing so we have to use the following theorem.

<u>Theorem 3.2 :</u> If $(x_1, x_2, \ldots, x_j)$ represents an integers X in a system of mutually prime moduli $m_1, \ldots, m_j$ such that

$$x_i = [X]_{m_i}, \quad i = 1, \ldots j$$

then $$[[X]_{m_1 \cdot m_2 \cdot \ldots \cdot m_j}]_{m_i} = [X]_{m_i} = x_i$$

<u>Proof :</u> $$x_i = [X]_{m_i}$$

or $$X = y_i m_i + x_i, \quad y_i \in J$$

Let $$A = [X]_{m_1 \cdot m_2 \cdot \ldots \cdot m_j}$$

then $$X = Y \cdot m_1 \cdot m_2 \cdot \ldots \cdot m_j + A, \quad Y \in J.$$

or $$y_i m_i + x_i = Y \cdot m_1 \cdot \cdot m_i \ldots \cdot m_j + A$$

or $$[A]_{m_i} = x_i$$

or $$[[X]_{m_1 \cdot m_2 \cdot \ldots m_j}]_{m_i} = x_i = [X]_{m_i}$$

Therefore, $[[[X]_N]_{2M}]_{m_i} = [[X]_N]_{m_i}$, since $2M = m_1 \cdot \ldots \cdot m_i \cdot \ldots \cdot m_j$.

To determine $[X]_N$, we can use the mixed radix conversion process for the set of moduli $m_{j+1}, \ldots m_n$.

$$[X]_N = r_n m_{j+1} \cdot \ldots \cdot m_{n-1} + \ldots + r_{j+2} m_{j+1} + r_{j+1}$$

$$0 \le r_k < m_k, \; k = j+1, \ldots, n .$$

The mixed radix digits r_k can be determined from the residue digits as discussed in Chapter 2.

$$[[X]_N]_{m_i} = [r_n m_{j+1} \cdot \ldots \cdot m_{n-1} \oplus \ldots \oplus r_{j+2} m_{j+1} \oplus r_{j+1}]_{m_i}$$

Since $[A \oplus B]_m = [[A]_m \oplus [B]_m]_m$

$$[[X]_N]_{m_i} = [[r_n m_{j+1} \cdot \ldots \cdot m_{n-1}]_{m_i} \oplus \ldots \oplus [r_{j+2} m_{j+1}]_{m_i} \oplus [r_{j+1}]_{m_i}]_{m_i}$$

And since $[A \odot B]_m = [[A]_m \odot [B]_m]_m$

$$[[X]_N]_{m_i} = [[[r_n]_{m_i} \odot [m_{j+1}]_m \odot \ldots [m_{n-1}]_{m_i}]_{m_i} \oplus \ldots \oplus [r_{j+1}]_{m_i}]_{m_i}$$

Once the choice of the moduli has been decided, the fixed quantities in the above relation are known and can be stored. All we have to do is to compute the quantities $[r_k]_{m_i}$, $i = 1, \ldots j$; $k = j+1, \ldots n$. Once we have computed $[[X]_N]_{m_i}$, we compare this with $[[X]_{2M}]_{m_i}$. This can be done as each $[[X]_N]_{m_i}$ is computed. If all of $[[X]_N]_{m_i}$ and $[[X]_{2M}]_{m_i}$ compare, that is they are equal, then the number $(x_1, x_2, \ldots, x_n)$ is positive. If they do not compare, we add $\ell = [N]_{2M}$ to $[[X]_N]_{m_i}$ and again compare with $[[X]_{2M}]_{m_i}$. In the worst case $(M - 1)$ additions of ℓ will be necessary before we can say the number is not positive. A schematic flow chart for this method is shown in Fig. 3.3.

Example 3.2 : Consider the set of moduli 2, 3, 5, 7 and 11. Let us partition the set such that $2M = 2.3 = 6$ and $N = 5.7.11 = 385$. Let us consider a residue number $X = (0, 1, 0, 1, 4)$.

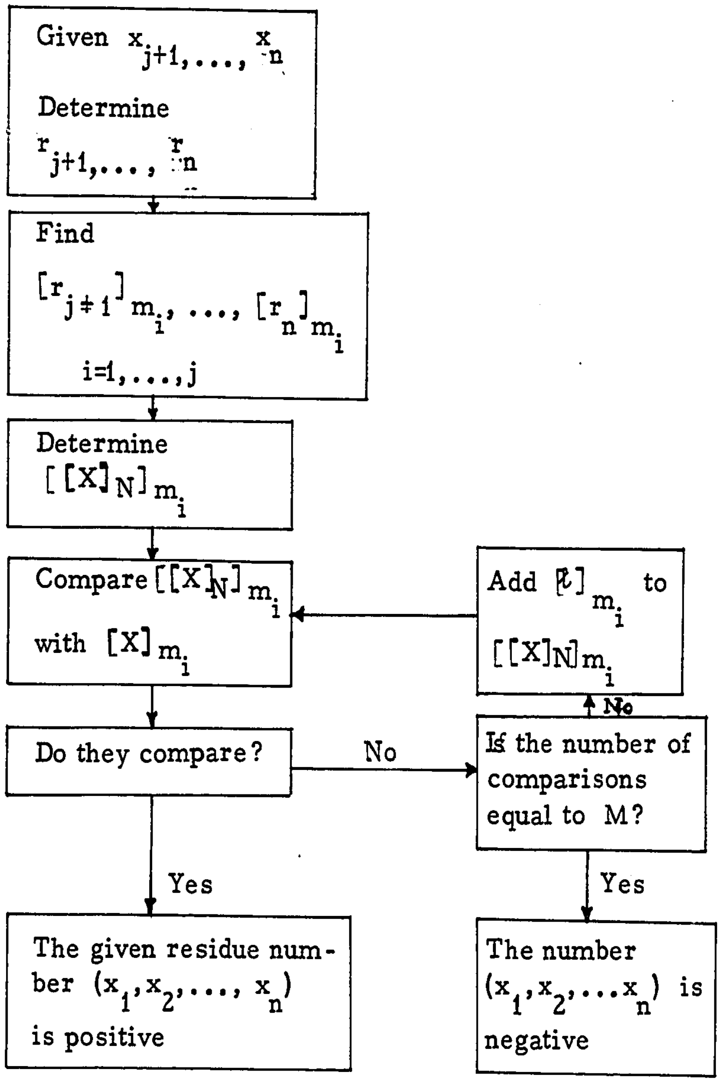

Given $x_{j+1}, \ldots, x_n$
Determine $r_{j+1}, \ldots, r_n$
Find $[r_{j+1}]_{m_i}, \ldots, [r_n]_{m_i}$ $i=1, \ldots, j$
Determine $[[X]_N]_{m_i}$
Compare $[[X]_N]_{m_i}$ with $[X]_{m_i}$
Add $[\ell]_{m_i}$ to $[[X]_N]_{m_i}$
Do they compare?
No
Is the number of comparisons equal to M?
No
Yes
Yes
The given residue number $(x_1, x_2, \ldots, x_n)$ is positive
The number $(x_1, x_2, \ldots x_n)$ is negative

Fig. 3.3 A General Flow Chart For Sign Determination

i varies from 0 to M-1 or from 0 to 2.

By Theorem 3.2 $[X]_6$ represented in mod 2 and mod 3 is given by

$$[[X]_6]_2 = [X]_2 = 0$$

$$[[X]_6]_3 = [X]_3 = 1$$

Now, we shall find $[[X]_{385}]_2$ and $[[X]_{385}]_3$, which is nothing but $[[X]_{385}]_6$ represented in mod 2 and mod 3. To find $[X]_{385}$ we have to use the mixed radix conversion process for moduli 5, 7 and 11.

	Mixed Radix Conversion			Mixed Radix Digits
Moduli	5	7	11	
Residues	0	1	4	$r_3 = 0$
Subtract $[r_3]_{m_i}$	0	0	0	
	0	1	4	
Multiply by $[\frac{1}{5}]_{m_i}$		$\odot$ 3	9	
		3	3	$r_4 = 3$
Subtract $[r_4]_{m_i}$		3	3	
		0	0	
Multiply by $[\frac{1}{7}]_{m_i}$			$\odot$ 8	
			0	$r_5 = 0$

$$[X]_{385} = r_5 \cdot m_3 \cdot m_4 + r_4 \cdot m_3 + r_3$$

$$= r_5 \cdot 35 + r_4 \cdot 5 + r_3$$

Therefore

$$[[X]_{385}]_2 = [[[r_5]_2 \odot [35]_2]_2 \oplus [[r_4]_2 \odot [5]_2]_2 \oplus [r_3]_2]_2$$

$$= [[0 \odot 1]_2 \oplus [1 \odot 1]_2 \oplus 0]_2$$

$$= 1$$

$$[[X]_{385}]_3 = [[[r_5]_3 \odot [35]_3]_3 \oplus [[r_4]_3 \odot [5]_3]_3 \oplus [r_3]_3]_3$$

$$= [[0 \odot 2]_3 \oplus [0 \odot 2]_3 \oplus 0]_3 = 0$$

In mod 2 and mod 3

$$[X]_6 = (0, 1)$$

$$[X]_{385} = (1, 0)$$

They do not compare. So we add ℓ to $[X]_{385}$.

$\ell = [385]_6 = 1 = (1, 1)$ in mod 2, mod 3

$$[X]_{385} + \ell \quad = \quad \begin{array}{r} (1, 0) \\ \oplus \; (1, 1) \\ \hline (0, 1) \end{array}$$

This compares with $[X]_6$. Hence the number $(0, 1, 0, 1, 4)$ is positive.

Consider another residue number $(0, 0, 0, 3, 1)$.

We can check that

$$[[X]_{385}]_2 = 1$$

$$[[X]_{385}]_3 = 0$$

In mod 2 and mod 3

$$[X]_6 = (0, 0)$$

$$[X]_{385} = (1, 0)$$

They do not compare. Hence we add ℓ to $[X]_{385}$; $[X]_{385} + \ell = (0, 1)$.

It still does not compare with $[X]_6$. Adding ℓ again

$$[X]_{385} + 2\ell = (1, 2)$$

This does not compare with $[X]_6$. We have made M = 3 comparisons. Hence the number (0,0,0,3,1) is negative. Fig. 3.4 shows schematically how to find $[[X]_{385}]_2$ and $[[X]_{385}]_3$. Boxes labelled C_7 and C_{11} are complement units modulo 7 and 11 respectively. L_1, L_2 and L_3 represent simple logic required to convert r_3, r_4 and r_5 to $[r_3]_3$, $[r_4]_3$ and $[r_5]_3$ respectively.

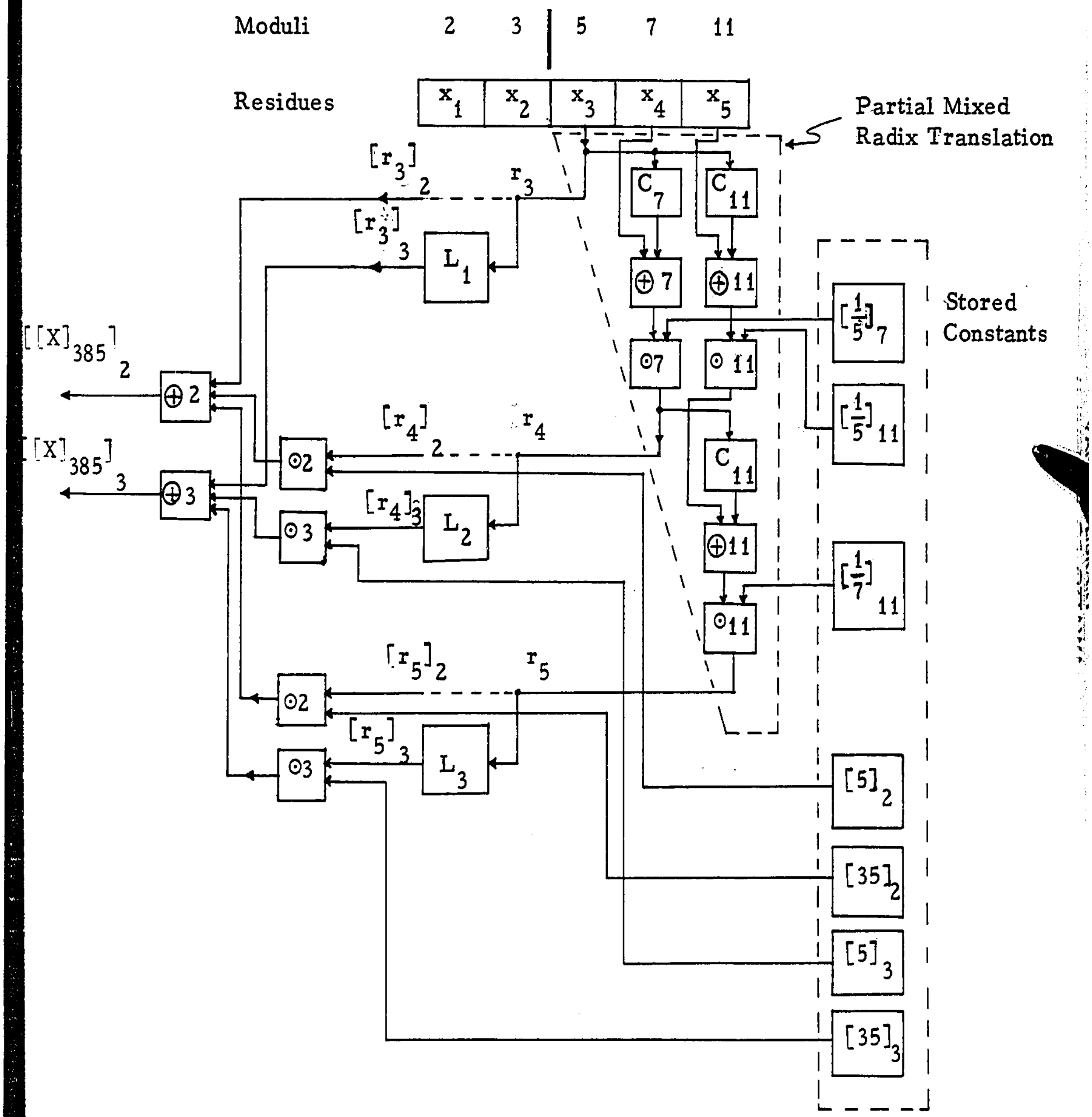


Fig. 3.4.

After $[[X]_{385}]_2$ and $[[X]_{385}]_3$ have been determined, they are compared with $x_1 = [X]_2$ and $x_2 = [X]_3$ respectively. The comparison logic is straightforward, and is shown in Figs. 3.5. It produces an output of 1 only when its inputs are identical.

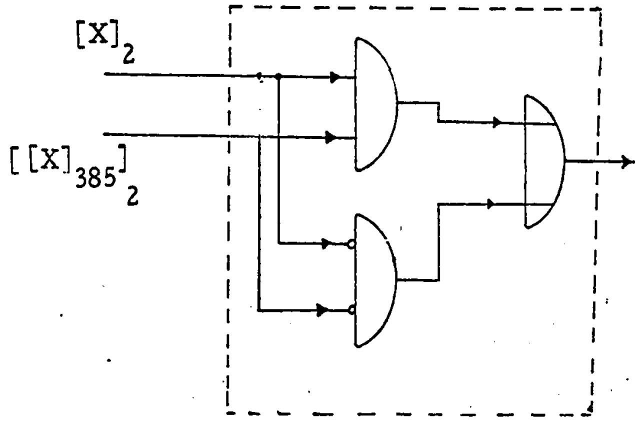


Fig. 3.5.a. Mod 2 Comparator

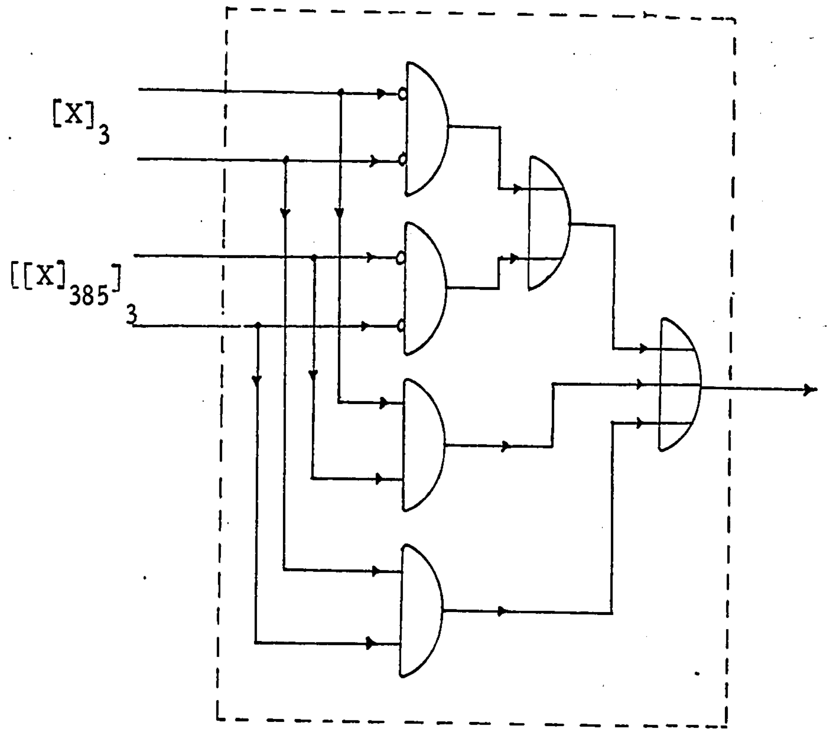


Fig. 3.5.b Mod 3 Comparator

When both comparators have an output of 1, the sign of the residue number $(x_1, x_2, x_3, x_4, x_5)$ is positive. If not, then the quantity i ℓ (i = 0, 1, 2; $\ell = 1$) must be added to $[[X]_{385}]_2$ and $[[X]_{385}]_3$ and this should again be compared with $[X]_2$ and $[X]_3$.

Let $y_1 = [[X]_{385}]_2$, $y_2 = [[X]_{385}]_3$, $y_1^* = [[X]_{385}]_2 + [\ell]_2]_2$ $y_2^* = [[X]_{385}]_3 + [\ell]_3]_3$, $y_1^{**} = [[X]_{385}]_2 + 2\,[\ell]_2]_2$ and $y_2^{**} = [[X]_{385}]_3 + 2\,[\ell]_3]_3$. Table 3.2 shows the combinations of y_1 and y_2 and the resulting combinations of y_1^*, y_2^* and y_1^{**}, y_2^{**}.

i = 0		i = 1		i = 2	
y_1	y_2	y_1^*	y_2^*	y_1^{**}	y_2^{**}
0	0	1	1	0	2
1	1	0	2	1	0
0	2	1	0	0	1
1	0	0	1	1	2
0	1	1	2	0	0
1	2	0	0	1	1

Table 3.2

It is clear that $y_1^* = \bar{y}_1$, $y_2^* = (y_2 \oplus 1) \bmod 3$, $y_1^{**} = y_1$ and $y_2^{**} = (y_2 \oplus 2) \bmod 3$. Fig. 3.6 shows the remaining part of the sign detection process after y_1 and y_2 have been determined. The stored constants 1 and 2 are added to y_2 in mod 3 adders to find y_2^* and y_2^{**} respectively.

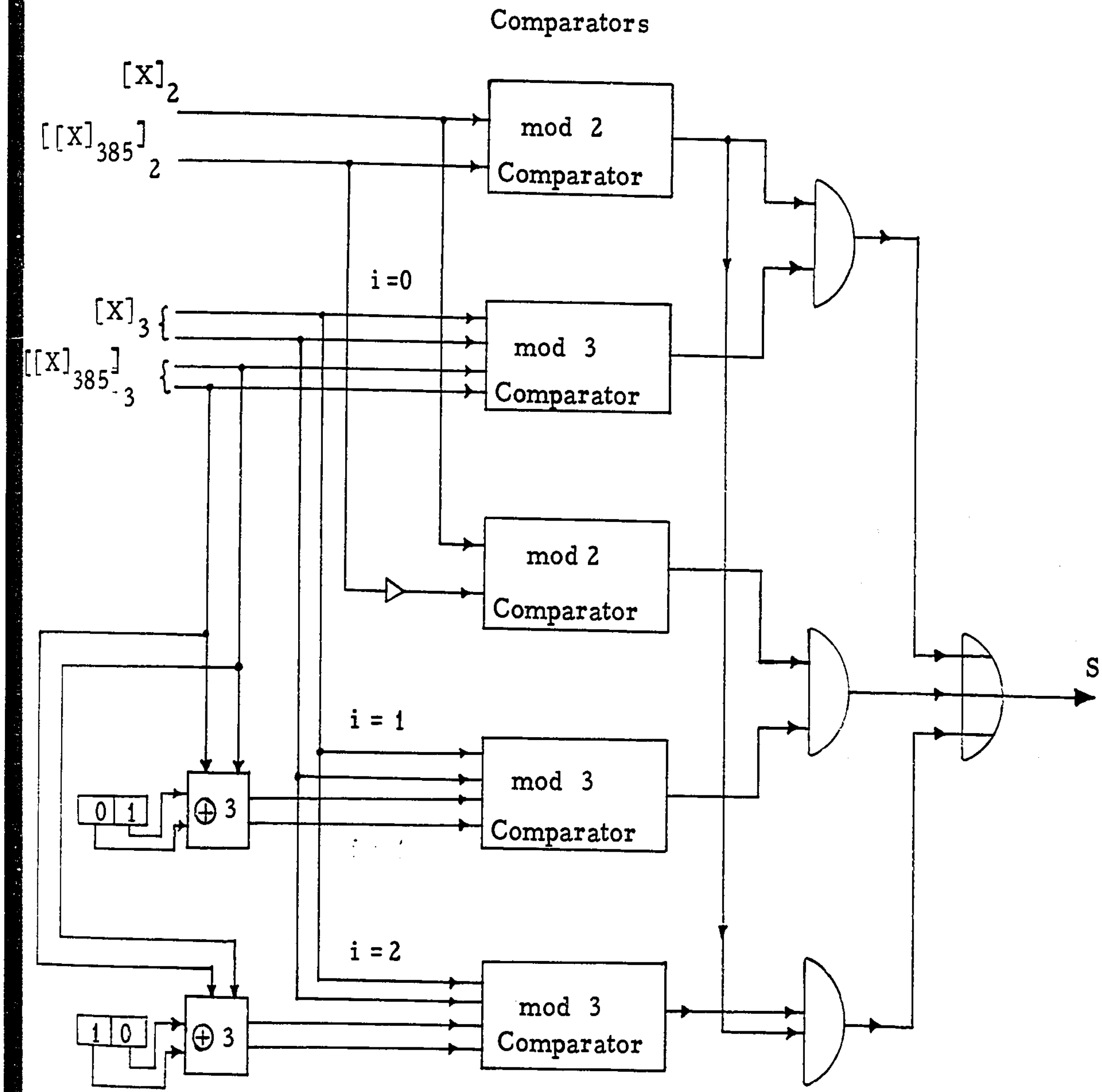


Fig. 3.6. Comparison Process for Sign Detection

The mixed radix translation can be reduced by partitioning the set of moduli into 30 (2, 3 and 5) and 77 (7 and 11) or into 42 (2, 3 and 7) and 55 (5 and 11). This would require one complementation, one addition and one multiplication, all modulo 11. Thus a reduction in hardware for translation process is possible. But then more hardware is required in the remaining sign determination process, after mixed radix digits r_4 and r_5 are determined. On the other hand, partitioning the set of moduli into 2 and 1155 (3, 5, 7 and 11) would require more hardware for mixed radix translation. The remaining process of sign detection requires only an EXCLUSIVE OR gate, to which $[X]_2$ and the parity bits of mixed radix digits are fed [See Sec. 3.4]. A compromise between mixed radix translation and the remaining sign detection process has to be finally made by the designer.

The sign detection scheme discussed so far uses a completely hardware approach. Of course, it is not necessary to determine sign by hardware alone. Using a combination of software and hardware will result in saving hardware. At the same time the process would require more time. Referring to Fig. 3.4, all the modular operations can be performed using the arithmetic unit of the machine, which will have modulo adders, multipliers and complement units. Thus no extra hardware would be needed for mixed radix translation. After the mixed radix digits are obtained, the remaining process requiring mod 2 and mod 3 adders and multipliers can be performed using the arithmetic unit. Comparators will still be needed but, by suitable programming, additions of $i\ell$ can be done using the arithmetic unit again. Thus the process allows the designer a lot of flexibility on the amount of hardware and software to be used in sign detection.

3.4. It is interesting to see what happens when in a system of n mutually prime moduli, the even modulus is 2. Then the following theorem can be used to establish the sign of a residue number.

Theorem 3.3 In a system consisting of n mutually prime moduli, m_1, $m_2, \ldots m_n$ where one of the moduli is 2, the sign of a number X with residue representation $(x_1, \ldots, x_n)$ is given by the EXCLUSIVE OR sum of x_1 and the least significant bits of the mixed digits corresponding to the residue representation $(x_2, \ldots x_n)$.

Proof : The proof follows directly from Corollary 3.1. Let $m_1 = 2$ and let $N = \prod_{i=2}^{n} m_i$. If we just had two moduli $m_1 = 2$ and the composite modulus N, then we could use Corollary 3.1 for sign determination. We can still use it because N consists of (n - 1) "component moduli". $[X]_N$ is given by

$$[X]_N = r_n m_2 \cdot m_3 \cdot \ldots \cdot m_{n-1} + \ldots + r_3 m_2 + r_2,$$

r_i's being the mixed radix digits corresponding to the residue representation $(x_2, \ldots, x_n)$.

Hence $X_m = [[X]_N]_2 = [[[r_n]_2 \odot [m_2]_2 \odot \ldots \odot [m_{n-1}]_2]_2 \oplus \ldots$

$$\oplus [[r_3]_2 \odot [m_2]_2]_2 \oplus [r_2]_2]_2 .$$

Since moduli $m_2, \ldots, m_n$ are all odd by assumption

$$[m_i]_2 = 1, \text{ for all } i = 2, \ldots, n .$$

Hence $X_m = [[r_n]_2 \oplus \ldots \oplus [r_3]_2 \oplus [r_2]_2]_2$

Also $X_o = [X]_2 = [X]_{m_1} = x_1$

Again, from Corollary 3.1, the sign function is given by $S = X_o \oplus X_m$, $\oplus$ here denotes the EXCLUSIVE OR sum. Therefore,

$$S = x_1 \oplus [r_2]_2 \oplus [r_3]_2 \oplus \ldots \oplus [r_n]_2.$$

This completes the proof.

If binary coding is used for all the modular operations, the mixed radix digits $r_2, \ldots, r_n$ will be in binary form and $[r_2]_2, \ldots, [r_n]_2$ will of x_1 and the least significant bits of the mixed digits corresponding to the residue representation $(x_2, \ldots x_n)$.

<u>Proof :</u> The proof follows directly from Corollary 3.1. Let $m_1 = 2$ and let $N = \prod_{i=2}^{n} m_i$. If we just had two moduli $m_1 = 2$ and the composite modulus N, then we could use Corollary 3.1 for sign determination. We can still use it because N consists of $(n - 1)$ " component moduli" . $[X]_N$ is given by

$$[X]_N = r_n m_2 \cdot m_3 \cdot \ldots \cdot m_{n-1} + \ldots + r_3 m_2 + r_2,$$

r_i's being the mixed radix digits corresponding to the residue representation $(x_2, \ldots, x_n)$.

Hence $X_m = [[X]_N]_2 = [[[r_n]_2 \odot [m_2]_2 \odot \ldots \odot [m_{n-1}]_2]_2 \oplus \ldots$

$$\oplus [[r_3]_2 \odot [m_2]_2]_2 \oplus [r_2]_2]_2 \ .$$

Since moduli $m_2, \ldots, m_n$ are all odd by assumption

$$[m_i]_2 = 1, \text{ for all } i = 2, \ldots, n \ .$$

Hence $X_m = [[r_n]_2 \oplus \ldots \oplus [r_3]_2 \oplus [r_2]_2]_2$

Also $X_o = [X]_2 = [X]_{m_1} = x_1$

Again, from Corollary 3.1, the sign function is given by $S = X_o \oplus X_m$, $\oplus$ here denotes the EXCLUSIVE OR sum. Therefore,

$$S = x_1 \oplus [r_2]_2 \oplus [r_3]_2 \oplus \ldots \oplus [r_n]_2.$$

This completes the proof.

If binary coding is used for all the modular operations, the mixed radix digits $r_2, \ldots, r_n$ will be in binary form and $[r_2]_2, \ldots, [r_n]_2$ will

4. IMPLEMENTATION STUDIES

In this chapter we shall discuss the modular adder and multiplier design, the complementation problem and the input-output translation..

In most of the available literature on modular arithmetic, there is a general trend to use magnetic core logic for the hardware implementation of various units. While this implementation scheme is conceptually simple, it does not take into account hardware considerations. In Ref. [10], it is pointed out that :

1. A significant amount of hardware is required for decoding binary coded residues modulo m to a code for input to the core matrix, and separate decoding is required for both operands.
2. A logic network is required for encoding the result to a binary coded residue for storing in memory.
3. 2 m core drivers and m sense amplifiers are required for a mod m adder, and similarly for a mod m multiplier.
4. The speed of a core matrix adder or multiplier is limited to approximately the cycle time of a core memory made from the same cores. A reasonable upper limit is a 1-microsecond cycle time as against 1/10 microsecond for semiconductors.

Because of the above limitations and disadvantages of core matrix mechanizations, we shall use logic gates in our designs.

4.1 Adder and Multiplier Design

In this section we consider the logical design of residue adders and multipliers. There are two approaches to the design problem.

4.1.1 Direct Implimentation

This is a straight forward method of design. For example, to mechanize an adder for modulus m, the sum mod m table is formed in binary code to obtain the Boolean functions for sum digits. Any of the standard minimization techniques are then used to reduce the sum functions. The same procedure is applied for finding the product digits of a mod m multiplier. Ref. [10] reports that using this technique a mod 31 adder requires several hundred logic gates, a rough estimate being 600 gates using two-level logic. The reason for such complexity of the mod 31 adder is that the five sum functions mod 31 are each functions of ten variables. It might seem likely that the sum functions modulo 32 would also be functions of ten variables. However, this is not so and it has been shown that a mod 32 adder can be implemented using a total of 68 AND and OR gates and 10 inverters. Such simplicity arises because instead of five functions each of ten variables, we have one function of ten, one of eight, one of six, one of four and one of two Boolean variables. It has been shown [12] that in general, if modulus m is an integral power of 2, then the Boolean function for bit S_i of the sum is a function of only the summand input variables X_j and Y_j, $j = 0, 1, \dots, i$. For a modulus $m = 2^n$, instead of n Boolean functions of 2n variables each, we get one function S_{n-1} of 2n variables, one function S_{n-2} of $2n - 2$ variables, and so on, to one function S_0 of 2 variables. The same result is true for a mod m multiplier, when m is an integral power of 2. Therefore, logic gate implementation for integral-power-of-2 moduli is considerably simplified. This property can be profitably exploited for only one modulus in a system of relatively prime moduli.

4.1.2 Modulus Substitution

It is well known that mod k addition can be performed on a mod m adder where m > k. This would, however, require detection of "overflow" mod k and then the correction of the sum mod m to get the correct sum mod k. Overflow will occur when the sum exceeds (k - 1) and this would indicate that a correction is necessary. The same is true for mod k multiplication. Table 4.1 shows the addition mod 7 and mod 8, with the overflow states mod 7 shown within triangles. It has been shown [10] that the number of overflow states is independent of the "parent"modulus (which is eight here) and that for a modulus k, k odd, the number of overflow states is equal to

$$\sum_{x=1}^{k-1} x = \frac{k(k-1)}{2}$$

⊕7	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

⊕8	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

Table 4.1 Addition mod 7 and mod 8 With Overflow mod 7.

One way to utilize the modulus substitution approach is to design one parent adder and multiplier and to carry out one mod m_i addition or multiplication per machine cycle. For a system of n moduli, an addition or multiplication would then require n machine cycles. This indicates the possibility of modular serial computation.

The second way to utilize modulus substitution is to combine it with the properties of integral-power-of-2 moduli and to design several parent adders and multipliers for different moduli. For example, in a system with moduli 2,3,5,7,11 and 13, a mod 4 adder could add mod 3, two mod 8 adders in parallel could add mod 5 and mod 7, and two mod 16 adders in parallel could add mod 11 and mod 13. This would require only one machine cycle. A second possibility is to use a serial-parallel combination : A mod 8 adder to add mod 3, mod 5 and mod 7, and a mod 16 adder to add mod 11 and mod 13. This would require 3 machine cycles. In the first cycle mod 2, mod 3, and mod 11 additions can be performed, in the second cycle mod 5 and mod 13 can be performed, and in the third cycle mod 7 addition can be done. The final choice is obviously governed by speed and hardware requirements. Ref. [10] tabulates the results of various implementation studies and the same table is shown here [Ref. Table 4.2].

Table 4. 2 Comparison of Different Adder Implementation Schemes (41)

Modulus	Adder Gates
Parallel Adder Add Time = 1 Clock Time	
32	130
13	102
11	100
7	40
5	26
3	10
$\Pi m_i \approx 2^{19}$	Total Gates = 408
Serial Adder Using Mod Substitution Add Time = 6 Clock Times	
32	130
13	20
11	18
7	12
5	7
3	4
$\Pi m_i \approx 2^{19}$	Total Gates = 191
Serial-Parallel Adder Add Time = 3 Clock Times	
32	130
13	20
11	18
7	40
5	13
3	10
$\Pi m_i \approx 2^{19}$	Total Gates = 231

It is mentioned that for the parallel adder, the adders for moduli 32, 7, 5 and 3 are mechanized directly ; those for moduli 11 and 13 are derived from mod 16 adders using modulus substitution. For the serial adder using modulus substitution, a mod 32 adder is mechanized directly and used through modulus substitution to add for the moduli 13, 11, 7, 5 and 3, one modulus per clock time. The serial-parallel adder uses a mod 32 direct adder for moduli 32, 13 and 11 and a mod 7 direct adder for moduli 7, 5 and 3. For comparison with the parallel adder, direct implementation of adders for all these moduli would require about 500 logic gates.

Another way to mechanize a mod m adder would be to use binary half and full adders as is done in case of decimal adders. Mechanization of a mod 32 adder using binary adders is very simple and requires 44 AND and OR gates. In mod 32 addition, whenever the sum of two addends exceeds 32, we have to subtract 32 from the sum to get the correct sum mod 32. Subtraction of 32 is done simply by ignoring the carry generated from the 16-order. For example, addition of 20 and 30 in mod 32 produces 18.

```
                10100  Twenty
                11110  Thirty
Ignore      1  ───────
the carry       10010  Eighteen
```

A schematic of this scheme is shown in Fig. 4.1. The use of binary adders to get addition mod 13 is shown in Fig. 4.2. Additional circuitry shown is for detecting overflow mod 13 and for correcting the actual sum obtained to get the correct sum mod 13. Overflow mod 13 occurs only when in the sum bits, 1's appear in 8, 4 and 1 orders or in 8, 4 and 2 orders or when there is a carry generated in the 8-order. When overflow does occur, the correct sum mod 13 is obtained by subtracting 13 from the indicated sum. This is achieved by adding 3 to the indicated sum and then subtracting 16 from it, subtraction of 16 being achieved by ignoring the carry from the 8-order of corrected sum. Table 4.3 shows the number of gates required for modular adders when binary adders are used to implement

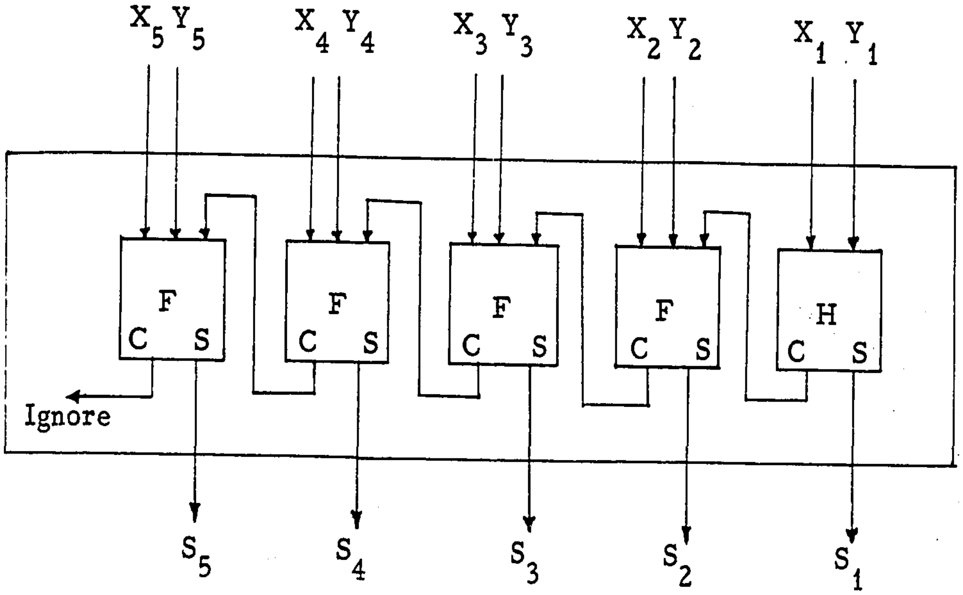


Fig. 4.1. Mod 32 Adder Using Binary Adders

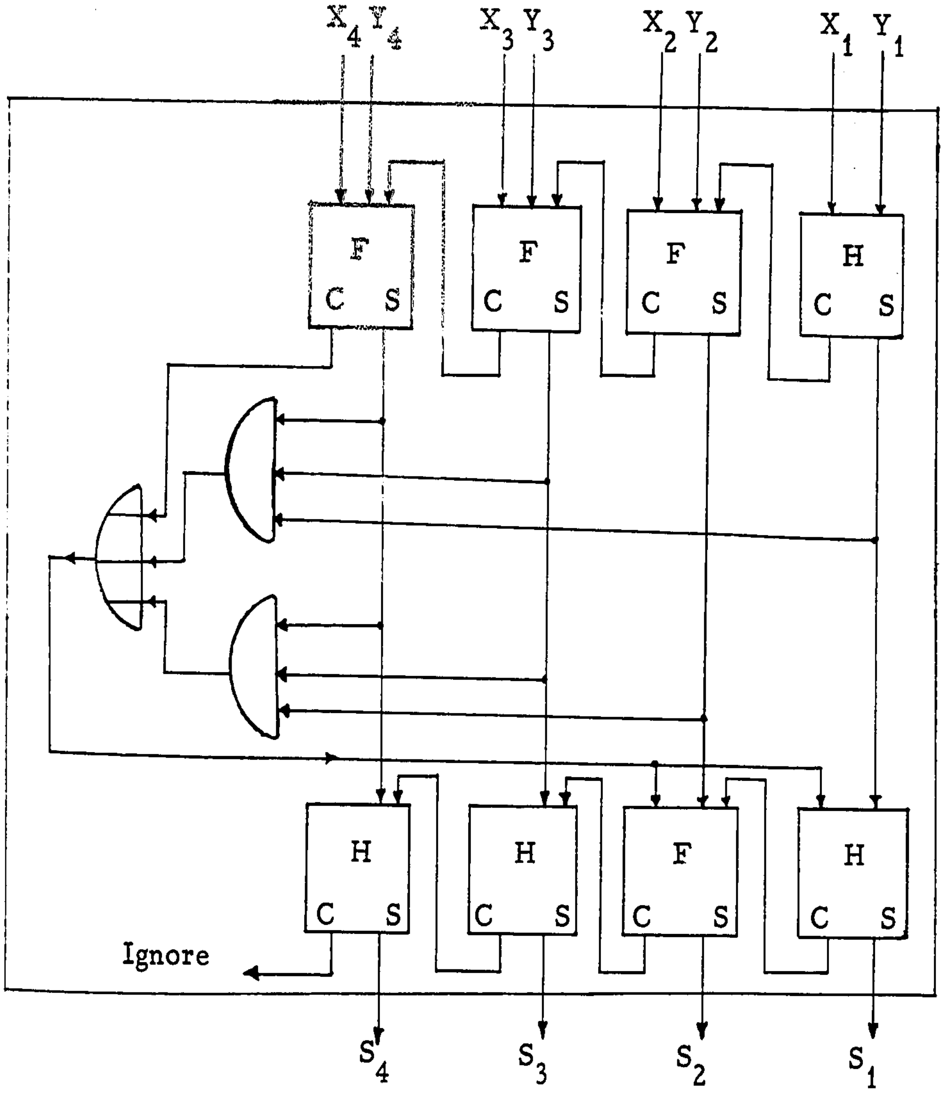


Fig. 4.2 Mod 13 Adder Using Binary Adders

them. This scheme does not result in a reduction of logic for the mod 3 adder, and it is implemented directly. It should be mentioned that the gate count is for AND and OR gates and the maximum number of inputs to a gate is three.

Modulus	Adder Gates
Parallel Adder Add Time = Time for carry to propagate by 5 positions	
32	44
13	60
11	60
7	38
5	45
3	8
$\Pi m_i \approx 2^{19}$	255

Table 4.3

The same design principles hold for modular multipliers. However, modulus substitution is of limited use for the multipliers due to the multiple overflow problem encountered in modular multiplication. The reason is as follows.

In a mod m addition process, the direct sum of two operands never exceeds 2m - 2 and then the sum mod m is obtained by subtracting m from the direct sum. However, in mod m multiplication, the maximum value of the direct product of two operands can be $(m - 1)^2$ and then the product mod m is obtained by subtracting m or multiples of m from the direct product depending on how many times the product overflows m. For example, if the product lies between m and 2m, we have to subtract m from it to get the correct product mod m ; if it lies between 2m and 3m, we subtract 2m from it and so on. This multiple overflow presents problems in modular multiplier design because we must have a way of detecting how many times the product overflows m and how many m's must be subtracted from it to get the correct result. Obviously, the use of modulus substitution would complicate the problem more because of multiple overflow in both the "parent" modulus and the "substituted" modulus.

Not much work seems to have been done in this direction except for using direct implementation. This problem deserves further study. However, it is outside the scope of this thesis and we shall leave it here.

4.2. Complementation : It has been discussed in Chapter 2, how subtraction is performed using complement addition. Therefore, we must have a means of finding the complement X' of a number X in some modulus m. By definition

$$X' = m - X.$$

There are two solutions proposed for this problem.

1. Let us consider a modulus $m = 11$ and a number $X = 4$. Code m and its least positive residues in ordinary binary code.

0 ---- 0000
1 ---- 0001
2 ---- 0010
3 ---- 0011
4 ---- 0100
5 ---- 0101
6 ---- 0110
7 ---- 0111
8 ---- 1000
9 ---- 1001
10 ---- 1010

and m = 11 ---- 1011

To find X', complement all the bits of X to get $\bar{X}$ and add to m. This addition is the ordinary binary addition.

$X = 4 = 0100$

$\overline{X} = 1011$

$$\begin{array}{rr} m+\overline{X} = & 1011 \\ + & 1011 \\ \hline & 0110 \\ \text{End around Carry } ① \hookrightarrow & 1 \\ \hline & 0111, \end{array}$$

which is 7. The complement (or additive inverse) of 4 in mod 11 is, of course, 7.

(2) The second method is based on a binary assignment to the least positive residues such that they form a self-complementing code with respect to m ; that is the assignment for any residue x_i is such that

$$x_i + \overline{x}_i = m$$

0	-	0000
1	-	0001
2	-	0011
3	-	0010
4	-	0110
5	-	0111
		0101
		0100
9	-	1100
8	-	1101
11	-	1111
10	-	1110
		1010
		1011
7	-	1001
6	-	1000

Table 4.4.

The mutually complementary numbers are shown paired together [Ref. Table 4.4]. Now if, $X = 4$, which is assigned 0110, $X' = \overline{X} = 1001$. This corresponds to 7, which is the additive inverse of 4 mod 11. Therefore, using such an assignment scheme the additive inverse of a number X can be found by complementing the bits of the assignment for X.

As far as modular complementation (finding additive inverse) is concerned,. the second solution is the most desirable one when using binary coding for residues. This, however, may result in more complex hardware for adders and multipliers and may not, therefore, be worthwhile. Hence, its effect on hardware deserves attention. However, this problem is outside of the scope of this thesis.

4.3 Input Translation

In this section we consider the problem of converting the input data into modular form. The input will be in a fixed radix form, with radix r, Any number X can be expressed by a polynomial

$$X = C_n r^n + C_{n-1} r^{n-1} + \ldots . + C_1 r + C_0, 0 \le C_i < r.$$

Then

$$[X]_{m_i} = [[C_n r^n]_{m_i} + [C_{n-1} r^{n-1}]_{m_i} + \ldots + [C_0]_{m_i}]_{m_i}$$

If the quantities $[C_i r^i]_{m_i}$ can be evaluated without the actual multiplication in a mod m_i multiplier, then $[X]_{m_i}$ can be determined using a mod m_i adder.

Let us consider binary inputs for a system with moduli 2, 3, 5 and 7. This system is capable of uniquely representing a total of 210 integers. The number of bits required for coding is equal to $\langle \log_2 210 \rangle$, where $\langle I \rangle$ denotes the least integer greater than or equal to I. Here $\langle \log_2 210 \rangle = 8$. Therefore, this system requires 8 bits to code all the 210 integers. Any X within the range of representation can be expressed as a polynomial.

$$X = C_7 2^7 + C_6 2^6 + C_5 2^5 + C_4 2^4 + C_3 2^3 + C_2 2^2 + C_1 2^1 + C_0 .$$

Any C_i can now take only two values — 0 or 1.

Then $[X]_2 = [C_0]_2 = C_0$

$$[X]_3 = [2C_7 + C_6 + 2C_5 + C_4 + 2C_3 + C_2 + 2C_1 + C_0]_3$$

$$[X]_5 = [3C_7 + 4C_6 + 2C_5 + C_4 + 3C_3 + 4C_2 + 2C_1 + C_0]_5$$

$$[X]_7 = [2C_7 + C_6 + 4C_5 + 2C_4 + C_3 + 4C_2 + 2C_1 + C_0]_7$$

In the worst case, determination of $[X]_3$ would require 4 multiplications and 7 additions ; determination of $[X]_5$ would require 6 multiplications and 7 additions and $[X]_7$ would require 5 multiplications and 7 additions. All these multiplications and additions in different moduli occur simultaneously. The maximum time taken for input translation is then the time required in the worst of the three above cases. We notice that the worst case is the determination of $[X]_5$. The translation process can be speeded up, at the expense of no extra hardware, by not using the modular multipliers. Then the process will need at most 7 additions. To discuss this in a little more detail let us focus our attention to the translation equations. The inputs to the mod 3 adder must all be 2-bit numbers. Therefore, C_0, C_2, C_4 and C_6 (all 1-bit numbers) must be converted into 2-bit numbers by adding a 0 at the left. $2C_1$, $2C_3$, $2C_5$ and $2C_7$ can only take the values 0 and 2, which in binary code are 00 and 10. Therefore, we can take C_1, C_3, C_5 and C_7 and place a 0 on the right before feeding into the mod 3 adder. In other words depending upon whether the coefficient of a C_i is 1 or 2, we have to add a 0 to the left or right of C_i, as the case may be, before feeding it to the mod 3 adder. Similar additions of 0's to the left or right of C_i would be necessary before feeding them to the mod 5 and mod 7 adders. If $y_1\, y_2$, $y_3\, y_4\, y_5$ and $y_6\, y_7\, y_8$ are the binary inputs to mod 3, mod 5 and mod 7 adders respectively, then the conversion of the C_i's to y_i's is schematically shown in Fig. 4.1. Tables 4.4 show the effect of coefficients of C_i's on the addition of 0's and 1's to the left or right of C_i's.

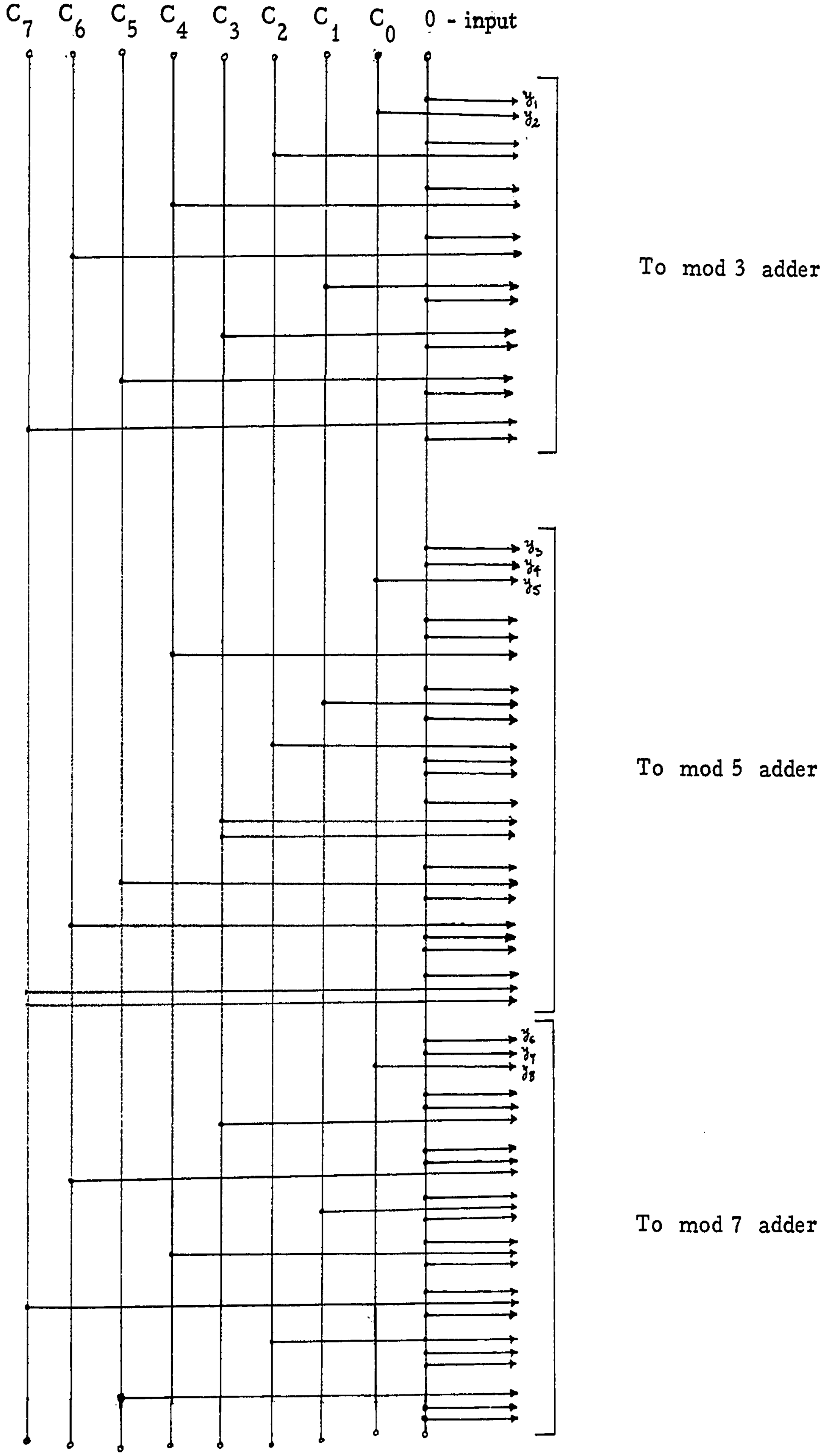


Fig. 4.1 Conversion of Input bits to Proper Form for Adder Inputs.

C_i	Coefficient of C_i	Operation
0	1	Add 0 Left
1	2	Add 0 Right

Table 4.4.a. Conversion of C_i for Proper Input to Mod 3 Adder

C_i	Coefficient of C_i	Operation
0	1	Add 00 Left
	2	Add 0 Left, Add 0 Right
1	3	Add 0 C_i Left
	4	Add 00 Right

Table 4.4.b Conversion of C_i for Proper Input to Mod 5 Adder

C_i	Coefficient of C_i	Operation
0	1	Add 00 Left
	2	Add 0 Left, Add 0 Right
1	4	Add 00 Right

Table 4.4.c. Conversion of C_i for Proper Input to Mod 7 Adder.

4.4 Output Translation

The problem of output translation is a little more complicated than that of input translation. In Sec. 2.5, we discussed the mathematical equations describing this process. In this section we will discuss the mechanization of the conversion process from a residue number to its corresponding natural number in binary form. The system of moduli considered for input translation, will again be used here. Let $m_1 = 2$, $m_2 = 3$, $m_3 = 5$ and $m_4 = 7$.

4.4.1 Mixed Radix Conversion Process : In this section we discuss the use of mixed radix conversion for output translation. In the above system of moduli, any X in the range $0 \leq X < 210$ can be represented in the mixed radix form as :

$$X = r_4 \cdot m_1 \cdot m_2 \cdot m_3 + r_3 \cdot m_1 \cdot m_2 + r_2 \cdot m_1 + r_1, \; 0 \leq r_i < m_i ,$$

r_i being the mixed radix digits. Binary representation for any X in the above range requires 8 bits. Addition of all the quantities in the mixed radix equation can be done using an 8-cell binary adder. The least significant bits are added in a half-adder. Full adders are needed for other bit positions. The sum never exceeds 210 and, therefore, there is no need for detecting overflow mod 210. We must convert all the quantities to be added into 8-bit form. To find the product of the mixed radix digits with the moduli, we would require binary multipliers. To save hardware, the multiplication can be replaced by a table look-up procedure : $m_1 \cdot m_2 \cdot m_3$, $m_1 \cdot m_2$ and m_1 are all fixed quantities. It is also known that any r_i can have only one of the values from 0 to $m_i - 1$, for a particular residue number. Therefore, all the possible values of $r_4 \cdot m_1 \cdot m_2 \cdot m_3$, $r_3 \cdot m_1 \cdot m_2$ and $r_2 \cdot m_1$ can be stored and depending upon a particular value of each r_2, r_3 and r_4, the corresponding products with the moduli can be read from the table. Tables 4.5. show the possible values of these quantities and their binary representations. Conversion of $r_3 \cdot m_1 \cdot m_2$, $r_2 \cdot m_1$ and r_1 to 8-bit representation can be achieved in the same way as outlined in Sec. 4.3.

r_4	$r_4 \cdot m_1 \cdot m_2 \cdot m_3$	Binary Code
0	0	0 0 0 0 0 0 0 0
1	30	0 0 0 1 1 1 1 0
2	60	0 0 1 1 1 1 0 0
3	90	0 1 0 1 1 0 1 0
4	120	0 1 1 1 1 0 0 0
5	150	1 0 0 1 0 1 1 0
6	180	1 0 1 1 0 1 0 0

Table 4.5.a Possible Values of $r_4 \cdot m_1 \cdot m_2 \cdot m_3$

r_3	$r_3 \cdot m_1 \cdot m_2$	Binary Code
0	0	0 0 0 0 0
1	6	0 0 1 1 0
2	12	0 1 1 0 0
3	18	1 0 0 1 0
4	24	1 1 0 0 0

Table 4.5.b Possible Values of $r_3 \cdot m_1 \cdot m_2$

r_2	$r_2 \cdot m_1$	Binary Code
0	0	0 0 0
1	2	0 1 0
2	4	1 0 0

Table 4.5.c Possible Values of $r_2 \cdot m_1$

An example will clearly illustrate the output translation process. Consider a number X with residue representation (1, 2, 1, 6) where $[X]_2 = 1$, $[X]_3 = 2$, $[X]_5 = 1$ and $[X]_7 = 6$. The mixed radix digits for these residue digits are $r_1 = 1$, $r_2 = 2$, $r_3 = 1$, $r_4 = 1$. For these values of r_2, r_3 and r_4, the table look up gives the corresponding values of $r_2 \cdot m_1$, $r_3 \cdot m_1 \cdot m_2$ and $r_4 \cdot m_1 \cdot m_2 \cdot m_3$ as (in binary code) 100, 00110 and 00011110 respectively. After conversion of all the quantities to 8-bit representation, and adding in the binary adder, we get X=00101001, which is the correct answer.

If desired, table look-up can be replaced by combinational logic.

4.4.2. Chinese Remainder Theorem : If for some special high output rate purposes, the method of Sec. 4.4.1 is too slow, then the Remainder Theorem can be used for faster conversion. For a system of n moduli, the conversion process described in Sec. 4.4.1 requires 3(n - 1) addition, multiplication and complementation cycles, (n - 1) table look-ups plus the time required for binary addition of n, $\langle \log_2 M \rangle$ - bit numbers,

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

Referring to Sec 2.5, we find that in a general system

$$X \equiv (x_1 X_1 \frac{M}{m_1} + \ldots + x_n X_n \frac{M}{m_n}) \mod M,$$

where $X_i \dfrac{M}{m_i}$ are constants which can be calculated in advance and stored to be retrieved when needed. If fixed radix multiplication is used to obtain $x_i X_i \dfrac{M}{m_i}$ and fixed radix addition to sum these products and further, to obtain rapid solutions, if parallel addition and multiplication are used, then an enormous amount of hardware would be required. In this case, the hardware for conversion would be comparable to that of an entire arithmetic unit of a fixed radix parallel machine and this would offset much of the advantage of modular arithmetic.

It is noted that multiplication is not necessary to obtain $x_i X_i \frac{M}{m_i}$. Each such quantity has only m_i values, 0 through $(m_i - 1) X_i \frac{M}{m_i}$. Therefore, for each m_i, $x_i X_i \frac{M}{m_i}$ can be stored in a read-only memory and the conversion problem is reduced to a series of memory retrievals and additions. For fast output a parallel adder is still required, but the hardware requirement is much smaller than that when parallel multiplication is required. This solution would require n table look ups and addition of n, $\langle \log_2 M \rangle$-bit numbers.

The total requirements of this process are a read-only memory, a parallel adder, the necessary control logic and the detection of "overflow" mod M and the correction necessary to get the correct value of X. This conversion process would require n table look-ups and (n - 1) additions.

Coming back to the example considered in Sec. 4.4.1, the conversion equation for this system is :

$$X = (105\, x_1 + 70\, x_2 + 126\, x_3 + 120\, x_4) \bmod 210.$$

considering the residue number (1, 2, 1, 6) again, table look-up will yield the right hand side quantities as 01101001, 10001100, 01111110 and 1011010000. These are added in a parallel adder, overflow mod 210 is detected by procedures outlined in Sec. 4.1.2 and the indicated sum is corrected to give the correct value of X = 00101001.

5. COMPARISON OF SIGN DETECTION METHODS

5.1 In this chapter we shall consider the sign detection problem again and attempt to present a comparison between the various possible schemes that can be used to establish the sign of a residue number. The evaluation and comparison is made difficult because of the following three factors :

(1) The evaluation methods vary widely and depend upon the availability of the type of hardware to be used and upon the entire system. For example, it is not very meaningful to make a comparison based on diode or transistor count when logic gates with a fixed number of inputs are to be used for implementation. In this case, a gate count will be the meaningful comparison. In some other case, the number of interconnections may be a more realistic way of comparison.

(2) The costs of component units are not known. In Chapter 4, some definite figures for the number of gates required to implement modular adders were obtained, but we have no such figure on modular multipliers. We could argue that we can always find the number of gates required for direct implementation of modular multipliers, but we cannot be sure that this would be the most economic implementation. Also, we do not have complete information about the complement units. Complementation can be done by using the methods of Sec. 4.2 or by combinational logic which directly transforms a binary coded residue to its complement form. Since we are not able to decide how complementation is to be done we cannot state definite costs.

(3) The methods of sign determination which are to be compared vary widely. For example, it is extremely difficult to compare a software solution with a hardware solution, unless the entire system is known. We shall, however, try to present some insight to this problem.

5.1.1 Purely Combinational Method

This is the brute force method of sign detection. The sign function S is assigned a 0 in the positive region of the residue representation and 1 in the negative region. The residues are coded in binary form, Using any of the standard minimization techniques, S is expressed as a Boolean function of the binary coded residues. This method, though straightforward, is not practical because of hardware considerations. The expression for S contains a large number of prime implicants and as a result a large number of logic gates are required to detect the sign. For example a system with moduli 2, 3, 5 and 7, which is too small for any practical-sized computer, requires about 70 AND and OR gates, with the maximum number of inputs to a gate being 7. It, therefore, appears that for any practical-sized computer, this method will be too expensive to use.

5.1.2 Mixed Radix Translation Process

In this method, a given residue number is converted to its corresponding binary or decimal form which will indicate whether it lies in the positive or the negative region. In a system of n mutually prime moduli where binary coding is used for all the quantities the sign detection by mixed radix translation involves $3(n-1)$ modular operation cycles, $(n-1)$ table look-ups, addition of n, $\langle \log_2 M \rangle$-bit numbers and comparison of this sum with $M/2$, which draws the line between the positive and the negative regions. It is difficult to assess at this point, how much time is involved in this process because it can be implemented either completely by hardware or by a combination of hardware and software, in which case only one modular operation (addition, multiplication or complementation) is performed per cycle time. The cycle time can vary depending upon the type of components being used.

In a system with moduli 2, 3, 5, 7 and 11, we require 10 complement units, 10 modular adders and 10 modular multipliers to determine the mixed radix digits by hardware alone. The remaining process of sign detection would require 4 table look-ups, addition of 5, 12-bit numbers and comparison of the sum with $\frac{M}{2} = 1155$. It is not economic or practical to use so much hardware for finding the mixed radix digits alone. It is much more economical to perform one modular operation per cycle, though the sign detection would then require more time.

5.1.3 Use of The Remainder Theorem

The theorem is used for converting a given residue number to a fixed radix form, and the comparison of this with a fixed number will yield the sign information. For an n-moduli system, this process involves n table look-ups, addition of n, $\langle \log_2 M \rangle$ -bit numbers, detection of overflow mod M, the correction to get the actual sum mod M and comparison of the sum with M/2. In the example under consideration, therefore, this process requires 5 table look-ups, addition of 5, 12-bit numbers, detection of overflow mod 2310, correction of the sum and finally comparison of this with 1155.

It should be clear that from the point of view of hardware, this process is better than the mixed-radix translation, when the latter is implemented by hardware alone. The use of the Remainder Theorem does require additional logic for overflow detection and correction of the sum but this is less than the logic requirements for mixed-radix translation.

From the consideration of speed also, the Remainder Theorem is faster. For the mixed-radix process, the signal has to go through 3(n-1) stages of logic to yield all the mixed-radix digits and it is reasonable to expect that the time required for this is greater than that required for detection of overflow and correction of the sum.

If only one modular operation is performed per cycle time, then the mixed-radix process can be implemented using the modular adders, multipliers and the complement units of the arithmetic unit.

In this case it would require less hardware than the Remainder Theorem. But, then it would be considerably slower than the Remainder Theorem method.

5.1.4 Here we consider the approach of Sec. 3.3. Consider the same example again, with moduli 2, 3, 5, 7 and 11. If we partition the set into two composite moduli 6 and 385, the sign detection (as outlined in Sec. 3.3) requires six stages of logic to determine the three required mixed radix digits and then another four stages for the remaining process of sign detection. Both from hardware and speed considerations, this method is better than the mixed-radix translation. The four stages of logic through which the signal has to go, represent a simpler realization than the complete mixed radix translation. Also, the latter requires twelve stages of logic, just to determine the mixed radix digits whereas this method requires ten stages to determine the sign and is, therefore, faster. It is difficult to compare this method with the Remainder Theorem.

Now let the set of moduli be partitioned into 30 (2, 3 and 5) and 77 (7 and 11). In the worst case, sign detection requires three stages of logic to determine two mixed radix digits, one stage of combinational logic to convert these to mod 2, 3 and 5 representation and then fifteen comparisons. Such a large number of comparisons makes the comparison logic large, if hardware is used, or makes it time consuming if software is used. A detailed analysis to determine which partition scheme for the moduli is best is postponed for future studies. However, it can be stated for certain that if Theorem 3.3 is used, the sign information is obtained faster than the mixed-radix-translation. In the former case, the signal has to go through nine stages of logic (in the example under consideration) to determine the four mixed radix digits and then through an EXCLUSIVE OR gate to give the sign information.

5.2 The discussion of Sec. 5.1 was not very quantitative in nature because of the limitations pointed out in the beginning. To establish which of the methods is fastest or most economical requires a thorough systems study which is outside of the scope of this thesis.

5.3 Conclusions

The work carried out for the purpose of this thesis has led to the following conclusions.

The sign detection problem in residue number systems should not be treated as an isolated problem. There is no unique "best" solution, but rather a variety of solutions, and each solution has to be evaluated with respect to the overall system design.

The contributions to the residue arithmetic computing techniques are summarized below.

In the Two-Moduli Theorem (Sec. 3.2) we have extended Szabo's Coding

Theorem by finding an explicit formula for the sign function S. In the special case when one of the two moduli is 2, S is the EXCLUSIVE OR sum of two bits. The Theorem has been extended to the general case of n moduli, and it has been shown that in the special case when one of the moduli is 2, S is the EXCLUSIVE OR sum of n bits.

It should be pointed out that the use of the mixed-radix translation process has been previously suggested for sign detection. In the suggested method [5], the mixed-radix notation is used as an intermediate step in translating from the residue code to a binary or decimal code. However, as a result of the Two-Moduli Theorem, we have been able to use partial mixed-radix conversion and additional combinational circuitry to obtain the sign function without translating to decimal or binary. The use of the theorem with its extension to the general case provides a degree of flexibility in implementing the method. It has been shown that varying amounts of hardware and software can be used, depending on the system requirements and design.

An extremely simple solution to the input translation problem has been found. It has been shown (Sec. 4.3) that translation from binary to residue representation can be achieved by using modular adders only. There is no need for modular multiplication for this process, for it can be replaced by simple wire-splitting (Fig. 4.1). As pointed out in Chapter 4, modular multiplier design is a difficult problem and replacing it by wire-splitting certainly reduces the complexity and the time required for input translation.

5.4 Recommendations for Further Research

One of the areas that deserves investigation is fast output translation. The existing methods are too slow compared to the speed of the arithmetic unit and hence not desirable for use in the sign detection problem. A fast translation method will not only solve this problem but also the related problems of magnitude comparison and overflow detection.

The multiple overflow problem in modular multiplier design poses serious problems. An investigation in this area should lead to some interesting results.

Another area of investigation is to find an optimum coding scheme for the overall system. We have seen how self-complementing codes simplify the complementation problem, but its effect on other processes has not been studied. It is strongly felt that this aspect of residue arithmetic will prove to be quite challenging.

REFERENCES

1. Antonin Svoboda, "The Numerical System of Residual Classes (SRC)", Computer Progress in Czechoslovakia, Prague, Czechoslovakia.

2. H.L. Garner, "The Residue Number System", IRE Transactions on Electronic Computers , Vol. EC-8, June, 1959, pp. 140-147.

3. H. Aiken, W. Semon, Advanced Digital Computer Logic, Report No. WADC TR-59-472, Computing Labs, Harvard University, July 1959.

4. P.W. Cheney, "A Digital Correlator Based on the Residue Number System," IRE Transactions on Electronic Computers, Vol. EC-10, March 1961, pp. 63 - 70.

5. N. Szabo, "Sign Detection in Nonredundant Residue Systems", IRE Transactions on Electronic Computers, Vol. EC-11, August, 1962, pp. 494-500.

6. W.E. Deskins, "Abstract Algebra" Collier-Macmillan, 1964.

7. G.H. Hardy, E.M. Wright, "An Introduction to the Theory of Numbers", Oxford Press, 1954.

8. R.W. Watson and C.W. Hastings, "Self-Checked Computation Using Residue Arithmetic," Proceedings of the IEEE, Vol. 54, No. 12, December 1966, pp. 1920-1931.

9. "Research on Residue Number Serial Computation Techniques", Technical Documentary Report No. RTD-TDR-634034, Systems Research Laboratories, Wright-Patterson Air Force Base, Ohio, November 1963.

10. "Modular Arithmetic Computing Techniques," Technical Documentary Report No. ASD-TDR-63-280, Electronic Technology Division, Aeronautical Systems Division, Wright-Patterson Air Force Base, Ohio, May 1963.

11. Ivan Flores, "The Logic of Computer Arithmetic", Prentice-Hall, 1963.

12. H.L. Garner, et al, "Residue Number Systems for Computers", ASD Technical Report No. 61-483, University of Michigan, October 1961.

VITA

Name : Dilip K. Banerji

Born : November 3, 1943, India

Educated :

Primary : Kanpur, India

Secondary : Kanpur, India

University : The Indian Institute of Technology,
Kanpur, India.
1965 Bachelor of Technology in
Electrical Engineering
(B. Tech.)