CONVERSION SCHEME IN RESIDUE CODE

R.K. Arora and Saroj Kaushik

Computer Centre, Indian Institute of Technology
New Delhi, India

ABSTRACT

In this paper, an implementation scheme involving decoders and residue adders has been suggested to convert input data in fixed radix representation to residue representation. A method dealing with the reverse process is also demonstrated. A comparison has been made with the methods known hitherto.

INTRODUCTION

In Residue Number System, the arithmetic operations of addition, subtraction and multiplication are executed in the same period of time without the need for interpositional carry. But these advantages of modular (residue) arithmetic are nullified by the complexity of the algorithms for division, sign determination, overflow detection etc. When these operations are required frequently in conjunction with addition, subtraction and multiplication, the use of modular arithmetic can be justified only if fast means of conversion into and out of the residue representation are available. In this paper, the problems of converting a number of its fixed radix representation to residue representation and vice versa are considered. A comparative study has been carried out with respect to the existing methods.

RESIDUE CODE

Consider an ordered set of n positive integers $(m_1,m_2,\ldots,m_n)$ such that $m_i \geq 2$ for any $1 \leq i \leq n$ and are relatively prime to each other. These m_i's are called moduli and the corresponding ordered n-tuple $(x_1,x_2,\ldots,x_n)$ of least non-negative residues of a number X with respect to the above moduli is called the residue representation of X. Such a representation of numbers forms a Residue Number System (RNS). Since all moduli are relatively prime, each $X \in [0,M)$, where $M = \prod_{i=1}^{n} m_i$, is uniquely represented in the RNS.

CONVERSION FROM FIXED RADIX REPRESENTATION TO RESIDUE REPRESENTATION

Definition of the Problem

The problem of converting a base b integer X (in fixed radix representation) with respect to the moduli $m_1,m_2,\ldots,m_n$ is called 'Input Conversion' or 'Input Translation' problem[1,3]. Since each residue can be computed independently of the other residues, it is sufficient to consider only the problem of finding the residue of X with respect to one modulus m.

Let the polynomial representation of X as a base b number be:

$$X = a_{p-1}\cdot b^{p-2} + \ldots + a_0,$$

where $0 \leq a_i < b$, $i=0,1,\ldots,p-1$;

or $$X = \sum_{i=0}^{p-1} a_i \cdot b^i .$$

Then
$$|X|_m = \left|\sum_{i=0}^{p-1} a_i \cdot b^i\right|_m,$$
$$= \left|\sum_{i=0}^{p-1} |a_i \cdot b^i|_m\right|_m,$$
$$= \left|\sum_{i=0}^{p-1} |c_i|_m\right|_m,$$

where $c_i = a_i \cdot b^i$.

The problem of input conversion has been considered earlier by Banerji[4,5]. He assumes the binary representation of a digit a_i and $|2^j \cdot b^i|_m$ for each i and j. It is briefly described as follows.

Let $a_{i,j}$ be the bits in the representation of a_i,

CH1630-3/81/0000/0152$00.75 © 1981 IEEE

then $a_i = \sum_{j=0}^{k-1} a_{i,j} \cdot 2^j$, for some positive integer k.

Then $|c_i|_m = \left| \sum_{j=0}^{k-1} a_{i,j} \cdot 2^j \cdot b^i \right|_m$,

$$= \left| \sum_{j=0}^{k-1} |a_{i,j} \cdot 2^j \cdot b^i|_m |_m \right| ,$$

$$= \left| \sum_{j=0}^{k-1} |d_{i,j}|_m |_m \right| ,$$

where $d_{i,j} = a_{i,j} \cdot 2^j \cdot b^i$.

He suggested that if the $|d_{i,j}|_m$ were available in binary coded form, $|c_i|_m$ could be obtained by using a k-input mod m adder in level 1. The required residue could be obtained by further adding $|c_i|_m$ in a p-input mod m adder in level 2. He also observed that in a typical case when b = 10, the number of combinations was sufficiently small so that combinational logic could be considered to compute $|c_i|_m$ directly from the bits of a_i.

The Proposed Scheme

In our scheme, we first completely decode a_i (a base b digit of X) by producing binary signals α_{i,g_i} for each g_i, $0 \leq g_i < b$ such that

$\alpha_{i,g_i} = 0$ iff $a_i = g_i$;

and $\alpha_{i,g_i} = 1$ otherwise.

Next we compute the binary representation $(c_{i,q-1}, c_{i,q-2}, \ldots, c_{i,1}, c_{i,0})$ of $|c_i|_m$ by a translation circuit, which is assumed to be stored in a register of q bits.

Finally, the $|c_i|_m$ (in binary code) for i=0,1,...,p-1, are added in p-input mod m adder to get the required residue of X modulo m. The over-all schematic of the input conversion process discussed above is shown in figure 1.

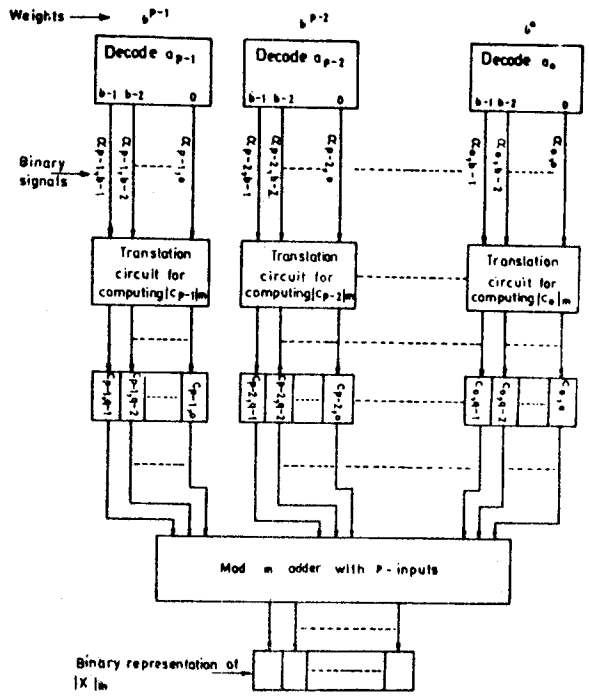

Fig. 1 Overall schematic of input conversion process

Example 1 :

Let X = 234, b = 10 and m = 17. Here p = 3, $a_0 = 4$, $a_1 = 3$ and $a_2 = 2$.

To compute $|X|_{17}$, we first find the value of $|c_0|_{17}$, $|c_1|_{17}$ and $|c_2|_{17}$ and then add them in 3-input mod 17 adder. The computation of $|c_1|_{17}$ and $|c_2|_{17}$ in binary code is shown in Tables 1 and 2 respectively. Here m = 17 b = 10, so $|c_0|_{17} = |a_0|_{17} = a_0$.

Binary signals of Decoded a_1	$\|c_1\|_{17} = \|a_1 \cdot 10\|_{17}$	Binary representation of $\|c_1\|_{17}$ $c_{1,4}\,c_{1,3}\,c_{1,2}\,c_{1,1}\,c_{1,0}$
$\alpha_{1,0}$	0	0 0 0 0 0
$\alpha_{1,1}$	10	0 1 0 1 0
$\alpha_{1,2}$	3	0 0 0 1 1
$\alpha_{1,3}$	13	0 1 1 0 1
$\alpha_{1,4}$	6	0 0 1 1 0
$\alpha_{1,5}$	16	1 0 0 0 0
$\alpha_{1,6}$	9	0 1 0 0 1
$\alpha_{1,7}$	2	0 0 0 1 0
$\alpha_{1,8}$	12	0 1 1 0 0
$\alpha_{1,9}$	5	0 0 1 0 1

Table 1

Table 2

Binary signals of Decoded a_2	$\lvert c_2 \rvert_{17} = \lvert a_2 \cdot 10^2 \rvert_{17}$	Binary representation of c_2 17 $c_{2,4}c_{2,3}c_{2,2}c_{2,1}c_{2,0}$				
$\alpha_{2,0}$	0	0	0	0	0	0
$\alpha_{2,1}$	15	0	1	1	1	1
$\alpha_{2,2}$	13	0	1	1	0	1
$\alpha_{2,3}$	11	0	1	0	1	1
$\alpha_{2,4}$	9	0	1	0	0	1
$\alpha_{2,5}$	7	0	0	1	1	1
$\alpha_{2,6}$	5	0	0	1	0	1
$\alpha_{2,7}$	3	0	0	0	1	1
$\alpha_{2,8}$	1	0	0	0	0	1
$\alpha_{2,9}$	16	1	0	0	0	0

Now we design a translation circuit whose input is the decoded values of a_i and output is the binary representation of $\lvert c_i \rvert_{17}$, i = 1,2. We get the following Boolean equations for this translation, For i = 1 (refer Table 1) we get

$$c_{1,0}=(\alpha_{1,0}\cdot\alpha_{1,1}\cdot\alpha_{1,4}\cdot\alpha_{1,5}\cdot\alpha_{1,7}\cdot\alpha_{1,8})$$

$$c_{1,1}=(\alpha_{1,0}\cdot\alpha_{1,3}\cdot\alpha_{1,5}\cdot\alpha_{1,6}\cdot\alpha_{1,8}\cdot\alpha_{1,9})$$

$$c_{1,2}=(\alpha_{1,0}\cdot\alpha_{1,1}\cdot\alpha_{1,2}\cdot\alpha_{1,5}\cdot\alpha_{1,6}\cdot\alpha_{1,7})$$

$$c_{1,3}=(\alpha_{1,0}\cdot\alpha_{1,2}\cdot\alpha_{1,4}\cdot\alpha_{1,5}\cdot\alpha_{1,7}\cdot\alpha_{1,9})$$

$$c_{1,4}=(\alpha_{1,0}\cdot\alpha_{1,1}\cdot\alpha_{1,2}\cdot\alpha_{1,3}\cdot\alpha_{1,4}\cdot\alpha_{1,5}\cdot\alpha_{1,7}\cdot\alpha_{1,8}\cdot\alpha_{1,9});$$

and for i = 2 (refer Table 2), we get

$$c_{2,0}=(\alpha_{2,0}\cdot\alpha_{2,9});$$

$$c_{2,1}=(\alpha_{2,0}\cdot\alpha_{2,2}\cdot\alpha_{2,4}\cdot\alpha_{2,6}\cdot\alpha_{2,8}\cdot\alpha_{2,9})$$

$$c_{2,2}=(\alpha_{2,0}\cdot\alpha_{2,3}\cdot\alpha_{2,4}\cdot\alpha_{2,7}\cdot\alpha_{2,8}\cdot\alpha_{2,9})$$

$$c_{2,3}=(\alpha_{2,0}\cdot\alpha_{2,5}\cdot\alpha_{2,6}\cdot\alpha_{2,7}\cdot\alpha_{2,8}\cdot\alpha_{2,9})$$

$$c_{2,4}=(\alpha_{2,0}\cdot\alpha_{2,1}\cdot\alpha_{2,2}\cdot\alpha_{2,3}\cdot\alpha_{2,4}\cdot\alpha_{2,5}\cdot\alpha_{2,6}\cdot\alpha_{2,7}\cdot\alpha_{2,8}).$$

The functional arrangements for the computation of $\lvert c_1 \rvert_{17}$ in its binary representations ($c_{1,4}$, $c_{1,3}$, $c_{1,2}$, $c_{1,1}$, $c_{1,0}$) is shown in Figs. 2. Similarly $\lvert c_2 \rvert_{17}$

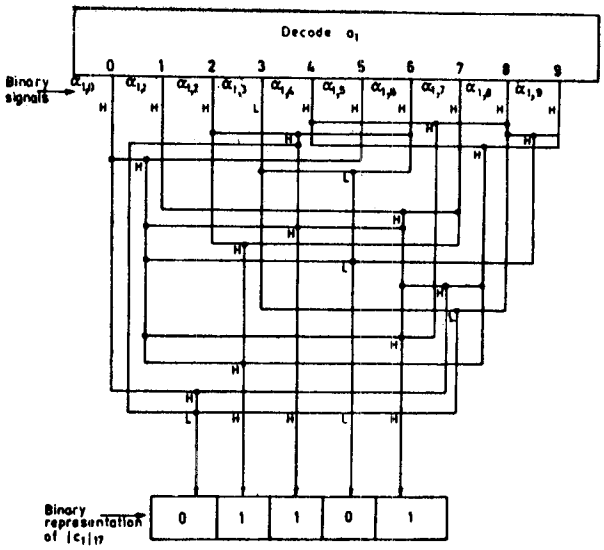

Fig. 2 Translation circuit for computing $|c_1|_{17}$

can be computed.

We use semiconductor logic gates in the decoders of the above design. It is also assumed that the decoders produce a 'Low' on the selected output line and a 'H8gh' on all the others.

CONVERSION FROM RESIDUE REPRESENTATION TO FIXED RADIX REPRESENTATION

Here we consider the output conversion problem, i.e., the conversion of a number X from its residue representation $(x_1,x_2,\ldots,x_n)$ with respect to the moduli $m_1,m_2,\ldots,m_n$ to its fixed radix representation or a base b representation. This problem was also considered earlier by Aiken and Semon[2] and Banerji[4,5]. The method proposed by Aiken and Semon is based on computing an auxiliary function A(X). The computation of A(X) requires introducing a redundant modulus (called the 'Super modulus'). In addition to this redundancy, this method requires more modular operations in general than those in the method suggested by Banerji and Brzozowski[5]. The latter method is based on the standard algorithm for conversion of an integer from one base to

another[3] and requires only the residue arithmetic operations.

Description of the Proposed Method

Our method is also based on the standard algorithm for conversion of an integer from one base to another. It requires less number of Base Extension Operations (BEO) as compared to those required for the method suggested by Banerji and Brzozowski[4] but with an increase in hardware complexity. The method is described as follows:

Consider the RNS of n pairwise relatively prime moduli $m_1, m_2, \ldots m_n$ and let X be any integer in the range, $0 \leq X < M$, where $M = \prod_{i=1}^{n} m_i$. We want to convert $X \longleftrightarrow (x_1, x_2, \ldots, x_n)$ into fixed radix representation (base b representation).

$$\text{Now} \quad X = \sum_{i=0}^{p-1} a_i . b^i \ .$$

$$\text{Then } |X|_{bt} = \Big| \sum_{i=0}^{p-1} a_i . b^i \Big|_{bt}, \ 1 \leq t < p;$$

$$= \sum_{i=0}^{t-1} a_i . b^i \ .$$

It is assumed that all the residues are binary coded. The binary representation of the least significant 't' base b digits ($|X|_{bt}$) is obtained by using the BEO[3].

Now let,

$$X(1) = \frac{X - |X|_{bt}}{b^t} = \sum_{i=t}^{p-1} a_i . b^{i-t} \quad (1)$$

$$\text{Then} \quad |X(1)|_{bt} = \Big| \sum_{i=t}^{p-1} a_i . b^{i-t} \Big|_{bt} \ ,$$

$$= \sum_{i=t}^{2t-1} a_i . b^{i-t} \ .$$

Use the BEO on the residues of X(1) (obtained from equation (1)) to compute the binary representation of $|X(1)|_{bt}$.

In general,

$$X(j) = \frac{X(j-1) - |X(j-1)|_{bt}}{b^t} \ ,$$

for $j = 1, 2, \ldots, \lfloor \frac{p-1}{t} \rfloor$ and $X(0) = X$.

$$\text{Then } |X(j)|_{bt} = \sum_{i=j.t}^{(j+1).t-1} a_i . b^{i-j.t} .$$

We see that the computation of $|X|_{bt}$ gives the binary representation of 't' least significant digits of X, the computation of $|X(1)|_{bt}$ gives the binary representation of next least significant 't' base b digits of X and so on. Thus we get the sets of 't' base b digits of X (in binary code) which can be converted simultaneously to base b representation by combinational logic.

The computation of the base b digits from $(x_1, x_2, \ldots, x_n)$ depends upon the relationship between $_bt$ and the moduli $m_1, m_2, \ldots, m_n$ as suggested by Banerji[4].

COMPARISON OF CONVERSION SCHEMES

In this section, we compare the proposed schemes for conversion into and out of residue representation with the existing ones.

Comparison of Input Conversion

Let T be the time required to compute $|X|_m$, where X is a p-digit base b number. Let T_1 be the time required to compute $|c_i|_m$ and T_2 be the time required to add all

$$|c_i|_m, i = 0, 1, 2, \ldots, p-1.$$

Then $\quad T = T_1 + T_2$.

If we use only one residue adder for one modulus, then

- $T = \sigma + (p-1).t_a$ - using table look up for $|c_i|_m$,
- $T = 2\Delta + (p-1).t_a$ - using combinational logic for computing $|c_i|_m$,
- $T = (p.\lceil \log_2 b \rceil^* - 1).t_a$ - using the method proposed by Banerji[5],
- $T = \lambda + (p-1).t_a$ - using the proposed scheme.

Where t_a denotes the time for one addition cycle, σ is the read only memory cycle time, λ is the time required to decode a base b digit and Δ is a single gate delay.

---

* $\lceil I \rceil$ denotes the ceiling of I, i.e., smallest integer $\geq I$.

To get clear idea about the magnitude of T, consider σ = 40 ns. Δ = 10 ns, b = 10 and p = 6. Assume $t_a = 3\Delta$ (adder of the method proposed by Banerji[5], assuming a single level decode logic). Let $\lambda = \Delta$ = 10 ns.

Then

$T = \sigma + (p-1).t_a = 0.19$ μs. - using table look up for $|c_i|_m$,

$T = 2\Delta + (p-1).t_a = 0.17$ μs. - using combinational logic for computing $|c_i|_m$,

$T = (p.\lceil \log_2 b \rceil - 1).t_a = 0.69$ μs. - using the method proposed by Banerji[5],

$T = \lambda + (p-1).t_a = 0.16$ μs. - using the proposed scheme.

We see that T depends on the number of residue adders used.

If we assume a decoder of single gate delay, then our scheme turns out to be faster than the others.

The number of gates required for computing $|c_i|_m, i \geq 0$ is the same as the number of gates required in the decoder to decode a base b digit a_i.

Comparison of Output Conversion

Let G and G_t be the number of modular operations required by the method suggested by Banerji[4] and the method presented here respectively. Now X be represented in base b representation by p digits. The number of modular operations required to compute a_0 are 2n+1. Further, 2 modular operations are required to compute $\frac{X-a_0}{b}$. Hence the total number of modular operations are

$$G = 2n+2(n-k_1)+\ldots+2(n-k_{p-1})+3p-2,$$

$$= 2n.p-2\left(\sum_{i=1}^{p-1} k_i\right) + 3p-2,$$

where k_i are the number of the moduli whose product is less than b at the time of computing a_i digit. The number of modular operations required to compute a_i digit are $2(n-k_i)+1$.

In the proposed method, (2n+1) modular operations are required to compute 't' least significant digits. Note that the residues are encoded in the binary code. So we obtain the binary code of 't' least significant digits. Excluding j_1 moduli whose product is less than b^t, the next t least significant digits are computed as suggested earlier. The number of modular operations required are $2(n-j_1)+1$. Similarly let $j_{\lfloor (p-1)/t \rfloor}$ be those moduli whose product is less than b^t at the time of computing the last 't_1' base be digits, where $t_1 \leq t$. So the total number of modular operations required to compute p digits are

$$G_t = 2n+2(n-j_1)+\ldots+2(n-j_{\lfloor (p-1)/t \rfloor}) + 2\{\lfloor (p-1)/t \rfloor\} + \lfloor (p-1)/t \rfloor + 1,$$

$$= 2n.\{1+\lfloor (p-1)/t \rfloor\} - 2(j_1+j_2+\ldots+j_{\lfloor (p-1)/t \rfloor}) + 3\lfloor (p-1)/t \rfloor + 1.$$

Table 3 shows the number of modular operations for different values of t, in the case when p = 8, n = 11 and b =10. Assume $j_i = i.t$, $i = 1,2,\ldots,\lfloor (p-1)/t \rfloor$, and $k_i = i$, $i = 1,2,\ldots,p-1$.

Then

$$G = 2 \times 11 \times 8 - 2\left(\sum_{i=1}^{7} i\right) + 24-2,$$

$$= 176-56+22 = 142.$$

Table 3

Value of t	Number of modular operations G_t
2	74
3	55
4	40
5	38
6	36
7	34

We observe from the Table 3 that for t = 2 and 3 the reduction in the number of modular operations is maximum.

We also see that the modular operations for t = 2 (or 3) are reduced to nearly (or less than) half of those required by the method suggested by Banerji[4], but the hardware complexity increases, since the proposed method requires an implementation of mod b^t subtractors and multipliers whereas his method requires an implementation of mod b subtractors and multipliers. Thus the proposed method certainly increases the speed of output conversion process but at the same time increases the hardware complexity.