Residue Number Systems
Ivor Page[1]

5.1 Arithmetic in a modulus system

The great speed of arithmetic in Residue Number Systems (RNS) comes from a simple theorem from number theory:

$$x \Box y \ mod \ m = (x \ mod \ m \Box y \ mod \ m) \ mod \ m$$

where the binary operator $\Box \in \{+, -, \times\}$.

$$\begin{aligned} \text{For example } (x + y) \ mod \ m &= ((x \ mod \ m) + (y \ mod \ m)) \ mod \ m) \\ \text{and } (x \times y) \ mod \ m &= ((x \ mod \ m) \times (y \ mod \ m)) \ mod \ m) \end{aligned}$$

Say we want to compute the sum $(187 + 259) \ mod \ 10$. That is, we only need the least significant decimal digit of the answer. A simple strategy would be to add the least significant digits of the two operands, $7 + 9$, and extract the least significant digit of the result. i.e.

$$(187 + 259) \ mod \ 10 = (187 \ mod \ 10 + 259 \ mod \ 10) \ mod \ 10 = 6$$

In this example, the two operands were represented (incompletely) by their *residues* relative to the modulus 10.

A RNS is defined by a vector of k moduli, $(m_1, m_2, \cdots, m_k)$. The moduli must be pairwise co-prime, which means that, for any pair of moduli, the only common factor is 1. In a RNS each operand is represented by a list of its residues, one for each modulus. For example, if we use $RNS(8, 7, 5, 3)$ (as in the text) the two operands from the previous sum become: $187 \rightarrow (3, 5, 2, 1)$ and $259 \rightarrow (3, 0, 4, 1)$. The sum of the two RNS values proceeds in each column independently. For example, in the third column, the result is $2 + 4 \ mod \ 5 = 1$. The complete result is $(6, 5, 1, 2)$. We shall see how to convert the answer back to decimal or binary notation later. For the moment, let's work back from the correct answer obtained using decimals: $187 + 259 = 446 \rightarrow (6, 5, 1, 2)$.

The potential for high speed arithmetic comes from the ability to calculate the result in each column independent of other columns, and therefore in parallel. There is no carry propagation between the columns. Within each column the residues will most likely be represented by unsigned binary integers. If a particular column requires p bits to store its residues, a p bit adder (or a table-lookup scheme) will be needed. The speed of that column adder is determined by p. The speed of RNS arithmetic is therefore (partly) determined by the largest column width.

[1]University of Texas at Dallas

5.1.1 Range

A RNS with moduli $(m_1, m_2, m_3, \cdots, m_k)$ has *dynamic range* $m_1 \times m_2 \times m_3 \cdots \times m_k$.

For example, $RNS(3, 2)$ can represent 6 values. This dynamic range can be used to represent any interval of 6 consecutive values on the number line. The interval $[0, 5]$ is an obvious choice. Note that the decimal values $0, 6, 12, 18, 24, \cdots$ all have representations (0,0) in this RNS. Similar ambiguities exist in any fixed length number representation. In an 8 bit unsigned binary system, the values $0, 256, 512, \cdots$ all have the same representation.

$RNS(3, 2)$ can also represent the range $[-3, 2]$ if a bias of 3 is added to each operand before conversion to RNS and the appropriate adjustments are made to the final result during conversion back to 2's complement. It is better, however, to map the positive values without the addition of a bias The negative values that result from this approach have an important property:

Decimal	RNS
-3	(0,1)
-2	(1,0)
-1	(2,1)
0	(0,0)
1	(1,1)
2	(2,0)

Each positive value is represented without the addition of a bias and each negative value is represented by the complement of its magnitude. To complement (r_1, r_2), subtract each residue from its modulus. In the special case where $r_i = 0$ is to be complemented, the answer is 0 since $m_i - 0 = m_i$, which becomes 0 after the application of the modulus m_i. The negative of (1,1) is (2,1), and the negative of (2,0) is (1,0).

5.1.2 Speed

The speed of the arithmetic operations within the RNS is dependent on the arithmetic process and the maximum residue size. Assume that residues are stored as unsigned binary values. This is appropriate since, within the RNS, all the residues are positive. In the scheme from the text, $RNS(8, 7, 5, 3)$, the column widths are 3, 3, 3, and 2 bits respectively. The speed of arithmetic operations is determined by the width of the largest residue.

Speed is also affected by the arithmetic processes employed. For example, we could use adders in each column or we could use ROMs and get the results by table-lookup.

Using ROMs

In $RNS(8, 7, 5, 3)$ The ROMs for addition would have 64, 64, 64, and 16 entries respectively, and they would hold results of widths 3, 3, 3, and 2 bits. ROMS of the same dimensions

would be needed for multiplication. Only two sets of ROMs are needed since subtraction is performed by negation of the subtrahend followed by addition. Figure 1 shows the schematic of a 16 bit ROM. In CMOS, the cells would be implemented using transistors rather than diodes and the X-decode and Y-select circuits would probably use a clocking scheme to support pre-charging of the lines within the matrix.

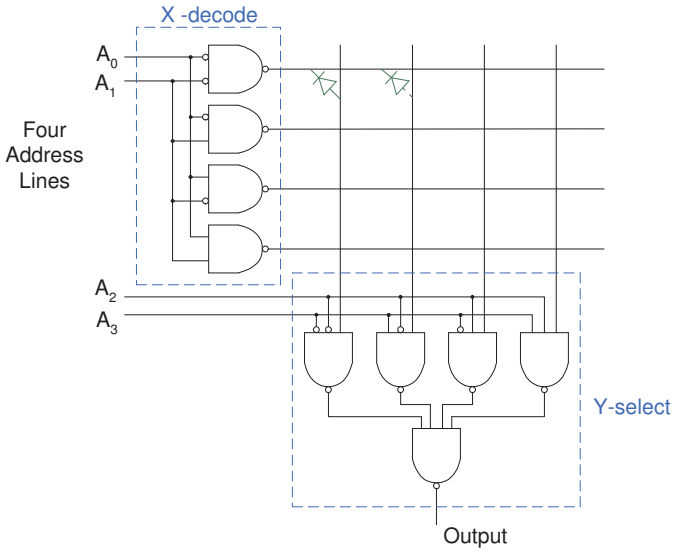

Figure 1: 16-Bit ROM (4 Address Lines)

The dimensions of a ROM determine its speed. The X-decoder will, in general, be a log-depth gate configuration since fan-in is limited. The Y-selector has two components illustrated by the two layers of gates in Figure 1. Both layers will, in general, be log-depth gate configurations. The delay components for a ROM with L address lines and gates with maximum fan-in r are as follows:

$$\begin{aligned} \text{X-decode} &= D\left\lceil log_r \left(\frac{L}{2}\right)\right\rceil \\ \text{Diode Matrix} &= D \\ \text{Y-select} &= D\left\lceil log_r \left(\frac{L}{2}+1\right)\right\rceil + \lceil log_r\, 2^{L/2}\rceil \end{aligned}$$

Here, we assume unit gate delay for the cell matrix. The X-decoder in general will require $\lceil log_r\ (L/2)\rceil$ layers of gates. Similarly, the first row of gates in the Y-seletor will be replaced by $\lceil log_r\ (L/2+1)\rceil$ layers, and the final output gate will be replaced by $\lceil log_r\ 2^{(L/2)}\rceil = \lceil (L/2) log_r\ 2\rceil$ layers of gates.

For large L, the final term dominates, but it is unlikely that we would use moduli greater

that 100, and correspondingly values of L greater than 7 for schemes implementing up to 64 bit arithmetic.

Examples:

The following table summarizes delays for small ROM sizes:

Width of Residue Column	ROM delay
5	5D
6	5D
8	6D
10	8D
12	8D

The delays are small and apply to addition, subtraction, and multiplication. Modern CMOS ROM circuits have even better delay characteristics, although log-depth seems inevitable in the address decoder(s).

Using Adders

If adders are used, we must find a fast way to take the modulus of the intermediate result in each column. If the modulus is a power of 2, say 8, an unsigned 3-bit adder will automatically take the modulus during the addition process. Similarly, if the modulus is one less than a power of 2, say 7, then a 3-bit 1's complement adder with end-around carry will automatically take the modulus during the addition process. In the case of modulus 5, or any modulus that isn't of form 2^p or 2^p-1, a more complex and time consuming process is needed. For addition in such a column with modulus m_i, the result r after addition is in the range $[0, 2m_i - 2]$. It would be necessary to detect the cases where $r \geq m_i$ and subtract m_i from the result.

In the case of multiplication, the column result is in the range $[0, (m_i - 1)^2]$. Taking the modulus here appears to be more of a challenge. A ROM approach would be better suited to these moduli.

If adders are to be used, the moduli must be of form 2^p or $2^p - 1$. These are known as *low-cost moduli*. Moduli should be carefully chosen to cover the desired numeric range such that the column widths of the RNS system are similar. i.e. we want to minimize the width of the largest column.

5.1.3 Efficiency

In $RNS(8, 7, 5, 3)$ above, 11 bits were needed for a dynamic range of only 804. An 11 bit unsigned binary representation (or an 11 bit 2's complement representation) can represent $2^{11} = 2048$ different values. The efficiency of the RNS above is only $804/2048 = 39.3\%$. Care in choosing the moduli can greatly improve the efficiency of the system.

5.1.4 Conversion from 2's Complement to RNS

Conversion from 2's complement to a RNS simply requires conversion of the magnitude followed by negation of the result if the operand being converted is negative. For example, say an 8-bit 2's complement system was to be converted to a suitable RNS. The range of this system is $[-128, 127]$, i.e. 256 different values. The RNS would need at least this range. The system $RNS(16, 7, 3)$ covers 336 values and has column widths 4, 3, and 2 bits, making a total of 9 bits with an efficiency of 50%.

The following table shows some examples of values in the representation:

Decimal	Binary 2's complement	$RNS(16, 7, 3)$
-128	10000000	(0,5,1)
-127	10000001	(1,6,2)
-126	10000010	(2,0,0)
-1	11111111	(15,6,2)
0	00000000	(0,0,0)
1	00000001	(1,1,1)
126	01111110	(14,0,0)
127	01111111	(15,1,1)

Since we are not using the entire dynamic range of values of the RNS, the addition of 1 to the most positive value represented (127+1) does not yield the most negative value represented (-128). This suggests that overflow detection may not be easy (see later).

We are still left with the task of applying the modulus to the magnitude of the binary number being converted. This process is simplified if a table is kept of the residues of the powers of 2 within the required positive range. For example, in $RNS(16, 7, 3)$, we would need a table of the RNS representations of $1, 2, 4, \cdots 64$, i.e. 7 entries. Then, to convert the binary value 01011001, we add the RNS values corresponding to 1's in the binary value. These additions are done using the RNS adder.

Here is the table for $RNS(16, 7, 3)$:

Decimal	$RNS(15, 7, 3)$
1	(1,1,1)
2	(2,2,2)
4	(4,4,1)
8	(8,1,2)
16	(0,2,1)
32	(0,2,2)
64	(0,1,1)

To convert the value $89_{10} = 01011001_2$, we compute the following sum in the RNS adder: (0,1,1) + (0,2,1) + (8,1,2) + (1,1,1) = (9,5,2).

For a k bit 2's complement value, the number of entries in the conversion table is $k-1$ and the worst case number of additions is $k-2$.

5.1.5 Choice of Moduli

As mentioned earlier, the moduli are chosen to maximize efficiency (minimize the total number of bits in the RNS system) and to simplify arithmetic operations. The latter requirement leads to the constraint that each modulus should be low-cost, i.e. of form 2^p or $2^p - 1$. Two approaches are given. One leads to the optimal efficiency using unrestricted moduli, while the other uses only low-cost moduli.

Unrestricted Moduli

In the first approach, the moduli can be primes or powers of primes or products of these. We begin by using all the primes in sequence until the desired range is achieved.

To cover the range of a 16 bit binary system, a dynamic range of 65,536 is needed. $RNS(17, 13, 11, 7, 5, 3, 2)$ gives dynamic range 510,510, 7.78 times what is needed. The number of bits needed is 5+4+4+3+3+2+1 = 22, and the efficiency is less than 1%. There are many possible improvements. We could remove the 7, giving $RNS(17, 13, 11, 5, 3, 2)$, which has dynamic range 72,930, 1.11 times more than needed, and efficiency 25%.

It would be preferable to remove the 17 if possible, since it is the only value that implies the (widest) column width of 5. $RNS(13, 11, 7, 5, 3, 2)$ has dynamic range 30,030, 0.458 of what is needed. Squaring the 3 gives $RNS(13, 11, 7, 5, 9, 2)$, which has dynamic range 90,090, 1.37 times what is needed and efficiency less than 1%. Cubing the 2 and removing the 5, gives $RNS(13, 11, 7, 9, 8)$, which has dymanic range 72,072, 1.0999 times what is needed, and efficiency 0.25%. We can combine the 7 and 9 (since the 9 is inefficient) to get 63: $RNS(13, 11, 63, 8)$. This has the same dynamic range as the previous version, 1.0999 times what is needed, but the efficiency is 50%. However, the widest column is now 6 bits.

Summary so far:

Moduli	Dynamic Range	Excess Range	Widest Column	Efficiency
(17,13,11,7,5,3,2)	510,510	7.78	5	<1%
(17,13,11,5,3,2)	72,930	1.11	5	25%
(13,11,7,5,9,2)	90,090	1.37	4	<1%
(13,11,7,9,8)	72,072	1.1	4	25%
(13,11,63,8)	72,072	1.1	6	50%

A strategy emerges:

- Keep all moduli less than 17 so the maximum column width is 4
- Raise primes to powers and/or combine them by multiplication to get moduli that are just less than or equal to powers of 2.
- Remove values from the list of candidate moduli and use step 2 to compensate.

Higher efficiencies may be obtained in general if the first constraint is relaxed. There is a tradeoff between efficiency and maximum column width.

If the maximum column width is to be kept to 4 bits, the moduli must be selected from the values in the following table:

2 4 8 16	3 9	5	7	11	13
6	10	12	14	15	

There are only 15 values and we are only interested in products of pairwise co-prime moduli that exceed the target value 65,536.

The constraint that the moduli must be pairwise co-prime can be represented in a table as follows:

2	4	6	8	10	12	14	16
3	6	9	12	15			
5	10	15					
7	14						
11							
13							

If any value from row i of this table is used as a modulus, then no other value from that row can be used. Some values appear in more than one row. Using these techniques, an exhaustive search of all possible solutions can be programmed and the optimum set of moduli found.

As mentioned before, higher efficiency might be achieved if the maximum column width was allowed to increase to 5 bits.

Low-Cost Moduli

When restricted to low-cost moduli, we get

$$RNS(2^{a_{k-2}}, 2^{a_{k-2}} - 1, 2^{a_{k-3}} - 1, \cdots, 2^{a_1} - 1, 2^{a_0} - 1)$$

Two values $2^a - 1$ and $2^b - 1$ are co-prime if and only if a and b are co-prime.

First, form a sequence as follows:

RNS	Basis	Dynamic Range
$RNS(2^3, 2^3 - 1, 2^2 - 1)$	3,2	168
$RNS(2^4, 2^4 - 1, 2^3 - 1)$	4,3	1680
$RNS(2^5, 2^5 - 1, 2^3 - 1, 2^2 - 1)$	5,3,2	20,832
$RNS(2^5, 2^5 - 1, 2^4 - 1, 2^3 - 1)$	5,4,3	104,160

The final system, $RNS(32, 31, 15, 7)$, has dynamic range 1.59 times what is needed, but has efficiency 50%. Its widest column is 5 bits, but the speed of arithmetic is greatly helped by using low-cost moduli.

Once sufficient dynamic range is obtained, the problem of selecting the lower order terms can be tackled as follows: Say that we have established that the largest two moduli are 2^b and $2^b - 1$. Candidates for the lower order moduli are $2^{b-1} - 1, 2^{b-2} - 1, \cdots, 7, 3$. We form a $b - 2$ bit binary number Q to represent the presence of these terms such that bit i of this number is a 1 if $2^i - 1$ is included in the list of moduli. For example, if the largest two moduli are 128 and 127, then this number Q has 5 bits. These bits represent the presence of values 63, 31, 15, 7, and 3. All possible sets of low-cost moduli with largest values 128, 127 are found by counting through all possible values of Q, excluding those values where any pair of bits 0, 2, and 4 are 1. That eliminates 14 of the 32 possible sets of moduli.

5.1.6 Conversion from RNS to 2's Complement

There are two methods of conversion. The first makes use of an intermediate mixed-radix representation, while the second is a direct conversion method based on the Chinese Remainder Theorem.

Mixed Radix Form

Associated with $RNS(m_{k-1}, \cdots, m_1, m_0)$ is a mixed radix system, $MRS(m_{k-1}, \cdots, m_1, m_0)$, with column weights

$$m_{k-2} \times \cdots \times m_1 \times m_0, \quad \cdots, \quad m_2 \times m_1 \times m_0, \quad m_1 \times m_0, \quad m_0, \quad 1$$

and digit sets $[0, m_{k-1} - 1], \cdots, [0, m_2 - 1], [0, m_1 - 1], [0, m_0 - 1]$ in its k digit positions, exactly the same as in the corresponding RNS. For example, $MRS(8, 7, 5, 3)$ has column

weights $7 \times 5 \times 3$, 5×3, 3, and 1 = 105, 15, 3, 1.

$$(0, 3, 1, 0)_{MRS} = (0 \times 105) + (3 \times 15) + (1 \times 3) + (0 \times 1) = 48$$

To convert from RNS to MRS, we have to find the digits z_i so that

$$y = (x_{k-1}, \cdots, x_1, x_0)_{RNS} = (z_{k-1}, \cdots, z_1, z_0)_{MRS}$$

From the definition of MRS,

$$y = z_{k-1}(m_{k-2} \cdots m_2 m_1 m_0) + \cdots + z_2(m_1 m_0) + z_1(m_0) + z_0$$

Clearly $z_0 = x_0$. Subtracting this equation, we get:

$$y' = y - x_0 = (x'_{k-1}, \cdots, x'_2, x'_1, 0)_{RNS} = (z_{k-1}, \cdots, z_1, 0)_{MRS}$$

where $x'_j = (x_j - x_0)\ mod\ m_j$. If we divide both representations by m_0 we get the following:

$$(x''_{k-1}, \cdots x''_2, x''_1)_{RNS} = (z_{k-1}, \cdots, z_2, z_1)_{MRS}$$

The process is applied iteratively to yield the result. How do we divide a RNS number by the modulus m_0?

Dividing the RNS number y' by m_0 is known as *scaling* and is much simpler than general division in RNS. Division by m_0 is achieved by multiplying each residue by the *multiplicative inverse* of m_0 relative to the modulus in that column. The multiplicative inverses of 3 relative to moduli 8, 7, and 5 are 3, 5, and 2 respectively since:

$$(3 \times 3) mod 8 = (3 \times 5) mod 7 = (3 \times 2) mod 5 = 1$$

The number $(0, 6, 3, 0)_{RNS}$ can be divided by 3 by multiplication by $(3, 5, 2, 0)_{RNS}$, giving $(0, 2, 1, -)_{RNS}$.

Multiplicative inverses can be precomputed and stored in tables.

Example:

Convert $y = (0, 6, 3, 0)$ from $RNS(8, 7, 5, 3)$ to mixed-radix representation.

$z_0 = x_0 = 0$. Dividing by $m_0 = 3$ is achieved by multiplying by $(3, 5, 2, 0)_{RNS}$, giving $(0, 2, 1, -)_{RNS}$.

Therefore $z_1 = 1$. Subtracting 1 and dividing by 5 gives:

$$\frac{(7, 1, 0, -)_{RNS}}{5} = (7, 1, 0, -)_{RNS} \times (5, 3, -, -)_{RNS} = (3, 3, -, -)_{RNS}$$

So $z_2 = 3$. Subtracting 3 and dividing my 7, we get:

$$\frac{(0, 0, -, -)_{RNS}}{7} = (0, 0, -, -)_{RNS} \times (7, -, -, -)_{RNS} = (0, -, -, -)_{RNS}$$

$z_3 = 0$.

$$y = (0, 6, 3, 0)_{RNS} = (0, 3, 1, 0)_{MRS} = 48$$

We can use mixed-radix form to compare the magnitudes of two RNS values or to detect the sign of a RNS number. These operations are impossible within the RNS.

Using the Chinese Remainder Theorem (CRT)

Consider the conversion of $y = (3, 2, 4, 2)$ in $RNS(8, 7, 5, 3)$ to decimal.

We can split up the number into one RNS value per column as follows:

$$\begin{aligned}(3,2,4,2)_{RNS} &= (3,0,0,0)_{RNS} + (0,2,0,0)_{RNS} + (0,0,4,0)_{RNS} + (0,0,0,2)_{RNS} \\ &= 3 \times (1,0,0,0) + 2 \times (0,1,0,0) + 4 \times (0,0,1,0) + 2 \times (0,0,0,1)\end{aligned}$$

Knowing the decimal (or binary) equivalents of the following RNS values would easily enable us to do the conversion.

$$\begin{aligned}(1,0,0,0) &= 105 \\ (0,1,0,0) &= 120 \\ (0,0,1,0) &= 336 \\ (0,0,0,1) &= 280\end{aligned}$$

So, $(3, 2, 4, 2)_{RNS} = ((3 \times 105) + (2 \times 120) + (4 \times 336) + (2 \times 280)) \, mod 840 = 779$

Once more, a table with precomputed values is used to aid the conversion. How are these values obtained?

Chinese Remainder Theorem:

$$x = (x_{k-1}, \cdots, x_2, x_1, x_0)_{RNS} = \left(\sum_{i=0}^{k-1} M_i(\alpha_i x_i \ mod \ m_i) \right) \ mod \ M$$

where, $M_i = M/m_i$, $\alpha_i = M_i^{-1} \ mod \ m_i$ is the multiplicative inverse of M_i with respect to m_i, and M is the product of the moduli.

Multiplication can be avoided if tables are kept in each column for the conversion of every possible residue value. In the following table, $T_i = (M_i(\alpha_i x_i \ mod \ m_i) \ mod \ M$

i	m_i	x_i	T_i
3	8	0	0
		1	105
		2	210
		3	315
		4	420
		5	525
		6	630
		7	735
2	7	0	0
		1	120
		2	240
		3	360
		4	480
		5	600
		6	720

i	m_i	x_i	T_i
1	5	0	0
		1	336
		2	672
		3	168
		4	504
0	3	0	0
		1	280
		2	560