Residue Number Systems

"God created the integers, all else is the work of man"
LEOPOLD KRONECKER, 1886

By converting arithmetic on large numbers to arithmetic on a collection of smaller numbers, residue number system (RNS) representations produce significant speedup for some classes of arithmetic-intensive algorithms in signal processing applications. Additionally, RNS arithmetic is a valuable tool for theoretical studies of the limits of fast arithmetic. In this chapter, we study RNS representations and arithmetic, along with their advantages and drawbacks. Chapter topics include:

4.1 RNS REPRESENTATION AND ARITHMETIC

What number has the remainders of 2, 3, and 2 when divided by the numbers 7, 5, and 3, respectively? This puzzle, written in the form of a verse by the Chinese scholar Sun Tsu more than 1500 years ago [Jenk93], is perhaps the first documented use of number representation using multiple residues. The puzzle essentially asks us to convert the coded representation (2|3|2) of a residue number system, based on the moduli (7|5|3), into standard decimal format.

In a residue number system (RNS), a number x is represented by the list of its residues with respect to k pairwise relatively prime moduli $m_{k-1} > \cdots > m_1 > m_0$. The residue x_i of x with respect to the ith modulus m_i is akin to a digit and the entire k-residue representation of x can be viewed as a k-digit number, where the digit set for the ith position is $[0, m_i - 1]$. Notationally, we write

$$x_i = x \bmod m_i = \langle x \rangle_{m_i}$$

and specify the RNS representation of x by enclosing the list of residues, or digits, in parentheses. For example,

$$x = (2|3|2)_{\text{RNS}(7|5|3)}$$

represents the puzzle given at the beginning of this section. The list of moduli can be deleted from the subscript when we have agreed on a default set. In many of the examples of this chapter, the following RNS is assumed:

$$\text{RNS}(8|7|5|3) \qquad \text{Default RNS for Chapter 4}$$

The product M of the k pairwise relatively prime moduli is the number of different representable values in the RNS and is known as its *dynamic range*.

$$M = m_{k-1} \times \cdots \times m_1 \times m_0$$

For example, $M = 8 \times 7 \times 5 \times 3 = 840$ is the total number of distinct values that are representable in our chosen 4-modulus RNS. Because of the equality

$$\langle -x \rangle_{m_i} = \langle M - x \rangle_{m_i}$$

the 840 available values can be used to represent numbers 0 through 839, -420 through $+419$, or any other interval of 840 consecutive integers. In effect, negative numbers are represented using a complement system with the complementation constant M.

Here are some example numbers in RNS(8|7|5|3):

$(0 \mid 0 \mid 0 \mid 0)_{\text{RNS}}$ Represents 0 or 840 or $\cdots$

$(1 \mid 1 \mid 1 \mid 1)_{\text{RNS}}$ Represents 1 or 841 or $\cdots$

$(2 \mid 2 \mid 2 \mid 2)_{\text{RNS}}$ Represents 2 or 842 or $\cdots$

$(0 \mid 1 \mid 3 \mid 2)_{\text{RNS}}$ Represents 8 or 848 or $\cdots$

$(5 \mid 0 \mid 1 \mid 0)_{\text{RNS}}$ Represents 21 or 861 or $\cdots$

$(0 \mid 1 \mid 4 \mid 1)_{\text{RNS}}$ Represents 64 or 904 or $\cdots$

$(2 \mid 0 \mid 0 \mid 2)_{\text{RNS}}$ Represents -70 or 770 or $\cdots$

$(7 \mid 6 \mid 4 \mid 2)_{\text{RNS}}$ Represents -1 or 839 or $\cdots$

Given the RNS representation of x, the representation of $-x$ can be found by complementing each of the digits x_i with respect to its modules m_i (0 digits are left unchanged).

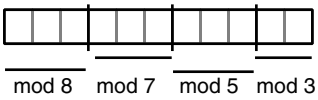

Figure 4.1 Binary-coded number format for RNS (8 | 7 | 5 | 3)

Thus, given that $21 = (5 \mid 0 \mid 1 \mid 0)_{\text{RNS}}$, we find

$$-21 = (8 - 5 \mid 0 \mid 5 \ - 1 \mid 0)_{\text{RNS}} = (3 \mid 0 \mid 4 \mid 0)_{\text{RNS}}$$

Any RNS can be viewed as a weighted representation. We will present a general method for determining the position weights (the Chinese remainder theorem) in Section 4.3. For RNS(8|7|5|3), the weights associated with the four positions are

$$105 \qquad 120 \qquad 336 \qquad 280$$

As an example, $(1 \mid 2 \mid 4 \mid 0)_{\text{RNS}}$ represents the number

$$\langle (105 \times 1) + (120 \times 2) + (336 \times 4) + (280 \times 0) \rangle_{840} = \langle 1689 \rangle_{840} = 9$$

In practice, each residue must be represented or encoded in binary. For our example RNS, such a representation would require 11 bits (Fig. 4.1). To determine the number representation efficiency of our 4-modulus RNS, we note that 840 different values are being represented using 11 bits, compared with 2048 values possible with binary representation. Thus, the representational efficiency is

$$840/2048 = 41\%$$

Since $\log_2 840 = 9.714$, another way to quantify the representational efficiency is to note that in our example RNS, about 1.3 bits of the 11 bits go to waste.

As noted earlier, the sign of an RNS number can be changed by independently complementing each of its digits with respect to its modulus. Similarly, addition, subtraction, and multiplication can be performed by independently operating on each digit. The following examples for RNS(8 | 7 | 5 | 3) illustrate the process:

$(5 \mid 5 \mid 0 \mid 2)_{\text{RNS}}$ Represents $x = +5$

$(7 \mid 6 \mid 4 \mid 2)_{\text{RNS}}$ Represents $y = -1$

$(4 \mid 4 \mid 4 \mid 1)_{\text{RNS}}$ $x + y : \langle 5 + 7 \rangle_8 = 4, \langle 5 + 6 \rangle_7 = 4,$ etc.

$(6 \mid 6 \mid 1 \mid 0)_{\text{RNS}}$ $x - y : \langle 5 - 7 \rangle_8 = 6, \langle 5 - 6 \rangle_7 = 6,$ etc.

(alternatively, find $-y$ and add to x)

$(3 \mid 2 \mid 0 \mid 1)_{\text{RNS}}$ $x \times y : \langle 5 \times 7 \rangle_8 = 3, \langle 5 \times 6 \rangle_7 = 2,$ etc.

Figure 4.2 depicts the structure of an adder, subtractor, or multiplier for RNS arithmetic. Since each digit is a relatively small number, these operations can be quite fast and simple in RNS. This speed and simplicity are the primary advantages of RNS arithmetic. In the case of addition, for example, carry propagation is limited to within a single

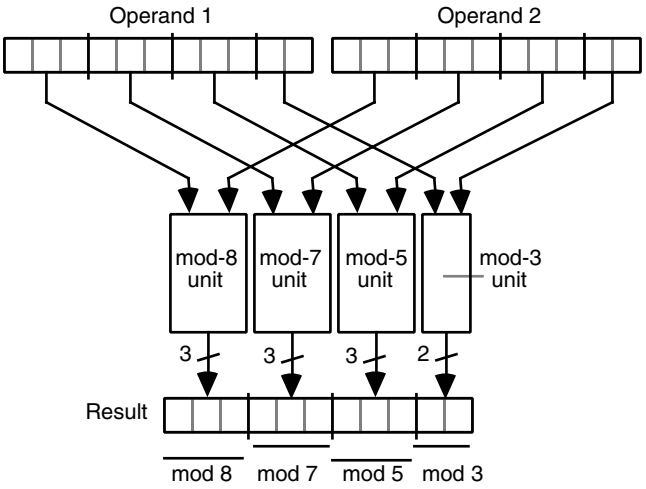

Figure 4.2 The structure of an adder, subtractor, or multiplier for RNS(8 | 7 | 5 | 3).

residue (a few bits). Thus, RNS representation pretty much solves the carry-propagation problem. As for multiplication, a 4×4 multiplier for example is considerably more than four times simpler than a 16×16 multiplier, besides being much faster. In fact, since the residues are small (say, 6 bits wide), it is quite feasible to implement addition, subtraction, and multiplication by direct table lookup. With 6-bit residues, say, each operation requires a $4K \times 6$ table. Thus, excluding division, a complete arithmetic unit module for one 6-bit residue can be implemented with 9 KB of memory.

Unfortunately, however, what we gain in terms of the speed and simplicity of addition, subtraction, and multiplication can be more than nullified by the complexity of division and the difficulty of certain auxiliary operations such as sign test, magnitude comparison, and overflow detection. Given the numbers

$$(7 \mid 2 \mid 2 \mid 1)_{\text{RNS}} \qquad \text{and} \qquad (2 \mid 5 \mid 0 \mid 1)_{\text{RNS}}$$

we cannot easily tell their signs, determine which of the two is larger, or find out whether $(1 \mid 0 \mid 2 \mid 2)_{\text{RNS}}$ represents their true sum as opposed to the residue of their sum modulo 840.

These difficulties have thus far limited the application of RNS representations to certain signal processing problems in which additions and multiplications are used either exclusively or predominantly and the results are within known ranges (e.g., digital filters, Fourier transforms). We discuss division and other "difficult" RNS operations in Section 4.4.

4.2 CHOOSING THE RNS MODULI

The set of the moduli chosen for RNS affects both the representational efficiency and the complexity of arithmetic algorithms. In general, we try to make the moduli as small as

possible, since it is the magnitude of the largest modulus m_{k-1} that dictates the speed of arithmetic operations. We also often try to make all the moduli comparable in magnitude to the largest one, since with the computation speed already dictated by m_{k-1}, there is usually no advantage in fragmenting the design of Fig. 4.2 through the use of very small moduli at the right end.

We illustrate the process of selecting the RNS moduli through an example. Let us assume that we want to represent unsigned integers in the range 0 to $(100\ 000)_{\text{ten}}$, requiring 17 bits with unsigned binary representation.

A simple strategy is to pick prime numbers in sequence until the dynamic range M becomes adequate. Thus, we pick $m_0 = 2, m_1 = 3, m_2 = 5$, etc. After we add $m_5 = 13$ to our list, the dynamic range becomes

$$\text{RNS}(13 \mid 11 \mid 7 \mid 5 \mid 3 \mid 2) \qquad M = 30\ 030$$

This range is not yet adequate, so we add $m_6 = 17$ to the list:

$$\text{RNS}(17 \mid 13 \mid 11 \mid 7 \mid 5 \mid 3 \mid 2) \qquad M = 510\ 510$$

The dynamic range is now 5.1 times as large as needed, so we can remove the modulus 5 and still have adequate range:

$$\text{RNS}(17 \mid 13 \mid 11 \mid 7 \mid 3 \mid 2) \qquad M = 102\ 102$$

With binary encoding of the six residues, the number of bits needed for encoding each number is

$$5 + 4 + 4 + 3 + 2 + 1 = 19 \text{ bits}$$

Now, since the speed of arithmetic operations is dictated by the 5-bit residues modulo m_5, we can combine the pairs of moduli 2 and 13, and 3 and 7, with no speed penalty. This leads to:

$$\text{RNS}(26 \mid 21 \mid 17 \mid 11) \qquad M = 102\ 102$$

This alternative RNS still needs $5 + 5 + 5 + 4 = 19$ bits per operand, but has two fewer modules in the arithmetic unit.

Better results can be obtained if we proceed as above, but include powers of smaller primes before moving to larger primes. The chosen moduli will still be pairwise relatively prime, since powers of any two prime numbers are relatively prime. For example, after including $m_0 = 2$ and $m_1 = 3$ in our list of moduli, we note that 2^2 is smaller than the next prime 5. So we modify m_0 and m_1 to get

$$\text{RNS}(2^2 \mid 3) \qquad M = 12$$

This strategy is consistent with our desire to minimize the magnitude of the largest modulus. Similarly, after we have included $m_2 = 5$ and $m_3 = 7$, we note that both 2^3

and 3^2 are smaller than the next prime 11. So the next three steps lead to

$$\begin{array}{ll} \text{RNS}(3^2 \mid 2^3 \mid 7 \mid 5) & M = 2520 \\ \text{RNS}(11 \mid 3^2 \mid 2^3 \mid 7 \mid 5) & M = 27\,720 \\ \text{RNS}(13 \mid 11 \mid 3^2 \mid 2^3 \mid 7 \mid 5) & M = 360\,360 \end{array}$$

The dynamic range is now 3.6 times as large as needed, so we can replace the modulus 9 with 3 and then combine the pair 5 and 3 to obtain

$$\text{RNS}(15 \mid 13 \mid 11 \mid 2^3 \mid 7) \qquad M = 120\,120$$

The number of bits needed by this last RNS is

$$4 + 4 + 4 + 3 + 3 = 18 \text{ bits}$$

which is better than our earlier result of 19 bits. The speed has also improved because the largest residue is now 4 bits wide instead of 5.

Other variations are possible. For example, given the simplicity of operations with power-of-2 moduli, we might want to backtrack and maximize the size of our even modulus within the 4-bit residue limit

$$\text{RNS}(2^4 \mid 13 \mid 11 \mid 3^2 \mid 7 \mid 5) \qquad M = 720\,720$$

We can now remove 5 or 7 from the list of moduli, but the resulting RNS is in fact inferior to RNS(15|13|11|2^3|7). This might not be the case with other examples; thus, once we have converged on a feasible set of moduli, we should experiment with other sets that can be derived from it by increasing the power of the even modulus at hand.

The preceding strategy for selecting the RNS moduli is guaranteed to lead to the smallest possible number of bits for the largest modulus, thus maximizing the speed of RNS arithmetic. However, speed and cost do not just depend on the widths of the residues but also on the moduli chosen. For example, we have already noted that power-of-2 moduli simplify the required arithmetic operations, so that the modulus 16 might be better than the smaller modulus 13 (except, perhaps, with table-lookup implementation). Moduli of the form $2^a - 1$ are also desirable and are referred to as low-cost moduli [Merr64], [Parh76]. From our discussion of addition of 1's-complement numbers in Section 2.4, we know that addition modulo $2^a - 1$ can be performed using a standard a-bit binary adder with end-around carry.

Hence, we are motivated to restrict the moduli to a power of 2 and odd numbers of the form $2^a - 1$. One can prove (left as exercise) that the numbers $2^a - 1$ and $2^b - 1$ are relatively prime if and only if a and b are relatively prime. Thus, any list of relatively prime numbers $a_{k-2} > \cdots > a_1 > a_0$ can be the basis of the following k-modulus RNS

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

for which the widest residues are a_{k-2}-bit numbers. Note that to maximize the dynamic range with a given residue width, the even modulus is chosen to be as large as possible.

Applying this strategy to our desired RNS with the target range [0, 100 000], leads to the following steps:

RNS $(2^3 \mid 2^3 - 1 \mid 2^2 - 1)$	Basis: 3, 2	$M = 168$
RNS $(2^4 \mid 2^4 - 1 \mid 2^3 - 1)$	Basis: 4, 3	$M = 1680$
RNS $(2^5 \mid 2^5 - 1 \mid 2^3 - 12^2 - 1)$	Basis: 5, 3, 2	$M = 20\,832$
RNS $(2^5 \mid 2^5 - 1 \mid 2^4 - 1 \mid 2^3 - 1)$	Basis: 5, 4, 3	$M = 104\,160$

This last system, RNS(32 | 31 | 15 | 7), possesses adequate range. Note that once the number 4 is included in the base list, 2 must be excluded because 4 and 2, and thus $2^4 - 1$ and $2^2 - 1$, are not relatively prime.

The derived RNS requires $5 + 5 + 4 + 3 = 17$ bits for representing each number, with the largest residues being 5 bits wide. In this case, the representational efficiency is close to 100% and no bit is wasted. In general, the representational efficiency of low-cost RNS is provably better than 50% (yet another exercise!), leading to the waste of no more than 1 bit in number representation.

To compare the RNS above to our best result with unrestricted moduli, we list the parameters of the two systems together:

RNS $(15 \mid 13 \mid 11 \mid 2^3 \mid 7)$	18 bits	$M = 120\,120$
RNS $(2^5 \mid 2^5 - 1 \mid 2^4 - 1 \mid 2^3 - 1)\vert$	17 bits	$M = 104\,160$

Both systems provide the desired range. The latter has wider, but fewer, residues. However, the simplicity of arithmetic with low-cost moduli makes the latter a more attractive choice. In general, restricting the moduli tends to increase the width of the largest residues and the optimal choice is dependent on both the application and the target implementation technology.

4.3 ENCODING AND DECODING OF NUMBERS

Since input numbers provided from the outside (machine or human interface) are in standard binary or decimal and outputs must be presented in the same way, conversions between binary/decimal and RNS representations are required.

Conversion from binary/decimal to RNS

The binary-to-RNS conversion problem is stated as follows: Given an integer y, find its residues with respect to the moduli $m_i, 0 \leq i \leq k-1$. Let us assume that y is an unsigned binary integer. Conversion of signed-magnitude or 2's-complement numbers can be accomplished by converting the magnitude and then complementing the RNS representation if needed.

To avoid time-consuming divisions, we take advantage of the following equality:

$$\langle (y_{k-1} \cdots y_1 y_0)_{\text{two}} \rangle_{m_i} = \langle \langle 2^{k-1} y_{k-1} \rangle_{m_i} + \cdots + \langle 2y_1 \rangle_{m_i} + \langle y_0 \rangle_{m_i} \rangle_{m_i}$$

Table 4.1 Precomputed residues of the first 10 powers of 2

j	2^j	$\langle 2^j \rangle_7$	$\langle 2^j \rangle_5$	$\langle 2^j \rangle_3$
0	1	1	1	1
1	2	2	2	2
2	4	4	4	1
3	8	1	3	2
4	16	2	1	1
5	32	4	2	2
6	64	1	4	1
7	128	2	3	2
8	256	4	1	1
9	512	1	2	2

If we precompute and store $\langle 2^j \rangle_{m_i}$ for each i and j, then the residue x_i of y (mod m_i) can be computed by modulo-m_i addition of some of these constants.

Table 4.1 shows the required lookup table for converting 10-bit binary numbers in the range [0, 839] to RNS(8 | 7 | 5 | 3). Only residues mod 7, mod 5, and mod 3 are given in the table, since the residue mod 8 is directly available as the three least-significant bits of the binary number y.

■ **EXAMPLE 4.1** Represent $y = (1010\ 0100)_{\text{two}} = (164)_{\text{ten}}$ in RNS(8 | 7 | 5 | 3). The residue of y mod 8 is $x_3 = (y_2 y_1 y_0)_{\text{two}} = (100)_{\text{two}} = 4$. Since $y = 2^7 + 2^5 + 2^2$, the required residues mod 7, mod 5, and mod 3 are obtained by simply adding the values stored in the three rows corresponding to $j = 7, 5, 2$ in Table 4.1:

$$x_2 = \langle y \rangle_7 = \langle 2 + 4 + 4 \rangle_7 = 3$$
$$x_1 = \langle y \rangle_5 = \langle 3 + 2 + 4 \rangle_5 = 4$$
$$x_0 = \langle y \rangle_3 = \langle 2 + 2 + 1 \rangle_3 = 2$$

Therefore, the RNS(8 | 7 | 5 | 3) representation of $(164)_{\text{ten}}$ is $(4 \mid 3 \mid 4 \mid 2)_{\text{RNS}}$.

In the worst case, k modular additions are required for computing each residue of a k-bit number. To reduce the number of operations, one can view the given input number as a number in a higher radix. For example, if we use radix 4, then storing the residues of 4^i, 2×4^i and 3×4^i in a table would allow us to compute each of the required residues using only $k/2$ modular additions.

The conversion for each modulus can be done by repeatedly using a single lookup table and modular adder or by several copies of each arranged into a pipeline. For a low-cost modulus $m = 2^a - 1$, the residue can be determined by dividing up y into a-bit segments and adding them modulo $2^a - 1$.

Conversion from RNS to mixed-radix form

Associated with any residue number system RNS($m_{k-1} \mid \cdots \mid m_2 \mid m_1 \mid m_0$) is a mixed-radix number system MRS($m_{k-1} \mid \cdots \mid m_2 \mid m_1 \mid m_0$), which is essentially a k-digit positional number system with position weights

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

and digit sets $[0, m_{k-1} - 1], \cdots, [0, m_2 - 1], [0, m_1 - 1]$, and $[0, m_0 - 1]$ in its k-digit positions. Hence, the MRS digits are in the same ranges as the RNS digits (residues). For example, the mixed-radix system MRS(8 | 7 | 5 | 3) has position weights $7 \times 5 \times 3 = 105, 5 \times 3 = 15, 3$, and 1, leading to

$$(0 \mid 3 \mid 1 \mid 0)_{\text{MRS}(8 \mid 7 \mid 5 \mid 3)} = (0 \times 105) + (3 \times 15) + (1 \times 3) + (0 \times 1) = 48$$

The RNS-to-MRS conversion problem is that of determining the z_i digits of MRS, given the x_i digits of RNS, so that

$$y = (x_{k-1} \mid \cdots \mid x_2 \mid x_1 \mid x_0)_{\text{RNS}} = (z_{k-1} \mid \cdots \mid z_2 \mid z_1 \mid z_0)_{\text{MRS}}$$

From the definition of MRS, we have

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

It is thus immediately obvious that $z_0 = x_0$. Subtracting $z_0 = x_0$ from both the RNS and MRS representations, we get

$$y - x_0 = (x'_{k-1} \mid \cdots \mid x'_2 \mid x'_1 \mid 0)_{\text{RNS}} = (z_{k-1} \mid \cdots \mid z_2 \mid z_1 \mid 0)_{\text{MRS}}$$

where $x'_j = \langle x_j - x_0 \rangle_{m_j}$. If we now divide both representations by m_0, we get the following in the reduced RNS and MRS from which m_0 has been removed:

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

Thus, if we demonstrate how to divide the number $y' = (x'_{k-1} \mid \cdots \mid x'_2 \mid x'_1 \mid 0)_{\text{RNS}}$ by m_0 to obtain $(x''_{k-1} \mid \cdots \mid x''_2 \mid x''_1)_{\text{RNS}}$, we have converted the original problem to a similar problem with one fewer modulus. Repeating the same process then leads to the determination of all the z_i digits in turn.

Dividing y', which is a multiple of m_0, by a given constant (in this case m_0) is known as *scaling* and is much simpler than general division in RNS. Division by m_0 can be accomplished by multiplying each residue by the *multiplicative inverse* of m_0 with respect to the associated modulus. For example, the multiplicative inverses of 3 relative to 8, 7, and 5 are 3, 5, and 2, respectively, because

$$\langle 3 \times 3 \rangle_8 = \langle 3 \times 5 \rangle_7 = \langle 3 \times 2 \rangle_5 = 1$$

Thus, the number $y' = (0 \mid 6 \mid 3 \mid 0)_{\text{RNS}}$ can be divided by 3 through multiplication by $(3 \mid 5 \mid 2 \mid -)_{\text{RNS}}$:

$$\frac{(0 \mid 6 \mid 3 \mid 0)_{\text{RNS}}}{3} = (0 \mid 6 \mid 3 \mid 0)_{\text{RNS}} \times (3 \mid 5 \mid 2 \mid -)_{\text{RNS}} = (0 \mid 2 \mid 1 \mid -)_{\text{RNS}}$$

Multiplicative inverses of the moduli can be precomputed and stored in tables to facilitate RNS-to-MRS conversion.

■ **EXAMPLE 4.2** Convert $y = (0 \mid 6 \mid 3 \mid 0)_{\text{RNS}}$ to mixed-radix representation.
We have $z_0 = x_0 = 0$. Based on the preceding discussion, dividing y by 3 yields:

$$\begin{aligned}\frac{(0 \mid 6 \mid 3 \mid 0)_{\text{RNS}}}{3} &= (0 \mid 6 \mid 3 \mid 0)_{\text{RNS}} \times (3 \mid 5 \mid 2 \mid -)_{\text{RNS}} \\ &= (0 \mid 2 \mid 1 \mid -)_{\text{RNS}}\end{aligned}$$

Thus we have $z_1 = 1$. Subtracting 1 and dividing by 5, we get:

$$\begin{aligned}\frac{(7 \mid 1 \mid 0 \mid -)_{\text{RNS}}}{5} &= (7 \mid 1 \mid 0 \mid -)_{\text{RNS}} \times (5 \mid 3 \mid - \mid -)_{\text{RNS}} \\ &= (3 \mid 3 \mid - \mid -)_{\text{RNS}}\end{aligned}$$

Next, we get $z_2 = 3$. Subtracting 3 and dividing by 7, we find:

$$\begin{aligned}\frac{(0 \mid 0 \mid - \mid -)_{\text{RNS}}}{7} &= (0 \mid 0 \mid - \mid -)_{\text{RNS}} \times (7 \mid - \mid - \mid -)_{\text{RNS}} \\ &= (0 \mid - \mid - \mid -)_{\text{RNS}}\end{aligned}$$

We conclude by observing that $z_3 = 0$. The conversion is now complete:

$$y = (0 \mid 6 \mid 3 \mid 0)_{\text{RNS}} = (0 \mid 3 \mid 1 \mid 0)_{\text{MRS}} = 48$$

Mixed-radix representation allows us to compare the magnitudes of two RNS numbers or to detect the sign of a number. For example, the RNS representations $(0 \mid 6 \mid 3 \mid 0)_{\text{RNS}}$ and $(5 \mid 3 \mid 0 \mid 0)_{\text{RNS}}$ of 48 and 45 provide no clue to their relative magnitudes, whereas the equivalent mixed-radix representations $(0 \mid 3 \mid 1 \mid 0)_{\text{MRS}}$ and $(0 \mid 3 \mid 0 \mid 0)_{\text{MRS}}$, or $(000 \mid 011 \mid 001 \mid 00)_{\text{MRS}}$ and $(000 \mid 011 \mid 000 \mid 00)_{\text{MRS}}$, when coded in binary, can be compared as ordinary numbers.

Conversion from RNS to binary/decimal

One method for RNS-to-binary conversion is to first derive the mixed-radix representation of the RNS number and then use the weights of the mixed-radix positions to complete the conversion. We can also derive position weights for the RNS directly based on the Chinese remainder theorem (CRT), as discussed below.

Consider the conversion of $y = (3 \mid 2 \mid 4 \mid 2)_{\text{RNS}}$ from RNS(8 | 7 | 5 | 3) to decimal. Based on RNS properties, we can write

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

Thus, knowing the values of the following four constants (the RNS position weights) would allow us to convert any number from RNS(8 | 7 | 5 | 3) to decimal using four multiplications and three additions.

$$\begin{aligned}(1 \mid 0 \mid 0 \mid 0)_{\text{RNS}} &= 105 \\ (0 \mid 1 \mid 0 \mid 0)_{\text{RNS}} &= 120 \\ (0 \mid 0 \mid 1 \mid 0)_{\text{RNS}} &= 336 \\ (0 \mid 0 \mid 0 \mid 1)_{\text{RNS}} &= 280\end{aligned}$$

Thus, we find

$$(3 \mid 2 \mid 4 \mid 2)_{\text{RNS}} = \langle (3 \times 105) + (2 \times 120) + (4 \times 336) + (2 \times 280) \rangle_{840} = 779$$

It only remains to show how the preceding weights were derived. How, for example, did we determine that $w_3 = (1 \mid 0 \mid 0 \mid 0)_{\text{RNS}} = 105$?

To determine the value of w_3, we note that it is divisible by 3, 5, and 7, since its last three residues are 0s. Hence, w_3 must be a multiple of 105. We must then pick the appropriate multiple of 105 such that its residue with respect to 8 is 1. This is done by multiplying 105 by its multiplicative inverse with respect to 8. Based on the preceding discussion, the conversion process can be formalized in the form of CRT.

THEOREM 4.1 (The Chinese remainder theorem) The magnitude of an RNS number can be obtained from the CRT formula:

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

where, by definition, $M_i = M/m_i$, and $\alpha_i = \langle M_i^{-1} \rangle_{m_i}$ is the multiplicative inverse of M_i with respect to m_i.

To avoid multiplications in the conversion process, we can store the values of $\langle M_i \langle \alpha_i x_i \rangle_{m_i} \rangle_M$ for all possible i and x_i in tables of total size $\sum_{i=0}^{k-1} m_i$ words. Table 4.2

Table 4.2 Values needed in applying the Chinese remainder theorem to RNS (8 | 7 | 5 | 3)

i	m_i	x_i	$\langle M_i \langle \alpha_i x_i \rangle_{m_i} \rangle_M$
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
1	5	0	0
		1	336
		2	672
		3	168
		4	504
0	3	0	0
		1	280
		2	560

shows the required values for RNS(8 | 7 | 5 | 3). Conversion is then performed exclusively by table lookups and modulo-M additions.

4.4 DIFFICULT RNS ARITHMETIC OPERATIONS

In this section, we discuss algorithms and hardware designs for sign test, magnitude comparison, overflow detection, and general division in RNS. The first three of these operations are essentially equivalent in that if an RNS with dynamic range M is used for representing signed numbers in the range $[-N, P]$, with $M = N + P + 1$, then sign test is the same as comparison with P and overflow detection can be performed based on the signs of the operands and that of the result. Thus, it suffices to discuss magnitude comparison and general division.

To compare the magnitudes of two RNS numbers, we can convert both to binary or mixed-radix form. However, this would involve a great deal of overhead. A more

efficient approach is through approximate CRT decoding. Dividing the equality in the statement of Theorem 4.1 by M, we obtain the following expression for the scaled value of x in [0, 1):

$$\frac{x}{M} = \frac{(x_{k-1} \mid \cdots \mid x_2 \mid x_1 \mid x_0)_{\text{RNS}}}{M} = \left\langle \sum_{i=0}^{k-1} m_i^{-1} \langle \alpha_i x_i \rangle_{m_i} \right\rangle_1$$

Here, the addition of terms is performed modulo 1, meaning that in adding the terms $m_i^{-1}\langle \alpha_i x_i \rangle_{m_i}$, each of which is in [0, 1), the whole part of the result is discarded and only the fractional part is kept; this is much simpler than the modulo-M addition needed in conventional CRT decoding.

Again, the terms $m_i^{-1}\langle \alpha_i x_i \rangle_{m_i}$ can be precomputed for all possible i and x_i and stored in tables of total size $\sum_{i=0}^{k-1} m_i$ words. Table 4.3 shows the required lookup table for approximate CRT decoding in RNS(8 | 7 | 5 | 3). Conversion is then performed exclusively by

Table 4.3 Values needed in applying approximate Chinese remainder theorem decoding to RNS (8 | 7 | 5 | 3)

i	m_i	x_i	$\langle m_i^{-1}\langle \alpha_i x_i \rangle_{m_i} \rangle_1$
3	8	0	.0000
		1	.1250
		2	.2500
		3	.3750
		4	.5000
		5	.6250
		6	.7500
		7	.8750
2	7	0	.0000
		1	.1429
		2	.2857
		3	.4286
		4	.5714
		5	.7143
		6	.8571
1	5	0	.0000
		1	.4000
		2	.8000
		3	.2000
		4	.6000
0	3	0	.0000
		1	.3333
		2	.6667

table lookups and modulo-1 additions (i.e., fractional addition, with the carry-out simply ignored).

■ **EXAMPLE 4.3** Use approximate CRT decoding to determine the larger of the two numbers $x = (0 \mid 6 \mid 3 \mid 0)_{\text{RNS}}$ and $y = (5 \mid 3 \mid 0 \mid 0)_{\text{RNS}}$.
Reading values from Table 4.3, we get:

$$\frac{x}{M} \approx \langle .0000 + .8571 + .2000 + .0000 \rangle_1 = .0571$$
$$\frac{y}{M} \approx \langle .6250 + .4286 + .0000 + .0000 \rangle_1 = .0536$$

Thus, we can conclude that $x > y$, subject to approximation errors to be discussed next.

If the maximum error in each table entry is ε, then approximate CRT decoding yields the scaled value of an RNS number with an error of no more than $k\varepsilon$. In Example 4.3, assuming that the table entries have been rounded to four decimal digits, the maximum error in each entry is $\varepsilon = 0.000\,05$ and the maximum error in the scaled value is $4\varepsilon = 0.0002$. The conclusion $x > y$ is, therefore, safe.

Of course we can use highly precise table entries to avoid the possibility of erroneous conclusions altogether. But this would defeat the advantage of approximate CRT decoding in simplicity and speed. Thus, in practice, a two-stage process might be envisaged: a quick approximate decoding process is performed first, with the resulting scaled value(s) and error bound(s) used to decide whether a more precise or exact decoding is needed for arriving at a conclusion.

In many practical situations, an exact comparison of x and y might not be required and a ternary decision result $x < y, x \approx y$ (i.e., too close to call), or $x > y$ might do. In such cases, approximate CRT decoding is just the right tool. For example, in certain division algorithms (to be discussed in Chapter 14), the sign and the magnitude of the partial remainder s are used to choose the next quotient digit q_j from the redundant digit set $[-1, 1]$ according to the following:

$$\begin{array}{ll} s < 0 & \text{quotient digit} = -1 \\ s \approx 0 & \text{quotient digit} = \ \ 0 \\ s > 0 & \text{quotient digit} = \ \ 1 \end{array}$$

In this case, the algorithm's built-in tolerance to imprecision allows us to use it for RNS division. Once the quotient digit in $[-1, 1]$ has been chosen, the value $q_j d$, where d is the divisor, is subtracted from the partial remainder to obtain the new partial remainder for the next iteration. Also, the quotient, derived in positional radix-2 format using the digit set $[-1, 1]$, is converted to RNS on the fly.

In other division algorithms, to be discussed in Chapters 14 and 15, approximate comparison of the partial remainder s and divisor d is used to choose a radix-r quotient digit in $[-\alpha, \beta]$. An example includes radix-4 division with the redundant quotient digit set $[-2, 2]$. In these cases, too, approximate CRT decoding can be used to facilitate RNS division [Hung94].

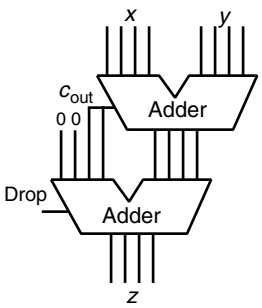

Figure 4.3 Adding a 4-bit ordinary mod-13 residue x to a 4-bit pseudoresidue y, producing a 4-bit mod-13 pseudoresidue z.

4.5 REDUNDANT RNS REPRESENTATIONS

Just as the digits in a positional radix-r number system do not have to be restricted to the set $[0, r-1]$, we are not obliged to limit the residue digits for the modulus m_i to the set $[0, m_i - 1]$. Instead, we can agree to use the digit set $[0, \beta_i]$ for the mod-m_i residue, provided $\beta_i \geq m_i - 1$. If $\beta_i \geq m_i$, then the resulting RNS is redundant.

One reason to use redundant residues is to simplify the modular reduction step needed after each arithmetic operation. Consider, for example, the representation of mod-13 residues using 4-bit binary numbers. Instead of using residues in [0, 12], we can use pseudoresidues in [0, 15]. Residues 0, 1, and 2 will then have two representations, since $13 = 0 \bmod 13$, $14 = 1 \bmod 13$, and $15 = 2 \bmod 13$. Addition of such a pseudoresidue y to an ordinary residue x, producing a pseudoresidue z, can be performed by a 4-bit binary adder. If the carry-out is 0, the addition result is kept intact; otherwise, the carry-out, which is worth 16 units, is dropped and 3 is added to the result. Thus, the required mod-13 addition unit is as shown in Fig. 4.3. Addition of two pseudoresidues is possible in a similar way [Parh01].

One can go even further and make the pseudoresidues $2h$ bits wide, where normal mod-m residues would be only h bits wide. This simplifies a multiply-accumulate operation, which is done by adding the $2h$-bit product of two normal residues to a $2h$-bit running total, reducing the $(2h + 1)$-bit result to a $2h$-bit pseudoresidue for the next step by subtracting $2^h m$ from it if needed (Fig. 4.4). Reduction to a standard h-bit residue is then done only once at the end of accumulation.

4.6 LIMITS OF FAST ARITHMETIC IN RNS

How much faster is RNS arithmetic than conventional (say, binary) arithmetic? We will see later in Chapters 6 and 7 that addition of binary numbers in the range $[0, M - 1]$ can be done in O(log log M) time and with O(log M) cost using a variety of methods such as carry-lookahead, conditional-sum, or multilevel carry-select. Both these are optimal to within constant factors, given the fixed-radix positional representation. For example, one can use the constant fan-in argument to establish that the circuit depth of an adder must be at least logarithmic in the number $k = \log_r M$ of digits. Redundant

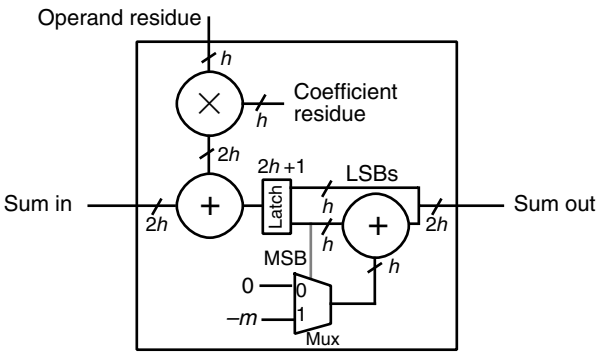

Figure 4.4 A modulo-*m* multiply-add cell that accumulates the sum into a double width redundant pseudoresidue.

representations allow O(1)-time, O(log M)-cost addition. What is the best one can do with RNS arithmetic?

Consider the residue number system RNS($m_{k-1} \mid \cdots \mid m_1 \mid m_0$). Assume that the moduli are chosen as the smallest possible prime numbers to minimize the size of the moduli, and thus maximize computation speed. The following theorems from number theory help us in figuring out the complexity.

THEOREM 4.2 The ith prime p_i is asymptotically equal to $i \ln i$.

THEOREM 4.3 The number of primes in $[1, n]$ is asymptotically equal to $n/(\ln n)$.

THEOREM 4.4 The product of all primes in $[1, n]$ is asymptotically equal to e^n.

Table 4.4 lists some numerical values that can help us understand the asymptotic approximations given in Theorems 4.2 and 4.3.

Armed with these results from number theory, we can derive an interesting limit on the speed of RNS arithmetic.

THEOREM 4.5 It is possible to represent all k-bit binary numbers in RNS with O(k/log k) moduli such that the largest modulus has O(log k) bits.

Proof: If the largest needed prime is n, by Theorem 4.4, we must have $e^n \approx 2^k$. This equality implies $n < k$. The number of moduli required is the number of primes less than n, which by Theorem 4.3 is O(n/log n) = O(k/log k).

As a result, addition of such residue numbers can be performed in O(log log log M) time and with O(log M) cost. So, the cost of addition is asymptotically comparable to that of binary representation whereas the delay is much smaller, though not constant.

Table 4.4 The ith-prime p_i and the number of primes in $[1, n]$ versus their asymptotic approximations

i	p_i	$i \ln i$	Error (%)	n	Primes in $[1, n]$	$n/(\ln n)$	Error (%)
1	2	0.000	100	5	2	3.107	55
2	3	1.386	54	10	4	4.343	9
3	5	3.296	34	15	6	5.539	8
4	7	5.545	21	20	8	6.676	17
5	11	8.047	27	25	9	7.767	14
10	29	23.03	21	30	10	8.820	12
15	47	40.62	14	40	12	10.84	10
20	71	59.91	16	50	15	12.78	15
30	113	102.0	10	100	25	21.71	13
40	173	147.6	15	200	46	37.75	18
50	229	195.6	15	500	95	80.46	15
100	521	460.5	12	1000	170	144.8	15

If for implementation ease, we limit ourselves to moduli of the form 2^a or $2^a - 1$, the following results from number theory are applicable.

THEOREM 4.6 The numbers $2^a - 1$ and $2^b - 1$ are relatively prime if and only if a and b are relatively prime.

THEOREM 4.7 The sum of the first i primes is asymptotically $O(i^2 \ln i)$.

These theorems allow us to prove the following asymptotic result for low-cost residue number systems.

THEOREM 4.8 It is possible to represent all k-bit binary numbers in RNS with $O((k/\log\ k)^{1/2})$ low-cost moduli of the form $2^a - 1$ such that the largest modulus has $O((k\ \log\ k)^{1/2})$ bits.

Proof: If the largest modulus that we need is $2^l - 1$, by Theorem, 4.7, we must have $l^2\ \ln\ l \approx k$. This implies that $l = O((k/\log\ k)^{1/2})$. By Theorem 4.2, the lth prime is approximately $p_l \approx l\ \ln\ l \approx O((k\ \log\ k)^{1/2})$. The proof is complete upon noting that to minimize the size of the moduli, we pick the ith modulus to be $2^{p_i} - 1$.

As a result, addition of low-cost residue numbers can be performed in $O(\log \log M)$ time with $O(\log M)$ cost and thus, asymptotically, offers little advantage over binary representation.

PROBLEMS

4.1 RNS representation and arithmetic

Consider the RNS system RNS(15 | 13 | 11 | 8 | 7) derived in Section 4.2.

a. Represent the numbers $x = 168$ and $y = 23$ in this RNS.
b. Compute $x + y, x - y, x \times y$, checking the results via decimal arithmetic.
c. Knowing that x is a multiple of 56, divide it by 56 in the RNS. *Hint*: $56 = 7 \times 8$.
d. Compare the numbers $(5 \mid 4 \mid 3 \mid 2 \mid 1)_{\text{RNS}}$ and $(1 \mid 2 \mid 3 \mid 4 \mid 5)_{\text{RNS}}$ using mixed-radix conversion.
e. Convert the numbers $(5 \mid 4 \mid 3 \mid 2 \mid 1)_{\text{RNS}}$ and $(1 \mid 2 \mid 3 \mid 4 \mid 5)_{\text{RNS}}$ to decimal.
f. What is the representational efficiency of this RNS compared with standard binary?

4.2 RNS representation and arithmetic

Consider the low-cost RNS system RNS(32 | 31 |15 | 7) derived in Section 4.2.

a. Represent the numbers $x = 168$ and $y = -23$ in this RNS.
b. Compute $x + y, x - y, x \times y$, checking the results via decimal arithmetic.
c. Knowing that x is a multiple of 7, divide it by 7 in the RNS.
d. Compare the numbers $(4 \mid 3 \mid 2 \mid 1)_{\text{RNS}}$ and $(1 \mid 2 \mid 3 \mid 4)_{\text{RNS}}$ using mixed-radix conversion.
e. Convert the numbers $(4 \mid 3 \mid 2 \mid 1)_{\text{RNS}}$ and $(1 \mid 2 \mid 3 \mid 4)_{\text{RNS}}$ to decimal.
f. What is the representational efficiency of this RNS compared with standard binary?

4.3 RNS representation

Find all numbers for which the RNS(8 | 7 | 5 | 3) representation is palindromic (i.e., the string of four "digits" reads the same forward and backward).

4.4 RNS versus GSD representation

We are contemplating the use of 16-bit representations for fast integer arithmetic. One option, radix-8 GSD representation with the digit set $[-5, 4]$, can accommodate four-digit numbers. Another is RNS(16 | 15 | 13 | 11) with complement representation of negative values.

a. Compute and compare the range of representable integers in the two systems.
b. Represent the integers $+441$ and -228 and add them in the two systems.
c. Briefly discuss and compare the complexity of multiplication in the two systems.

4.5 RNS representation and arithmetic

Consider an RNS that can be used to represent the equivalent of 24-bit 2's-complement numbers.

a. Select the set of moduli to maximize the speed of arithmetic operations.
b. Determine the representational efficiency of the resulting RNS.
c. Represent the numbers $x = +295$ and $y = -322$ in this number system.
d. Compute the representations of $x + y, x - y$, and $x \times y$; check the results.

4.6 Binary-to-RNS conversion

In an RNS, 11 is used as one of the moduli.

a. Design a mod-11 adder using two standard 4-bit binary adders and a few logic gates.
b. Using the adder of part a and a 10-word lookup table, show how the mod-11 residue of an arbitrarily long binary number can be computed by a serial-in, parallel-out circuit.
c. Repeat part a, assuming the use of mod-11 pseudoresidues in [0, 15].
d. Outline the changes needed in the design of part b if the adder of part c is used.

4.7 Low-cost RNS

Consider RNSs with moduli of the form 2^{a_i} or $2^{a_i} - 1$.

a. Prove that $m_i = 2^{a_i} - 1$ and $m_j = 2^{a_j} - 1$ are relatively prime if and only if a_i and a_j are relatively prime.
b. Show that such a system wastes at most 1 bit relative to binary representation.
c. Determine an efficient set of moduli to represent the equivalent of 32-bit unsigned integers. Discuss your efficiency criteria.

4.8 Special RNS representations

It has been suggested that moduli of the form $2^{a_i} + 1$ also offer speed advantages. Evaluate this claim by devising a suitable representation for the $(a_i + 1)$-bit residues and dealing with arithmetic operations on such residues. Then, determine an efficient set of moduli of the form 2^{a_i} and $2^{a_i} \pm 1$ to represent the equivalent of 32-bit integers.

4.9 Overflow in RNS arithmetic

Show that if $0 \leq x, y < M$, then $(x + y) \bmod M$ causes overflow if and only if the result is less than x (thus the problem of overflow detection in RNS arithmetic is equivalent to the magnitude comparison problem).

4.10 Discrete logarithm

Consider a prime modulus p. From number theory, we know that there always exists an integer generator g such that the powers $g^0, g^1, g^2, g^3, \ldots \pmod p$ produce all the integers in $[1, p - 1]$. If $g^i = x \bmod p$, then i is called the mod-p, base-g discrete logarithm of x. Outline a modular multiplication scheme using discrete log and $\log^{-1}$ tables and an adder.

4.11 Halving even numbers in RNS

Given the representation of an even number in an RNS with only odd moduli, find an efficient algorithm for halving the given number.

4.12 Symmetric RNS

In a symmetric RNS, the residues are signed integers, possessing the smallest possible absolute values, rather than unsigned integers. Thus, for an odd modulus m, symmetric residues range from $-(m-1)/2$ to $(m-1)/2$ instead of from 0 to $m-1$. Discuss the possible advantages of a symmetric RNS over ordinary RNS.

4.13 Approximate CRT decoding

Consider the numbers $x = (0 \mid 6 \mid 3 \mid 0)_{\text{RNS}}$ and $y = (5 \mid 3 \mid 0 \mid 0)_{\text{RNS}}$ of Example 4.3 in Section 4.4.

a. Redo the example and its associated error analysis with table entries rounded to two decimal digits. How does the conclusion change?
b. Redo the example with table entries rounded to three decimal digits and discuss.

4.14 Division of RNS numbers by the moduli

a. Show how an RNS number can be divided by one of the moduli to find the quotient and the remainder, both in RNS form.
b. Repeat part a for division by the product of two or more moduli.

4.15 RNS base extension

Consider a k-modulus RNS and the representation of a number x in that RNS. Develop an efficient algorithm for deriving the representation of x in a $(k+1)$-modulus RNS that includes all the moduli of the original RNS plus one more modulus that is relatively prime with respect to the preceding k. This process of finding a new residue given k existing residues is known as base extension.

4.16 Automorphic numbers

An n-place *automorph* is an n-digit decimal number whose square ends in the same n digits. For example, 625 is a 3-place automorph, since $625^2 = 390\,625$.

a. Prove that $x > 1$ is an n-place automorph if and only if $x \bmod 5^n = 0$ or 1 and $x \bmod 2^n = 1$ or 0, respectively.
b. Relate n-place automorphs to a 2-residue RNS with $m_1 = 5^n$ and $m_0 = 2^n$.
c. Prove that if x is an n-place automorph, then $(3x^2 - 2x^3) \bmod 10^{2n}$ is a $2n$-place automorph.

4.17 RNS moduli and arithmetic

We need an RNS to represent the equivalent of 8-bit 2's-complement numbers.

a. Suggest an efficient 3-modulus RNS for this purpose.
b. Design logic circuits that can negate (change the sign of) a given number represented in your number system.
c. Design an adder circuit for one of the moduli that is not a power of 2 (pick the most convenient one).

d. Design a multiplier circuit for one of your moduli (pick the most convenient one).

4.18 Binary-to-RNS conversion

Design the needed algorithms and circuits to translate a 16-bit 2's-complement binary number into RNS(32 | 31 | 15 | 7) format.

4.19 RNS arithmetic

Consider the residue number system RNS(31 | 16 | 15 | 7) leading to a 16-bit representation.

a. Represent the numbers $x = 98$ and $y = -54$ in this system.
b. Compute $x + y$, $x - y$, and $x \times y$.
c. Check the computations of part b by reconverting the results, showing all computation steps.
d. Compare the range of numbers to that of the 16-bit 2's-complement system.

4.20 RNS arithmetic

Consider the residue number system RNS(16 | 15 | 13 | 11) leading to a 16-bit representation.

a. Represent the numbers $x = 56$ and $y = -23$ in this system.
b. Compute $x + y$, $x - y$, and $x \times y$. Check the results.
c. Represent the number 3 and compute 3^8. Check the result.
d. Compare the range of numbers to that of the 16-bit 2's-complement system.

4.21 RNS moduli and range

a. Determine the smallest possible RNS moduli for representing the equivalent of 64-bit integers.
b. Find the largest RNS representation range that is possible with residues that are no wider than 6 bits.

4.22 Selection of RNS moduli

We would like to represent the equivalent of 4-digit unsigned decimal integers in an RNS.

a. Choose a suitable set of moduli if there is no restriction on your choice.
b. Choose a suitable set of low-cost moduli.

4.23 RNS range and arithmetic

Consider the residue number system RNS(32 | 31 | 29 | 27 | 25 | 23 | 19).

a. Compare the efficiency of this RNS to that of binary representation.
b. Discuss the structure of a table-based adder/subtractor/multiplier using 1K $\times$ 8 ROM modules.

c. What is the maximum possible range extension without sacrificing the speed of arithmetic?

4.24 RNS representations and arithmetic

Consider the residue number system RNS(16 | 15 | 7).

a. Derive the range of signed integers that can be represented, assuming an almost symmetric range.
b. Find the representations of $x = +38$ and $y = -5$ and compute their sum, difference, and product.
c. Determine what values are represented by $(0 \mid 1 \mid 0)_{\text{RNS}}$ and $(1 \mid 0 \mid 1)_{\text{RNS}}$.

4.25 RNS with redundant range

Consider the residue number system RNS(8 | 7 | 3) used to represent numbers in the range [0, 63] by means of 8 bits. This number system possesses some redundancy in that only 64 of its 168 combinations are used. Is this redundancy adequate to detect any single-bit error in one of the residues? Fully justify your answer.

4.26 RNS design

Select the best RNS moduli for representing the equivalent of 16-bit signed-magnitude integers with the highest possible speed of arithmetic.

4.27 RNS arithmetic

Consider RNS(16 | 15), with 4-bit binary encoding of the residues, for representing the equivalent of 7-bit unsigned binary integers.

a. Show the representations of the unsigned integers 25, 60, and 123.
b. Using only standard 4-bit adders and 4×4 multipliers (and no other component), design an adder and a multiplier for the given RNS.
c. Using the same components as in part b, design a 7-bit binary adder and a 7×7 unsigned binary multiplier, producing a 7-bit result.
d. Compare the circuits of parts b and c with regard to speed and complexity.

4.28 Special RNSs

a. Show that residue-to-binary conversion for an RNS with moduli $m_2 = 2^h + 1$, $m_1 = 2^h$, and $m_0 = 2^h - 1$ can be performed using a 3-to-2 (carry-save) adder and a $2h$-bit fast adder with end-around carry. Therefore, latency and cost of the converter are only slightly more than those for a $2h$-bit fast adder.
b. Generalize the result of part a to an RNS with more than two moduli of the form $2^h \pm 1$ as well as one that is a power of 2.

4.29 A four-modulus RNS

Consider the 4-modulus number system RNS($2^n + 3 \mid 2^n + 1 \mid 2^n - 1 \mid 2^n - 3$).

a. Show that the four moduli are pairwise relatively prime.

b. Derive a simple residue-to-binary conversion scheme for the number system RNS($2^n + 3 \mid 2^n - 1$).
c. Repeat part b for RNS($2^n + 1 \mid 2^n - 3$).
d. Show how the results of parts b and c can be combined into a simple residue-to-binary converter for the original 4-modulus RNS.

REFERENCES AND FURTHER READINGS

[Garn59] Garner, H. L., "The Residue Number System," *IRE Trans. Electronic Computers*, Vol. 8, pp. 140–147, 1959.

[Hung94] Hung, C. Y., and B. Parhami, "An Approximate Sign Detection Method for Residue Numbers and Its Application to RNS Division," *Computers & Mathematics with Applications*, Vol. 27, No. 4, pp. 23–35, 1994.

[Hung95] Hung, C. Y., and B. Parhami, "Error Analysis of Approximate Chinese-Remainder-Theorem Decoding," *IEEE Trans. Computers*, Vol. 44, No. 11, pp. 1344–1348, 1995.

[Jenk93] Jenkins, W. K., "Finite Arithmetic Concepts," in *Handbook for Digital Signal Processing*, S. K. Mitra and J. F. Kaiser (eds.), Wiley, 1993, pp. 611–675.

[Merr64] Merrill, R.D., "Improving Digital Computer Performance Using Residue Number Theory," *IEEE Trans. Electronic Computers*, Vol. 13, No. 2, pp. 93–101, 1964.

[Omon07] Omondi, A., and B. Premkumar, *Residue Number Systems: Theory and Implementation*, Imperial College Press, 2007.

[Parh76] Parhami, B., "Low-Cost Residue Number Systems for Computer Arithmetic," *AFIPS Conf. Proc.*, Vol. 45 (1976 National Computer Conference), AFIPS Press, 1976, pp. 951–956.

[Parh93] Parhami, B., and H.-F. Lai, "Alternate Memory Compression Schemes for Modular Multiplication," *IEEE Trans. Signal Processing*, Vol. 41, pp. 1378–1385, 1993.

[Parh96] Parhami, B., "A Note on Digital Filter Implementation Using Hybrid RNS-Binary Arithmetic," *Signal Processing*, Vol. 51, pp. 65-67, 1996.

[Parh01] Parhami, B., "RNS Representations with Redundant Residues," *Proc. 35th Asilomar Conf. Signals, Systems, and Computers*, pp. 1651–1655, 2001.

[Sode86] Soderstrand, M. A., W. K. Jenkins, G. A. Jullien, and F. J. Taylor (eds.), *Residue Number System Arithmetic*, IEEE Press, 1986.

[Szab67] Szabo, N. S., and R. I. Tanaka, *Residue Arithmetic and Its Applications to Computer Technology*, McGraw-Hill, 1967.

[Verg08] Vergos, H. T., "A Unifying Approach for Weighted and Diminished-1 Modulo $2^n + 1$ Addition," *IEEE Trans. Circuits and Systems II*, Vol. 55, No. 10, pp. 1041–1045, 2008.