

Chapter 1

Introduction

Our main aim in this chapter is to introduce the basic concepts underlying residue number systems (RNS), arithmetic, and applications and to give a brief but almost complete summary. We shall, however, start with a discussion of certain aspects of more commonly used number systems and then review the main number systems used in conventional computer arithmetic. We shall also briefly discuss one other unconventional number system that has found some practical use in computer arithmetic; this is the *redundant¹ signed-digit* number system. We have two objectives in these preliminary discussions. The first is to facilitate a contrast between RNS and commonly used number systems. The second is to recall a few basic properties of the conventional number systems, as, ultimately, it is these that form the basis of implementations of residue arithmetic. The subsequent introduction to RNS consists of some basic definitions, a discussion of certain desirable features of a residue number system, and a discussion of the basic arithmetic operations.

A basic number system consists of a correspondence between sequences of digits and numbers. In a *fixed-point* number system, each sequence corresponds to exactly one number², and the *radix-point* —the “decimal point” in the ordinary decimal number system— that is used to separate the integral and fractional parts of a representation is in a fixed position. In contrast, in a *floating-point* number system, a given sequence may correspond to several numbers: the position of the radix-point is not fixed, and each position in a digit-sequence indicates the particular number represented. Usually, floating-point systems are used for the representation

¹In a redundant number system, a given number may have more than one representation.

²Notable exceptions occur in certain unconventional number systems.

of real numbers, and fixed-point systems are used to represent integers (in which the radix point is implicitly assumed to be at the right-hand end) or as parts of floating-point representations; but there are a few exceptions to this general rule. Almost all applications of RNS are as fixed-point number systems.

If we consider a number³ such as 271.834 in the ordinary decimal number system, we can observe that each digit has a *weight* that corresponds to its *position*: hundred for the 2, ten for the 7, . . . , thousandth for the 4. This number system is therefore an example of a *positional* (or *weighted*) number system; residue number systems, on the other hand, are non-positional. The decimal number system is also a *single-radix* (or *fixed-radix*) system, as it has only one base (i.e. ten). Although *mixed-radix* (i.e. multiple-radix) systems are relatively rare, there are a few useful ones. Indeed, for the purposes of conversion to and from other number systems, as well as for certain operations, it is sometimes useful to associate a residue number system with a weighted, mixed-radix number system.

1.1 Conventional number systems

In this section we shall review the three standard notations used for fixed-point computer arithmetic and then later point out certain relationships with residue arithmetic.

In general, numbers may be signed, and for binary digital arithmetic there are three standard notations that have been traditionally used for the binary representation of signed numbers. These are *sign-and-magnitude*, *one's complement*, and *two's complement*. Of these three, the last is the most popular, because of the relative ease and speed with which the basic arithmetic operations can be implemented. Sign-and-magnitude notation has the convenience of having a sign-representation that is similar to that used in ordinary decimal arithmetic. And one's complement, although a notation in its own right, more often appears only as an intermediate step in arithmetic involving the other two notations.

The sign-and-magnitude notation is derived from the conventional written notation of representing a negative number by prepending a sign to a magnitude that represents a positive number. For binary computer hardware, a single bit suffices for the sign: a sign bit of 0 indicates a positive

³For clarity of expression, we shall not always distinguish between a number and its representation.

number, and a sign bit of 1 indicates a negative number. For example, the representation of the number positive-five in six bits is 000101, and the corresponding representation of negative-five is 100101. Note that the representation of the sign is independent of that of the magnitude and takes up exactly one bit; this is not the case both with one's complement and two's complement notations.

Sign-and-magnitude notation has two representations, $000\dots 0$ and $100\dots 0$, for the number zero; it is therefore redundant. With one exception (in the context of floating-point numbers) this existence of two representations for zero can be a nuisance in an implementation. Addition and subtraction are harder to implement in this notation than in one's complement and two's complement notations; and as these are the most common arithmetic operations, true sign-and-magnitude arithmetic is very rarely implemented.⁴

In one's complement notation, the representation of the negation of a number is obtained by inverting the bits in its binary representation; that is, the 0s are changed to 1s and the 1s are changed to 0s. For example, the representation of the number positive-five in six bits is 000101 and negative-five therefore has the representation 111010. The leading bit again indicates the sign of the number, being 0 for a positive number and 1 for a negative number. We shall therefore refer to the most significant digit as the *sign bit*, although here the sign of a negative number is in fact represented by an infinite string of 1s that in practice is truncated according to the number of bits used in the representations and the magnitude of the number represented. It is straightforward to show that the n -bit representation of the negation of a number N is also, when interpreted as the representation of an unsigned number, that of $2^n - 1 - N$. (This point will be useful in subsequent discussions of basic residue arithmetic.) The one's complement system too has two representations for zero— $00\dots 0$ and $11\dots 1$ —which can be a nuisance in implementations. We shall see that a similar problem occurs with certain residue number systems. Addition and subtraction in this notation are harder to implement than in two's complement notation (but easier than in sign-and-magnitude notation) and multiplication and division are only slightly less so. For this reason, two's complement is the preferred notation for *implementing* most computer arithmetic.

⁴It is useful to note here that the notation of representation and the notation for the actual arithmetic implementation need not be the same.

Negation in two's complement notation consists of a bit-inversion (that is, a translation into the one's complement) followed by the addition of a 1, with any carry from the addition being ignored. Thus, for example, the result of negating 000101 is 111011. As with one's complement notation, the leftmost bit here too indicates the sign: it is 0 for a positive number and 1 for a negative number; but again, strictly, the sign is actually represented by the truncation of an infinite string. For n -bit representations, representing the negation of the number N may also be viewed as the representation of the positive number $2^n - N$.

In contrast with the first two conventional notations, the two's complement has only one representation for zero, i.e. $00\dots 0$. The two's complement notation is the most widely used of the three systems, as the algorithms and hardware designs required for its implementation are quite straightforward. Addition, subtraction, and multiplication are relatively easy to implement with this notation, and division is only slightly less so.

All of the notations above can be readily extended to non-binary radices. The extension of binary sign-and-magnitude to an arbitrary radix, r , involves representing the magnitude in radix- r and using 0 in the sign digit for positive numbers and $r - 1$ for negative numbers. An alternative representation for the sign is to use half of the permissible values of the sign digit (that is, $0\dots r/2 - 1$, assuming r is even) for the positive numbers and the other half (that is, $r/2\dots r - 1$, for an even radix) for the negative numbers. The generalization of one's complement to an arbitrary radix is known as *diminished-radix complement*, the name being derived from the fact that to negate a number in this notation, each digit is subtracted from the radix diminished by one, i.e. from $r - 1$. Alternatively, the representation of the negation may also be viewed as the result of subtracting the number from $r^n - 1$, where n is the number of digits used in the representations. Thus, for example, the negation of 01432 in radix-8 is 76345, i.e. $77777 - 01432$. The sign digit will be 0 for a positive number and $r - 1$ for a negative number. The generalization of two's complement to an arbitrary radix is known as *radix complement notation*. In radix complement notation, the radix- r negation of a number is obtained, essentially, by subtracting from r^n , where n is the number of digits used in the representations. Alternatively, negation may also be taken as the formation of the diminished-radix complement followed by the addition of a 1. Thus, for example, the radix-8 negation of 01432 is 76346, i.e. $100000 - 01432$ or $77777 - 01432 + 1$. The determination of sign is similar to that for the radix- r diminished-radix complement.

Residue number systems are more complex than the three standard notations reviewed above, and, consequently, cannot be implemented directly with the same efficiency as the conventional arithmetic. Therefore, in practice, residue arithmetic is often realized in terms of lookup-tables (to avoid the complex combinational-logic circuits) and conventional arithmetic (i.e. arithmetic in some standard notation). For example, the sign-and-magnitude approach may be convenient for representing signed numbers in RNS, but actual arithmetic operations might be best realized in terms of radix-complement (two's complement) arithmetic. We shall also see that certain choices of representational parameters in RNS naturally lead to diminished-radix complement (one's complement) arithmetic.

1.2 Redundant signed-digit number systems

Many unconventional number systems have been proposed, and some even have been put to practical use. Nevertheless, very few have had widespread or sustained application. Therefore, other than RNS, we shall restrict ourselves here to just the *redundant signed-digit* (RSD) systems, which have long been used for high-speed arithmetic. As far as practical unconventional notations go, RSD systems are the only serious competitor to RNS. RSD systems may also have some applications in the implementation of residue arithmetic, just as the conventional notations do; this is a largely unexplored area, although there has been some relevant work.

Two of the conventional systems discussed above have some *redundancy*, which means that in each there is at least one number with two or more representations; but the redundancy there is hardly of a useful variety. In contrast, redundant signed-digit number systems have much greater degrees of deliberate and useful redundancy. These systems are mainly used for high-speed arithmetic, especially in multiplication (in which they are used indirectly) and division (in which they are used more directly). They are also used in some algorithms for the evaluation of elementary (i.e. transcendental) functions.

In a typical RSD system, the number of values that any one digit may assume exceeds the value of the radix, and the digits are individually signed. If such a system employs a radix r , then the number of distinct values that a digit may assume lies between $r + 1$ and $2r - 1$. (In conventional number systems, such as those above, the size of the digit-set does not exceed r .) The digit-set in a redundant signed-digit number system is

$\{-a, -(a-1), \dots, -1, 0, 1, \dots, b-1, b\}$, where $\lceil (r-1)/2 \rceil \leq a, b \leq r-1$; usually $a = b$. In contrast, for a conventional number system the digit-set is usually $\{0, 1, \dots, r-1\}$.

Given the above, it is evident that in an RSD system a given number will have several distinct representations. For example, in the radix-2 RSD system, which employs the digit-set $\{\bar{1}, 0, 1\}$ (where $\bar{1}$ stands for -1), three five-bit representations of the number eleven are 01011 (that is, $8+2+1$), $10\bar{1}0\bar{1}$ (that is, $16-4+1$), and $0110\bar{1}$ (that is, $8+4-1$). The representation of the negation of a number represented in a redundant signed-digit notation is obtained by changing the sign of every non-zero digit. For example, three representations, obtained from the preceding example, of negative-eleven are $0\bar{1}0\bar{1}\bar{1}$ (that is, $-8-2-1$), $\bar{1}0101$ (that is, $-16+4+1$), and $0\bar{1}\bar{1}01$ (that is, $-8-4+1$).

In high-speed multiplication, RSD systems appear implicitly in multiplier recoding [7]. For example, straightforward multiplication by the positive number 11111111 requires eight additions, but only two are required if the multiplier is recoded (on-the-fly) into $\bar{1}0000001$. For division, generating a quotient in RSD notation helps speed up the process by allowing the individual quotient digits to be formed from only approximate comparisons of the partial dividend and the divisor. The redundancy permits later correction of any errors in the choice of digits. For example, if two successive quotient-digits should be 01 , but the first digit is guessed to be 1 , a correction can subsequently be made by selecting the next digit as $\bar{1}$, since 01 and $1\bar{1}$ represent the same number. The RSDs also find some use in the evaluation of elementary functions, in a manner similar to that of their use in division; the main algorithms here are the CORDIC ones [4]. As far as RNS goes, there has been a few proposals to use RSD to speed up the implementations of RNS-arithmetic [3].

1.3 Residue number systems and arithmetic

Residue number systems are based on the *congruence* relation, which is defined as follows. Two integers a and b are said to be *congruent modulo* m if m divides exactly the difference of a and b ; it is common, especially in mathematics tests, to write $a \equiv b \pmod{m}$ to denote this. Thus, for example, $10 \equiv 7 \pmod{3}$, $10 \equiv 4 \pmod{3}$, $10 \equiv 1 \pmod{3}$, and $10 \equiv -2 \pmod{3}$. The number m is a *modulus* or *base*, and we shall assume that its values exclude unity, which produces only trivial congruences.

If q and r are the quotient and remainder, respectively, of the integer division of a by m —that is, $a = q.m + r$ —then, by definition, we have $a \equiv r \pmod{m}$. The number r is said to be the *residue* of a with respect to m , and we shall usually denote this by $r = |a|_m$. The set of m smallest values, $\{0, 1, 2, \dots, m - 1\}$, that the residue may assume is called the set of *least positive residues modulo m* . Unless otherwise specified, we shall assume that these are the only residues in use.

Suppose we have a set, $\{m_1, m_2, \dots, m_N\}$, of N positive and pairwise relatively prime moduli⁵. Let M be the product of the moduli. Then every number $X < M$ has a unique representation in the residue number system, which is the set of residues $\{|X|_{m_i} : 1 \leq i \leq N\}$. A partial proof of this is as follows. Suppose X_1 and X_2 are two different numbers with the same *residue-set*. Then $|X_1|_{m_i} = |X_2|_{m_i}$, and so $|X_1 - X_2|_{m_i} = 0$. Therefore $X_1 - X_2$ is the least common multiple (lcm) of m_i . But if the m_i are relatively prime, then their lcm is M , and it must be that $X_1 - X_2$ is a multiple of M . So it cannot be that $X_1 < M$ and $X_2 < M$. Therefore, the set $\{|X|_{m_i} : 1 \leq i \leq N\}$ is unique and may be taken as the representation of X . We shall write such a representation in the form $\langle x_1, x_2, \dots, x_N \rangle$, where $x_i = |X|_{m_i}$, and we shall indicate the relationship between X and its residues by writing $X \cong \langle x_1, x_2, \dots, x_N \rangle$. The number M is called the *dynamic range* of the RNS, because the number of numbers that can be represented is M . For unsigned numbers, that range is $[0, M - 1]$.

Representations in a system in which the moduli are not pairwise relatively prime will be not be unique: two or more numbers will have the same representation. As an example, the residues of the integers zero through fifteen relative to the moduli two, three, and five (which are pairwise relatively prime) are given in the left half of Table 1.1. And the residues of the same numbers relative to the moduli two, four, and six (which are not pairwise relatively prime) are given in the right half of the same table. Observe that no sequence of residues is repeated in the first half, whereas there are repetitions in the second.

The preceding discussions (and the example in the left-half of Table 1.1) define what may be considered *standard residue number systems*, and it is with these that we shall primarily be concerned. Nevertheless, there are useful examples of “non-standard” RNS, the most common of which are the *redundant residue number systems*. Such a system is obtained by, essentially, adding extra (redundant) moduli to a standard system. The

⁵That is, for every j and k , if $j \neq k$, then m_j and m_k have no common divisor larger than unity.

dynamic range then consists of a “legitimate” range, defined by the non-redundant moduli and an “illegitimate” range; for arithmetic operations, initial operands and results should be within legitimate range. Redundant number systems of this type are especially useful in fault-tolerant computing. The redundant moduli mean that digit-positions with errors may be excluded from computations while still retaining a sufficient part of the dynamic range. Furthermore, both the detection and correction of errors are possible: with k redundant moduli, it is possible to detect up to k errors and to correct up to $\lfloor k/2 \rfloor$ errors. A different form of redundancy can be introduced by extending the size of the digit-set corresponding to a modulus, in a manner similar to RSDs. For a modulus m , the normal digit set is $\{0, 1, 2, \dots, m - 1\}$; but if instead the digit-set used is $\{0, 1, 2, \dots, \tilde{m}\}$, where $\tilde{m} \geq m$, then some residues will have redundant representations. Redundant residue number systems are discussed in slightly more detail in Chapters 2 and 8.

Table 1.1: Residues for various moduli

N	Relatively prime moduli			Relatively non-prime moduli		
	$m_1 = 2$	$m_2 = 3$	$m_3 = 5$	$m_1 = 2$	$m_2 = 4$	$m_3 = 6$
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	0	2	2	0	2	2
3	1	0	3	1	3	3
4	0	1	4	0	0	4
5	1	2	0	1	1	5
6	0	0	1	0	2	0
7	1	1	2	1	3	1
8	0	2	3	0	0	2
9	1	0	4	1	1	3
10	0	1	0	0	2	4
11	1	2	1	1	3	5
12	0	0	2	0	0	0
13	1	1	3	1	1	1
14	0	2	4	0	2	2
15	1	0	0	1	3	3

1.3.1 Choice of moduli

Ignoring other, more “practical”, issues, the best moduli are probably prime numbers—at least from a purely mathematical perspective. A particularly useful property of such moduli is that of “generation”. If a modulus, m , is prime, then there is at least one *primitive root* (or *generator*), $p \leq m - 1$, such that the set $\{|p^i|_m : i = 0, 1, 2, \dots, m - 2\}$ is the set of all the non-zero residues with respect to m . As an example, if $m = 7$, then we may take $p = 3$, since $\{|3^0|_7 = 1, |3^1|_7 = 3, |3^2|_7 = 2, |3^3|_7 = 6, |3^4|_7 = 4, |3^5|_7 = 5\} = \{1, 2, 3, 4, 5, 6\}$; 5 is also a primitive root. Evidently, for such moduli, multiplication and powering of residues may be carried out in terms of simple operations on indices of the power of the primitive root, in a manner similar to the use of logarithms and anti-logarithms in ordinary multiplication. More on the subject will be found in Chapters 2 and 5.

For computer applications, it is important to have moduli-sets that facilitate both efficient representation and balance, where the latter means that the differences between the moduli should be as small as possible.⁶ Take, for example, the choice of 13 and 17 for the moduli, these being adjacent prime numbers; the dynamic range is 221. With a straightforward binary encoding, four bits and five bits, respectively will be required to represent the corresponding residues. In the former case, the representational efficiency is 13/16, and in the latter it is 17/32. If instead we chose 13 and 16, then the representational efficiency would be improved—to 16/16 in the second case—but at the cost of reduction in the range (down to 208). On, the other hand, with the better balanced pair, 15 and 16, we would have both better efficiency and greater range: 15/16 and 16/16 for the former, and 240 for the latter.

It is also useful to have moduli that simplify the implementation of the arithmetic operations. This invariably means that arithmetic on residue digits should not deviate too far from conventional arithmetic, which is just arithmetic modulo a power of two. A common choice of prime modulus that does not complicate arithmetic and which has good representational efficiency is $m_i = 2^i - 1$. Not all pairs of numbers of the form $2^i - 1$ are relatively prime, but it can be shown that that $2^j - 1$ and $2^k - 1$ are relatively prime if and only if j and k are relatively prime. Many moduli sets are based on these choices, but there are other possibilities; for example,

⁶Unbalanced moduli-sets lead to uneven architectures, in which the role of the largest moduli, with respect to both cost and performance, is excessively dominant. An example of a moduli-set with good balance is $\{2^n - 1, 2^n, 2^n + 1\}$.

moduli-sets of the form $\{2^n - 1, 2^n, 2^n + 1\}$ are among the most popular in use.

In general, then, there are at least four considerations that should be taken into account in the selection of moduli. First, the selected moduli must provide an adequate range whilst also ensuring that RNS representations are unique. The second is, as indicated above, the efficiency of binary representations; in this regard, a balance between the different moduli in a given moduli-set is also important. The third is that, ideally, the implementations of arithmetic units for RNS should to some extent be compatible with those for conventional arithmetic, especially given the “legacy” that exists for the latter. And the fourth is the size of individual moduli: Although, as we shall see, certain RNS-arithmetic operations do not require carries between digits, which is one of the primary advantages of RNS, this is so only between *digits*. Since a digit is ultimately represented in binary, there will be carries between bits, and therefore it is important to ensure that digits (and, therefore, the moduli) are not too large. Low-precision digits also make it possible to realize cost-effective table-lookup implementations of arithmetic operations. But, on the other hand, if the moduli are small, then a large number of them may be required to ensure a sufficient dynamic range. Of course, ultimately the choices made, and indeed whether RNS is useful or not, depend on the particular applications and technologies at hand.

1.3.2 *Negative numbers*

Some applications require that it be possible to represent negative numbers as well as positive ones. As with the conventional number systems, any one of the radix complement, diminished-radix complement, or sign-and-magnitude notations may be used in RNS for such representation. The merits and drawbacks of choosing one over the other are similar to those for the conventional notations. In contrast with the conventional notations, however, the determination of sign is much more difficult with the residue notations, as is magnitude-comparison. This is the case even with sign-and-magnitude notation, since determining the sign of the result of an arithmetic operation such as addition or subtraction is not easy—even if the signs of the operands are known. This problem, which is discussed in slightly more detail below and in Chapter 6, imposes many limitations on the application of RNS.

The extension of sign-and-magnitude notation to RNS involves the use of a single sign-digit or prepending to each residue in a representation an extra bit or digit for the sign; we shall assume the former. For the complement notations, the range of representable numbers is usually partitioned into two approximately equal parts, such that approximately half of the numbers are positive and the rest are negative. Thus, if the moduli used are m_1, m_2, \dots, m_N , then there are $M \triangleq \prod_{i=1}^N m_i$ representable numbers, and every representable number, X , satisfies one of two relations:

$$-\frac{M-1}{2} \leq X \leq \frac{M-1}{2} \quad \text{if } M \text{ is odd}$$

$$-\frac{M}{2} \leq X \leq \frac{M}{2} - 1 \quad \text{if } M \text{ is even}$$

Then, for complement notation, if $\langle x_1, x_2, \dots, x_N \rangle$ is the representation of X , where $x_i = |X|_{m_i}$, then the representation of $-X$ is $\langle \bar{x}_1, \bar{x}_2, \dots, \bar{x}_N \rangle$, where \bar{x}_i is the m_i 's-complement, i.e. the radix complement, or the $(m_i - 1)$'s-complement, i.e. the diminished-radix complement, of x_i . For example, with the moduli-set $\{2, 3, 5, 7\}$, the representation of seventeen is $\langle 1, 2, 2, 3 \rangle$ and the radix-complement representation of its negation is $\langle 1, 1, 3, 4 \rangle$, from $\langle 2 - 1, 3 - 2, 5 - 2, 7 - 3 \rangle$. The justification for taking the complement (negation) of each residue digit is that $|m_i - x_i|_{m_i} = -x_i$.

If we again take $[0, M - 1]$ as the nominal range of an RNS, then it will be seen that in the last example $\langle 1, 1, 3, 4 \rangle$ is also the representation of 193. That is, in splitting the range $[0, M - 1]$, we take for positive numbers the subrange $[0, M/2 - 1]$ if M is even, or $[0, (M - 1)/2]$ if M is odd, and correspondingly, for the negative numbers we take $[M/2, M - 1]$ or $[(M + 1)/2, M - 1]$. This makes sense since $|M - X|_{M} = -X$ (in the last example $210 - 17 = 193$) and fits in with the discussion in Section 1.1 as to how we may also view diminished-radix complement and radix-complement representations as the representations of positive numbers.

Unless otherwise stated, we shall generally assume that we are dealing with just the positive numbers.

1.3.3 Basic arithmetic

The standard arithmetic operations of addition/subtraction and multiplication are easily implemented with residue notation, depending on the choice of the moduli, but division is much more difficult. The latter is not surpris-

ing, in light of the statement above on the difficulties of sign-determination and magnitude-comparison. Residue addition is carried out by individually adding corresponding digits, relative to the modulus for their position. That is, a carry-out from one digit position is not propagated into the next digit position.

If the given moduli are m_1, m_2, \dots, m_N , $X \cong \langle x_1, x_2, \dots, x_N \rangle$ and $Y \cong \langle y_1, y_2, \dots, y_N \rangle$, i.e. $x_i = |X|_{m_i}$ and $y_i = |Y|_{m_i}$, then we may define the addition $X + Y = Z$ by

$$\begin{aligned} X + Y &\cong \langle x_1, x_2, \dots, x_N \rangle + \langle y_1, y_2, \dots, y_N \rangle \\ &= \langle z_1, z_2, \dots, z_N \rangle && \text{where } z_i = |x_i + y_i|_{m_i} \\ &\cong Z \end{aligned}$$

As an example, with the moduli-set $\{2, 3, 5, 7\}$, the representation of seventeen is $\langle 1, 2, 2, 3 \rangle$, that of nineteen is $\langle 1, 1, 4, 5 \rangle$, and adding the two residue numbers yields $\langle 0, 0, 1, 1 \rangle$, which is the representation for thirty-six in that system.

Subtraction may be carried out by negating (in whatever is the chosen notation) the subtrahend and adding to the minuend. This is straightforward for numbers in diminished-radix complement or radix complement notation. For numbers represented in residue sign-and-magnitude, a slight modification of the algorithm for conventional sign-and-magnitude is necessary: the sign digit is fanned out to all positions in the residue representation, and addition then proceeds as in the case for unsigned numbers but with a conventional sign-and-magnitude algorithm.

Multiplication too can be performed simply by multiplying corresponding residue digit-pairs, relative to the modulus for their position; that is, multiply digits and ignore or adjust an appropriate part of the result. If the given moduli are m_1, m_2, \dots, m_N , $X \cong \langle x_1, x_2, \dots, x_N \rangle$ and $Y \cong \langle y_1, y_2, \dots, y_N \rangle$, i.e. $x_i = |X|_{m_i}$ and $y_i = |Y|_{m_i}$, then we may define the multiplication $X \times Y = Z$ by

$$\begin{aligned} X \times Y &\cong \langle x_1, x_2, \dots, x_N \rangle \times \langle y_1, y_2, \dots, y_N \rangle \\ &= \langle z_1, z_2, \dots, z_N \rangle && \text{where } z_i = |x_i \times y_i|_{m_i} \\ &\cong Z \end{aligned}$$

As an example, with the moduli-set $\{2, 3, 5, 7\}$, the representation of seventeen is $\langle 1, 2, 2, 3 \rangle$, that of nineteen is $\langle 1, 1, 4, 5 \rangle$, and that of their product, three hundred and twenty-three, is $\langle 1, 2, 3, 1 \rangle$. As with addition, obtaining the modulus with respect to m_i can be implemented without division, and quite efficiently, if m_i is of a suitable form.

Basic fixed-point division consists, essentially, of a sequence of subtractions, magnitude-comparisons, and selections of the quotient-digits. But comparison in RNS is a difficult operation, because RNS is not positional or weighted. Consider, for example, the fact that with the moduli-set $\{2, 3, 5, 7\}$, the number represented by $\langle 0, 0, 1, 1 \rangle$ is almost twice that represented by $\langle 1, 1, 4, 5 \rangle$, but this is far from apparent. It should therefore be expected that division will be a difficult operation, and it is. One way in which division could be readily implemented is to convert the operands to a conventional notation, use a conventional division procedure, and then convert the result back into residue notation. The conversions are, however, time consuming, and a direct algorithm should be used if possible. The essence of a good RNS division algorithm will therefore be a relatively efficient method of performing magnitude-comparisons. Such algorithms are discussed in Chapter 6. In a sense, all of them require what are essentially conversions out of the RNS, and, compared with conventional division algorithms, all of them are rather unsatisfactory.

1.3.4 Conversion

The most direct way to convert from a conventional representation to a residue one, a process known as *forward conversion*, is to divide by each of the given moduli and then collect the remainders. This, however, is likely to be a costly operation if the number is represented in an arbitrary radix and the moduli are arbitrary. If, on the other hand, the number is represented in radix-2 (or a radix that is a power of two) and the moduli are of a suitable form (e.g. $2^n - 1$), then there are procedures that can be implemented with more efficiency. The conversion from residue notation to a conventional notation, a process known as *reverse conversion*, is more difficult (conceptually, if not necessarily in the implementation) and so far has been one of the major impediments to the adoption of RNS. One way in which it can be done is to assign weights to the digits of a residue representation and then produce a “conventional” (i.e. positional, weighted) mixed-radix representation from this. This mixed-radix representation can then be converted into whatever conventional form is desired. In practice, the use of a direct conversion procedure for the latter can be avoided by carrying out the arithmetic of the conversion in the notation for the result. Another approach involves the use of the Chinese Remainder Theorem, which is the basis for many algorithms for conversion from residue to conventional notation; this too involves, in essence, the extraction of a mixed-radix representation.

1.3.5 Base extension

We have so far assumed that once the moduli-set has been determined, then all operations are carried out with respect to only that set. That is not always so. A frequently occurring computation is that of *base extension*, which is defined as follows. Given a residue representation $\langle |X|_{m_1}, |X|_{m_2}, \dots, |X|_{m_N} \rangle$ and an additional set of moduli, $m_{N+1}, m_{N+2}, \dots, m_{N+K}$, such that $m_1, m_2, \dots, m_N, m_{N+1}, \dots, m_{N+K}$ are all pairwise relatively prime, we want to compute the residue representation $\langle |X|_{m_1}, |X|_{m_2}, \dots, |X|_{m_N}, |X|_{m_{N+1}}, \dots, |X|_{m_{N+K}} \rangle$. Base extension is very useful in dealing with the difficult operations of reverse conversion, division, dynamic-range extension, magnitude-comparison, overflow-detection, and sign-determination. The operation is discussed in more detail in Chapters 2 and 7.

1.3.6 Alternative encodings

The basic encoding for values (residues, etc.) in almost all RNS implementations is conventional binary. That is, if the largest possible value is N , then each value is represented in $\lceil \log_2 N \rceil$ bits, and the weighted sum of the bits in a representation is the value represented. This is the encoding we shall assume throughout the book. Nevertheless, alternative encodings are possible, and one that is perhaps worthy of note is *one-hot encoding* (OHE), which is a special case of *n-of-m* encoding⁷. In OHE, if the largest possible value is N , then each data value is represented in N bits. Such a representation for a value n has 0s in all bit positions, except the n th, which is set to 1. Evidently OHE has less representational efficiency than straightforward binary encoding. The basic idea of OHE is not new, but its application is to RNS is. This is because OHE is not suitable for the representation of large values; but in RNS the values generally tend to be small, and so OHE is more practical.

The nominal advantages of OHE residue numbers are as follows [6]. First, a change in the value of the represented number requires changing at most two bits; this is the smallest change possible and is beneficial for power consumption, since, in current technology, that is related to the number of transitional activities. Second, arithmetic operations are simply implemented by shifting: addition consists of rotating one operand by an amount equal to the value of the other operand, and multiplication

⁷In *n-hot* (*n-of-m*) encoding, a value is represented by 1s on n out of m lines.

(assuming moduli-sets with primitive roots) can be realized by rotations and index (“log”) and reverse-index (“anti-log”) operations that are simple permutations of wires. (Multiplication where there is no primitive root can be realized by using barrel shifters.) Other operations that are easily realized are inverse-calculation (the analogues, discussed in Chapter 2, of negation and reciprocation in conventional arithmetic), index computation, and modulus conversion. Of course, whether the nominal advantages can be turned into practical advantages is highly dependent on the technology at hand—for example, on whether, for arithmetic operations, barrel shifters can be realized with better efficiency than in conventional circuits. Nevertheless, initial results appear promising.

All that said, OHE does have two significant limitations. First, the sizes of basic arithmetic circuits, such as adders and multipliers, grow at a rate $O(m^2)$, where m is the modulus, in contrast, with conventional binary circuits, in which the rate of growth can be constrained to $O(m \log m)$. Second, the poor representational efficiency means that the cost of interconnections will be much higher than with conventional circuits—at least, $O(m)$ growth versus $O(\log m)$; this may be critical in current technologies, in which interconnections play an increasingly larger role.

1.4 Using residue number systems

We now give an example that demonstrates a typical application of residue number systems: a multiply-accumulate operation over a sequence of scalars. This is an operation that occurs very frequently in digital-signal-processing applications, one of the areas for which RNS is suitable.

EXAMPLE. Let the moduli-set be $\{m_i\} = \{2, 3, 5, 7\}$. The dynamic range of this moduli-set 210. Suppose we wish to evaluate the sum-of-products $7 \times 3 + 16 \times 5 + 47 \times 2$. The residue-sets are

$$|2|_{m_i} = \{0, 2, 2, 2\}$$

$$|3|_{m_i} = \{1, 0, 3, 3\}$$

$$|5|_{m_i} = \{1, 2, 0, 5\}$$

$$|7|_{m_i} = \{1, 1, 2, 0\}$$

$$|16|_{m_i} = \{0, 1, 1, 2\}$$

$$|47|_{m_i} = \{1, 2, 2, 5\}$$

We proceed by first computing the products by multiplying the corresponding residues:

$$\begin{aligned} |7 \times 3|_{m_i} &= \{|1 \times 1|_2, |1 \times 0|_3, |2 \times 3|_5, |0 \times 3|_7\} \\ &= \{1, 0, 1, 0\} \end{aligned}$$

$$\begin{aligned} |16 \times 5|_{m_i} &= \{|0 \times 1|_2, |1 \times 2|_3, |1 \times 0|_5, |2 \times 5|_7\} \\ &= \{0, 2, 0, 3\} \end{aligned}$$

$$\begin{aligned} |47 \times 2|_{m_i} &= \{|1 \times 0|_2, |2 \times 2|_3, |2 \times 2|_5, |5 \times 2|_7\} \\ &= \{0, 1, 4, 3\} \end{aligned}$$

Now that we have computed the products, the sum of products can be evaluated by adding the corresponding residues:

$$\begin{aligned} |7 \times 3 + 16 \times 5 + 47 \times 2|_{m_i} &= \{|1 + 0 + 0|_2, |0 + 2 + 1|_3, |1 + 0 + 4|_5, \\ &\quad |0 + 3 + 3|_7\} \\ &= \{1, 0, 0, 6\} \end{aligned}$$

END EXAMPLE

The residue representation $\langle 1, 0, 0, 6 \rangle$ corresponds to the decimal 195, which is correct. Consider now the cost of the multipliers required if RNS was not used. We need to multiply 7 and 3, and this requires a 5-bit multiplier. Similarly, the multipliers required for the products of 16 and 5 and 47 and 2 are 8-bit and 9-bit, respectively. But with the use of RNS, 6-bit multipliers would suffice for all the multiplications. Thus the use of RNS can result in considerable savings in operational-time, area, and power. Moreover, all multiplications can be done in parallel in order to increase the overall speed. This shows a primary advantage of RNS relative to conventional notations. Although there is a final overhead in conversion between the number systems, it may be considered as a one-off overhead.

Whether RNS is useful or not crucially depends on the application. For appropriate applications, there have been some clear evidence of the advantages. As an example, [5] gives the design of a digital-signal processor whose arithmetic units are fully in RNS. An evaluation of this processor shows better performance at lower cost (chip area) than with a conventional system.

1.5 Summary

The main point of this chapter has been to introduce the essential aspects of residue number systems and to give a brief, but essentially complete, summary of what is to come in the rest of the book. We have also reviewed the three standard number systems and one other unconventional number system used for arithmetic in computers; all of these have some relationships with residue arithmetic, especially in implementation. Major differences between residue arithmetic and standard computer arithmetic are the absence of carry-propagation (in the former), in the two main operations of addition and multiplication, and the relatively low precisions required, which leads to practical table-lookup implementations. In practice, these may make residue arithmetic worthwhile, even though in terms of asymptotic bounds such arithmetic has little advantage over conventional arithmetic. A wealth of information on residue number systems, and their applications, will be found in [1, 2], although these may currently be difficult to obtain.

We have noted above that addition and multiplication are easy to realize in a residue number system but that operations that require the determination of magnitudes (e.g. magnitude-comparison, division, overflow, and sign-determination) are difficult; reverse conversion is also difficult.⁸ Together, these suggest that the best applications for residue number systems are those with numerous additions and multiplications, but relatively few conversions or magnitude operations, and in which the results of all arithmetic operations lie within a known range. Such applications are typical of digital signal processing, in which, for example, the computation of inner-products (essentially, sequences of multiply-add operations) is quite common.

Residue number systems are also useful in error detection and correction. This is apparent, given the independence of digits in a residue-number representation: an error in one digit does not corrupt any other digits. In general, the use of redundant moduli, i.e. extra moduli that play no role in determining the dynamic range, facilitates both error detection and correction. But even without redundant moduli, fault-tolerance is possible, since computation can still continue after the isolation of faulty digit-positions, provided that a smaller dynamic range is acceptable. Lastly, RNS can help speed up complex-number arithmetic: for example, the multiplication of

⁸The essence of the problem with these operations is that they require interactions between all the digits of an RNS representation.

two complex numbers requires four multiplications and two additions when done conventionally but only two multiplications and two additions with the right sort of RNS. (See Chapter 2.)

Figure 1.1 shows the general structure of an RNS processor.

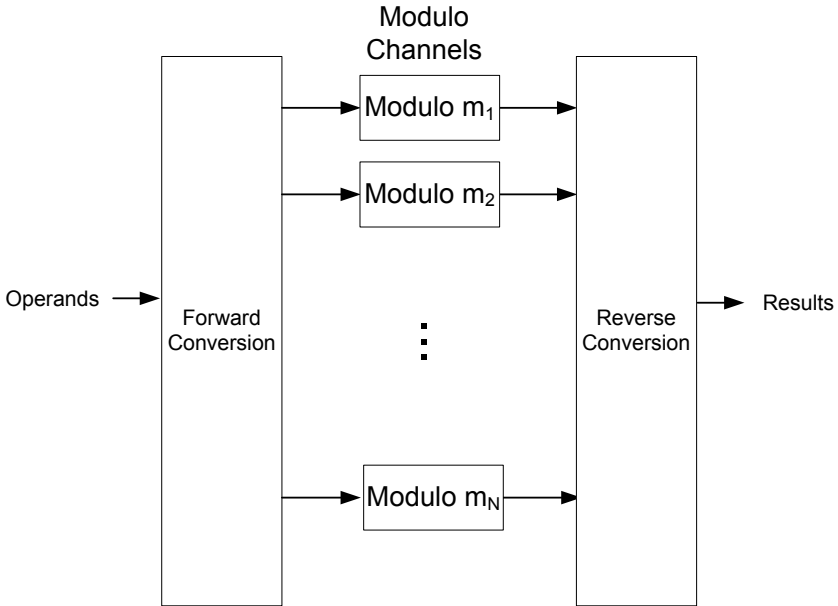


Figure 1.1: General structure of an RNS processor

References

- (1) N. S. Szabo and R. I. Tanaka. 1967. *Residue Arithmetic and Its Applications to Computer Technology*. McGraw-Hill, New York.
- (2) M. A. Soderstrand et. al. 1986. *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. IEEE Press, California.
- (3) A. Lindstrom, M. Nordseth, L. Bengtsson, and A. Omondi. 2003. Arithmetic circuits combining residue and signed-digit representations. In: A. R. Omondi and S. Sedhukin, Eds. *Advances in Computer Systems Architecture, Lecture Notes In Computer Science, vol. 2823* (Springer-Verlag, Heidelberg) pp 246–257.

- (4) J.-M. Muller. 1997. *Elementary Functions: Algorithms and Implementations*. Birkhauser, Boston.
- (5) R. Charles and L. Sousa. 2003. "RDSP: A RISC DSP based on residue number system." In: *Proceedings, Euromicro Symposium on Digital System Design*.
- (6) W. A. Chren. 1998. One-hot residue encoding for low delay-power product CMOS design. *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, 45(3):303–313.
- (7) A. R. Omondi. 1994. *Computer Arithmetic Systems*. Prentice-Hall, UK.