Redundant Number Systems

"Numbers constitute the only universal language."

NATHANAEL WEST

This chapter deals with the representation of signed fixed-point numbers using a positive integer radix *r* and a redundant digit set composed of more than *r* digit values. After showing that such representations eliminate carry propagation, we cover variations in digit sets, addition algorithms, input/output conversions, and arithmetic support functions. Chapter topics include:

3.1 COPING WITH THE CARRY PROBLEM

Addition is a primary building block in implementing arithmetic operations. If addition is slow or expensive, all other operations suffer in speed or cost. Addition can be slow and/or expensive because:

a. With k-digit operands, one has to allow for O(k) worst-case carry-propagation stages in simple ripple-carry adder design.

b. The carry computation network is a major source of complexity and cost in the design of carry-lookahead and other fast adders.

The carry problem can be dealt with in several ways:

1. Limit carry propagation to within a small number of bits.
2. Detect the end of propagation rather than wait for worst-case time.
3. Speed up propagation via lookahead and other methods.
4. Ideal: Eliminate carry propagation altogether!

As examples of option 1, hybrid-redundant and residue number system representations are covered in Section 3.4 and Chapter 4, respectively. Asynchronous adder design (option 2) is considered in Section 5.4. Speedup methods for carry propagation are covered in Chapters 6 and 7.

In the remainder of this chapter, we deal with option 4, focusing first on the question: Can numbers be represented in such a way that addition does not involve carry propagation? We will see shortly that this is indeed possible. The resulting number representations can be used as the primary encoding scheme in the design of high-performance systems and are also useful in representing intermediate results in machines that use conventional number representation.

We begin with a decimal example ($r = 10$), assuming the standard digit set [0, 9]. Consider the addition of the following two decimal numbers without carry propagation. For this, we simply compute "position sums" and write them down in the corresponding columns. We can use the symbols $A = 10, B = 11, C = 12$, etc., for the extended digit values or simply represent them with two standard digits.

$$\begin{array}{rrrrrrrl} & 5 & 7 & 8 & 2 & 4 & 9 & \\ + & 6 & 2 & 9 & 3 & 8 & 9 & \text{Operand digits in [0, 9]} \\ \cline{2-7} & 11 & 9 & 17 & 5 & 12 & 18 & \text{Position sums in [0, 18]} \end{array}$$

So, if we allow the digit set [0, 18], the scheme works, but only for the first addition! Subsequent additions will cause problems.

Consider now adding two numbers in the radix-10 number system using the digit set [0, 18]. The sum of digits for each position is in [0, 36], which can be decomposed into an interim sum in [0, 16] and a transfer digit in [0, 2]. In other words

$$[0, 36] = 10 \times [0, 2] + [0, 16]$$

Adding the interim sum and the incoming transfer digit yields a digit in [0, 18] and creates no new transfer. In interval notation, we have

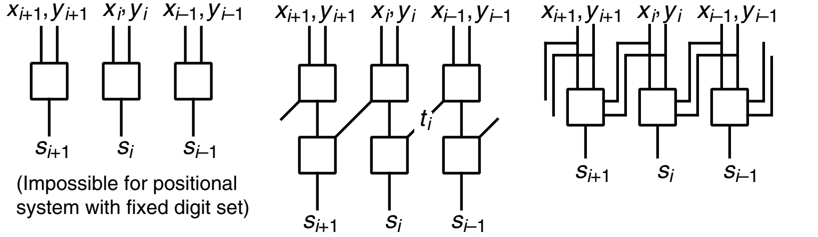
$$[0, 16] + [0, 2] = [0, 18]$$

Figure 3.1 shows an example addition.

So, even though we cannot do true carry-free addition (Fig. 3.2a), the next best thing, where carry propagates by only one position (Fig. 3.2b), is possible if we use the digit set [0, 18] in radix 10. We refer to this best possible scheme as "carry-free" addition. The key to the ability to do carry-free addition is the representational redundancy that provides multiple encodings for some numbers. Figure 3.2c shows that the single-stage

Figure 3.1 Adding radix-10 numbers with the digit set [0, 18].

```
        11    9   17   10   12   18
    +    6   12    9   10    8   18    Operand digits in [0, 18]
      -----------------------------
        17   21   26   20   20   36    Position sums in [0, 36]
         |    |    |    |    |    |
         7   11   16    0   10   16    Interim sums in [0, 16]
     /    /    /    /    /    /
    1    1    1    2    1    2         Transfer digits in [0, 2]
  ---------------------------------
    1    8   12   18    1   12   16    Sum digits in [0, 18]
```



(a) Ideal single-stage carry-free. **(b)** Two-stage carry-free. **(c)** Single-stage with lookahead.

Figure 3.2 Ideal and practical carry-free addition schemes.

Figure 3.3 Adding radix-10 numbers with the digit set [0, 11].

```
        11   10    7   11    3    8
    +    7    2    9   10    9    8    Operand digits in [0, 11]
      -----------------------------
        18   12   16   21   12   16    Position sums in [0, 22]
         |    |    |    |    |    |
         8    2    6    1    2    6    Interim sums in [0, 9]
     /    /    /    /    /    /
    1    1    1    2    1    1         Transfer digits in [0, 2]
  ---------------------------------
    1    9    3    8    2    3    6    Sum digits in [0, 11]
```

propagation of transfers can be eliminated by a simple lookahead scheme; that is, instead of first computing the transfer into position i based on the digits x_{i-1} and y_{i-1} and then combining it with the interim sum, we can determine s_i directly from x_i, y_i, x_{i-1}, and y_{i-1}. This may make the adder logic somewhat more complex, but in general the result is higher speed.

In the decimal example of Fig. 3.1, the digit set [0, 18] was used to effect carry-free addition. The 9 “digit” values 10 through 18 are redundant. However, we really do not need this much redundancy in a decimal number system for carry-free addition; the digit set [0, 11] will do. Our example addition (after converting the numbers to the new digit set) is shown in Fig. 3.3.

A natural question at this point is: How much redundancy in the digit set is needed to enable carry-free addition? For example, will the example addition of Fig. 3.3 work with the digit set [0, 10]? (Try it and see.) We will answer this question in Section 3.5.

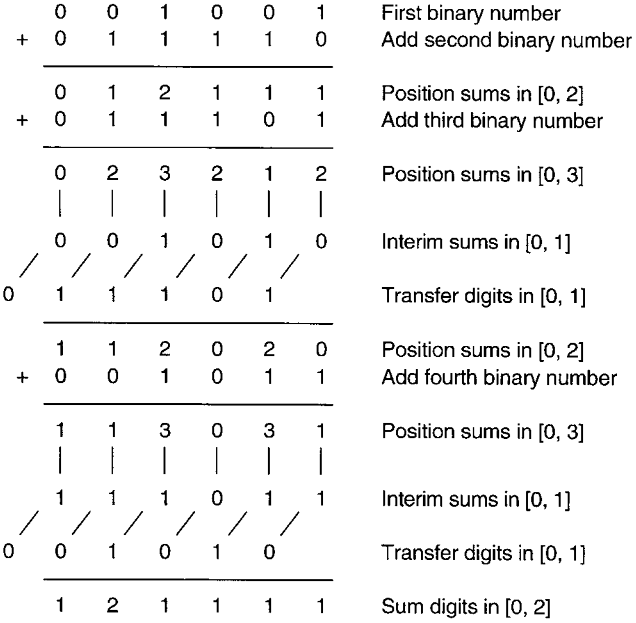
3.2 REDUNDANCY IN COMPUTER ARITHMETIC

Redundancy is used extensively for speeding up arithmetic operations. The oldest example, first suggested in 1959 [Metz59], pertains to carry-save or stored-carry numbers using the radix-2 digit set [0, 2] for fast addition of a sequence of binary operands. Figure 3.4 provides an example, showing how the intermediate sum is kept in stored-carry format, allowing each subsequent addition to be performed in a carry-free manner.

Why is this scheme called carry-save or stored-carry? Figure 3.5 provides an explanation. Let us use the 2-bit encoding

$$0 : (0,0), \qquad 1 : (0,1) \text{ or } (1,0), \qquad 2 : (1,1)$$

to represent the digit set [0, 2]. With this encoding, each stored-carry number is really composed of two binary numbers, one for each bit of the encoding. These two binary numbers can be added to an incoming binary number, producing two binary numbers composed of the sum bits kept in place and the carry bits shifted one position to the left. These sum and carry bits form the partial sum and can be stored in two registers for the next addition. Thus, the carries are "saved" or "stored" instead of being allowed to propagate.



0 0 1 0 0 1 First binary number
+ 0 1 1 1 1 0 Add second binary number
0 1 2 1 1 1 Position sums in [0, 2]
+ 0 1 1 1 0 1 Add third binary number
0 2 3 2 1 2 Position sums in [0, 3]
0 0 1 0 1 0 Interim sums in [0, 1]
0 1 1 1 0 1 Transfer digits in [0, 1]
1 1 2 0 2 0 Position sums in [0, 2]
+ 0 0 1 0 1 1 Add fourth binary number
1 1 3 0 3 1 Position sums in [0, 3]
1 1 1 0 1 1 Interim sums in [0, 1]
0 0 1 0 1 0 Transfer digits in [0, 1]
1 2 1 1 1 1 Sum digits in [0, 2]

Figure 3.4 Addition of four binary numbers, with the sum obtained in stored-carry form.

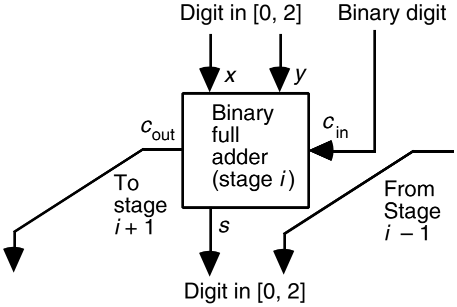


Figure 3.5 Using an array of independent binary full adders to perform carry-save addition.

Figure 3.5 shows that one stored-carry number and one standard binary number can be added to form a stored-carry sum in a single full-adder delay (2–4 gate levels, depending on the full adder's logic implementation of the outputs $s = x \oplus y \oplus c_{\text{in}}$ and $c_{\text{out}} = xy \vee xc_{\text{in}} \vee yc_{\text{in}}$). This is significantly faster than standard carry-propagate addition to accumulate the sum of several binary numbers, even if a fast carry-lookahead adder is used for the latter. Of course once the final sum has been obtained in stored-carry form, it may have to be converted to standard binary by using a carry-propagate adder to add the two components of the stored-carry number. The key point is that the carry-propagation delay occurs only once, at the very end, rather than in each addition step.

Since the carry-save addition scheme of Fig. 3.5 converts three binary numbers to two binary numbers with the same sum, it is sometimes referred to as a 3/2 reduction circuit or (3; 2) counter. The latter name reflects the essential function of a full adder: it counts the number of 1s among its three input bits and outputs the result as a 2-bit binary number. More on this in Chapter 8.

Other examples of the use of redundant representations in computer arithmetic are found in fast multiplication and division schemes, where the multiplier or quotient is represented or produced in redundant form. More on these in Parts III and IV.

3.3 DIGIT SETS AND DIGIT-SET CONVERSIONS

Conventional radix-r numbers use the standard digit set $[0, r - 1]$. However, many other redundant and nonredundant digit sets are possible. A necessary condition is that the digit set contain at least r different digit values. If it contains more than r values, the number system is redundant.

Conversion of numbers between standard and other digit sets is quite simple and essentially entails a digit-serial process in which, beginning at the right end of the given number, each digit is rewritten as a valid digit in the new digit set and a transfer (carry or borrow) into the next higher digit position. This conversion process is essentially like carry propagation in that it must be done from right to left and, in the worst case, the most significant digit is affected by a "carry" coming from the least significant position. The following examples illustrate the process (see also the examples at the end of Section 2.6).

■ **EXAMPLE 3.1** Convert the following radix-10 number with the digit set [0, 18] to one using the conventional digit set [0, 9].

	11	9	17	10	12	18	Rewrite 18 as 10 (carry 1) +8
	11	9	17	10	13	8	13 = 10 (carry 1) + 3
	11	9	17	11	3	8	11 = 10 (carry 1) + 1
	11	9	18	1	3	8	18 = 10 (carry 1) + 8
	11	10	8	1	3	8	10 = 10 (carry 1) + 0
	12	0	8	1	3	8	12 = 10 (carry 1) + 2
1	2	0	8	1	3	8	Answer: all digits in [0, 9]

■ **EXAMPLE 3.2** Convert the following radix-2 carry-save number to binary; that is, from digit set [0, 2] to digit set [0, 1].

	1	1	2	0	2	0	Rewrite 2 as 2 (carry 1) + 0
	1	1	2	1	0	0	2 = 2 (carry 1) + 0
	1	2	0	1	0	0	2 = 2 (carry 1) + 0
	2	0	0	1	0	0	2 = 2 (carry 1) + 0
1	0	0	0	1	0	0	Answer: all digits in [0, 1]

Another way to accomplish the preceding conversion is to decompose the carry-save number into two numbers, both of which have 1s where the original number has a digit of 2. The sum of these two numbers is then the desired binary number.

	1	1	1	0	1	0	First number: "sum" bits
+	0	0	1	0	1	0	Second number: "carry" bits
1	0	0	0	1	0	0	Sum of the two numbers

■ **EXAMPLE 3.3** Digit values do not have to be positive. We reconsider Example 3.1 using the asymmetric target digit set [−6, 5].

	11	9	17	10	12	18	Rewrite 18 as 20 (carry 2) − 2
	11	9	17	10	14	−2	14 = 10 (carry 1) + 4
	11	9	17	11	4	−2	11 = 10 (carry 1) + 1
	11	9	18	1	4	−2	18 = 20 (carry 2) − 2
	11	11	−2	1	4	−2	11 = 10 (carry 1) + 1
	12	1	−2	1	4	−2	12 = 10 (carry 1) + 2
1	2	1	−2	1	4	−2	Answer: all digits in [−6, 5]

On line 2 of this conversion, we could have rewritten 14 as 20 (carry 2) − 6, which would have led to a different, but equivalent, representation. In general, several representations may be possible with a redundant digit set.

■ **EXAMPLE 3.4** If we change the target digit set of Example 3.2 from [0, 1] to [−1, 1], we can do the conversion digit-serially as before. However, carry-free conversion is possible for this example if we rewrite each 2 as 2 (carry 1) + 0 and each 1 as 2 (carry 1) −1. The resulting interim digits in [−1, 0] can absorb an incoming carry of 1 with no further propagation.

	1	1	2	0	2	0	Given carry-save number
	−1	−1	0	0	0	0	Interim digits in [−1, 0]
1	1	1	0	1	0		Transfer digits in [0, 1]
1	0	0	0	1	0	0	Answer: all digits in [−1, 1]

3.4 GENERALIZED SIGNED-DIGIT NUMBERS

We have seen thus far that the digit set of a radix-r positional number system need not be the standard set $[0, r-1]$. Using the digit set $[-1, 1]$ for radix-2 numbers was proposed by E. Collignon as early as 1897 [Glas81]. Whether this was just a mathematical curiosity, or motivated by an application or advantage, is not known. In the early 1960s, Avizienis [Aviz61] defined the class of signed-digit number systems with symmetric digit sets $[-\alpha, \alpha]$ and radix $r > 2$, where α is any integer in the range $\lfloor r/2 \rfloor + 1 \le \alpha \le r - 1$. These number systems allow at least $2\lfloor r/2 \rfloor + 3$ digit values, instead of the minimum required r values, and are thus redundant.

Subsequently, redundant number systems with general, possibly asymmetric, digit sets of the form $[-\alpha, \beta]$ were studied as tools for unifying all redundant number representations used in practice. This class is called "generalized signed-digit (GSD) representation" and differs from the ordinary signed-digit (OSD) representation of Avizienis in its more general digit set as well as the possibility of higher or lower redundancy.

Binary stored-carry numbers, with $r = 2$ and digit set [0, 2], offer a good example for the usefulness of asymmetric digit sets. Higher redundancy is exemplified by the digit set $[-7, 7]$ in radix 4 or [0, 3] in radix 2. An example for lower redundancy is the binary signed-digit (BSD) representation with $r = 2$ and digit set $[-1, 1]$. None of these is covered by OSD.

An important parameter of a GSD number system is its *redundancy index*, defined as $\rho = \alpha + \beta + 1 - r$ (i.e., the amount by which the size of its digit set exceeds the size r of a nonredundant digit set for radix r). Figure 3.6 presents a taxonomy of redundant and nonredundant positional number systems showing the names of some useful subclasses and their various relationships. Note that the redundancy index ρ is quite general and can be applied to any digit set. Another way of quantifying the redundancy of a number system with the symmetric digit set $[-\alpha, \alpha]$ in radix r is to use the ratio $h = \alpha/(r - 1)$. This formulation of redundancy, which is inapplicable to the general digit set $[-\alpha, \beta]$, has been used in connection with high-radix division algorithms, to be discussed in Chapter 14. Besides its general inapplicability, the index h suffers from the problem that it varies from $\frac{1}{2}$ (for no redundancy), through 1 (for $\alpha = r - 1$), to values larger than 1

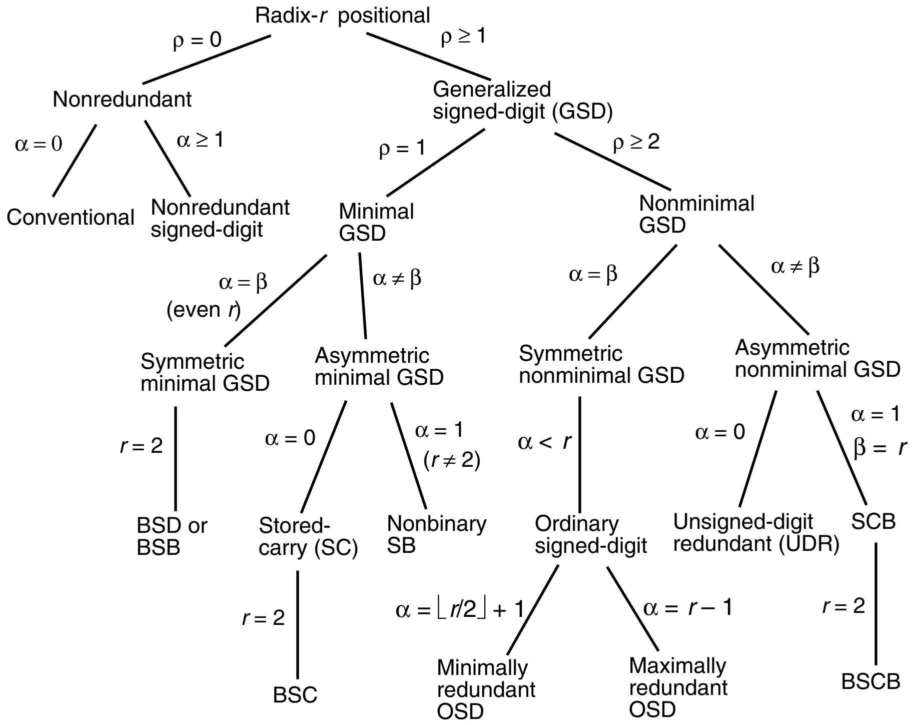


Figure 3.6 A taxonomy of redundant and nonredundant positional number systems.

x_i	1	−1	0	−1	0	Representation of +6
(*s*, *v*)	01	11	00	11	00	Sign and value encoding
2's-compl	01	11	00	11	00	2-bit 2's-complement
(*n*, *p*)	01	10	00	10	00	Negative and positive flags
(*n*, *z*, *p*)	001	100	010	100	010	1-out-of-3 encoding

Figure 3.7 Four encodings for the BSD digit set $[-1, 1]$.

for highly redundant number representation systems. Encountering redundancy indices below 1 is unusual and could be misleading.

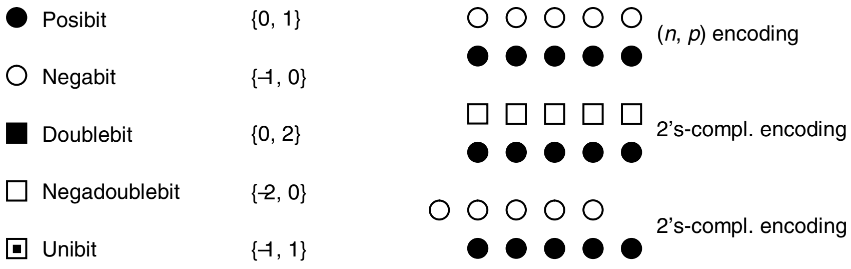
Any hardware implementation of GSD arithmetic requires the choice of a binary encoding scheme for the $\alpha + \beta + 1$ digit values in the digit set $[-\alpha, \beta]$. Multivalued logic realizations have been considered, but we limit our discussion here to binary logic and proceed to show the importance and implications of the encoding scheme chosen through some examples.

Consider, for example, the BSD number system with $r = 2$ and the digit set $[-1, 1]$. One needs at least 2 bits to encode these three digit values. Figure 3.7 shows four of the many possible encodings that can be used.

With the (n, p) encoding, the code $(1, 1)$ may be considered an alternate representation of 0 or else viewed as an invalid combination. Many implementations have shown that the (n, p) encoding tends to simplify the hardware and also increases the speed by reducing the number of gate levels [Parh88]. The 1-out-of-3 encoding requires more bits per number but allows the detection of some storage and processing errors.

The (n,p) and 2's-complement encodings of Fig. 3.7 are examples of encodings in which two-valued signals having various weights collectively represent desired values. Figure 3.8a depicts three new symbols, besides posibits and negabits previously introduced in Figs. 1.4 and 2.13. A *doublebit* represents one of the two values in the set $\{0, 2\}$. A *negadoublebit* is a negatively weighted doublebit. Finally, a *unibit* assumes one of the two values in $\{-1, 1\}$. A posibit and a negabit together represent one of the values in the set $\{-1, 0, 1\}$, yielding the (n,p) encoding of a BSD. A negadoublebit and a posibit form a 2-bit 2's-complement number capable of representing a value in $[-2, 1]$ and thus a BSD. These two encodings for a 5-digit BSD number are shown in Fig. 3.8b. The third representation in Fig. 3.8b is derived from the second one by shifting the negadoublebits to the left by one position and changing them into negabits. Each BSD digit now spans two digit positions in its encoding. These weighted bit-set encodings have been found quite useful for the efficient representation and processing of redundant numbers [Jabe05].

Hybrid signed-digit representations [Phat94] came about from an attempt to strike a balance between algorithmic speed and implementation cost by introducing redundancy in selected positions only. For example, standard binary representation may be used with BSD digits allowed in every third position, as shown in the addition example of Fig. 3.9.



(a) Extended dot notation (b) Encodings for a BSD number

Figure 3.8 Extended dot notation and its use in visualizing some BSD encodings.

Figure 3.9 Example of addition for hybrid signed-digit numbers.

	BSD	B	B	BSD	B	B	BSD	B	B	Type
	1	0	1	–1	0	1	–1	0	1	x_i
+	0	1	1	–1	1	0	0	1	0	y_i
	1	1	2	–2	1	1	–1	1	1	p_i
	–1			0			–1			w_i
1			–1			0			0	t_{i+1}
1	–1	1	1	0	1	1	–1	1	1	s_i

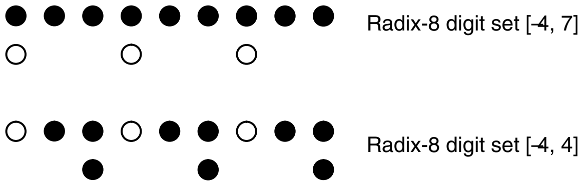


Figure 3.10 Two hybrid-redundant representations in extended dot notation.

The addition algorithm depicted in Fig. 3.9 proceeds as follows. First one completes the position sums p_i that are in [0, 2] for standard binary and $[-2, 2]$ in BSD positions. The BSD position sums are then broken into an interim sum w_i and transfer t_{i+1}, both in $[-1, 1]$. For the interim sum digit, the value 1 (-1) is chosen only if it is certain that the incoming transfer cannot be 1 (-1); that is, when the two binary operand digits in position $i - 1$ are (not) both 0s. The worst-case carry propagation spans a single group, beginning with a BSD that produces a transfer digit in $[-1, 1]$ and ending with the next higher BSD position.

More generally, the group size can be g rather than 3. A larger group size reduces the hardware complexity (since the adder block in a BSD position is more complex than that in other positions) but adds to the carry-propagation delay in the worst case; hence, the hybrid scheme offers a trade-off between speed and cost.

Hybrid signed-digit representation with uniform spacing of BSD positions can be viewed as a special case of GSD systems. For the example of Fig. 3.9, arranging the numbers in 3-digit groups starting from the right end leads to a radix-8 GSD system with digit set $[-4, 7]$: that is, digit values from $(-1\ 0\ 0)_{\text{two}}$ to $(1\ 1\ 1)_{\text{two}}$. So the hybrid scheme of Fig. 3.9 can be viewed as an implementation of (digit encoding for) this particular radix-8 GSD representation.

The hybrid-redundant representation of Fig. 3.9, constituting an encoding for the radix-8 digit set $[-4, 7]$, is depicted in Fig. 3.10 using extended dot notation. The asymmetry of the digit set, and thus of the number representation range, is an unfortunate feature of such representations that allow only posibits in nonredundant positions. By removing the latter restriction, we can obtain more desirable symmetric hybrid-redundant representations, exemplified by the second encoding of Fig. 3.10, which constitutes an encoding for the radix-8 digit set $[-4, 4]$. Arithmetic on all such extended hybrid-redundant representations can be performed with equal ease [Jabe06].

3.5 CARRY-FREE ADDITION ALGORITHMS

The GSD carry-free addition algorithm, corresponding to the scheme of Fig. 3.2b, is as follows:

Carry-free addition algorithm for GSD numbers

Compute the position sums $p_i = x_i + y_i$.

Divide each p_i into a transfer t_{i+1} and an interim sum $w_i = p_i - rt_{i+1}$.

Add the incoming transfers to obtain the sum digits $s_i = w_i + t_i$.

Let us assume that the transfer digits t_i are from the digit set $[-\lambda, \mu]$. To ensure that the last step leads to no new transfer, the following condition must be satisfied:

$$\underset{\substack{|\\ \text{Smallest interim sum}\\ \text{if a transfer of } -\lambda\\ \text{is to be absorbable}}}{-\alpha+\lambda} \quad \le \quad \underset{\text{interim sum}}{p_i - rt_{i+1}} \quad \le \quad \underset{\substack{|\\ \text{Largest interim sum}\\ \text{if a transfer of } \mu\\ \text{is to be absorbable}}}{\beta-\mu}$$

From the preceding inequalities, we can easily derive the conditions $\lambda \ge \alpha/(r-1)$ and $\mu \ge \beta/(r-1)$. Once λ and μ are known, we choose the transfer digit value by comparing the position sum p_i against $\lambda + \mu + 2$ constants $C_j, -\lambda \le j \le \mu + 1$, with the transfer digit taken to be j if and only if $C_j \le p_i < C_{j+1}$. Formulas giving possible values for these constants can be found in [Parh90]. Here, we describe a simple intuitive method for deriving these constants.

■ **EXAMPLE 3.5** For $r = 10$ and digit set $[-5, 9]$, we need $\lambda \ge 5/9$ and $\mu \ge 1$. Given minimal values for λ and μ that minimize the hardware complexity, we find by choosing the minimal values for λ and μ

$$\lambda_{\min} = \mu_{\min} = 1 \qquad \text{(i.e., transfer digits are in } [-1, 1])$$

$$-\infty = C_{-1} \qquad -4 \le C_0 \le -1 \qquad 6 \le C_1 \le 9 \qquad C_2 = +\infty$$

We next show how the allowable values for the comparison constant C_1, shown above, are derived. The position sum p_i is in $[-10, 18]$. We can set t_{i+1} to 1 for p_i values as low as 6; for $p_i = 6$, the resulting interim sum of -4 can absorb any incoming transfer in $[-1, 1]$ without falling outside $[-5, 9]$. On the other hand, we must transfer 1 for p_i values of 9 or more. Thus, for $p_i \ge C_1$, where $6 \le C_1 \le 9$, we choose an outgoing transfer of 1. Similarly, for $p_i < C_0$, we choose an outgoing transfer of -1, where $-4 \le C_0 \le -1$. In all other cases, the outgoing transfer is 0.

Assuming that the position sum p_i is represented as a 6-bit, 2's-complement number *abcdef*, good choices for the comparison constants in the above ranges are $C_0 = -4$ and $C_1 = 8$. The logic expressions for the signals g_1 and g_{-1} then become

$$g_{-1} = a(\bar{c} \vee \bar{d}) \qquad \text{Generate a transfer of } -1$$

$$g_1 \;\; = \bar{a}(b \vee c) \qquad \text{Generate a transfer of } 1$$

An example addition is shown in Fig. 3.11.

It is proven in [Parh90] that the preceding carry-free addition algorithm is applicable to a redundant representation if and only if one of the following sets of conditions is satisfied:

a. $r > 2, \rho \ge 3$
b. $r > 2, \rho = 2, \alpha \ne 1, \beta \ne 1$

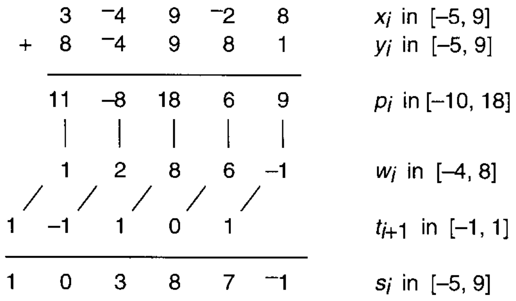
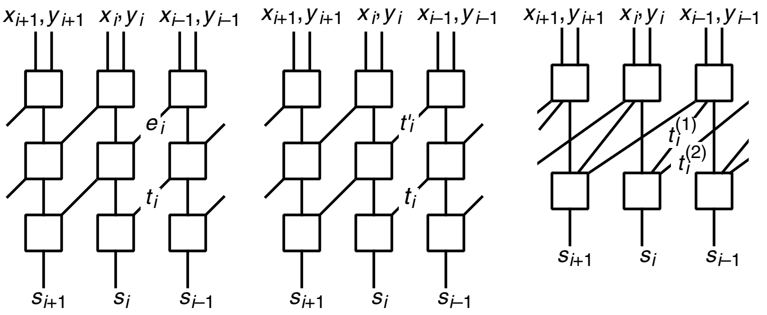


Figure 3.11 Adding radix-10 numbers with the digit set $[-5, 9]$.



(a) Three-stage carry estimate. **(b)** Three-stage repeated carry. **(c)** Two-stage parallel carries.

Figure 3.12 Some implementations for limited-carry addition.

In other words, the carry-free algorithm is not applicable for $r = 2, \rho = 1$, or $\rho = 2$ with $\alpha = 1$ or $\beta = 1$. In such cases, a limited-carry addition algorithm is available:

Limited-carry addition algorithm for GSD numbers

Compute the position sums $p_i = x_i + y_i$.

Compare each p_i to a constant to determine whether e_{i+1} = "low" or "high" (e_{i+1} is a binary range estimate for t_{i+1}).

Given e_i, divide each p_i into a transfer t_{i+1} and an interim sum $w_i = p_i - rt_{i+1}$.

Add the incoming transfers to obtain the sum digits $s_i = w_i + t_i$.

This "limited-carry" GSD addition algorithm is depicted in Fig. 3.12a; in an alternative implementation (Fig. 3.12b), the "transfer estimate" stage is replaced by another transfer generation/addition phase.

Even though Figs. 3.12a and 3.12b appear similar, they are quite different in terms of the internal designs of the square boxes in the top and middle rows. In both cases, however, the sum digit s_i depends on $x_i, y_i, x_{i-1}, y_{i-1}, x_{i-2}$, and y_{i-2}. Rather than wait for the limited transfer propagation from stage $i-2$ to i, one can try to provide the necessary

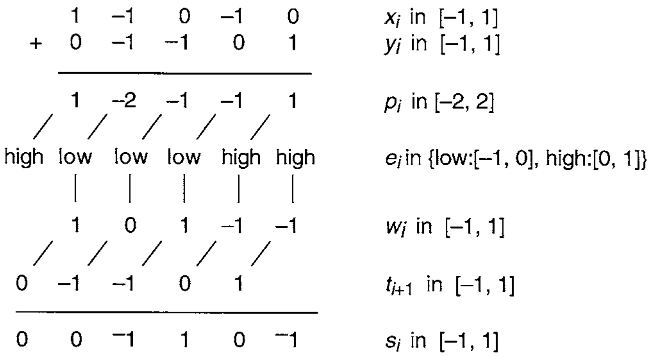


Figure 3.13 Limited-carry addition of radix-2 numbers with the digit set $[-1, 1]$ by means of carry estimates. A position sum of -1 is kept intact when the incoming transfer is in $[0, 1]$, whereas it is rewritten as 1 with a carry of -1 if the incoming transfer is in $[-1, 0]$. This scheme guarantees that $t_i \neq w_i$ and thus $-1 \leq s_i \leq 1$.

information directly from stage $i - 2$ to stage i. This leads to an implementation with parallel carries $t^{(1)}_{i+1}$ and $t^{(2)}_{i+2}$ from stage i, which is sometimes applicable (Fig. 3.12c).

■ **EXAMPLE 3.6** Figure 3.13 depicts the use of carry estimates in limited-carry addition of radix-2 numbers with the digit set $[-1, 1]$. Here we have $\rho = 1, \lambda_{\min} = 1$, and $\mu_{\min} = 1$. The "low" and "high" subranges for transfer digits are $[-1, 0]$ and $[0, 1]$, respectively, with a transfer t_{i+1} in "high" indicated if $p_i \geq 0$.

■ **EXAMPLE 3.7** Figure 3.14 shows another example of limited-carry addition with $r = 2$, digit set $[0, 3]$, $\rho = 2, \lambda_{\min} = 0$, and $\mu_{\min} = 3$, using carry estimates. The "low" and "high" subranges for transfer digits are $[0, 2]$ and $[1, 3]$, respectively, with a transfer t_{i+1} in "high" indicated if $p_i \geq 4$.

■ **EXAMPLE 3.8** Figure 3.15 shows the same addition as in Example 3.7 ($r = 2$, digit set $[0, 3]$, $\rho = 2, \lambda_{\min} = 0, \mu_{\min} = 3$) using the repeated-carry scheme of Fig. 3.12b.

■ **EXAMPLE 3.9** Figure 3.16 shows the same addition as in Example 3.7 ($r = 2$, digit set $[0, 3]$, $\rho = 2, \lambda_{\min} = 0, \mu_{\min} = 3$) using the parallel-carries scheme of Fig. 3.12c.

Subtraction of GSD numbers is very similar to addition. With a symmetric digit set, one can simply invert the signs of all digits in the subtractor y to obtain a representation of

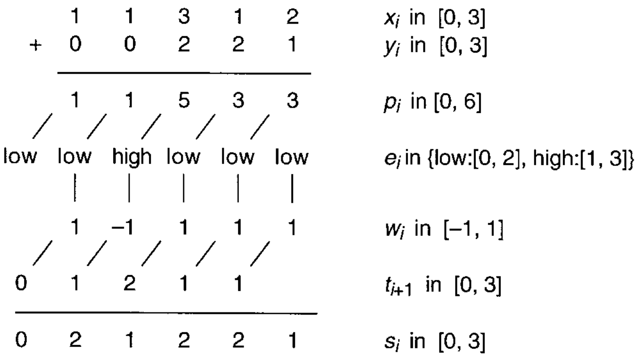


Figure 3.14 Limited-carry addition of radix-2 numbers with the digit set [0, 3] by means of carry estimates. A position sum of 1 is kept intact when the incoming transfer is in [0, 2], whereas it is rewritten as −1 with a carry of 1 if the incoming transfer is in [1, 3].

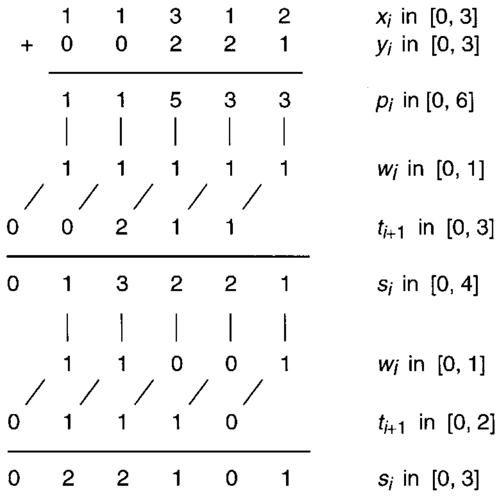


Figure 3.15 Limited-carry addition of radix-2 numbers with the digit set [0, 3] by means of the repeated-carry scheme.

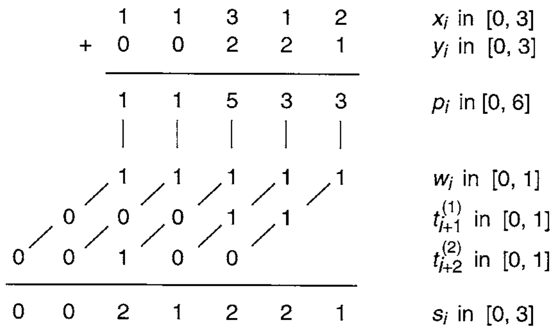


Figure 3.16 Limited-carry addition of radix-2 numbers with the digit set [0, 3] by means of the parallel-carries scheme.

$-y$ and then perform the addition $x+(-y)$ using a carry-free or limited-carry algorithm as already discussed. Negation of a GSD number with an asymmetric digit set is somewhat more complicated, but can still be performed by means of a carry-free algorithm [Parh93]. This algorithm basically converts a radix-r number from the digit set $[-\beta, \alpha]$, which results from changing the signs of the individual digits of y, to the original digit set $[-\alpha, \beta]$. Alternatively, a direct subtraction algorithm can be applied by first computing position differences in $[-\alpha - \beta, \alpha + \beta]$, then forming interim differences and transfer digits. Details are omitted here.

3.6 CONVERSIONS AND SUPPORT FUNCTIONS

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 or decimal and GSD representations are required.

■ **EXAMPLE 3.10** Consider number conversions from or to standard binary to or from BSD representation. To convert from signed binary to BSD, we simply attach the common number sign to each digit, if the (s, v) code of Fig. 3.7 is to be used for the BSD digits. Otherwise, we need a simple digitwise converter from the (s, v) code to the desired code. To convert from BSD to signed binary, we separate the positive and negative digits into a positive and a negative binary number, respectively. A subtraction then yields the desired result. Here is an example:

1	−1	0	−1	0	BSD representation of +6
1	0	0	0	0	Positive part (1 digits)
0	1	0	1	0	Negative part (−1 digits)
0	0	1	1	0	Difference = conversion result

The positive and negative parts required above are particularly easy to obtain if the BSD number is represented using the (n, p) code of Fig. 3.7. The reader should be able to modify the process above for dealing with numbers, or deriving results, in 2's-complement format.

The conversion from redundant to nonredundant representation essentially involves carry propagation and is thus rather slow. It is expected, however, that we will not need conversions very often. Conversion is done at the input and output. Thus, if long sequences of computation are performed between input and output, the conversion overhead can become negligible.

Storage overhead (the larger number of bits that may be needed to represent a GSD digit compared to a standard digit in the same radix) used to be a major disadvantage of redundant representations. However, with advances in VLSI (very large-scale integration) technology, this is no longer a major drawback; though the increase in the number of pins for input and output may still be a factor.

In the rest of this section, we review some properties of GSD representations that are important for the implementation of arithmetic support functions: zero detection, sign test, and overflow handling [Parh93].

In a GSD number system, the integer 0 may have multiple representations. For example, the three-digit numbers 0 0 0 and $^{-}1$ 4 0 both represent 0 in radix 4. However, in the special case of $\alpha < r$ and $\beta < r$, zero is uniquely represented by the all-0s vector. So despite redundancy and multiple representations, comparison of numbers for equality can be simple in this common special case, since it involves subtraction and detecting the all-0s pattern.

Sign test, and thus any relational comparison ($<, \leq$, etc.), is more difficult. The sign of a GSD number in general depends on all its digits. Thus sign test is slow if done through signal propagation (ripple design) or expensive if done by a fast lookahead circuit (contrast this with the trivial sign test for signed-magnitude and 2's-complement representations). In the special case of $\alpha < r$ and $\beta < r$, the sign of a number is identical to the sign of its most significant nonzero digit. Even in this special case, determination of sign requires scanning of all digits, a process that can be as slow as worst-case carry propagation.

Overflow handling is also more difficult in GSD arithmetic. Consider the addition of two k-digit numbers. Such an addition produces a transfer-out digit t_k. Since t_k is produced using the worst-case assumption about the as yet unknown t_{k-1}, we can get an overflow indication ($t_k \neq 0$) even when the result can be represented with k digits. It is possible to perform a test to see whether the overflow is real and, if it is not, to obtain a k-digit representation for the true result. However, this test and conversion are fairly slow.

The difficulties with sign test and overflow detection can nullify some or all of the speed advantages of GSD number representations. This is why applications of GSD are presently limited to special-purpose systems or to internal number representations, which are subsequently converted to standard representation.

PROBLEMS

3.1 Stored-carry and stored-borrow representations

The radix-2 number systems using the digit sets [0, 2] and [−1, 1] are known as binary stored-carry and stored-borrow representations, respectively. The general radix-r stored-carry and stored-borrow representations are based on the digit sets [0, r] and [−1, $r-1$], respectively.

a. Show that carry-free addition is impossible for stored-carry/borrow numbers. Do not just refer to the results in [Parh90]; rather, provide your own proof.

b. Supply the details of limited-carry addition for radix-r stored-carry numbers.

c. Supply the details of limited-carry addition for radix-r stored-borrow numbers.

d. Compare the algorithms of parts b and c and discuss.

3.2 Stored-double-carry and stored-triple-carry representations

The radix-4 number system using the digit set [0, 4] is a stored-carry representation. Use the digit sets [0, 5] and [0, 6] to form the radix-4 stored-double-carry and stored-triple-carry number systems, respectively.

a. Find the relevant parameters for carry-free addition in the two systems (i.e., the range of transfer digits and the comparison constants). Where there is a choice, select the best value and justify your choice.

b. State the advantages (if any) of one system over the other.

3.3 Stored-carry-or-borrow representations

The general radix-r stored-carry-or-borrow representations use the digit set $[-1, r]$.

a. Show that carry-free addition is impossible for stored-carry-or-borrow numbers.

b. Develop a limited-carry addition algorithm for such radix-r numbers.

c. Compare the stored-carry-or-borrow representation to the stored-double-carry representation based on the digit set $[0, r + 1]$ and discuss.

3.4 Addition with parallel carries

a. The redundant radix-2 representation with the digit set [0, 3], used in several examples in Section 3.5, is known as the binary stored-double-carry number system [Parh96]. Design a digit slice of a binary stored-double-carry adder based on the addition scheme of Fig. 3.16.

b. Repeat part a with the addition scheme of Fig. 3.14.

c. Repeat part a with the addition scheme of Fig. 3.15.

d. Compare the implementations of parts a–c with respect to speed and cost.

3.5 Addition with parallel or repeated carries

a. Develop addition algorithms similar to those discussed in Section 3.5 for binary stored-triple-carry number system using the digit set [0, 4].

b. Repeat part a for the binary stored-carry-or-borrow number system based on the digit set $[-1, 2]$.

c. Develop a sign detection scheme for binary stored-carry-or-borrow numbers.

d. Can one use digit sets other than [0, 3], [0, 4], and $[-1, 2]$ in radix-2 addition with parallel carries?

e. Repeat parts a–d for addition with repeated carries.

3.6 Nonredundant and redundant digit sets

Consider a fixed-point, symmetric radix-3 number system, with k whole and l fractional digits, using the digit set $[-1, 1]$.

a. Determine the range of numbers represented as a function of k and l.

b. What is the representation efficiency relative to binary representation, given that each radix-3 digit needs a 2-bit code?

c. Devise a carry-free procedure for converting a symmetric radix-3 positive number to an unsigned radix-3 number with the redundant digit set [0, 3], or show that such a procedure is impossible.

d. What is the representation efficiency of the redundant number system of part c?

3.7 Digit-set and radix conversions

Consider a fixed-point, radix-4 number system, with k whole and l fractional digits, using the digit set $[-3, 3]$.

a. Determine the range of numbers represented as a function of k and l.
b. Devise a procedure for converting such a radix-4 number to a radix-8 number that uses the digit set $[-7, 7]$.
c. Specify the numbers K and L of integer and fractional digits in the new radix of part b as functions of k and l.
d. Devise a procedure for converting such a radix-4 number to a radix-4 number that uses the digit set $[-2, 2]$.

3.8 Hybrid signed-digit representation

Consider a hybrid radix-2 number representation system with the repeating pattern of two standard binary positions followed by one BSD position. The addition algorithm for this system is similar to that in Fig. 3.9. Show that this algorithm can be formulated as carry-free radix-8 GSD addition and derive its relevant parameters (range of transfer digits and comparison constants for transfer digit selection).

3.9 GSD representation of zero

a. Obtain necessary and sufficient conditions for zero to have a unique representation in a GSD number system.
b. Devise a 0 detection algorithm for cases in which 0 has multiple representations.
c. Design a hardware circuit for detecting 0 in an 8-digit radix-4 GSD representation using the digit set $[-2, 4]$.

3.10 Imaginary-radix GSD representation

Show that the imaginary-radix number system with $r = 2j$, where $j = \sqrt{-1}$, and digit set $[-2, 2]$ lends itself to a limited-carry addition process. Define the process and derive its relevant parameters.

3.11 Negative-radix GSD representation

Do you see any advantage to extending the definition of GSD representations to include the possibility of a negative radix r? Explain.

3.12 Mixed redundant-conventional arithmetic

We have seen that BSD numbers cannot be added in a carry-free manner but that a limited-carry process can be applied to them.

a. Show that one can add a conventional binary number to a BSD number to obtain their BSD sum in a carry-free manner.
b. Supply the complete logic design for the carry-free adder of part a.
c. Compare your design to a carry-save adder and discuss.

3.13 Negation of GSD numbers

One disadvantage of GSD representations with asymmetric digit sets is that negation (change of sign) becomes nontrivial. Show that negation of GSD numbers is always a carry-free process and derive a suitable algorithm for this purpose.

3.14 Digit-serial GSD arithmetic

Generalized signed-digit representations allow fast carry-free or limited-carry parallel addition. Generalized signed-digit representations may seem less desirable for digit-serial addition because the simpler binary representation already allows very efficient bit-serial addition. Consider a radix-4 GSD representation using the digit set $[-3, 3]$.

a. Show that two such GSD numbers can be added digit-serially beginning at the most significant end (most-significant-digit-first arithmetic).

b. Present a complete logic design for your digit-serial adder and determine its latency.

c. Do you see any advantage for most-significant-digit-first, as opposed to least-significant-digit-first, arithmetic?

3.15 BSD arithmetic

Consider BSD numbers with digit set $[-1, 1]$ and the 2-bit (n, p) encoding of the digits (see Fig. 3.7). The code (1, 1) never appears and can be used as don't-care.

a. Design a fast sign detector for a 4-digit BSD input operand using full lookahead.

b. How can the design of part a be used for 16-digit inputs?

c. Design a single-digit BSD full adder producing the sum digit s_i and transfer t_{i+1}.

3.16 Unsigned-digit redundant representations

Consider the hex-digit decimal number system with $r = 10$ and digit set $[0, 15]$ for representing unsigned integers.

a. Find the relevant parameters for carry-free addition in this system.

b. Design a hex-digit decimal adder using 4-bit binary adders and a simple postcorrection circuit.

3.17 Double-least-significant-bits 2's-complement numbers

Consider k-bit 2's-complement numbers with an extra least-significant bit attached to them [Parh08]. Show that such redundant numbers have symmetric range, allow for bitwise 2's-complementation, and can be added using a standard k-bit adder.

3.18 Choice of digit set in redundant representations

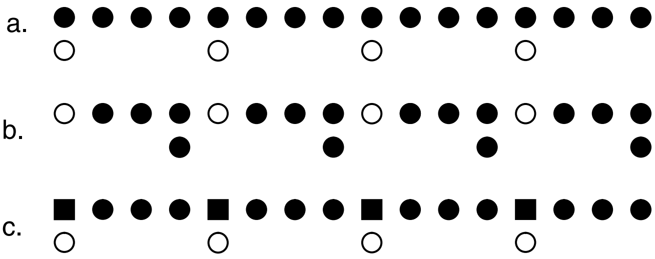
Prove or disprove the following assertions about the range $[-\lambda, \mu]$ of the transfer digits in a radix-r redundant number system.

a. The transfer digit set $[-\lambda, \mu]$ is a function of $\sigma = \alpha + \beta$ only (i.e., it is unaffected if α is changed to $\alpha - \delta$ and β to $\beta + \delta$).

b. The transfer digit set $[-\lambda, \mu]$ is minimized for a given even value of $\sigma = \alpha + \beta$ if $\alpha = \beta$.

3.19 Hybrid-redundant representations

For each of the hybrid-redundant representations, shown in the following diagram using extended dot notation, specify the digit set in the corresponding radix-16 GSD interpretation and devise an appropriate addition algorithm.



3.20 Digit-set conversions

Convert the radix-10 number $2\ {}^{-}8\ 9\ 6\ {}^{-}7\ 8$, using the digit set $[-9, 9]$ into each of the following digit sets.

a. $[0, 9]$
b. $[0, 12]$
c. $[-4, 5]$
d. $[-7, 7]$

3.21 Over-redundant digit sets

The digit set $[-\alpha, \beta]$ in radix r is over-redundant if $\alpha \geq r$ and $\beta \geq r$. Even though this level of redundancy may appear excessive, such representations do find some applications. For example, the over-redundant digit set $[-2, 2]$ in radix 2 is useful for fast division. Convert the over-redundant radix-2 number $1\ {}^{-}1\ 2\ 0\ {}^{-}2\ 2$ into each of the following digit sets.

a. $[0, 1]$
b. $[0, 2]$
c. $[-1, 1]$
d. $[-2, 1]$

3.22 Carry-free addition algorithm

Find the relevant parameters for carry-free addition (similar to Example 3.5) in the case of radix-4 addition with the following digit sets.

a. $[0, 5]$
b. $[-2, 3]$

3.23 Limited-carry addition algorithm

Find the relevant parameters for limited-carry addition (similar to Examples 3.6 and 3.7) in the case of radix-4 addition with the following digit sets.

a. [0, 4]
b. [−2, 2]

3.24 Shifting of stored-carry numbers

An unsigned binary number can be divided by 2 via a single-bit right shift. The same property holds for 1's- and 2's-complement numbers, provided that the sign bit is shifted into the vacated bit at the left end of the number (sign extension).

a. Show, by means of an example, that a stored-carry number cannot be divided by 2 via independent right shift of both the sum and carry bit-vectors [Tenc06].
b. Design a simple circuit that allows division by 2 via right shift, by supplying correct bit values to be shifted into the vacated positions at the left end.
c. Show that the modification of part b is not needed when the right-shifted stored-carry number is generated by adding an ordinary binary number to a standard stored-carry number.

3.25 Redundancy of a number system

Near the beginning of Section 3.4, we introduced the redundancy index $\rho = \alpha + \beta + 1 - r$ for a radix-r number system with the digit set $[-\alpha, \beta]$. We also mentioned that the ratio $h = \alpha/(r-1)$ has been used for quantifying redundancy in the special case of $\alpha = \beta$ in connection with high-radix division algorithms.

a. Derive an equation that relates the two redundancy indices ρ and h in the case of $\alpha = \beta$.
b. What would be a suitable formulation, applicable to an arbitrary digit set $[-\alpha, \beta]$ in radix r, if we were to define our redundancy index as a ratio, rather than a difference?

REFERENCES AND FURTHER READINGS

[Aviz61] Avizienis, A., "Signed-Digit Number Representation for Fast Parallel Arithmetic," *IRE Trans. Electronic Computers*, Vol. 10, pp. 389–400, 1961.

[Glas81] Glaser, A., *History of Binary and Other Nondecimal Numeration*, rev. ed., Tomash Publishers, 1981.

[Jabe05] Jaberipur, G., B. Parhami, and M. Ghodsi, "Weighted Two-Valued Digit-Set Encodings: Unifying Efficient Hardware Representation Schemes for Redundant Number Systems," *IEEE Trans. Circuits and Systems I*, Vol. 52, No. 7, pp. 1348–1357, 2005.

[Jabe06] Jaberipur, G., B. Parhami, and M. Ghodsi, "An Efficient Universal Addition Scheme for All Hybrid-Redundant Representations with Weighted Bit-Set Encoding," *J. VLSI Signal Processing*, Vol. 42, pp. 149–158, 2006.

[Korn94] Kornerup, P., "Digit-Set Conversions: Generalizations and Applications," *IEEE Trans. Computers*, Vol. 43, No. 8, pp. 622–629, 1994.

[Metz59] Metze, G., and J. E. Robertson, "Elimination of Carry Propagation in Digital Computers," *Information Processing '59* (Proceedings of a UNESCO Conference), 1960, pp. 389–396.

[Parh88] Parhami, B., "Carry-Free Addition of Recoded Binary Signed-Digit Numbers," *IEEE Trans. Computers*, Vol. 37, No. 11, pp. 1470–1476, 1988.

[Parh90] Parhami, B., "Generalized Signed-Digit Number Systems: A Unifying Framework for Redundant Number Representations," *IEEE Trans. Computers*, Vol. 39, No. 1, pp. 89–98, 1990.

[Parh93] Parhami, B., "On the Implementation of Arithmetic Support Functions for Generalized Signed-Digit Number Systems," *IEEE Trans. Computers*, Vol. 42, No. 3, pp. 379–384, 1993.

[Parh96] Parhami, B., "Comments on 'High-Speed Area-Efficient Multiplier Design Using Multiple-Valued Current Mode Circuits,'" *IEEE Trans. Computers*, Vol. 45, No. 5, pp. 637–638, 1996.

[Parh08] Parhami, B., "Double-Least-Significant-Bits 2's-Complement Number Representation Scheme with Bitwise Complementation and Symmetric Range," *IET Circuits, Devices & Systems*, Vol. 2, No. 2, pp. 179–186, 2008.

[Phat94] Phatak, D. S., and I. Koren, "Hybrid Signed-Digit Number Systems: A Unified Framework for Redundant Number Representations with Bounded Carry Propagation Chains," *IEEE Trans. Computers*, Vol. 43, No. 8, pp. 880–891, 1994.

[Phat01] Phatak, D. S., T. Goff, and I. Koren, "Constant-Time Addition and Simultaneous Format Conversion Based on Redundant Binary Representations," *IEEE Trans. Computers*, Vol. 50, No. 11, pp. 1267–1278, 2001.

[Tenc06] Tenca, A. F., S. Park, and L. A. Tawalbeh, "Carry-Save Representation Is Shift-Unsafe: The Problem and Its Solution," *IEEE Trans. Computers*, Vol. 55, No. 5, pp. 630–635, 2006.