On the Implementation of Arithmetic Support Functions for Generalized Signed-Digit Number Systems

Behrooz Parhami

Abstract—**Ordinary signed-digit (OSD) number representation systems have been defined for any radix $r \geq 3$ with digit values ranging over the set $\{-\alpha, \cdots, -1, 0, 1, \cdots, \alpha\}$, where α is an arbitrary integer in the range $r/2 < \alpha < r$. The most important property of OSD number representation systems is the possibility of performing carry-free addition and (by changing all the digit signs in the subtrahend) borrow-free subtraction. However, OSD number systems are not the only representations with this property. Previously, the author has defined the class of *generalized signed-digit* (GSD) number systems having the digit set $\{-\alpha, -\alpha+1, \cdots, \beta-1, \beta\}$ in radix r with $\alpha \geq 0$, $\beta \geq 0$, and $\alpha + \beta \geq r$. GSD number systems cover all useful redundant number representations as special cases. Most GSD number systems support carry-free addition and borrow-free subtraction and even those that do not, can be dealt with using limited-carry or limited-borrow algorithms which yield the ith sum or difference digit z_i as a function of the digits x_i, y_i, x_{i-1}, y_{i-1}, x_{i-2}, and y_{i-2} of the operands x and y. In this paper, we treat additional topics that are important for practical implementation of arithmetic functions using GSD number systems. Because GSD number systems may have asymmetric digit sets, we need to consider subtraction (or at least sign change for representations with $\alpha > 0$ and $\beta > 0$) explicitly. Zero detection, sign detection and overflow handling are also treated in depth.**

Index Terms—**Asymmetric signed-digit number systems, overflow, redundant number systems, sign change, sign detection, signed-digit arithmetic, zero detection.**

I. Introduction

Two basic approaches for reducing the delays caused by signal propagation in arithmetic operations are *speeding up* or *eliminating* the carry and borrow chains. The first approach is exemplified by carry-skip [7] and carry-lookahead [8] designs which reduce the ripple-carry delay of $O(n)$ to a delay of $O(\sqrt{n})$ and $O(\log n)$, respectively, for n-digit operands. Even though in the theory of algorithms, $O(\log n)$ time complexity is definitely superior to $O(\sqrt{n})$, the length of operands in conventional digital computers is short enough to make the carry-skip method competitive, particularly in view of its greater suitability for VLSI implementation [5]. The second approach is exemplified by stored-carry [9] and signed-digit [1] number representation methods which limit the propagation of carries at the expense of some overhead in storage and data-path width and in processing time for the initial conversion and the final reconversion. This paper concerns the latter approach (for further discussion of purpose and motivation, see [15]).

For any radix $r \geq 3$, there are one or more *signed-digit* (SD) number representation systems [1]–[3]. These *ordinary* SD (OSD) number systems correspond to different values of α in the range $r/2 < \alpha < r$, from the minimally redundant system ($\alpha = \lfloor r/2 \rfloor + 1$) to the maximally redundant one ($\alpha = r - 1$), where α determines the set $\{-\alpha, \cdots, -1, 0, 1, \cdots, \alpha\}$ of the $2\alpha + 1$ digit values used. The most important property of OSD number representation systems is the possibility of performing carry-free addition and (by changing all the digit signs in the subtrahend) borrow-free subtraction. The most serious drawbacks of OSD number systems are difficult sign detection (sign of a number is the sign of its most significant nonzero digit) and the overhead for conversion to/from conventional nonredundant representation. The propagation-free arithmetic property of OSD numbers has caused renewed interest in redundant number representations for systolic and/or VLSI implementation (see, e.g., [6]).

However, OSD number systems are not the only representations with this property. I have previously defined the class of *generalized signed-digit* (GSD) number systems having the digit set $\{-\alpha, -\alpha+1, \cdots, \beta-1, \beta\}$ in radix r with $\alpha \geq 0$, $\beta \geq 0$, and $\rho = \alpha + \beta + 1 - r$, where $\rho \geq 1$ is the *redundancy index* of the number system [11], [15]. Table I shows that GSD number systems cover all previously known useful redundant number representation systems and also lead to new ones. An example of such new representations is provided by the class of stored-carry-or-borrow (SCB) number systems with applications in the design of systolic counters [4], [10]. Any GSD number system with $r > 2$ and $\rho \geq 3$ (or with $\rho \geq 2$, provided that $\alpha \neq 1$ and $\beta \neq 1$) supports carry-free addition and even in the exceptional cases (i.e., for $r = 2$, $\rho = 1$, or $\rho = 2$ with $\alpha = 1$ or $\beta = 1$), a limited-carry addition algorithm is applicable that yields the ith sum digit s_i as a function of the digits x_i, y_i, x_{i-1}, y_{i-1}, x_{i-2}, and y_{i-2} of the operands x and y [15].

In this paper, we treat additional topics that are important for practical implementation of arithmetic functions using GSD number systems. Because GSD number systems may have asymmetric digit sets, we need to consider subtraction (or at least sign change for representations with $\alpha > 0$ and $\beta > 0$) explicitly. In Section II, we present results on borrow-free and limited-borrow subtraction of GSD numbers. Results pertaining to a negation (sign change) unit, that can be placed on the path of the addend in a GSD adder to allow the sharing of addition and subtraction hardware, are presented in Section III. Zero/sign detection and overflow handling are the topics of Sections IV and V, respectively. Conclusions and directions for further research appear in Section VI.

II. Subtraction of GSD Numbers

Proofs have not been provided for results in this section since they are quite similar to the proofs for carry-free and limited-carry addition presented in [15]. For the sake of completeness, results that are identical in form to those for addition are stated for both.

Algorithm 1 (propagation-free addition/subtraction): Let the augend/minuend and the addend/subtrahend have x_i and y_i as the ith digits. For each position i, a *position sum/difference* $p_i = x_i \pm y_i$ is computed which is then broken into a *transfer digit* t_{i+1} and an *interim sum/difference* $w_i = p_i \pm r t_{i+1}$). The *final sum/difference* digit is $z_i = w_i \pm t_i$ whose computation should produce no new transfer. □

Since transfer digits in GSD arithmetic are signed (unlike standard carries and borrows that are unsigned), we could have used "additive" transfer digits whose values are added into the next digit position for subtraction as well as for addition. However, we have chosen to use a "subtractive" transfer digit here since it leads to

$$-\lambda \leq t_j \leq \mu \tag{1}$$

as the common range for the transfer digits in addition and subtraction. The nonnegative integers $\lambda \leq \alpha$ and $\mu \leq \beta$ will be specified later. The transfer digit selection process consists of a number of

Manuscript received January 10, 1988; revised June 15, 1990 and April 15, 1992.

The author is with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, 93106.

IEEE Log Number 9202840.

TABLE I
MAJOR SUBCLASSES OF GSD NUMBER REPRESENTATION SYSTEMS AND THEIR PARAMETERS

Name of the GSD Subclass	Abbr.	r	α	β	ρ
Binary stored-carry	BSC	2	0	2	1
General stored-carry	SC	r	0	r	1
Binary stored-double-carry	BSDC	2	0	3	2
General stored-double-carry	SDC	r	0	$r+1$	2
Binary stored-borrow (signed-digit)	BSB, BSD	2	1	1	1
General stored-borrow	SB	r	1	$r-1$	1
Binary stored-carry-or-borrow	BSCB	2	1	2	2
General stored-carry-or-borrow	SCB	r	1	r	2
Minimally redundant symmetric SD		$r \geq 4$	$r/2$	$r/2$	1
Ordinary signed-digit	OSD	$r \geq 3$	$r/2 < \alpha < r$	α	$2 \leq \rho < r$
Minimally redundant OSD		$r \geq 3$	$\lfloor r/2 \rfloor + 1$	α	$2 \leq \rho \leq 3$
Maximally redundant OSD		$r \geq 3$	$r-1$	α	$r-1$
Unsigned-digit redundant	UDR	r	0	β	$\beta - r + 1$

(approximate) comparisons between p_i and known constants. Thus, Algorithm 1 is simplified if λ and μ are minimized.

Lemma 1: The set $\{-\lambda, -\lambda+1, \cdots, \mu-1, \mu\}$ of possible transfer digit values for propagation-free addition/subtraction of GSD numbers is of minimal size if λ and μ are chosen to be

$$\lambda^{\min} = \lceil \alpha/(r-1) \rceil \tag{2}$$

$$\mu^{\min} = \lceil \beta/(r-1) \rceil. \quad \square \tag{3}$$

Lemma 2: Assuming that $\lambda \geq \lambda^{\min}$ and $\mu \geq \mu^{\min}$, Algorithm 1 is applicable to a GSD number system iff

$$\rho \geq \lambda + \mu. \quad \square \tag{4}$$

Theorem 1: The propagation-free addition/subtraction defined by Algorithm 1 is applicable to a GSD number system iff $r > 2$ and either $\rho \geq 3$ or $\rho = 2$, $\alpha \neq 1$, $\beta \neq 1$. $\square$

Algorithm 2 (transfer digit selection for borrow-free subtraction): The transfer digit t_{i+1} is selected to be k iff $C_{k+1} < p_i \leq C_k$, where $C_{-\lambda} = \infty$, $C_{\mu+1} = -\infty$, and each $C_j (-\lambda < j \leq \mu)$ is a known comparison constant. $\square$

Clearly, the values of comparison constants in Algorithm 2 directly affect the complexity of the GSD subtractor. In general, there may be several valid choices for each C_j and thus one which results in the simplest possible hardware realization can be selected. The following theorem specifies the range of valid choices for C_j. The corresponding result for addition can be found in Theorem 2 of [15].

Theorem 2: The comparison constants C_k of Algorithm 2 must satisfy

$$-(k-1)r - (\alpha - \mu) - 1 \leq C_k \leq -kr + \beta - \lambda. \quad \square \tag{5}$$

Next, we consider a limited-propagation subtraction algorithm that can be used in the following cases where Algorithm 1 is inapplicable: 1) $r = 2$, 2) $\rho = 1$, or 3) $\rho = 2$, with $\alpha = 1$ or $\beta = 1$.

Algorithm 3 (limited-propagation addition/subtraction): Let the augned/minuend and the addend/subtrahend have x_i and y_i as the ith digits. In stage 1, for each position i, a *position sum/difference* $p_i = x_i \pm y_i$ is computed and used to generate a *range estimate* e_{i+1} for the final transfer digit t_{i+1}. In stage 2, the position sum/difference p_i and the range estimate e_i are used to compute a *transfer digit* t_{i+1} and an *interim sum/difference* $w_i = p_i \pm r t_{i+1}$. The *final sum/difference* digit is $z_i = w_i \pm t_i$ whose computation should produce no new transfer. $\square$

The new aspects of this algorithm vis-a-vis Algorithm 1 are: 1) Computation of the range estimates e_{i+1}, and 2) Computation of t_{i+1} as a function of both p_i and e_i. The range estimate e_i may be presented in many different formats. In our subsequent discussion, we consider the simplest case where a binary range estimate is used, since this can be shown to be sufficient for limited-propagation addition/subtraction in all cases [15]. Let λ' and μ' be constant (positive or negative) integers satisfying

$$-\lambda < -\lambda' \leq \mu' < \mu. \tag{6}$$

The binary range estimate $e_j \in \{l, h\}$ restricts the transfer digit t_j into one of two closed subintervals; the *low* subinterval $[-\lambda, \mu']$ or the *high* subinterval $[-\lambda', \mu]$.

Algorithm 4 (range estimate for limited-borrow subtraction): Select $e_{i+1} = h$ iff $p_i < E$ and $e_{i+1} = l$ iff $p_i \geq E$, where E is a known comparison constant. $\square$

Theorem 3: The comparison constant E of Algorithm 4 must satisfy

$$-(\mu'+1)r + \beta - \lambda' < E \leq (\lambda'+1)r - (\alpha - \mu'). \quad \square \tag{7}$$

Algorithm 5 (transfer digit selection for limited-borrow subtraction): The transfer digit t_{i+1} is selected to be k iff $C_{k+1}(e_i) < p_i \leq C_k(e_i)$, where $C_{-\lambda}(l) = C_{-\lambda}(h) = \infty$, $C_{\mu+1}(l) = C_{\mu+1}(h) = -\infty$, and each $C_j(e_i)$ $(-\lambda < j \leq \mu,\ e_i \in \{l, h\})$ is a known comparison constant. $\square$

Theorem 4: The comparison constants $C_k(l)$ and $C_k(h)$ of Algorithm 5 must satisfy

$$-(k-1)r - (\alpha - \mu') - 1 \leq C_k(l) \leq -kr + \beta - \lambda \tag{8a}$$

$$-(k-1)r - (\alpha - \mu) - 1 \leq C_k(h) \leq -kr + \beta - \lambda'. \quad \square \tag{8b}$$

Lemma 3: Propagation-free addition/subtraction is applicable to a GSD number system iff

$$\rho \geq \max(\lambda + \mu', \lambda' + \mu) \tag{9}$$

$$\lambda' + \mu' \geq \lfloor \rho/(r+1) \rfloor. \quad \square \tag{10}$$

Theorem 5: Limited-propagation addition/subtraction (Algorithm 3) is applicable to all GSD number systems. $\square$

III. Negation of GSD Numbers

Negation (sign change) may be viewed as a special case of subtraction. Thus, a subtractor built according to the results of Section II can be used to effect a borrow-free or limited-borrow sign change. However, because the minuend is fixed, negation can be accomplished by a much simpler circuit than that required for general subtraction. In fact we will see that the negation process is always borrow-free, even for classes of GSD numbers that do not support borrow-free subtraction. Clearly, negation is meaningful for a GSD number system only if $\beta > 0$ and $\alpha > 0$. Also, for $\alpha \neq \beta$, negation may lead to apparent or real overflow that must be handled by techniques discussed in Section V.

The algorithm for negation of a number y is identical to Algorithm 1, with q_i replaced by $-y_i$. For the sake of completeness, we restate this special case of Algorithm 1 as follows. We will see that borrow-free negation is always possible so that the analog of Algorithm 3 need never be used.

Algorithm 5 (borrow-free negation): Let the number y to be negated have y_i as the ith digit. The *position negate* $-y_i$ is broken into a *transfer digit* t_{i+1} and an *interim negate* $v_i = -y_i + rt_{i+1}$. The *final negate* digit is $n_i = v_i - t_i$ whose computation should produce no new transfer. □

Lemma 4: The set $\{\tau, \tau+1, \cdots, \tau'\}$ of possible transfer digit values for GSD negation is of minimal size if τ and τ' are chosen to be

$$\tau^{\max} = \lfloor(\beta - \alpha)/(r-1)\rfloor \tag{11}$$

$$\tau'^{\min} = \lceil(\beta - \alpha)/(r-1)\rceil. \tag{12}$$

Proof: Using the extreme values of the parameters in the ith stage, the following inequalities are easily derived:

$$y_i - \alpha + \tau' \leq rt_{i+1} \leq y_i + \beta + \tau. \tag{13}$$

To determine the maximal value for τ, we set $y_i = -\alpha$ in (13) to obtain

$$(-2\alpha + \tau')/r \leq \min_{y_i}(t_{i+1}) \leq (-\alpha + \beta + \tau)/r.$$

Thus, we select

$$\tau^{\max} = \lfloor(\beta - \alpha + \tau^{\max})/r\rfloor. \tag{14}$$

Similarly:

$$(\beta - \alpha + \tau')/r \leq \max_{y_i}(t_{i+1}) \leq (2\beta + \tau)/r$$

$$\tau'^{\min} = \lceil(\beta - \alpha + \tau'^{\min})/r\rceil. \tag{15}$$

The desired results are obtained by proceeding as in the proof of Lemma 2 in [15]. □

Lemma 5: Assuming $\tau \leq \tau^{\max}$ and $\tau' \geq \tau'^{\min}$, Algorithm 5 is applicable to a GSD number system iff $\rho \geq \tau' - \tau$.

Proof: Similar to the proof of Lemma 1 in [15], with $-y_i$, τ, and τ' replacing p_i, $-\lambda$, and μ, respectively. □

Note that τ and τ' can be of either sign, as will be seen later. This is the main reason for the change of notation from the $-\lambda$ and μ introduced in our discussion on addition/subtraction.

For a geometric verification of (14) and (15), one can refer to Fig. 1 where the two lines bounding the value of t_{i+1} according to (13) have been drawn. Note that in the special case of $\alpha = \beta$ (i.e., symmetric GSD representations), we have $\tau^{\max} = \tau'^{\min} = 0$. Thus, transfer digits are not needed for negating such GSD numbers; a fact that is intuitively obvious. Somewhat less obvious is the generalization stated in the following theorem.

Theorem 6: The need for generating transfer digits in GSD negation is obviated iff $\alpha = \beta \bmod (r-1)$. In all other cases, two-valued transfer digits are sufficient.

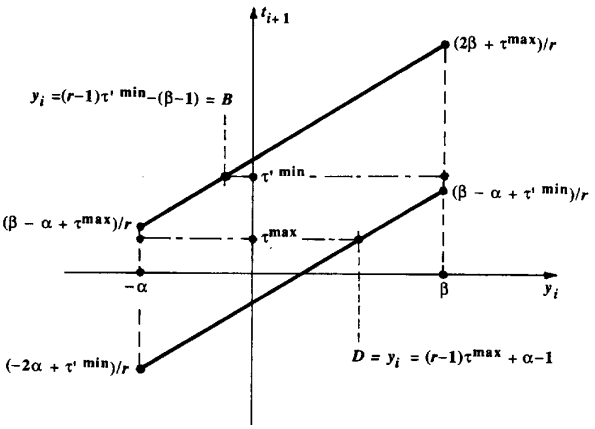

Fig. 1. The range of transfer digit values as a function of the operand digit for borrow-free negation.

Proof: Immediate upon noting that $\tau^{\max} = \tau'^{\min}$ implies the divisibility of $\beta - \alpha$ by $r-1$ and that in all other cases $\tau^{\max}$ and $\tau'^{\min}$ given by (11) and (12) differ by 1. □

Corollary 1: The borrow-free negation defined by Algorithm 5 is applicable to all GSD number systems.

Proof: Immediate from Lemma 5 upon noting that $\rho \geq 1$ and $\tau'^{\min} - \tau^{\max} \leq 1$ (by Theorem 6). □

For hardware implementation, a constant transfer digit value $\tau^{\max} = \tau'^{\min}$ means a simpler and faster negation circuit. Thus, SCB number representation systems having $\alpha = 1$ and $\beta = r$ (see Table I), and more generally GSD number systems with $\alpha = \beta \bmod (r-1)$, offer advantages in terms of the speed and simplicity of negation. When $\tau^{\max} \neq \tau'^{\min}$, we must specify how one of the two transfer digit values is selected for Algorithm 5.

Algorithm 6 (transfer digit selection for borrow-free negation): The transfer digit t_{i+1} is selected to be $\tau'^{\min}$ iff $y_i \geq C$ and to be $\tau^{\max}$ iff $y_i < C$, where C is a known comparison constant. □

Theorem 7: The comparison constant C of Algorithm 6 must satisfy

$$(r-1)\tau'^{\min} - (\beta - 1) \leq C \leq (r-1)\tau^{\max} + \alpha. \tag{16}$$

Proof: From Fig. 1, we find the values of y_i at the intersections of the lines $t_{i+1} = \tau'^{\min}$ and $t_{i+1} = \tau^{\max}$ with the upper and lower oblique lines as

$$B = (r-1)\tau'^{\min} - (\beta - 1)$$

$$D = (r-1)\tau^{\max} + \alpha - 1.$$

So, the boundary between selecting $t_{i+1} = \tau^{\max}$ and $t_{i+1} = \tau'^{\min}$ (i.e., the value of C) must lie between B and $D+1$. The selection of a suitable value for C is possible if $D + 1 \geq B$. To see when $D + 1 \geq B$ holds, we compute

$$(D+1) - B = (r-1)(\tau^{\max} - \tau'^{\min}) + \alpha + \beta - 1$$

$$= (r-1)(\tau^{\max} - \tau'^{\min} + 1) + \rho - 1.$$

Since $\tau^{\max} - \tau'^{\min} + 1 \geq 0$ and $\rho \geq 1$, the above difference is always nonnegative. □

Example 1: For the BSCB system (see Table I), we have $t_{i+1} = \tau^{\max} = \tau'^{\min} = 1$. Table II shows the negation process. The adjustment to n_7* is necessary because $t_8 \neq 0$. Hardware for this adjustment can be incorporated into the logic for the last negator stage. Similarly, the first negator stage is different since $t_0 = 0$ (the circled entry). □

TABLE II
EXAMPLE OF BORROW-FREE NEGATION FOR THE BSCB NUMBER SYSTEM

i	8	7	6	5	4	3	2	1	0
y_i		0	1	−1	1	0	2	−1	1
$-y_i$		0	−1	1	−1	0	−2	1	−1
v_i		2	1	3	1	2	0	3	1
t_i	1	1	1	1	1	1	1	1	(0)
n_i^*		1	0	2	0	1	−1	2	1
Adjustment									
n_i		−1	0	2	0	1	−1	2	1

IV. ZERO AND SIGN DETECTION

Whenever zero has the unique all-zeros representation, zero detection for GSD numbers is similar to that of conventional nonredundant positional systems. Thus, we first prove the following uniqueness theorem.

Theorem 8: Zero has the unique k-digit all-zeros representation in a GSD number system iff at least one of the following four conditions holds: 1) $k = 1$, 2) $\alpha = 0$, 3) $\beta = 0$, 4) $\max(\alpha, \beta) < r$.

Proof: For $k = 1$, $\alpha = 0$, or $\beta = 0$, uniqueness is obvious. Suppose that $k > 1$, $\alpha > 0$, $\beta > 0$, and $\max(\alpha, \beta) < r$. Then, the value of the number $z_{k-1}z_{k-2}\cdots z_1 z_0$ can be written as

$$\text{value}(z_{k-1}z_{k-2}\cdots z_1 z_0) = z_0 + r \times \text{value}(z_{k-1}z_{k-2}\cdots z_1).$$

A necessary condition for the above value to be 0 is $z_0 = 0 \bmod r$. Thus, since $\max(\alpha, \beta) < r$, we must have $z_0 = 0$. We can deduce by induction that $z_1 = \cdots = z_{k-2} = z_{k-1} = 0$. This concludes the sufficiency proof. To show the necessity of at least one of the four conditions, we prove that zero has multiple representations if $k > 1$, $\alpha > 0$, $\beta > 0$, and $\max(\alpha, \beta) \geq r$. If $\alpha \geq r$, then $0\ 0\ \cdots\ 0\ 1\ -r$ is an alternate representation of zero. Similarly, if $\beta \geq r$, then $0\ 0\ \cdots\ 0\ -1\ r$ has the same value as the all-zeros vector. □

Fortunately, even if the conditions of Theorem 8 do not hold, zero detection is not fundamentally more difficult. A logic signal ζ_i is propagated from right to left, indicating at each position i, whether the digits to the right represent a multiple of r (i.e., zero with an appropriate transfer digit). The procedure is formalized in the following algorithm.

Algorithm 7 (zero detection for GSD numbers): Let the number under test have the k-digit GSD representation $z_{k-1}z_{k-2}\cdots z_1 z_0$. Set $z_k = 0$, $\zeta_0 = \textbf{true}$, $t_0 = 0$ and compute sequentially for $i = 0, 1, \cdots, k$:

$u_i := z_i + t_i$;

$t_{i+1} := \textbf{if } u_i = 0 \bmod r \textbf{ then } u_i/r \textbf{ else}$ anything;

$\zeta_{i+1} := \textbf{if } u_i = 0 \bmod r \textbf{ then } \zeta_i \textbf{ else false}.$

The test result is ζ_{k+1}. □

Note that in the special cases covered by Theorem 8, we have $t_i = 0$, $u_i = z_i$, and $\zeta_{i+1} := \textbf{if } z_i = 0 \textbf{ then } \zeta_i \textbf{ else false}$. Thus, the test result is the logical AND of *position test* signals indicating whether $z_i = 0$. For large values of k, this must be implemented as a tree of AND gates due to fan-in limitations. In the general case, transfer-skip and transfer-lookahead techniques can be used in much the same way as conventional carry-skip and carry-lookahead to speed up the hardware realization of Algorithm 7.

Example 2: That the radix-10 SCB number (see Table I) 0 0 −1 9 9 9 9 10 represents zero is established by Algorithm 7 as shown in Table III. We have $t_k = 0$ with $\zeta_k = \textbf{true}$. Thus, the zero test result is affirmative. □

TABLE III
EXAMPLE OF ZERO DETECTION FOR A RADIX-10 SCB NUMBER SYSTEM

i	8	7	6	5	4	3	2	1	0
z_i		0	0	−1	9	9	9	9	10
u_i		0	0	0	10	10	10	10	10
t_i	0	0	0	1	1	1	1	1	0
ζ_i	true	true	true	true	true	true	true	true	true

Let the range of t_{i+1} in Algorithm 7 be $-\lambda_z \leq t_{i+1} \leq \mu_z$. To find minimal values for λ_z and μ_z, we proceed as in the proof of Lemma 1. We first note that

$$-(\alpha + \lambda_z) \leq u_i \leq \beta + \mu_z.$$

If $u_i \neq 0 \bmod r$, then the value of t_{i+1} is immaterial. So, we assume $u_i = rt_{i+1}$. Thus:

$$-(\alpha + \lambda_z)/r \leq t_{i+1} \leq (\beta + \mu_z)/r.$$

This yields

$$\lambda_z^{\min} = \left\lfloor (\alpha + \lambda_z^{\min})/r \right\rfloor \Rightarrow \lambda_z^{\min} = \lfloor \alpha/(r-1) \rfloor \quad (17a)$$

$$\mu_z^{\min} = \left\lfloor \left(\beta + \mu_z^{\min}\right)/r \right\rfloor \Rightarrow \mu_z^{\min} = \lfloor \beta/(r-1) \rfloor. \quad (17b)$$

Note that the values of $\lambda_z^{\min}$ and $\mu_z^{\min}$ given by (17) are less than or equal to the corresponding $\lambda^{\min}$ and $\mu^{\min}$ values given by Lemma 1 for the addition or subtraction process.

We now discuss the problem of sign detection for GSD numbers. Unlike OSD numbers, the sign of the most significant nonzero digit of a GSD number may not be identical to the sign of the number. For example, the BSCB number (see Table I) 0 0 −1 2 1 is positive. Obviously, the sign detection problem arises only if $\alpha, \beta > 0$.

Theorem 9: The sign of a GSD number is given by the sign of its most significant nonzero digit iff $\max(\alpha, \beta) < r$.

Proof: Suppose that the most significant nonzero digit of the GSD number $y_{k-1}y_{k-2}\cdots y_1 y_0$ is $y_j (0 \leq j \leq k-1)$ and that it is positive. Then, the minimum value that the number can have is

$$\begin{aligned} &\longleftarrow j \text{ digits} \longrightarrow \\ \text{value}(1 - \alpha \cdots - \alpha - \alpha) &= r^j - \alpha\left(r^{j-1} + \cdots + r + 1\right) \\ &= [\alpha + r^j(r - 1 - \alpha)]/(r-1). \end{aligned}$$

Clearly, the above value is positive for $\alpha \leq r - 1$. Similarly, if the most significant nonzero digit is negative and $\beta \leq r - 1$, the number is bound to be negative. This concludes the sufficiency proof. To prove the necessity, we show that with $\max(\alpha, \beta) \geq r$, we have positive (negative) numbers with leading negative (positive) digits. If $\alpha \geq r$, we have value$(1\ -r\ -1) = -1$. Similarly, for $\beta \geq r$, value$(-1\ r\ 1) = 1$. □

Thus, in special cases which satisfy the condition of Theorem 9, sign detection for GSD numbers is no more difficult than that of OSD numbers. As was the case for zero detection, a simple extension of the right-to-left scan will allow us to detect the sign of a GSD number in all cases.

Algorithm 8 (sign detection for GSD numbers): Let the number under test have the k-digit GSD representation $z_{k-1}z_{k-2}\cdots z_1 z_0$. Set $z_k = 0$, $\sigma_0 = \textbf{pos}$, $t_0 = 0$ and compute sequentially for $i = 0, 1, \cdots, k$:

$u_i := z_i + t_i$;

$t_{i+1} := \textbf{if } u_i \geq 0 \textbf{ then} \lfloor u_i/r \rfloor \textbf{else} \lceil u_i/r \rceil$;

$v_i := u_i - rt_{i+1}$;

$\sigma_{i+1} := \textbf{if } v_i = 0 \textbf{ then } \sigma_i \textbf{ else if } v_i > 0 \textbf{ then pos else neg}.$

The test result is σ_{k+1}. □

TABLE IV
EXAMPLE OF SIGN DETECTION FOR A RADIX-10 SCB NUMBER SYSTEM

i	8	7	6	5	4	3	2	1	0
z_i	0	0	0	−1	9	9	10	9	10
u_i	0	0	0	0	10	10	11	10	10
t_i	0	0	0	1	1	1	1	1	0
v_i	0	0	0	0	0	0	1	0	0
σ_i	pos	pos	pos	pos	pos	pos	pos	pos	pos

Note that in the special cases covered by Theorem 8, we have $t_i = 0$ $v_i = u_i = z_i$, and σ_{i+1} is determined by σ_i and the sign of z_i. Again, transfer-skip and transfer-lookahead techniques can be used in much the same way as conventional carry-skip and carry-lookahead to speed up the hardware realization of Algorithm 8.

Example 3: That the radix-10 SCB number (see Table I) 0 0 −1 9 9 10 9 10 represents a positive number is established by Algorithm 8 as shown in Table IV. We have $\sigma_k = \mathbf{pos}$. Thus, the number is positive. □

It is interesting to note that Algorithms 7 and 8 can be combined into a single algorithm whose hardware realization is only slightly more complex than either algorithm alone. Thus, the overall hardware requirement is reduced by using the following version of Algorithm 8 with three-valued $\sigma_i (\sigma_i \in \{-1, 0, 1\})$ for both zero and sign detection.

Algorithm 9 (combined zero and sign detection for GSD numbers): Let the number under test have the k-digit GSD representation $z_{k-1}z_{k-2}\cdots z_1 z_0$. Set $z_k = 0$, $\sigma_0 = 0$, $t_0 = 0$ and compute sequentially for $i = 0, 1, \cdots, k$:

$$u_i := z_i + t_i;$$

$$t_{i+1} := \textbf{if } u_i \geq 0 \textbf{ then} \lfloor u_i/r \rfloor \textbf{else} \lceil u_i/r \rceil;$$

$$v_i := u_i - r t_{i+1};$$

$$\sigma_{i+1} := \textbf{if } v_i = 0 \textbf{ then } \sigma_i \textbf{ else } \text{signum}(v_i).$$

The three-valued test result is σ_{k+1}, whose interpretation is the same as that of the signum function used above. □

The range of t_{i+1} in Algorithms 8 and 9 is the same as that obtained for Algorithm 7 in (17a) and (17b).

V. DETECTION OF OVERFLOW IN GSD ARITHMETIC

In dealing with fixed-length k-digit numbers, an *apparent overflow* occurs when the outgoing transfer digit t_k is nonzero. We call this an "apparent overflow" since even with $t_k \neq 0$, the result of the arithmetic operation may be representable as a k-digit GSD number. The apparent overflow simply signifies that a particular representation of the result has more than k digits. When the result has no k-digit representation, we say that a *real overflow* has occurred. Obviously, real overflow is much harder to detect than apparent overflow.

Fortunately, however, it is not necessary for the hardware to detect real overflow. In a special-purpose system (e.g., a systolic arithmetic engine), an intermediate result with apparent overflow can be tagged as invalid for the rest of the computation. While it is true that with this approach more results will be tagged than what is absolutely necessary, overflows are so infrequent that no serious problem is encountered. A partial remedy is to maintain a guard digit on the left which acts as the "invalid" tag when not zero. Frequently, it is possible to use idle cycles to reset a nonzero guard digit to 0 by changing the most significant digit of the number (e.g., changing 1 −6 to 0 4 in decimal), thus enabling the computation to continue. At any rate, the net effect of using apparent overflows instead of real overflows is that the range of numbers that we can deal with is occasionally restricted.

On the other hand, if overflow is dealt with by using exception-handling software, there is no problem at all since the value of t_k and the *apparent result* $x_{k-1}x_{k-2}\cdots x_1 x_0$ (which, as in the case of conventional overflow, differs from the correct result by a multiple of r^k) provide sufficient information for the overflow handler to deal with the situation; an attempt to obtain a valid k-digit representation of the result may be part of this routine's function.

The detection of real overflow and the correction procedure in the case of a nonreal overflow are discussed next. In the rest of this section, we will assume t_k to be an additive transfer digit, so that $t_k x_{k-1}x_{k-2}\cdots x_1 x_0$ is a correct $(k+1)$-digit representation of the result. Obviously, for a borrow-type transfer digit, the sign of t_k must be changed if the subsequent results are to be applicable.

Algorithm 10 (detection of real overflow): Given that an outgoing transfer digit $t_k \neq 0$ has been produced with the apparent result $x_{k-1}x_{k-2}\cdots x_1 x_0$, set $t'_0 = 0$ and compute sequentially for $i = 0, 1, \cdots, k-1$:

$$u_i := x_i + t'_i;$$

$$t'_{i+1} := \textbf{if } t_k > 0 \textbf{ then } -\lfloor (\beta - u_i)/r \rfloor \textbf{ else } \lfloor (u_i + \alpha)/r \rfloor;$$

$$x'_i := u_i - r t'_{i+1}.$$

Then compute $u_k := t_k + t'_k$. There is real overflow iff $t_k u_k > 0$. □

Note that the computation of x'_i is redundant in this algorithm. The reason for its inclusion is that is simplifies the description of Algorithm 11, as we will see shortly. An intuitive explanation of Algorithm 10 is that it obtains, in a right-to-left sequential scan, the largest (smallest) possible digits in the range $-\alpha \leq x'_i \leq \beta$ that can be used for representing the result producing $t_k > 0 (t_k < 0)$. If even with these digits, the outgoing transfer digit u_k is nonzero and of the same sign as t_k, then real overflow has occurred. In executing Algorithm 10, the sign of all transfer digits is opposite that of t_k. The magnitude of the transfer digit t'_j satisfies

$$|t'_j| \leq \lfloor (\alpha + \beta)/r \rfloor = 1 + \lfloor (\rho - 1)/r \rfloor. \tag{18}$$

Thus, for $\rho \leq r$, a binary transfer digit is sufficient. It is interesting to note that for nonredundant representations, we have $\alpha + \beta + 1 = r$. Thus, all transfer digits are zero and an apparent overflow is always a real overflow.

Example 4: Consider a GSD system with $r = 10$, $\alpha = 5$, and $\beta = 10$. The minuend 6 5 5 5 and the subtrahend −5 5 3 4 yield the apparent difference 1 0 2 1 with the outgoing transfer digit $t_4 = 1$. Algorithm 10 works as shown in Table V. Since $u_4 = t_4 + t'_4 = 0$, the overflow is not real. □

Algorithm 11 (correction of result with nonreal overflow): After executing Algorithm 10 and determining the values of u_k and x'_i for $i = 0, 1, \cdots, k-1$, set $t''_k = u_k$ and compute sequentially for $i = k-1, \cdots, 1, 0$:

$$v_i := x'_i + r t''_{i+1};$$

$$t''_i := \textbf{if } v_i > \beta \textbf{ then } v_i - \beta \textbf{ else if } v_i < -\alpha \textbf{ then } v_i + \alpha \textbf{ else } 0;$$

$$x''_i := v_i - t''_i;$$

The corrected result is $x''_{k-1}x''_{k-2}\cdots x''_1 x''_0$. Note that if $t''_i = 0$, then $t''_j = 0$ and $x''_j = x'_j$ for all $j < i$. Thus, execution of the algorithm can end as soon as t''_i becomes zero. □

To find the range of transfer digits in Algorithm 11, let $t_k > 0$. Then, we must have $u_k \leq 0$ for the overflow to be nonreal. Thus, all transfer digits t''_j in Algorithm 11 are nonpositive. Similarly if

TABLE V
EXAMPLE OF REAL OVERFLOW DETECTION IN A GSD NUMBER SYSTEM

i	4	3	2	1	0
x_i		1	0	2	1
u_i		0	0	2	1
t'_i	−1	−1	0	0	0
x'_i		10	10	2	1

$t_k < 0$, all transfer digits t''_j will be nonnegative. Since the sign of t''_j is determined by the sign of t_k, we need only represent its magnitude. In the following discussion, we take $t_k < 0$ and u_k, $t''_j \geq 0$. Let the range of t''_j be $[0, \theta]$. We have in Algorithm 11:

$$x''_i = v_i - t''_i = x'_i + rt''_{i+1} - t''_i. \tag{19}$$

Combining (19) with the restriction $x''_i \leq \beta$, we get

$$t''_{i+1} \leq (\beta - x'_i + t''_i)/r. \tag{20}$$

We have $x'_i \geq -\alpha$ and $t''_i \leq \theta$. Therefore, the right hand side of (20) is no more than $(\alpha+\beta+\theta)/r$. So, we must have $\theta \leq (\alpha+\beta+\theta)/r$, and this is equivalent to

$$\theta = \lfloor(\alpha+\beta)/r\rfloor = 1 + \lfloor(\rho-1)/r\rfloor. \tag{21}$$

So the range of t''_j is the same as the range of t'_j as specified by (18).

Example 5: Consider a GSD number system with $r = 10$, $\alpha = 5$, and $\beta = 10$. Since $\rho = 6$, (21) yields $\theta = 1$. Hence two-valued transfer digits $t''_j \in \{0, 1\}$ are sufficient. □

VI. CONCLUSION

We have completed a previously published study of redundant number representation systems which dealt only with carry-free and limited-carry addition [15] by considering subtraction, negation, zero detection, sign detection, and overflow detection algorithms. Using the results of this study and those of [15], it is possible to design special-purpose arithmetic "engines" for processing of large volumes of numerical data, provided that the nature of computations enables the conversion and reconversion overhead to be amortized over a long sequence of operations. Such conditions prevail in many signal processing algorithms and high-precision scientific computations.

It is natural to ask whether the generalization from OSD to GSD number representation is worthwhile. In other words, is it ever advantageous to use an asymmetric rather than a symmetric digit set for a redundant number system? Applications of the SC and SCB number systems lead us to an affirmative answer. One might try to justify this generalization by asking the equivalent question: Are asymmetric GSD number systems any more difficult to deal with than OSD representations? Clearly GSD addition and subtraction algorithms are no more complex than their OSD counterparts. The complexity does increase if ρ exceeds $r - 1$ (see Corollary 1 in [15]), but this is not related to the symmetry or asymmetry of the digit set used. Other advantages of this generalization have been discussed in [15].

Sign detection and overflow handling are difficult for all redundant number representations, independent of the digit set used. Theorem 8 indicates that zero detection can be equally simple for asymmetric digit sets, provided that certain conditions are satisfied. The most serious objection to the use of an asymmetric digit set is the increased complexity of negation (sign change), which is required to enable the sharing of addition and subtraction hardware. Theorem 6 tells us that the increased complexity is negligible if a GSD number system satisfies the condition $\alpha = \beta \bmod (r-1)$. Even if the above condition is not satisfied, negation is done totally in parallel with negligible speed penalty.

Finally, the results obtained here suggest that redundant representations with unsigned digits ($\alpha = 0$) may possess certain advantages if combined with a suitable representation of sign. Such unsigned-digit redundant (UDR) number systems eliminate the need for conversion from standard to redundant representation. The binary stored double-carry (BSDC) representation is a special UDR system ($r = 2$, $\beta = 3$) that finds application in the design of high-speed multipliers [14]. Although by Theorem 1, no redundant radix-2 representation supports carry-free addition, BSDC numbers can be added in a special scheme where two parallel carries are generated by stage i and are forwarded to stages $i + 1$ and $i + 2$ [14]. This obviates the need for range estimates required by Algorithm 3 and speeds up the "limited-carry" addition process at the expense of a somewhat more complex wiring.

REFERENCES

[1] A. Avizienis, "Signed-digit number representation for fast parallel arithmetic," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 389–400, 1961.

[2] ——, "On a flexible implementation of digital computer arithmetic," in *Inform. Processing '62 (Proc. IFIP Congress)*, North-Holland, Amsterdam, 1963, pp. 664–670.

[3] ——, "Binary-compatible signed-digit arithmetic," in *AFIPS Conf. Proc. (1964 Fall Joint Computer Conf.)*, pp. 663–672.

[4] L. J. Guibas and F. M. Liang, "Systolic stacks, queues, and counters," in *Proc. Conf. Advanced Res. VLSI*, M.I.T., 1982, pp. 155–164.

[5] A. Guyot, B. Hochet, and J.-M. Muller, "A way to build efficient carry-skip adders," *IEEE Trans. Comput.*, vol. C-36, no. 10, pp. 1144–1151, Oct. 1987.

[6] M. J. Irwin and R. M. Owens, "Digit-pipelined arithmetic as illustrated by the paste-up system," *IEEE Comput. Mag.*, vol. 20, no. 4, pp. 61–73, Apr. 1987.

[7] M. Lehman and N. Burla, "Skip techniques for high-speed carry propagation in binary arithmetic units," *IRE Trans. Electron. Comput.*, vol. EC-10, no. 4, pp. 691–698, Dec. 1961.

[8] O. L. MacSorley, "High-speed arithmetic in binary computers," *Proc. IRE*, vol. 49, pp. 67–91, Jan. 1961. Reprinted in [16].

[9] G. Metze and J. E. Robertson, "Elimination of carry propagation in digital computers," in *Inform. Processing '59 (Proc. UNESCO Conf.*, June 1959), 1960, pp. 389–396.

[10] B. Parhami, "Systolic up/down counters with zero and sign detection," in *Proc. Symp. Comput. Arithmetic*, Como, Italy, May 1987, pp. 174–178.

[11] ——, "A general theory of carry-free and limited-carry computer arithmetic," in *Proc. Canadian Conf. VLSI*, Winnipeg, Canada, Oct. 1987, pp. 167–172.

[12] ——, "Carry-free addition of recoded binary signed-digit numbers," *IEEE Trans. Comput.*, vol. 37, no. 11, pp. 1470–1476, Nov. 1988.

[13] ——, "Zero, sign, and overflow detection schemes for generalized signed-digit arithmetic," in *Proc. 22nd Asilomar Conf. Signals, Syst., and Comput.*, Pacific Grove, CA, Oct./Nov. 1988, pp. 636–639.

[14] ——, "A new method for designing highly parallel binary multipliers," in *Proc. 3rd Annu. Parallel Processing Symp.*, Fullerton, CA, Mar. 1989, pp. 176–185.

[15] ——, "Generalized signed-digit number systems: A unifying framework for redundant number representations," *IEEE Trans. Comput.*, vol. 39, no. 1, pp. 89–98, Jan. 1990.

[16] E. E. Swartzlander, Ed., *Computer Design Development, Principal Papers.* Hayden, 1976.