A Class of Stored-Transfer Representations for Redundant Number Systems

Ghassem Jaberipur
Shahid Beheshti Univ.
Tehran 19847, Iran, *and*
Sharif Univ. of Technology
jaberigh@mehr.sharif.ac.ir

Behrooz Parhami
Dept. Elec. & Computer Eng.
Univ. of California
Santa Barbara, CA 93106, USA
parhami@ece.ucsb.edu

Mohammad Ghodsi
Computer Engineering Dept.
Sharif Univ. of Technology
Tehran 11365, Iran
ghodsi@sharif.ac.ir

Abstract

Redundant representations play an important role in high-speed computer arithmetic. One key reason is that such representations support carry-free addition; i.e., addition in a small, constant time, independent of operand widths. We explore the implications of stored-transfer or transfer-save representation of digit sets for redundant number systems on the speed and cost of arithmetic algorithms and show that our methods lead to some of the fastest, most efficient implementations of carry-free arithmetic reported thus far. The speed/efficiency arises from storing or saving, instead of combining through addition, the transfer values generated during carry-free arithmetic.

Guide to Notation

$[\mu, \nu]$	Interval of values; denotes the empty set Φ if $\mu > \nu$
$', ''$	Marks a digit's main/stored-transfer component
a, b	See the definition of D
c_i	The ith transfer value in G, $0 \le i \le d-1$
D	$= [a, b]$, Digit set (main part)
d	Number of transfer values in G
G	$= \{c_0, c_1, \ldots, c_{d-1}\}$, Set of transfer values generated
h	$= \lceil \log_2 r \rceil$, Number of bits in one radix-r digit
k	Word width, in bits or digits
p	Position sum; digits are p_i
p_{min}	$= 2a + 2c_0$, Minimal value for p
p_{max}	$= 2b + 2c_{d-1}$, Maximal value for p
r	Number representation radix
s	Sum; digits are s_i
t	Transfer; digits are t_i
u, v	Sum and carry components of a carry-save value
w	Interim sum; digits are w_i
x, y	Addition operands; digits are x_i and y_i
α, β	See the definition of Δ
Δ	$= [\alpha, \beta]$, Digit set
Δ_i	Range of p where c_i is a valid transfer; $\Delta_{-1} = \Delta_d = \Phi$
δ	$= c_{d-1} - c_0 = \sum \delta_i$, Span of G
δ_i	$= c_{i+1} - c_i$, Difference of consecutive elements in G
λ_i	Overlap between Δ_i and Δ_{i+1} (cardinality of $\Delta_i \cap \Delta_{i+1}$)
ρ	$= \beta - \alpha + 1 - r$, Redundancy index of Δ
ρ'	$= b - a + 1 - r$, Redundancy index of D
σ', σ''	Functions generating main and stored transfer parts
τ, ω	Functions generating transfer and interim sum digits

1. Introduction

Redundancy in number representation aims to improve the speed or efficiency of arithmetic units [Metz59], [Aviz60] and is commonly used in modern digital systems. One reason for speed improvement with redundancy is the possibility of carry-free addition; i.e., addition in a small, constant time, independent of operand widths [Parh90]. Another reason is that redundancy allows some imprecision in the decision processes (such as quotient or root digit selection [Parh00], [Parh01]); this tolerance for imprecision removes enough complexity from the computation's critical path to yield significant performance improvement. Here, we focus on mechanisms that facilitate carry-free addition and allow its implementation with even greater speed.

In carry-free addition, one performs the following steps on all k digit positions of the two radix-r operands in parallel, where x_i and y_i belong to the redundant digit set $\Delta = [\alpha, \beta]$:

1. Compute the position sum digit $p_i = x_i + y_i$
2. Derive the interim sum digit w_i and transfer digit t_{i+1} satisfying $w_i = p_i - rt_{i+1}$
3. Form the final sum digit $s_i = w_i + t_i$

For step 3 to yield a valid digit in Δ without producing further transfers, w_i must be restricted in $D = [a, b]$, with the following holding for all possible values of t_i:

$$\alpha - t_i \le a < a + r - 1 \le b \le \beta - t_i$$

Note that the digit-size additions of steps 1 and 3, though quite fast compared to word-size additions required with nonredundant representations, are merely used for algorithm description and need not be explicitly performed in hardware. The addition in step 1 can be avoided, e.g., by noting that w_i and t_{i+1} are directly computable in hardware as functions of x_i and y_i. That is:

$$w_i = \omega(x_i, y_i) \qquad t_{i+1} = \tau(x_i, y_i)$$

This, in effect, fuses steps 1 and 2 and allows the designer to choose the best possible merged implementation. It may be the case, with certain digit sets and/or encodings, that some form of addition is still part of the best hardware implementation scheme for ω and τ, but this is not required. We are thus motivated to investigate methods for eliminating, or else simplifying, the addition in step 3.

0-7803-7147-X/01/$10.00©2001 IEEE

2. Stored-Transfer Representations

In a manner similar to the stored-carry or carry-save representation of binary numbers [Metz59], [Jabe99], we study the implications of stored-transfer or transfer-save representations of redundant digits where the pair (w_i, t_i) is viewed as an encoding of the sum digit s_i, thus obviating the need for the final addition. We call w_i the *main part* and t_i the *transfer part* of a digit's stored-transfer encoding.

Example 1: A main part that is a 4-bit 2's-complement number and a 4-valued stored transfer in [–1, 2] constitute a 6-bit encoding of the digit set [–9, 9]. Direct encoding of the digit set would require 5 bits. ■

The latter scheme leads to a two-step formulation of carry-free addition. In the following, we assume that any digit $z \in \Delta$ has a transfer-save encoding (z', z''), with $z' \in D$ and $z'' \in G = \{c_0, c_1, \ldots, c_{d-1}\}$; that is, primed and double-primed variables are used to designate the main and transfer parts of a digit.

1. Compute the position sum digit $p_i = x_i' + x_i'' + y_i' + y_i''$
2. Derive $s_i = (s_i', s_i'')$, satisfying $s_i' = p_i - r s_{i+1}''$

Note that the generated transfer set $G = \{c_0, c_1, \ldots, c_{d-1}\}$, satisfying $c_0 < c_1 < \ldots < c_{d-1}$, is d-valued but does not necessarily contain a set of d consecutive integers. We take this more general view in anticipation that it may provide added flexibility for optimizations. We will see later that even though such generalized transfer sets do not provide additional benefits directly, they can be used with minor modifications to the carry-free addition algorithm. On the other hand, the main part of a digit belongs to an interval $D = [a, b]$ of values. Whereas gaps in this set are also admissible, provided that the values in the set contain one member from each of the r residue equivalence classes j mod r $(0 \le j \le r - 1)$, we have not found this generality to lead to any speed or cost benefit.

Of course, steps 1 and 2 in this new two-step process can again be fused, in the manner previously outlined, leading to a merged, or single-step, implementation:

$$s_i' = \sigma'(x_i', x_i'', y_i', y_i'') \qquad s_{i+1}'' = \sigma''(x_i', x_i'', y_i', y_i'')$$

An objection may be raised that our scheme simply shifts the complexity of the original step 3 to the new step 1. That this is not the case will become clear as we describe our methods. Here, we just argue that the new scheme can, in principle, be faster than the original algorithm. For one thing, a 4-operand addition, where two of the operands (transfer parts) are fairly small, can indeed be faster and less complex than two separate additions [Koba85]. For another, the transfer-save representation (z', z'') may well contain the same total number of bits as the binary encoding of z [Jabe00]. In such a case, the function pairs (ω, τ) and (σ', σ'') have comparable bit-level complexities.

3. Some General Requirements

Equating the boundaries of the original digit set $\Delta = [\alpha, \beta]$ and its stored-transfer representation, i.e., $[\alpha, \beta] = [a, b] + \{c_0, c_1, \ldots, c_{d-1}\}$, leads to the requirements:

$$\alpha = a + c_0 \qquad \beta = b + c_{d-1}$$

For convenience, we define redundancy indices associated with the two digit sets $[\alpha, \beta]$ and $[a, b]$ as $\rho = \beta - \alpha + 1 - r$ and $\rho' = b - a + 1 - r$, respectively. We also designate $\delta = c_{d-1} - c_0$ as the span of the transfer set. It is easy to show that $\rho = \rho' + \delta$. If, for the sake of representational efficiency, we set $\rho' = 0$, it is the case that $\rho = \delta$. Furthermore, we define $\Delta_i = [a + rc_i, b + rc_i] \cap [p_{min}, p_{max}]$ as the range of p, where c_i is a valid (or *useful*, per Definition 1 below) transfer value, and $\lambda_i = b + rc_i - a - rc_{i+1} + 1 = \rho' + r - r\delta_i$ $(i < d - 1)$ as the overlap between Δ_i and Δ_{i+1}, where $\delta_i = c_{i+1} - c_i$.

Example 2: Stored-transfer representations of some redundant number systems are characterized in the following table. In all cases, D is irredundant ($\rho' = 0$, $\rho = \delta$) and is taken to be the unsigned set $[0, r - 1]$, except for the last entry where D is $[-r/2, r/2 - 1]$ with r even. For the two hybrid signed-digit entries, $r = 2^h$. ■

Number system	Δ	$\rho = \delta$	G
Stored-carry	$[0, r]$	1	$\{0, 1\}$
Stored-borrow	$[-1, r-1]$	1	$\{-1, 0\}$
Stored-carry-or-borrow	$[-1, r]$	2	$\{-1, 0, 1\}$
	$[-1, r]$	2	$\{-1, 1\}$
Stored-double-carry	$[0, r+1]$	2	$\{0, 1, 2\}$
Hybrid S-D (h–1 B, 1 BSD)	$[-1, 2^h-1]$	1	$\{-1, 0\}$
Hybrid S-D (1 BSD, h–1 B)	$[-2^{h-1}, 2^h-1]$	2^{h-1}	$\{-2^{h-1}, 0\}$
Minimally redundant asymmetric	$[-r/2-1, r/2]$	2	$\{-1, 0, 1\}$
	$[-r/2-1, r/2]$	2	$\{-1, 1\}$

We next explore constraints on the digit set and transfer values dictated by the requirements for carry-free addition, where we make use of the following definitions:

Definition 1: A transfer value $c_i \in G$ is *useful* if the set Δ_i is nonempty; i.e., there exists some position sum value p that may be decomposed as $p = w + rc_i$, where $w \in [a, b]$. ■

Definition 2: A transfer value $c_i \in G$ is *necessary* if the set $\Delta_i - (\Delta_i \cap \Delta_{i+1}) - (\Delta_{i-1} \cap \Delta_i)$, where c_i constitutes the only valid choice of transfer digit value, is nonempty. ■

Definition 3: The *necessity range* of p for c_i, $0 < i < d - 1$, is the possibly empty interval $[b + rc_{i-1} + 1, a + rc_{i+1} - 1]$ where c_i is *necessary*, and neither c_{i-1}, nor c_{i+1} is useful. ■

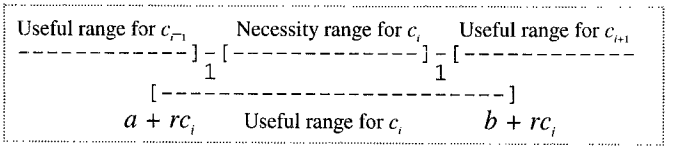

Fig. 1. Illustrating Definitions 1-3.

For a representation system to be closed under carry-free addition [Jabe00], the range $[2a + 2c_0, 2b + 2c_{d-1}]$ of the position sum p should be totally contained within $[a, b] + \{rc_0, rc_1, \ldots, rc_{d-1}\}$. This leads to the following results.

Lemma 1: If $m = \max_i \delta_i$ is the maximum spacing of values in G, we must have $\rho' \geq (m - 1)r$ for carry-free addition to be possible with stored transfer representation.

Proof: Consider consecutive transfer values j and $j + m$ in G. The ranges of p for which these transfer values can be chosen are $\Delta_j = [a + jr, b + jr]$ and $\Delta_{j+m} = [a + (j + m)r, b + (j + m)r]$, respectively. To avoid gaps in the p values, Δ_j and Δ_{j+m} must overlap:

$$b + jr + 1 \geq a + (j + m)r$$

This is easily converted to $\rho' = b - a + 1 - r \geq (m - 1)r$. ■

Corollary 1: Given a value for ρ', the maximum allowed spacing of values in G is $\rho'/r + 1$ (i.e., $\delta_i \leq \rho'/r + 1$). ■

Corollary 2: Given a value for ρ', the overlap between Δ_i and Δ_{i+1} (cardinality of $\Delta_i \cap \Delta_{i+1}$) is $\lambda_i = \rho' + r - r\delta_i$, i.e., the overlap λ_i is minimized, for $\delta_i = \lfloor \rho'/r \rfloor + 1$. ■

Corollary 3: For $\rho' \leq r - 1$, the transfer set G must contain an interval of integer values (i.e., $\delta_i = 1$ for all i). ■

Theorem 1: The transfer set G must be at least three-valued. Furthermore, a transfer set with four values is generally adequate, except for a few special cases.

Proof: The minimum and maximum transfer values, i.e., c_0 and c_{d-1}, should satisfy the following inequalities:

$$a + rc_0 \leq 2a + 2c_0 \quad \Rightarrow \quad c_0 \leq a/(r - 2)$$
$$2b + 2c_{d-1} \leq b + rc_{d-1} \quad \Rightarrow \quad b/(r - 2) \leq c_{d-1}$$

To minimize d, we aim to maximize the *necessity range* for each $c_i \in G$. We thus choose $c_0 = \lfloor a/(r - 2) \rfloor$ and minimize λ_i by choosing $\delta_i = \lfloor \rho'/r \rfloor + 1$, for all i, as prescribed by Corollary 2. The value of c_{d-1} can then be derived as:

$$c_{d-1} = c_0 + \sum \delta_i = \lfloor a/(r - 2) \rfloor + (d - 1)(\lfloor \rho'/r \rfloor + 1)$$

This equation, along with the lower bound for c_{d-1}, yields:

$$\lfloor a/(r-2) \rfloor + (d-1)(\lfloor \rho'/r \rfloor + 1) \geq (a+\rho'+r-1)/(r-2)$$

Letting $a = (r - 2)u + v$ and $\rho' = rq + y$, with $0 \leq v \leq r - 3$ and $0 \leq y \leq r - 1$, the condition above becomes:

$$u + (d-1)(q+1) \geq u + q + 1 + (v+y+2q+1)/(r-2)$$

Solving this inequality for d, we get:

$$d \geq 2 + \theta \quad \Rightarrow \quad d_{\min} = 2 + \lceil \theta \rceil$$

where $\theta = (v + y + 2q + 1) / [(r - 2)(q + 1)]$. Considering that $r \geq 3$, we next show that $\theta > 3$ ($d_{\min} > 5$) is impossible, and $\theta = 3$ ($d_{min} = 5$) is needed only for a few special cases. To show that $d_{min} > 5$ never holds, we note that

$$\theta = (v + y + 2q + 1) / ((r - 2)(q + 1)) > 3$$

implies $(3r - 8)q < v + y - 3r + 7 \leq 3 - r \leq 0$, which is impossible given that $(3r - 8)q < 0$ holds only if $q < 0$, whereas $q = \lfloor \rho'/r \rfloor \geq 0$. Similarly, setting $\theta > 2$ leads to:

$$(2r - 6)q < v + y - 2r + 5$$

Given that the right-hand side of the inequality above is no greater than 1, we must have $q = 0$ for $r > 3$. This leads to the following special cases for which $d_{\min} = 5$:

$r > 3$, $q = 0$, $v = r - 3$, and $y = r - 1$ or
$r = 3$, $v = 0$, and $y = 2$

For all other cases (i.e., $0 < \theta \leq 2$), we have $3 \leq d_{\min} \leq 4$. ■

The undesirable cases in Theorem 1, where $\theta = 3$, are unlikely to be of practical interest. The radix-3 case (besides not being a power of 2) implies at least five values each for D and G, leading to 6 or more bits per digit. For radix 2^h, $h \geq 2$, the high redundancy implied by $\rho' \geq r - 1$, coupled with 3 bits for the 5-valued stored transfer, can be easily avoided by suitable choice of a that ensures $v < r - 3$ (e.g., $0 \leq a \leq r - 4$, or $-r + 2 \leq a \leq -2$).

Corollary 4: For $\rho' = 0$, we have $d_{\min} = 3$. In this case, $G_{\min} = \{c_0, c_0 + 1, c_0 + 2\}$ is adequate, where $c_0 = \lfloor a/(r - 2) \rfloor$. ■

Corollary 5: For $0 < \rho' \leq r-1$, we have $\theta \leq 2$ and $d_{\min} \leq 4$, except when $v = r-3$ and $y = r-1$, in which case $d_{\min} = 5$. ■

Corollary 6: For carry-free addition to be possible with a digit set Δ, the condition $\rho \geq \delta \geq 2$ is necessary. ■

This last result is consistent with the fact that all the cases with $\rho = \delta = 1$ (e.g., some of those in Example 2) do not support carry-free addition [Parh90].

Lemma 2: The *necessity range* of p for $c_i \in G - \{c_0, c_{d-1}\}$ is nonempty if and only if $\delta_i + \delta_{i+1} > \rho'/r + 1$.

Proof: The requirement $b+1+rc_{i-1} \leq a-1+rc_{i+1}$, with $c_{i+1}-c_{i-1} = \delta_i+\delta_{i+1}$ and $b - a + 1 = \rho' + r$ lead to the desired result. ■

Corollary 7: For $\rho' \leq r - 1$ ($\delta_i + \delta_{i+1} = 2 > \rho'/r + 1$), all $c_i \in G - \{c_0, c_{d-1}\}$, are *necessary* transfer values. ■

Corollary 8: When D is a signed digit set (i.e., $a < 0 < b$) and $\rho' \leq r - 1$, we have $c_0 \leq a/(r - 2) < 0 < b/(r - 2) \leq c_{d-1}$, implying that 0 is a necessary transfer value. Furthermore, $G = \{-1, 0, 1\}$ is adequate. ■

Because a four-valued G is always sufficient (except in a few practically insignificant special cases), compared to the binary encoding of the nonredundant digit set $[0, r - 1]$, our stored-transfer representations need two bits of redundancy per digit. Virtually all practical redundant representations use power-of-two radices and thus imply at least one bit of redundancy. Therefore, the incremental cost of our scheme, in its initial form, and without the enhancement to be covered in Section 5, is one bit of redundancy per digit.

4. Speed and Cost Implications

The added cost of one bit per digit position buys us significant latency improvement in the basic operation of carry-free addition and all other arithmetic operations that use addition as a building block. In multioperand addition, and thus in multiplication, as well as in subtractive and multiplicative divison, the per-add savings are compounded over many addition levels.

Because the main digit part can be in 2's-complement format with $\rho' = 0$, much of digit-level addition circuits can be based on readily available, and well optimized, binary adder cells. For example, a digit adder can be built from an h-bit binary adder, computing the $(h + 1)$-bit sum $x_i' + y_i'$, followed by a special $(h + 5)$-input, $(h + 2)$-output circuit; the inputs are the aforementioned $(h + 1)$-bit sum and two 2-bit stored transfers x_i'' and y_i'', while the outputs are the h-bit sum digit s_i' and a 2-bit generated transfer s_{i+1}''. Except for an $O(h)$-time digit addition, the rest of the computation may be performed in a small constant time, independent of the radix (see Section 5).

One way to compare the speed of addition in the stored-transfer scheme with other representations, is to use the notion of *high-radix coefficient* introduced in [Jabe01], where signed-magnitude/1's-/2's-complement encodings of redundant digits are studied. The high radix coefficient corresponds to the number of simple digit-level addition and increment operations needed for adding two redundant numbers. As discussed above, stored-transfer representation has a high-radix coefficient of 1, whereas those of the other three representations are 2 for 2's-complement, 3 for 1's-complement, and 4 for signed-magnitude. A comparison between our stored-transfer scheme and hybrid signed-digit representation [Phat94] will be provided in Section 6.

5. Two-Valued Stored Transfers

The representational efficiency of our stored-transfer scheme can be improved by using the following "trick". Consider a 3-valued transfer $x'' \in \{-1, 0, 1\}$ attached to a main digit $x' = 2u' + v'$, where $v' = x' \bmod 2$ and $u' = \lfloor x'/2 \rfloor$. We assume that x' is encoded in two parts: a single bit denoting v' and an arbitrary encoding of u'. A given stored-transfer digit $\langle 2u' + 0, 0 \rangle$ can be recoded as $\langle 2u' + 1, -1 \rangle$, and $\langle 2u' + 1, 0 \rangle$ as $\langle 2u' + 0, 1 \rangle$, thus making it unnecessary to store the transfer value 0. The resulting 2-valued stored transfer renders the representational efficiency of our scheme competitive with the most efficient redundant representations. The cost of this recoding is small, given that it affects only a single bit v' in the encoding of x'. The case of a 3-valued transfer $x'' \in \{0, 1, 2\}$ is similar: recode $\langle 2u' + 0, 1 \rangle$ as $\langle 2u' + 1, 0 \rangle$, and $\langle 2u' + 1, 1 \rangle$ as $\langle 2u' + 0, 2 \rangle$.

This scheme, which may be viewed as reintroducing step 3 of the carry-free addition process, but in much simpler form involving single-bit logical operations, can be applied after each carry-free addition operation to keep representations efficient in the arithmetic circuits and their associated registers or it can be applied only at the interface between the arithmetic unit and storage system.

Ad-hoc simplifications and efficient implementations for special cases of ρ', and G, may be derived. For example we give the following algorithm for addition of two stored-transfer digits x_i, and y_i, where $\rho' = 0$ and $G = \{-1, 1\}$:

1. Form the h-bit 2's-complement value $z_i = x_i'' + y_i''$
2. Derive the carry-save sum $(u_i, v_i) = z_i + x_i' + y_i'$
3. Add u_i and v_i to form the binary position sum p_i
4. Derive s_i' and s_{i+1}'' satisfying $s_i' = p_i - r s_{i+1}''$
5. Adjust s_i'' and the least significant bit of s_i'

If we encode G as $\{0, 1\}$, the rightmost bit of z_i is always 0, the next bit is derived by an XNOR operation, and the identical leftmost $h - 2$ bits by a NOR operation. Standard full-adders may be used in step 2. Step 3 requires an h-bit ($h - 1$ if an extra half-adder is used in step 2) adder which can be of any suitable design. In step 4, s_i' and s_{i+1}'' are directly derived in constant time from p_i and its two most significant bits, respectively. Step 5 involves 1 gate delay, as previously discussed. Only step 3 has a latency that depends on h. Moreover, steps 1 & 2 and 3 & 4 may be partially overlapped to further reduce the constant-time component of the addition latency [Jabe00a].

6. Very High Radix Representations

One context in which our scheme is particularly cost-effective is when the radix r is rather large. In this case, we have both lower relative redundancy and greater latency improvement over other radix-r redundant representations. In particular, our scheme can be viewed as a competitor for the hybrid redundancy scheme that provides a mechanism for high-radix redundant representation via incorporating binary signed-digit positions after each group of $h - 1$ ordinary binary positions [Phat94], [Phat99]. Our scheme shares many advantages of hybrid redundancy, while being capable of providing full symmetry in the number system (if desired), offering lower latency, and providing greater flexibility in circuit implementation.

We first compare the representation of k-digit radix-2^h numbers in the hybrid scheme, having 1 BSD and $h - 1$ ordinary bits per digit, with the two-valued stored transfer representation containing an h-bit main part, with $\rho' = 0$ and $G = \{-1, 1\}$. Both schemes require a total of $k(h + 1)$ bits. The range of a k-digit number in the hybrid scheme and in our scheme are $[-r/2, r - 1]R$ and $[-r/2 - 1, r/2]R$, respectively, where $R = (r^k - 1)/(r - 1)$. The maximal symmetric subrange is $[-r/2, r/2]R$ in both cases; that is, where symmetry is required, the two schemes exhibit the same representational efficiency.

Details regarding speed and circuit-cost comparisons will be reported in future. Preliminary results [Jabe00a] indicate that, compared to hybrid redundancy, a few gates are saved in each digit position corresponding to a B position in hybrid redundancy while a comparable number of extra gates are needed for each position corresponding to a BSD position. It thus seems that circuit-cost advantage exists for even moderate radices ($h > 2$) and the advantage becomes significant as we go to higher radices. These observations, along with the fact that any h-bit adder design can be used with stored-transfer representation, whereas hybrid redundancy implies a rather rigid realization, allows for experimentation with various design options and flexibility in optimizing implementation parameters.

7. Conversion to/from 2's Complement

To convert a 2's-complement number to a stored-transfer representation in radix 2^h, where $0, 1 \in G$, we deal with the h-bit groups of the 2's-complement number in parallel. We sign-extend (if necessary) the input number to an equivalent 2's-complement number whose width is a multiple of h. Then we use the ith group as the ith digit's main part, and, except for the most significant group and $t_0 = 0$, set t_{i+1} equal to the most significant bit of the ith group. If $t_{i+1} = 0$, the transfer clearly has no effect and the numerical value is preserved. When $t_{i+1} = 1$, its worth within the h-bit group is 2^{h-1} which is the same as 2^h (transfer) plus -2^{h-1} (negatively weighted bit in the 2's-complement main part). A constant-time postconversion adjustment, such as the one discussed in Section 5, is needed if G does not include $\{0, 1\}$.

For the reverse conversion, we add the main parts with their corresponding transfers, all in parallel. This yields a redundant number with 2's-complement digits. The rest of the process follows conventional redundant-to-binary conversion techniques [Parh00]. We note that converting a 2's-complement number to its stored-transfer equivalent requires little or no circuitry, since it is done by inserting a copy of some bits in place of the transfers. But the reverse conversion, as for any other redundant representation, involes word-width carry propagation.

8. Conclusion

We have shown that the stored-transfer representation of certain redundant numbers offers speed and cost benefits in the carry-free addition process. We proved the necessity of at least three transfer digit values and sufficiency of four values (in all practical situations), for carry-free addition. We further showed that by a simple adjustment in final stage of the carry-free addition algorithm, one can reduce the number of stored transfers to two values, thus requiring one bit for storage. Our stored transfer scheme is thus competitive with other practical redundant representations with regard to storage cost. In particular it has cost, speed, and symmetry advantages over hybrid redundancy.

We also demonstrated that converting a 2's-complement number to stored-transfer form implies virtually no cost or latency, while the reverse conversion needs the obligatory carry propagation. This affinity with 2's-complement numbers, in representation and circuit implementation, is a key strength of the stored-transfer scheme.

Derivation of algorithms for stored-transfer multiplication and division is quite feasible. Very-high-radix SRT division with signed-digit partial remainders and signed-digit quotient [Flyn01] can be modified to accept stored-transfer operands. A series of arithmetic operations can thus be performed without carry propagation by representing the inputs, intermediate results, and outputs in stored-transfer format. Results on other operations, and associated arithmetic support functions such as shifting, will be reported in the near future.

References

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

[Flyn01] Flynn, M.J. and S.F. Oberman, *Advanced Computer Arithmetic Design*, Wiley, 2001.

[Jabe99] Jaberipur, G., "A Generalization of Carry Save Adders for Higher Radix Multi-Operand Addition," Public (unclassified) summary of a 1985 confidential technical report, Ref. # 1976, Iran Telecommunication Research Center, Nov. 1999.

[Jabe00] Jaberipur, G., "High Radix Carry-Free Computer Arithmetic: Ph.D. Dissertation Proposal", Computer Engineering Dept., Sharif Univ. of Technology, Tehran, Iran, Mar. 2000.

[Jabe00a] Jaberipur, G. and S.-M. Fakhraie, "Design of Inter-Digit Carry Free Adders for Signed Digit Stored-Borrow-or-Double-Carry (SSC) Numbers," Internal Report, Electrical & Computer Engineering Dept., Tehran Univ., Aug. 2000.

[Jabe01] Jaberipur, G. and Ghodsi, M., "High Radix Signed Digit Number Systems: Representation Paradigms," submited.

[Koba85] Kobayashi, H., "A Mutioperand Two's Complement Addition Algorithm," *Proc. 7th IEEE Symp. Computer Arithmetic*, pp. 16-19, June 1985.

[Metz59] Metze, G. and J.E. Robertson, "Elimination of Carry Propagation in Digital Computers," *Proc. Int'l Conf. Information Processing*, Paris, pp. 389-396, 1959.

[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, Jan. 1990.

[Parh00] Parhami, B., *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford, 2000.

[Parh01] Parhami, B., "Number Reprsentation and Computer Arithmetic," in *Encyclopedia of Information Systems*, Academic Press, to appear.

[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, pp. 880-891, Aug. 1994.

[Phat99] Phatak, D.S., T. Goff, and I. Koren, "Redundancy Management in Arithmetic Processing via Redundant Binary Representations", *Proc. 33rd Asilomar Conf. Signals Systems and Computers*, Oct. 1999, pp. 1475-1479.