Performing Arithmetic Operations on Round-to-Nearest Representations

Peter Kornerup, *Member*, *IEEE*, Jean-Michel Muller, *Senior Member*, *IEEE*, and Adrien Panhaleux

Abstract—During any composite computation, there is a constant need for rounding intermediate results before they can participate in further processing. Recently, a class of number representations denoted RN-Codings were introduced, allowing an unbiased rounding-to-nearest to take place by a simple truncation, with the property that problems with double-roundings are avoided. In this paper, we first investigate a particular encoding of the binary representation. This encoding is generalized to any radix and digit set; however, radix complement representations for even values of the radix turn out to be particularly feasible. The encoding is essentially an ordinary radix complement representation with an appended round-bit, but still allowing rounding-to-nearest by truncation, and thus avoiding problems with double-roundings. Conversions from radix complement to these round-to-nearest representations can be performed in constant time, whereas conversion the other way, in general, takes at least logarithmic time. Not only is rounding-to-nearest a constant time operation, but so is also sign inversion, both of which are at best log-time operations on ordinary two's complement representations. Addition and multiplication on such fixed-point representations are first analyzed and defined in such a way that rounding information can be carried along in a meaningful way, at minimal cost. The analysis is carried through for a compact (canonical) encoding using two's complement representation, supplied with a round-bit. Based on the fixed-point encoding, it is shown possible to define floating-point representations, and a sketch of the implementation of an FPU is presented.

Index Terms—Signed-digit, round-to-nearest, constant-time rounding and sign-inversion, floating-point representation, double-rounding.

1 Introduction

In a recent paper [1], a class of number representations denoted RN-Codings were introduced, the "RN" standing for "round-to-nearest," as these radix-β, signed-digit representations have the property that truncation yields rounding to the nearest representable value. They are based on a generalization of the observation that certain radix representations are known to possess this property, e.g., the balanced ternary ($\beta = 3$) system over the digit set $\{-1, 0, 1\}$. Another such representation is obtained by performing the original Booth-recoding [2] on a two's complement number into the digit set $\{-1, 0, 1\}$, where it is well-known that the nonzero digits of the recoded number alternate in sign. To distinguish between situations where we are not concerned with the actual encoding of a value, we shall here use the notation *RN-representation*.

We shall in Section 2 (extracted from [1]) cite some of the definitions and properties of the general RN-Codings/representations. However, we will, in particular, explore the binary representation, e.g., as obtained by the Booth recoding; the rounding by truncation property, including the feature that the effect of one rounding followed by another rounding yields the same result, as would be obtained by a single rounding to the same precision as the last.

Section 3 analyzes conversions between RN-representations and two's complement representations. Conversion from the latter to the former is performed by the Booth algorithm, yielding a signed-digit/borrow-save representation in a straightforward encoding, which, for an n-digit word, requires $2n$ bits. It is then realized that $n+1$ bits are sufficient, providing a simpler alternative encoding consisting of the bits of the truncated two's complement encoding, with a round-bit appended, termed the *canonical* encoding. Despite being based on a two's complement encoding, it is observed that sign-inversion (negation) is a constant-time operation on this canonical encoding. Conversion the other way, from RN-representation in this encoding into two's complement representation (essentially adding in the round-bit), is realizable by a parallel prefix structure. Section 4 generalizes the canonical representation to other radices and digit sets, showing that for even values of the radix the encodings employing radix-complement representations are particularly feasible.

Section 5 then analyzes possible implementations of addition and multiplication on fixed-point RN-represented numbers. Beuchat and Muller [3] discussed implementations of these basic operations based on the signed-digit representation of RN-coded numbers, whereas we here exploit the canonical encoding, which seems to be more convenient. Since it turns that there are two possible encodings of the result of an arithmetic operation, interpretations of the encodings as intervals may be used to uniquely define sums and products in a consistent way. Section 6 sketches how a floating point RN-representation may be defined and the basic arithmetic operations of an FPU may be realized. Then,

- *P. Kornerup is with the Department of Mathematics and Computer Science, University of Southern Denmark, Campusvej 55, DK-5230 Odense, Denmark. E-mail: kornerup@imada.sdu.dk.*
- *J.-M. Muller is with the CNRS-LIP-Arénaire, Ecole Normale Superieure de Lyon, 46 alle d'Italie, F-69364 Lyon Cedex 07, France. E-mail: Jean-Michel.Muller@ens-lyon.fr.*
- *A. Panhaleux is with the École Normale Supérieure de Lyon, 46 alle d'Italie, F-69364 Lyon Cedex 07, France. E-mail: Adrien.Panhaleux@ens-lyon.fr.*

Manuscript received 16 Oct. 2009; revised 12 Feb. 2010; accepted 3 Mar. 2010; published online 10 June 2010.
Recommended for acceptance by E. Antelo.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2009-10-0522.
Digital Object Identifier no. 10.1109/TC.2010.134.

Section 7 contains examples on some composite computations where fast and optimal roundings are useful, and may come for free when RN-representation in the canonical encoding is employed. Finally, Section 8 concludes the paper.

2 Definitions and Basic Properties (Cited from [1])

Definition 1 (RN-representations). *Let β be an integer greater than or equal to 2. The digit sequence $D = d_n d_{n-1} d_{n-2} \cdots$ (with $-\beta+1 \le d_i \le \beta-1$) is an RN-representation in radix β of x iff*

1. $x = \sum_{i=-\infty}^{n} d_i \beta^i$ *(that is D is a radix-β representation of x);*
2. *for any $j \le n$,*

$$\left| \sum_{i=-\infty}^{j-1} d_i \beta^i \right| \le \frac{1}{2}\beta^j,$$

that is, if the digit sequence is truncated to the right at any position j, the remaining sequence is always the number (or one of the two members in case of a tie) of the form $d_n d_{n-1} d_{n-2} d_{n-3} \ldots d_j$ that is closest to x.

Hence, truncating the RN-representation of a number at any position is equivalent to rounding it to the nearest.

Although it is possible to deal with infinite representations, we shall first restrict our discussions to finite representations. The following observations on such RN-representations for general $\beta \ge 2$ are then easily found:

Theorem 2 (Finite RN-representations).

- *if $\beta \ge 3$ is odd, then $D = d_m d_{m-1} \cdots d_\ell$ is an RN-representation iff*

$$\forall i, \frac{-\beta+1}{2} \le d_i \le \frac{\beta-1}{2};$$

- *if $\beta \ge 2$ is even, then $D = d_m d_{m-1} \cdots d_\ell$ is an RN-representation iff*
  - *all digits have absolute value less than or equal to $\frac{\beta}{2}$;*
  - *if $|d_i| = \frac{\beta}{2}$, then the first nonzero digit that follows on the right has the opposite sign, that is, the largest $j < i$ such that $d_j \ne 0$ satisfies $d_i \times d_j < 0$.*

Observe that for odd β, the system is nonredundant, whereas for β even, the system is redundant in the sense that some nonzero numbers have two representations. In particular, note that for radix 2, the digit set is $\{-1, 0, 1\}$, known by the names of "binary signed-digit" or "borrow-save," but here restricted such that the nonzero digits have alternating signs.

Theorem 3 (Uniqueness of finite representations).

- *if β is odd, then a finite RN-representation of x is unique;*
- *if β is even, then some numbers may have two finite representations. In that case, one has its least significant nonzero digit equal to $-\frac{\beta}{2}$, the other one has its least significant nonzero digit equal to $+\frac{\beta}{2}$.*

Proof. If β is odd, the result is an immediate consequence of the fact that the digit set is nonredundant. If β is even, then consider two different RN-representations representing the same value x, and consider the largest position j (that is, of weight β^j) such that these RN-representations differ, when truncated to the right of position j. Let x_a and x_b be the values represented by these digit strings. Obviously, $x_a - x_b \in \{-\beta^j, 0, \beta^j\}$. Now, $x_a = x_b$ would contradict the way that j was chosen. Without loss of generality, then assume $x_b = x_a + \beta^j$. This implies $x = x_a + \beta^j/2 = x_b - \beta^j/2$, since the maximal absolute value of a digit is $\beta/2$. Hence, the remaining digit strings (i.e., the parts that were truncated) are digit strings starting from position $j-1$, representing $\pm\beta^j/2$.

The only way of representing $\beta^j/2$ by an RN-representation starting from position $j-1$ is

$$\left(\frac{\beta}{2}\right)0000\cdots0.$$

This is seen as follows: If the digit at position $j-1$ of a number is less than or equal to $\frac{\beta}{2}-1$, then that number is less than or equal to

$$\left(\frac{\beta}{2}-1\right)\beta^{j-1} + \left(\frac{\beta}{2}\right)\sum_{i=\ell}^{j-2}\beta^i < \beta^j/2,$$

since the largest allowed digit is $\frac{\beta}{2}$. Also, the digit at position $j-1$ of an RN-representation cannot be larger than or equal to $\frac{\beta}{2}+1$. □

If β is even, then a number whose finite representation (by an RN-representation) has its last nonzero digit equal to $\frac{\beta}{2}$ has an alternative representation ending with $-\frac{\beta}{2}$ (just assume the last two digits are $d(\frac{\beta}{2})$: since the representation is an RN-representation, $d < \frac{\beta}{2}$, hence if we replace these two digits by $(d+1)(-\frac{\beta}{2})$ we still have a valid RN-representation). This has an interesting consequence; truncating a number, which is a tie, will round either way, depending on which of the two possible representations the number happens to have. Hence, there is no bias in the rounding.

Note that this rounding rule is different from the "round-to-nearest-even" rule required by the IEEE standard [4]. Both roundings provide a "round-to-nearest" in the case of a tie, but employ different rules when choosing which way to round. Also note that this rounding is also deterministic; the direction of rounding only depends on how the value to be rounded was derived, as the representation of the value is uniquely determined by the sequence of operations leading to the value.

Example.

- In radix 7, with digits $\{-3, -2, -1, 0, 1, 2, 3\}$, all representations are RN-representations, and no number has more than one representation;
- In radix 10 with digits $\{-5, \ldots, +5\}$, 15 has two RN-representations: 15 and $2\overline{5}$.

Theorem 4 (Uniqueness of infinite representations). *We now consider* infinite *representations, i.e., representations that do not ultimately terminate with an infinite sequence of zeros.*

- *if β is odd, then some numbers may have two infinite RN-representations. In that case, one is eventually finishing with the infinite-digit string*

$$\frac{\beta-1}{2}\frac{\beta-1}{2}\frac{\beta-1}{2}\frac{\beta-1}{2}\frac{\beta-1}{2}\frac{\beta-1}{2}\cdots$$

and the other one is eventually finishing with the infinite-digit string

$$\frac{-\beta+1}{2}\frac{-\beta+1}{2}\frac{-\beta+1}{2}\frac{-\beta+1}{2}\frac{-\beta+1}{2}\frac{-\beta+1}{2}\cdots;$$

- *if β is even, then two different infinite RN-representations necessarily represent different numbers. As a consequence, a number that is not an integer multiple of an integral (positive or negative) power of β has a unique RN-representation.*

Proof. If β is odd, the existence immediately comes from

$$1.\frac{-\beta+1}{2}\frac{-\beta+1}{2}\frac{-\beta+1}{2}\frac{-\beta+1}{2}\cdots = 0.\frac{\beta-1}{2}\frac{\beta-1}{2}\frac{\beta-1}{2}\frac{\beta-1}{2}\cdots = \frac{1}{2}.$$

Now, if for any β (odd or even) two different RN-representations represent the same number x, then consider them truncated to the right of some position j, such that the obtained digit strings differ. The obtained digit strings represent values x_a and x_b whose difference is $\pm\beta^j$ (a larger difference is impossible for obvious reasons).

First, consider the case where β is odd. From the definition of RN-representations, and assuming $x_a < x_b$, we have $x = x_a + \beta^j/2 = x_b - \beta^j/2$. Since β is odd, the only way of representing $\beta^j/2$ is with the infinite digit string (that starts from position $j-1$)

$$\frac{\beta-1}{2}\frac{\beta-1}{2}\frac{\beta-1}{2}\frac{\beta-1}{2}\cdots$$

and the result immediately follows.

Now, consider the case where β is even. Let us first show that $x_a = x_b$ is impossible. From Theorem 3, this would imply that one of the corresponding digit strings would terminate with the digit sequence $-\frac{\beta}{2}00\cdots00$, and the other one with the digit string $+\frac{\beta}{2}00\cdots00$. But from Theorem 2, this would imply that the remaining (truncated) terms are positive in the first case, and negative in the second case, which would mean (since $x_a = x_b$ implies that they are equal) that they would both be zero, which is not compatible with the fact that the representations of x are assumed infinite. Hence $x_a \neq x_b$. Assume $x_a < x_b$, which implies $x_b = x_a + \beta^j$. We necessarily have $x = x_a + \beta^j/2 = x_b - \beta^j/2$. Although $\beta^j/2$ has several possible representations in a "general" signed-digit radix-β system, the only way of representing it with an RN-representation is to put a digit $\frac{\beta}{2}$ at position $j-1$, and hence, no infinite representation is possible. □

Example.

- In radix 7, with digits $\{-3,-2,-1,0,1,2,3\}$, the number $3/2$ has two infinite representations, namely $1.3333333333\cdots$ and $2.\overline{3333333333}\cdots$
- in radix 10 with digits $\{-5,\ldots,+5\}$, the RN-representation of π is unique.

An important property of the RN-representation is that it avoids the *double rounding* problem occurring with some rounding methods, e.g., with the standard IEEE round-to-nearest-even. This may happen when the result of first rounding to a position j, followed by rounding to position k, does not yield the same result as if directly rounding to position k, as also discussed in [5]. We repeat from [1] the following result:

Observation 5 (Double rounding). *Let $\mathrm{rn}_i(x)$ be the function that rounds the value of x to nearest at position i by truncation. Then, for $k > j$, if x is represented in the RN-representation, then*

$$\mathrm{rn}_k(x) = \mathrm{rn}_k(\mathrm{rn}_j(x)).$$

3 Converting to and from Binary RN-Representation

3.1 Conversion from Two's Complement to RN-Representation

Consider an input value $x = -b_m 2^m + \sum_{i=\ell}^{m-1} b_i 2^i$ in two's complement representation:

$$x \sim b_m b_{m-1}\cdots b_{\ell+1} b_\ell$$

with $b_i \in \{0,1\}$ and $m > \ell$. Then, the digit string

$$\delta_m \delta_{m-1}\cdots\delta_{\ell+1}\delta_\ell \quad \text{with} \quad \delta_i \in \{-1,0,1\}$$

defined (by the Booth recoding [2]) for $i = \ell,\cdots,m$ as

$$\delta_i = b_{i-1} - b_i \quad (\text{with } b_{\ell-1} = 0 \text{ by convention}) \qquad (1)$$

is an RN-representation of x with $\delta_i \in \{-1,0,1\}$. That it represents the same value follows trivially by observing that the converted string represents the value $2x - x$. The alternation of the signs of nonzero digits is easily seen by considering how strings of the form $011\cdots10$ and $100\cdots01$ are converted.

Thus, the conversion can be performed in constant time. Actually, the digits of the two's complement representation directly provides for an encoding of the converted digits as a tuple: $\delta_i \sim (b_{i-1}, b_i)$ for $i = \ell,\ldots,m$, where

$$\begin{aligned} -1 &\sim (0,1),\\ 0 &\sim (0,0) \text{ or } (1,1),\\ 1 &\sim (1,0), \end{aligned} \qquad (2)$$

where the value of the digit is the difference between the first and the second component.

Example. Let $x = 110100110010$ be a sign-extended two's complement number and write the digits of $2x$ above the digits of x:

$2x$	1	0	1	0	0	1	1	0	0	1	0	0
x	1	1	0	1	0	0	1	1	0	0	1	0
x in RN-repr.		$\bar{1}$	1	$\bar{1}$	0	1	0	$\bar{1}$	0	1	$\bar{1}$	0

where it is seen that in any column the two uppermost bits provide the encoding defined above of the signed-digit below in the column. Since the digit in position $m+1$ will always be 0, there is no need to include the most significant position otherwise found in the two top rows.

If x is nonzero and b_k is the least significant nonzero bit of the two's complement representation of x, then $\delta_k = -1$, as confirmed in the example, and hence the last nonzero digit is always $\bar{1}$ and thus unique. However, if an RN-represented number is truncated for rounding somewhere, the resulting representation may have its last nonzero digit of value 1.

As mentioned in Theorem 3, there are exactly two finite binary RN-representations of any nonzero binary number of the form $a2^k$ for integral a and k, but requiring a specific sign of the last nonzero digit makes the representation unique. On the other hand, without this requirement, rounding by truncation makes the rounding unbiased in the tie-situation, by randomly rounding up or down, depending on the sign of the last nonzero digit in the remaining digit string.

Example. Rounding the value of x in Example 1 by truncating off the two least significant digits, we obtain

$\mathrm{rn}_2(2x)$	1	0	1	0	0	1	1	0	0	**1**
$\mathrm{rn}_2(x)$	1	1	0	1	0	0	1	1	0	0
$\mathrm{rn}_2(x)$ in RN-repr.		$\bar{1}$	1	$\bar{1}$	0	1	0	$\bar{1}$	0	1

where it is noted that the bit of value 1 in the upper rightmost corner (in boldface) acts as a round bit, assuring a round-up in cases where there is a tie-situation as here.

The example shows that there is another very compact encoding of RN-represented numbers derived directly from the two's complement representation, noting in the example that the upper row need not be part of the encoding, except for the round-bit. We will denote it as the *canonical encoding*, and note that it is a kind of "carry-save" in the sense that it contains a bit not yet added in. The same idea has previously been pursued in [6] in a floating-point setting, denoted "packet-forwarding."

Definition 6 (Binary canonical RN-encoding). *Let the number x be given in two's complement representation as the bit string $b_m \cdots b_{\ell+1} b_\ell$, such that $x = -b_m 2^m + \sum_{i=\ell}^{m-1} b_i 2^i$. Then, the* binary canonical encoding *of the RN-representation of x is defined as the pair*

$$x \sim (b_m b_{m-1} \cdots b_{\ell+1} b_\ell, \mathrm{r}) \text{ where the round-bit is } \mathrm{r} = 0$$

and after truncation at position k, for $m \geq k > \ell$

$$\mathrm{rn}_k(x) \sim (b_m b_{m-1} \cdots b_{k+1} b_k, r) \text{ with round-bit } r = b_{k-1}.$$

If (x, r_x) is the binary canonical (two's complement) RN-representation of X, then $X = x + r_x u$, where u is the weight of the least significant position, from which it follows that

$$-X = -x - r_x u = \bar{x} + u - r_x u = \bar{x} + (1 - r_x)u = \bar{x} + \bar{r}_x u.$$

Observation 7. *If (x, r_x) is the canonical RN-representation of a value X, then $(\bar{x}, \bar{r}_x)$ is the canonical RN-representation of $-X$, where $\bar{x}$ is the 1's complement of x. Hence, negation of a canonically encoded value is a constant-time operation.*

The signed-digit interpretation is available from the canonical encoding by pairing bits, (b_{i-1}, b_i) using the encoding (2) for $i > k$ and (r, b_k), when truncated at position k.

There are other equally compact encodings of RN-represented numbers, e.g., one could encode the signed-digit string simply by the string of bits obtained as the absolute values of the digits, together with say the sign of the most (or least) nonzero digit. Due to the alternating signs of the nonzero digits, this is sufficient to reconstruct the actual digit values. However, this encoding does not seem very convenient for arithmetic processing, as the correct signs will then have to be distributed over the bit string.

3.2 Conversion from Signed-Digit RN-Representation to two's Complement

The example of converting $0000000\bar{1}$ into its two's complement equivalent 11111111 shows that it is not possible to perform this conversion in constant time as information may have to travel an arbitrary distance to the left. Hence, a conversion may, in general, take at least logarithmic time. Since the RN-representation is a special case of the (redundant) signed-digit representation, this conversion is fundamentally equivalent to an addition.

If an RN-represented number is in canonical encoding, conversion into ordinary two's complement representation may require a nonzero round-bit to be added in; it simply consists in an incrementation, for which very efficient methods exist, based on parallel prefix trees with AND-gates as nodes.

4 Canonical Representation, the General Case

The binary canonical representation of RN-representation is specified by $x = (a, r_a)$, which is a pair of a number and a bit. We could decide to represent the value of a of that pair in something else than binary, say using a higher radix β and/or a digit set different from the set $\{0, \ldots, \beta - 1\}$.

Definition 8 (Canonical encoding: general case). *Let b be a number in radix β using the digit set D, such that $b = \sum_{i=\ell}^{m-1} b_i \beta^i$ with $b_i \in D$, and the rounding bit $r_b \in \{0, 1\}$. The pair (b, r_b) then represents the value $b + u r_b$, where u is the unit in the last place ($u = \beta^l$).*

The definition is very general as the representation doesn't necessarily allow rounding by truncation. We must redefine the rounding operation so that we avoid problems with double-roundings, basically by trying to convert the encoding into an RN-representation satisfying Definition 1.

4.1 Even Radix

The definition seems to make particular sense when a (nonnegative) number is represented in an even radix with the regular digit-set $\{0, \ldots, \beta - 1\}$. In that representation, the link between the canonical encoding and RN-representation is trivial enough, so that rounding-to-nearest can be done by truncation of the value b in the pair (b, r_b).

Consider an input value in radix-β with $0 \leq d_i \leq \beta - 1$

$$(x, r_x) = (d_m d_{m-1} d_{n-2} \cdots d_\ell, r_x),$$

and define variables c_k as

$$c_{k+1} = \begin{cases} 1, & \text{if } d_k \geq \frac{\beta}{2}, \\ 0, & \text{if } d_k < \frac{\beta}{2}. \end{cases} \quad (3)$$

With $c_\ell = r_x$, the digits δ_k of an RN-representation can be obtained using

$$\delta_k = d_k + c_k - \beta c_{k+1}.$$

The conversion for an even radix and digit set $\{0, \ldots, \beta - 1\}$ into an RN-representation gives us a way to easily perform rounding-to-nearest by truncation of (x, r_x) in the canonical encoding. For $k > \ell$:

$$\mathrm{rn}_k(x, r_x) \sim (d_n d_{n-1} \ldots d_{k+1} d_k, r),$$
$$\text{with round-bit } r = \begin{cases} 1, & \text{if } d_{k-1} \geq \beta/2, \\ 0, & \text{if } d_{k-1} < \beta/2, \end{cases}$$

and hence in RN-representation, the value can also be expressed by the digit string $\delta_n \delta_{n-1} \ldots \delta_k \in \{-\frac{\beta}{2}, \ldots, \frac{\beta}{2}\}$.

Example. With radix $\beta = 10$ and the regular digit-set $\{0, \ldots, 9\}$, for the value 9.25451 represented by $(9.25450, 1)$, we can truncate using the previous algorithm:

$$\mathrm{rn}_{-3}(9.25450, 1) = (9.254, 1),$$

the rounding-bit being 1 because $d_{-4} = 5 \Rightarrow c_{-3} = 1$. We only need to generate one carry (c_{-3}) to obtain the rounding bit.

To confirm that the rounding is correct, we may represent the value in RN-representation, by generating all the carries:

c_k	1	0	1	0	1	0	**1**
d_k	0	9	2	5	4	5	0
RN-representation	1	$\bar{1}$	3	$\bar{5}$	5	$\bar{5}$	1

With this RN-representation, truncating at position -3 gives:

c_k	1	0	1	0	**1**
d_k	0	9	2	5	4
RN-representation	1	$\bar{1}$	3	$\bar{5}$	5

corresponding to the previous canonical encoding $(9.254, 1)$. Note that to represent the given positive value, d_1 was set to zero. Had the value been a (negative) 10's complement represented number, then d_1 should, by sign extension, have been set to nine.

4.2 Other Representations

If we use other representations of b (say binary borrow-save/signed-digit, odd radices,...), the rounding may take time $O(\log(n))$:

Example [Borrow-save]. When trying to round x in a general borrow-save representation to the nearest integer, we have for any round bit $r_x \in \{0, 1\}$:

borrow-save	rounded
$(1\bar{1}\bar{1}0.00\ldots 0\bar{1}, r_x)$	$(1\bar{1}\bar{1}\bar{1}, 1)$
$(1\bar{1}\bar{1}0.00\ldots 00, r_x)$	$(1\bar{1}\bar{1}0, 0)$

Hence, we may have to look arbitrarily far to the right when rounding the values.

However, borrow-save could be interesting, since addition then can be performed in $O(1)$, instead of $O(\log(n))$ for binary canonical encoding using the regular digit-set $\{0, 1\}$. It is important to recall that borrow-save is not an RN-representation even though it uses the same digits. To have an RN-representation, the nonzero digits must alternate in signs, and translating an arbitrary number from borrow-save to RN-representation may take $O(\log(n))$ time.

Example [Odd radices]. When trying to round to the nearest integer, we have similarly:

radix 3	rounded
$(10.11\ldots 12, r_x)$	$(10, 1)$
$(10.11\ldots 11, r_x)$	$(10, 0)$

which means we may have to look arbitrarily far to the right when rounding the values. It is due to the fact that the midpoint between two representable numbers needs to be represented with an infinite number of digits. If we were to redefine arithmetic from scratch, odd radices could be a choice to be considered; but since we have to keep simple conversion to conventional number systems in mind, we decided in the following to focus on radix 2.

5 Performing Arithmetic Operations on Canonically Represented Values

The fundamental idea of the canonical radix-2 RN-representation is that it is a binary representation of some value using the digit set $\{-1, 0, 1\}$, but such that nonzero digits alternate in sign. We then introduced an encoding of such numbers, employing two's complement representation, in the form (a, r_a) representing the value

$$(2a + r_a u) - a = a + r_a u,$$

where u is the weight of the least significant position of a. Note that there is then no difference between $(a, 1)$ and $(a + u, 0)$, both being RN-representations of the same value:

$$\forall a, \mathcal{V}(a, 1) = \mathcal{V}(a + u, 0),$$

where we use the notation $\mathcal{V}(x, r_x)$ to denote the value of an RN-represented number.

5.1 An Interval Interpretation

Considered as intervals as described below, the two representations $(a, 1)$ and $(a + u, 0)$ describe different intervals. Since different representations of the same number can give different rounding results when truncated, it is then important to choose carefully the representation of the result when performing arithmetic operations like addition and multiplication. Hence, when defining the result, it is essential to choose the encoding of it to reflect the domains of the operands.

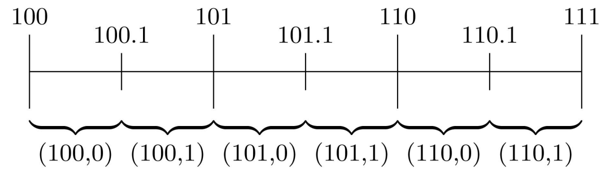


Fig. 1. Example of interpreting RN representations as intervals with $u = 1$.

Consider a value A to be rounded at some position of weight u where the round bit is 1, shown in boldface:

$$\begin{array}{r|l} \ldots\ 0\ 1\ 1\ \ldots\ \mathbf{1} & \mathrm{x}\ \ldots \\ \ldots\ 0\ 1\ \ldots\ 1 & 1\ \ \mathrm{x}\ \ldots \\ \hline \underbrace{\ldots\ 1\ 0\ \ldots\ 0}_{a+u} & \underbrace{\qquad\qquad}_{-\frac{u}{2} \le t \le 0} \end{array}$$

$$\begin{array}{r|l} \ldots\ 0\ \mathbf{1} & \mathrm{x}\ \ldots \\ \ldots\ 0 & 1\ \ \mathrm{x}\ \ldots \\ \hline \underbrace{\ldots\ 1}_{a+u} & \underbrace{\qquad\qquad}_{-\frac{u}{2} \le t \le 0} \end{array}$$

and similarly when the round bit is 0:

$$\begin{array}{r|l} \ldots\ 1\ 0\ 0\ \ldots\ \mathbf{0} & \mathrm{x}\ \ldots \\ \ldots\ 1\ 0\ \ldots\ 0 & 0\ \ \mathrm{x}\ \ldots \\ \hline \underbrace{\ldots\ 1\ 0\ \ldots\ 0}_{a} & \underbrace{\qquad\qquad}_{0 \le t \le \frac{u}{2}} \end{array}$$

$$\begin{array}{r|l} \ldots\ 1\ \mathbf{0} & \mathrm{x}\ \ldots \\ \ldots\ 1 & 0\ \ \mathrm{x}\ \ldots \\ \hline \underbrace{\ldots\ 1}_{a} & \underbrace{\qquad\qquad}_{0 \le t \le \frac{u}{2}} \end{array}$$

expressing bounds on the tail t thrown away during rounding by truncation. Observe that the right-hand ends of the intervals are closed, corresponding to a possibly infinite sequence of units having been thrown away. We find that the value A before rounding into $\mathcal{V}(a, r_a)$ must belong to the interval:

$$\begin{aligned} A \in & \left\{ \begin{array}{ll} \left[a \,;\, a + \frac{u}{2}\right], & \text{for } r_a = 0 \\ \left[a + \frac{u}{2} \,;\, a + u\right], & \text{for } r_a = 1 \end{array} \right\} \\ = & \left[a + r_a \frac{u}{2} \,;\, a + (1 + r_a)\frac{u}{2}\right] = \mathcal{I}(a, r_a). \end{aligned}$$

In the following, we shall use $\mathcal{I}(a, r_a)$ to denote the interval, the idea being to remember where the real number was before rounding.

We may interpret the representations of an encoding as an interval of length $u/2$, as in Fig. 1. In the figure, any number between 101 and 101.1 (for example, 101.01), when rounded to the nearest integer, will give the RN representation $(101, 0)$. So, we may say that $(101, 0)$ represents the interval $[101; 101.1]$ and, in particular,

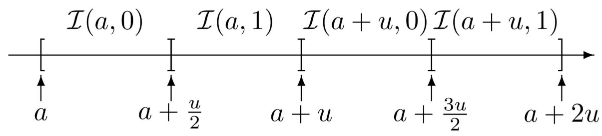


Fig. 2. Binary canonical RN-representations as intervals.

$$\mathcal{I}(a, 1) = \left[a + \frac{u}{2} \,;\, a + u\right],$$
$$\mathcal{I}(a + u, 0) = \left[a + u \,;\, a + \frac{3u}{2}\right].$$

Hence, even though the two encodings represent the same value $(a + u)$, when interpreting them as intervals according to what could have been thrown away, the intervals are essentially disjoint, except for sharing a single point. In general, we may express the interval interpretation as pictured in Fig. 2

We do not intend to define an interval arithmetic, but only require that the interval representation of the result of an arithmetic operation $\odot$ satisfies[1]

$$\mathcal{I}(A \odot B) \subseteq \mathcal{I}(A) \odot \mathcal{I}(B) = \{a \odot b | a \in A, b \in B\}.$$

To simplify the discussion, we will, in this section, only consider fixed-point representations for some fixed value of u. We will not discuss overflow problems, as we assume that we have enough bits to represent the result in canonically encoded representation.

5.2 Addition of RN-Represented Values

Employing the value interpretation, we have for addition:

$$\begin{array}{r} \mathcal{V}(a, r_a) = a + r_a u \\ +\mathcal{V}(b, r_b) = b + r_b u \\ \hline \mathcal{V}(a, r_a) + \mathcal{V}(b, r_b) = a + b + (r_a + r_b)u \end{array}.$$

The resulting value has two possible representations, depending on the rounding bit of the result. To determine what the rounding bit of the result should be, Table 1 shows the interval interpretations of the two possible representations of the result, depending on the rounding bits of the operands.

Since we want $\mathcal{I}(\mathcal{V}(a, r_a) + \mathcal{V}(b, r_b)) \subseteq \mathcal{I}(a, r_a) + \mathcal{I}(b, r_b)$, and $(a, r_a) + (0, 0) = (a, r_a)$, and in order to keep the addition symmetric, we define the addition of RN-encoded numbers as follows:

Definition 9 (Addition). *If u is the unit in the last place of the operands, let:*

$$(a, r_a) + (b, r_b) = ((a + b + (r_a \wedge r_b)u), r_a \vee r_b).$$

Recalling that $-(x, r_x) = (\bar{x}, \bar{r_x})$, we observe that using this definition, $(x, r_x) - (x, r_x) = (-u, 1)$, with $\mathcal{V}(-u, 1) = 0$. It is possible alternatively to define addition on RN-encoded numbers as $(a, r_a) +_2 (b, r_b) = ((a + b + (r_a \vee r_b)u), r_a \wedge r_b)$. Using this definition, $(x, r_x) -_2 (x, r_x) = (0, 0)$, but then the neutral element for addition is $(-u, 1)$, i.e., $(x, r_x) +_2 (-u, 1) = (x, r_x)$.

1. Note that this is the reverse inclusion of that required for ordinary interval arithmetic, e.g., [7].

TABLE 1
Interpretations of Additions as Intervals

	$\mathcal{I}(\mathcal{V}(a,r_a)+\mathcal{V}(b,r_b))$	$\mathcal{I}(a,r_a)+\mathcal{I}(b,r_b)$
$\mathbf{r_a = r_b = 0}$	$\mathcal{I}(a+b-u,1) = \left[a+b-\frac{u}{2}\ ;\ a+b\right]$	$\not\subseteq [a+b\ ;\ a+b+u]$
	$\mathcal{I}(a+b,0) = \left[a+b\ ;\ a+b+\frac{u}{2}\right]$	$\subseteq [a+b\ ;\ a+b+u]$
$\mathbf{r_a \oplus r_b = 1}$	$\mathcal{I}(a+b,1) = \left[a+b+\frac{u}{2}\ ;\ a+b+u\right]$	$\left.\right\} \subseteq \left[a+b+\frac{u}{2}\ ;\ a+b+\frac{3u}{2}\right]$
	$\mathcal{I}(a+b+u,0) = \left[a+b+u\ ;\ a+b+\frac{3u}{2}\right]$	
$\mathbf{r_a = r_b = 1}$	$\mathcal{I}(a+b+u,1) = \left[a+b+\frac{3u}{2}\ ;\ a+b+2u\right]$	$\subseteq [a+b+u\ ;\ a+b+2u]$
	$\mathcal{I}(a+b+2u,0) = \left[a+b+2u\ ;\ a+b+\frac{5u}{2}\right]$	$\not\subseteq [a+b+u\ ;\ a+b+2u]$

Example. Let us take two examples adding two numbers that were previously rounded to the nearest integer.

	Addition not rounded	Addition on rounded canonical representations
a_1	01011.1110	(01011, 1)
b_1	01001.1101	(01001, 1)
a_1+b_1	010101.1011	(010101, 1)
a_2	01011.1010	(01011, 1)
b_2	01001.1001	(01001, 1)
a_2+b_2	010101.0011	(010101, 1)

Using the definition above, $\mathrm{rn}_0(a_1+b_1)=\mathrm{rn}_0(a_1)+\mathrm{rn}_0(b_1)$ holds in the first case. Obviously, since some information may be lost during rounding, there are cases like in the second example, where $\mathrm{rn}_0(a_2+b_2)\neq \mathrm{rn}_0(a_2)+\mathrm{rn}_0(b_2)$. Also note that due to that information loss, a_2+b_2 is not in $\mathcal{I}((a_2,r_{a_2})+(b_2,r_{b_2}))$.

When interpreted as an interval, $\mathcal{I}(x,r_x)=[x+r_x\frac{u}{2};x+(1+r_x)\frac{u}{2}]$. Then, its "mirror image" interval of negated values is $-\mathcal{I}(x,r_x)=[\bar{x}+\bar{r}_x\frac{u}{2}\ ;\ \bar{x}+(1+\bar{r}_x)\frac{u}{2}]=\mathcal{I}(\bar{x},\bar{r}_x)$. Thus, we consider for subtraction

$$\begin{aligned}
&\mathcal{I}((a,r_a)-(b,r_b))\\
&\quad=\mathcal{I}\big((a,r_a)+(\bar{b},\bar{r}_b)\big)\\
&\quad=\mathcal{I}(a+\bar{b}+(r_a\vee\bar{r}_b)u,r_a\wedge\bar{r}_b)\\
&\quad=a+\bar{b}+u(r_a\vee\bar{r}_b)+\frac{u}{2}(r_a\wedge\bar{r}_b)+\left[0\ ;\ \frac{u}{2}\right]\\
&\quad=a-b+u(r_a-r_b)-\frac{u}{2}(r_a\wedge\bar{r}_b)+\left[0\ ;\ \frac{u}{2}\right]\\
&\quad=a-b+\frac{u}{2}(r_a-r_b)-\frac{u}{2}(\bar{r}_a\wedge r_b)+\left[0\ ;\ \frac{u}{2}\right].
\end{aligned}$$

On the other hand,

$$\begin{aligned}
&\mathcal{I}(a,r_a)-\mathcal{I}(b,r_b)\\
&\quad=\left[a+\frac{u}{2}r_a\ ;\ a+\frac{u}{2}(1+r_a)\right]-\left[b+\frac{u}{2}r_b\ ;\ b+\frac{u}{2}(1+r_b)\right]\\
&\quad=a-b+\frac{u}{2}(r_a-r_b)+\left[-\frac{u}{2}\ ;\ \frac{u}{2}\right].
\end{aligned}$$

Hence, for all points $x\in\mathcal{I}((a,r_a)-(b,r_b))$, x is in $\mathcal{I}(a,r_a)-\mathcal{I}(b,r_b)$.

Hence, subtraction of (x,r_x) can be realized by addition of the bitwise inverted tuple $(\bar{x},\bar{r}_x)$.

5.3 Multiplying Canonically Encoded RN-Represented Values

By definition, we have for the value of the product

$$\begin{aligned}
\mathcal{V}(a,r_a)&=a+r_au\\
\mathcal{V}(b,r_b)&=b+r_bu\\
\hline
\mathcal{V}(a,r_a)\mathcal{V}(b,r_b)&=ab+(ar_b+br_a)u+r_ar_bu^2,
\end{aligned}$$

noting that the unit of the result is u^2, assuming that $u\le 1$. Considering the operands as intervals, we find using (4):

$$\begin{aligned}
&\mathcal{I}(a,r_a)\times\mathcal{I}(b,r_b)\\
&\quad=\left[a+r_a\frac{u}{2}\ ;\ a+\frac{u}{2}(1+r_a)\right]\times\left[b+\frac{u}{2}r_b\ ;\ b+\frac{u}{2}(1+r_b)\right]\\
&\quad=a'b'\ +\begin{cases}
\left[0\ ;\ a'+b'+\frac{u}{2}\right]\times\frac{u}{2}, & \text{for } a>0,\ b>0,\\
[a'\ ;\ b']\times\frac{u}{2}, & \text{for } a<0,\ b>0,\\
[b'\ ;\ a']\times !\frac{u}{2}, & \text{for } a>0,\ b<0,\\
\left[a'+b'+\frac{u}{2}\ ;\ 0\right]\times\frac{u}{2}, & \text{for } a<0,\ b<0,
\end{cases}
\end{aligned}$$

where

$$a'=a+r_a\frac{u}{2}$$

and

$$b'=b+r_b\frac{u}{2},$$

and it is assumed that a and a' share the same sign, and similarly for b and b' (assumed satisfied if $a\neq 0$ and $b\neq 0$). But for $a<0$ and $b<0$, with

$$r_a=r_b=0,$$

we would expect the result $(ab,0)$ as interval

$$\mathcal{I}(ab,0)=ab+\left[0\ ;\ \frac{u}{2}\right],$$

to be in the appropriate interval defined by (5.3), which is NOT the case!

However, since negation of canonical (two's complement) RN-represented values can be obtained by constant-time bit inversion, multiplication of such operands can be realized by multiplication of the absolute values of the operands, the result being supplied with the correct sign by a conditional inversion.

Thus employing bitwise inversions, multiplication in two's complement RN-representations becomes equivalent to sign-magnitude multiplication; hence, assuming that both operands are nonnegative, the "interval product" is

$$
\begin{aligned}
&\mathcal{I}(a, r_a) \times \mathcal{I}(b, r_b) \\
&= \left[a + r_a \frac{u}{2};\; a + \frac{u}{2}(1 + r_a)\right] \times \left[b + r_b \frac{u}{2};\; b + \frac{u}{2}(1 + r_b)\right] \\
&= \left[\left(a + r_a \frac{u}{2}\right)\left(b + r_b \frac{u}{2}\right);\; \left(a + \frac{u}{2}(1 + r_a)\right)\left(b + \frac{u}{2}(1 + r_b)\right)\right] \\
&= \left(a + r_a \frac{u}{2}\right)\left(b + r_b \frac{u}{2}\right) + \left[0;\; \left(a + r_a \frac{u}{2}\right) + \left(b + r_b \frac{u}{2}\right) + \frac{u}{2}\right]\frac{u}{2} \\
&= ab + \left(ar_b + br_a + \frac{r_a r_b}{2}u\right)\frac{u}{2} + \left[0;\; a + b + (r_a + r_b + 1)\frac{u}{2}\right]\frac{u}{2}
\end{aligned}
$$

$$
= \begin{cases}
ab + \left[0;\; a + b + \frac{u}{2}\right]\frac{u}{2}, & \text{for } \mathbf{r_a = r_b = 0}, \\
ab + a\frac{u}{2} + [0;\; a + b + u]\frac{u}{2}, & \text{for } \mathbf{r_a = 0, r_b = 1}, \\
ab + b\frac{u}{2} + [0;\; a + b + u]\frac{u}{2}, & \text{for } \mathbf{r_a = 1, r_b = 0}, \\
ab + (a + b + \frac{u}{2})\frac{u}{2} + \left[0;\; a + b + \frac{3}{2}u\right]\frac{u}{2}, & \text{for } \mathbf{r_a = r_b = 1}.
\end{cases} \quad (4)
$$

It then follows that

$$\mathcal{I}((ab + (ar_b + br_a)u, r_a r_b) \subseteq \mathcal{I}((a, r_a) \times (b, r_b))$$

with unit u^2, since the left-hand RN-representation corresponds to the interval

$$\left[(ab + (ar_b + br_a)u) + (r_a r_b)\frac{u^2}{2};\; (ab + (ar_b + br_a)u) + (1 + r_a r_b)\frac{u^2}{2}\right]$$

and its lower endpoint is greater than or equal to the lower endpoint from (4):

$$ab + (ar_b + br_a)u + (r_a r_b)\frac{u^2}{2} \geq ab + \left(ar_b + br_a + \frac{r_a r_b}{2}u\right)\frac{u}{2}$$

together with the upper endpoint being smaller than or equal to that from (4):

$$
\begin{aligned}
&ab + (ar_b + br_a)u + (1 + r_a r_b)\frac{u^2}{2} \\
&\quad \leq ab + \left(ar_b + br_a + \frac{r_a r_b}{2}u\right)\frac{u}{2} + \left(a + b + (r_a + r_b + 1)\frac{u}{2}\right)\frac{u}{2}
\end{aligned}
$$

with both satisfied for $a \geq u$, $b \geq u$ (i.e., nonzero) and all permissible values of r_a, r_b.

Definition 10 (Multiplication). *If u is the unit in the last place, with $u \leq 1$, we define for nonnegative operands:*

$$(a, r_a) \times (b, r_b) = (ab + u(ar_b + br_a), r_a r_b),$$

and for general operands by appropriate sign inversions of the operands and thus the result. If $u < 1$, the unit is $u^2 < u$ and the result may often have to be rounded to unit u, which can be done by truncation.

For an implementation, some modifications to an unsigned multiplier will handle the r_a and r_b round bits, and we just have to calculate the double-length product with two additional rows consolidated into the partial product array. However, we shall not here go into the details of the consolidation.

The multiplier $(b + r_b u)$ may also be recoded into radix 2 (actually it is already so when interpreted as a signed-digit number) or into radix 4, and a term (bit) $d_i r_a$ may be added in the row below the row for the partial product $d_i a$, where d_i is the recoded ith digit of the multiplier. Hence, only at the very bottom of the array of partial products will there be a need for adding in an extra bit as a new row. The double-length product can be returned as (p, r_p), noticing that the unit now is $u' = u^2$, but the result may have to be rounded, which by simple truncation will define the rounded product as some (p', r'_p).

Example. As an example with $u = 1$:

	Not rounded	Canonical Representation
a	01011.1110	$(01011, 1)$
b	01001.1101	$(01001, 1)$
$a \times b$	01110100.10000110	$(01110111, 1)$

The multiplication in canonical representation was done according to the definition:

$$
\begin{aligned}
&ab + (ar_b + br_a) \\
&= 01100011 + (01011 + 01001) \\
&= 01100011 + 010100 = 01110111,
\end{aligned}
$$

where, we note that $(01110111, 1)$ corresponds to the interval:

$$[01110111.1\ ;\ 01111000.0]$$

which is clearly a subset of the interval

$$
\begin{aligned}
&[01011.1 \times 01001.1\ ;\ 01100 \times 01010] \\
&= [01101101.01\ ;\ 01111000.00].
\end{aligned}
$$

It is obvious that rounding results in larger errors when performing multiplication.

Similarly, some other arithmetic operations like squaring, square root, or even the evaluation of "well behaved" transcendental functions may be defined and implemented, just considering canonical RN-represented operands as two's complement values with a "carry-in" not yet absorbed, possibly using interval interpretation to define the resulting round bit.

6 Floating-Point Representations

For an implementation of a floating-point arithmetic unit (FPU), it is necessary to define a binary encoding, which, we assume, is based on the canonical two's complement for the encoding of the significand part (say s encoded in p bits, two's complement), supplied with the round bit (say r_s) and an exponent (say e in some biased binary encoding). It then seems natural to pack the three parts into a computer word (32, 64, or 128 bits) in the following order:

e	s	r_s

with the round bit in immediate continuation of the significand part, thus simplifying the rounding by truncation. As usual, we will require the value being represented to be in normalized form, say such that the radix point is between the first and second bit of the significand field. If the first bit is zero, the significand field then represents a fixed point value in the interval $\frac{1}{2} \leq s < 1$; if it is one, then $-1 \leq s < -\frac{1}{2}$.

We shall now sketch how the fundamental operations may be implemented on such floating-point RN-representations, not going into details on overflow, underflow, and exceptional values.

6.1 Multiplication

Since the exponents are handled separately, forming the product of the significands is precisely as described previously for fixed-point representations: sign-inverting negative operands, forming the double-length product, normalizing and rounding it, and possibly negating the result, supplying it with the proper sign.

Normalizing here may require a left shift, which is straightforward on the (positive) product before rounding by truncation.

6.2 Addition

In effective subtractions, after cancellation of leading digits, there is a need to left normalize; so a problem here is to consider what to shift in from the right. Thinking of the value as represented in signed digit, binary value, obviously zeroes, have to be shifted in.

In our encoding, say for a positive result (d, r_d) , we may have a two's complement bit pattern:

$$d \sim 0\ 0\ \cdots\ 0\ 1\ b_{k+2} \cdots b_{p-1} \text{ and round bit } r_d$$

to be left-normalized. Here, the least significant digit is encoded as $\{{}^{r_d}_{b_{p-1}}\}$.

It is then found that shifting in bits of value r_d will precisely achieve the effect of shifting in zeroes in the signed-digit interpretation:

$$2^k d \sim 0\ 1\ b_{k+2} \cdots b_{p-1} r_d \cdots r_d \text{ with round bit } r_d,$$

from $2 \times (x, r_x) = (x, r_x) + (x, r_x) = (2x + r_x u, r_x)$.

6.2.1 Subtraction, the "Near Case"

Addition is traditionally now handled in an FPU as two cases [8], where the "near case" is dealing with effective subtraction of operands whose exponents differ by no more than one. Here, a significant cancellation of leading digits may occur, and thus a variable amount of left normalization is required. As by the above, this left shifting is handled by shifting in copies of the round-bit.

6.2.2 Addition, the "Far Case"

The remaining cases dealt with are the "far case," where the result, at most, requires normalization by a single right or left shift. Otherwise, addition/subtraction takes place as for the similar operation in IEEE sign magnitude representation. There is no need, in general, to form the exact sum/difference when there is a great difference in exponents.

6.3 Division

As for multiplication, we assume that negative operands have been sign-inverted, and that exponents are treated separately.

Employing our interval interpretation, we must require the result of division of (x, r_x) by (y, r_y) to be in the interval:

$$\left[\frac{x + r_x \frac{u}{2}}{y + (1 + r_y)\frac{u}{2}} \; ; \; \frac{x + (1 + r_x)\frac{u}{2}}{y + r_y \frac{u}{2}}\right].$$

After some algebraic manipulations, it is found that the exact rational

$$q = \frac{x + r_x u}{y + r_y u}$$

belongs to that interval. Hence, any standard division algorithm may be used to develop an approximation to the value of q to $(p+1)$-bit precision, i.e., including the usual round-bit where the sign of the remainder may be used to determine if the exact result is just below or above the found approximation.

6.4 Discussion of Floating-Point Representations

As seen above, it is feasible to define a binary floating-point representation where the significand is encoded in the binary canonical two's complement encoding, together with the round-bit appended at the end of the encoding of the significand. An FPU implementation of the basic arithmetic operations is feasible at about the same complexity as the one based on the IEEE-754 standard for binary floating point, with a possible slight overhead in multiplication due to extra terms to be added. But, since the round-to-nearest functionality is achieved at much less hardware complexity, the arithmetic operations will generally be faster by avoiding the usual log-time rounding. The other (directed) roundings can also be realized at minimal cost. Benefits are obtained through faster rounding and sign inversion (both constant time); also note that the domain of representable values is symmetric.

7 Applications in Signal Processing

Let us consider here the fixed-point RN representations in high-speed digital signal processing applications, although there are similar benefits in floating point.

Two particular applications needing frequent roundings come to mind: calculation of inner products for filtering, and polynomial evaluations for approximation of standard functions. For the latter application, a very efficient way of evaluating a polynomial is to apply the *Horner Scheme*. Let $f(x) = \sum_{i=0}^{n} a_i x^i$ be such a polynomial approximation, then $f(x)$ is efficiently evaluated as

$$f(x) = (\cdots((a_n) * x + a_{n-1}) * x \cdots + a_1) * x + a_0,$$

where to avoid a growth in operand lengths, roundings are needed in each cycle of the algorithm, i.e., after each multiply-add operation. But here, the round-bits can easily be absorbed in a subsequent arithmetic operation, only at the very end a regular conversion may be needed; but normally the result is to be used in some calculation, hence a conversion may be avoided.

For inner product calculations, the most accurate result is obtained, if accumulation is performed in double precision; it will even be exact when performed in fixed-point arithmetic. However, if double precision is not available, it is essential that a fast and optimal rounding is employed during accumulation of the product terms.

8 Conclusions and Discussion

We have analyzed a general class of number representations for which truncation of a digit string yields the effect of rounding to nearest.

Concentrating on binary RN-represented operands, we have shown how a simple encoding, based on the ordinary two's complement representation, allows trivial (constant time) conversion from two's complement representation to the binary RN-representation. A simple parallel prefix (log time) algorithm is needed for conversion the other way. We have demonstrated how operands in this particular canonical encoding can be used at hardly any penalty in many standard calculations, e.g., addition and multiplication, with negation even being a constant-time operation, which often simplifies the implementation of arithmetic algorithms.

The particular feature of the RN-representation, that rounding-to-nearest is obtained by truncation, implies that repeated roundings ending in some precision yields the same result, as if a single rounding to that precision was performed. In [5] , it was proposed to attach some state information (2 bits) to a rounded result, allowing subsequent roundings to be performed in such a way that multiple roundings yield the same result as a single rounding to the same precision. It was shown that this property holds for any specific IEEE-754 [4] rounding mode, including, in particular, for the round-to-nearest-even mode. But these roundings may still require log-time incrementations, which are avoided with the proposed RN-representation.

The fixed-point encoding immediately allows for the definition of corresponding floating-point representations, which, in a comparable hardware FPU implementation, will be simpler and faster than an equivalent IEEE standard conforming implementation.

Thus, in applications, where many roundings are needed and conformance to the IEEE-754 standard is not required when employing the RN-representation, it is possible to avoid the penalty of intermediate log-time roundings. Signal processing may be an application area where specialized hardware (ASIC or FPGA) is often used anyway, and the RN-representation can provide faster arithmetic with round-to-nearest operations at reduced area and delay.

REFERENCES

[1] P. Kornerup and J.-M. Muller, "RN-Coding of Numbers: Definition and Some Properties," *Proc. Int'l Meeting on Automated Compliance Systems (IMACS '05)*, July 2005.

[2] A. Booth, "A Signed Binary Multiplication Technique," *Quarterly J. Mechanics and Applied Math.*, vol. 4, pp. 236-240, 1951.

[3] J.-L. Beuchat and J.-M. Muller, "Multiplication Algorithms for Radix-2 RN-Codings and Two's Complement Numbers," technical report, INRIA, http://www.inria.fr/rrrt/rr-5511.html, Feb. 2005.

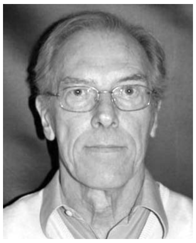
[4] IEEE, *IEEE Std. 754-2008 Standard for Floating-Point Arithmetic*, Aug. 2008.

[5] C. Lee, "Multistep Gradual Rounding," *IEEE Trans. Computers*, vol. 38, no. 4, pp. 595-600, Apr. 1989.

[6] A.M. Nielsen, D. Matula, C. Lyu, and G. Even, "An IEEE Compliant Floating-Point Adder that Conforms with the Pipelined Packet-Forwarding Paradigm," *IEEE Trans. Computers*, vol. 49, no. 1, pp. 33-47, Jan. 2000.

[7] R.E. Moore, *Interval Analysis.* Prentice Hall, 1963.

[8] P. Farmwald, "On the Design of High Performance Digital Arithmetic Units," PhD dissertation, Stanford Univ., Aug. 1981.



Peter Kornerup received the Mag. Scient. degree in mathematics from Aarhus University, Denmark, in 1967. After a period with the University Computing Center, from 1969, he was involved in establishing the computer science curriculum at Aarhus University; he helped found the Computer Science Department there in 1971 and served as its chairman until in 1988, when he became professor of computer science at Odense University, now the University of Southern Denmark. He has served on program committees for numerous IEEE, ACM, and other meetings. In particular, he has been on the program committees for the fourth through the 19th IEEE Symposium on Computer Arithmetic, and served as program cochair for these symposia in 1983, 1991, 1999, and 2007, respectively. He has been a guest editor for a number of journal special issues, and served as an associate editor of the *IEEE Transactions on Computers* from 1991 to 1995. He is a member of the IEEE and the IEEE Computer Society.

Jean-Michel Muller received the PhD degree in 1985 from the Institut National Polytechnique de Grenoble, France. He is directeur de recherches (senior researcher) at the Centre National de la Recherche Scientifique (CNRS), France, and is the former head of the Laboratoire de Informatique du Parallelisme (LIP laboratory, a joint laboratory of CNRS, Ecole Normale Supérieure de Lyon, INRIA, and Université Claude Bernard Lyon 1). His research interests are in computer arithmetic. He was coprogram chair of the 13th IEEE Symposium on Computer Arithmetic (Asilomar, June 1997), general chair of SCAN '97 (Lyon, France, September 1997), general chair of the 14th IEEE Symposium on Computer Arithmetic (Adelaide, Australia, April 1999). He is the author of several books, including *Elementary Functions, Algorithms and Implementation* (second edition, Birkhäuser Boston, 2006). He coordinated the writing of the *Handbook of Floating-Point Arithmetic* (Birkhäuser Boston, 2010). He served as an associate editor of the *IEEE Transactions on Computers* from 1996 to 2000. He is a senior member of the IEEE.

Adrien Panhaleux received the master's degree in 2008 from the École Normale Supérieure de Lyon, France. He is currently working toward the PhD degree at École Normale Supérieure de Lyon, under the supervision of Jean-Michel Muller and Nicolas Louvet.