

RN-Codings: New Insights and Some Applications

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 rounding to take place by a simple truncation, with the additional property that problems with double-roundings are avoided. In this paper we investigate a particular encoding of the binary representation, and conversions between the RN-Coding in this encoding and ordinary 2's complement representation. This encoding is essentially an ordinary 2's complement with an appended round-bit, but still allowing double-rounding without errors. Conversions from 2's complement to RN-Coding can be performed in constant time by the well-known Booth-recoding, whereas conversion the other way in general takes at least logarithmic time. A very fast parallel prefix algorithm for this conversion is defined and analyzed.

1 Introduction

In a recent paper [KM05] 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 $\{-1, 0, 1\}$ is an example. Another such representation is obtained by performing the original Booth-recoding [Boo51] on a 2's complement number into the digit set $\{-1, 0, 1\}$, where it is well-known that the non-zero digits of the recoded number alternate in sign.

We shall in Section 2 briefly from [KM05] cite some of the definitions and properties of the general RN-Codings/representations. However, we will here 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 accuracy as the two combined.

Section 3 then analyzes conversions between RN-Codings and 2'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, consisting of the bits of the truncated 2's complement encoding, with a round-bit appended.

Conversion the other way, from RN-Coding in this “canonical” encoding into 2's complement representation (essentially adding in the round-bit) is then shown to be realizable by a parallel prefix structure, which is then analyzed. Section 3 contains examples on some composite computations where fast and optimal roundings are useful, and may come for free when

RN-coding is employed. Although not based on these representations, the idea of forcing alternating signs in expansions is applied to a CORDIC algorithm [Vol59] as a sketch to be pursued. Section 4 then concludes with some examples of applications.

2 Definitions and Basic Properties (cited from [KM05])

Definition 1 (RN-codings) *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 $-\beta + 1 \leq d_i \leq \beta - 1$) is an RN-coding 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 $j \leq n$,

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

that is, if the digit sequence is truncated to the right at any position j , the obtained sequence is always the number of the form $d_n d_{n-1} d_{n-2} d_{n-3} \cdots d_j$ that is closest to x .

Hence, truncating the RN-coding 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 here restrict our discussions to finite representations. The following observations on such RN-Codings for general $\beta \geq 2$ are then easily found:

Observation 2 (Characterizations of finite RN-codings)

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

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

- if $\beta \geq 2$ is even then $D = d_m d_{m-1} \cdots d_\ell$ is an RN-coding iff

1. all digits have absolute value less than or equal to $\beta/2$;
2. if $|d_i| = \beta/2$, then the first non-zero digit that follows on the right has an opposite sign, that is, the largest $j < i$ such that $d_j \neq 0$ satisfies $d_i \times d_j < 0$.

Observe that for odd β the system is non-redundant, whereas for β even the system is redundant. In particular note that for radix 2 the digit set is $\{-1, 0, 1\}$, known by the names of “signed-digit” or “borrow-save”. Also, since here the non-zero digits all have absolute value one, their signs alternate. From now on we will only deal with radix 2 representations and algorithms for these.

An important property of the RN-coding is that no errors are introduced if repeated roundings take place, thus avoiding the *double rounding* problem with some roundings. It 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 . We repeat from [KM05] the following obvious result:

Observation 3 (Double rounding)

Let $\text{rn}_i(x)$ be the function that rounds the value of x to nearest at position i . Then for $k > j$, if x is represented in the RN-Coding, then

$$\text{rn}_k(x) = \text{rn}_k(\text{rn}_j(x))$$

3 Converting between RN-Coding and 2's Complement

3.1 Conversion from 2's Complement to RN-Coding

Consider an input value x in sign-extended 2's complement representation:

$$x = d_{m+1}d_m d_{m-1} \cdots d_{\ell+1}d_\ell \quad \text{where} \quad d_{m+1} = d_m$$

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

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

defined (by the Booth recoding) as

$$\delta_i = d_{i-1} - d_i \quad (\text{with } d_{\ell-1} = 0 \text{ by convention}) \quad (1)$$

is an RN-Coding 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 non-zero digits is easily seen by considering how strings of the form $01 \cdots 10$ and $10 \cdots 01$ converts.

Thus the conversion can be performed in constant time. Actually, the digits of the 2's complement representation directly provides for an encoding of the converted digits as a tuple: $\delta_i \sim (d_{i-1}, d_i)$ where

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

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

Example 1 Let $x = 110100110010$ be a sign-extended 2'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-Coding	0	$\bar{1}$	1	$\bar{1}$	0	1	0	$\bar{1}$	0	1	$\bar{1}$	0

where it is seen that in any column the two upper-most bits provide the encoding defined above of the signed-digit below in the column. \square

Note that by the definition, due to the sign extension, $\delta_{m+1} = 0$ and if d_k is the last nonzero digit of x then $\delta_k = -1$, confirmed in the example. Observe also that after the conversion the last non-zero digit is always $\bar{1}$ and thus unique. However, if an RN-Coded number is truncated for rounding somewhere, the resulting representation may have its last non-zero digit of value 1. But in this case the immediately preceding digit is either 0 or $\bar{1}$, and in both cases it is possible to rewrite these two digits, i.e., $01 \rightarrow 1\bar{1}$ or $\bar{1}1 \rightarrow 0\bar{1}$, such that the last non-zero digit is $\bar{1}$.

Hence there are exactly two finite binary RN-Codings of any non-zero binary number of the form $a2^k$ for integral a and k , but requiring a specific sign of the last non-zero 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 non-zero digit in the remaining digit string.

Example 2 Rounding the value of x in Example 1 by truncating off the two least significant digits we obtain

$\text{rn}_2(2x)$	1	0	1	0	0	1	1	0	0	1
$\text{rn}_2(x)$	1	1	0	1	0	0	1	1	0	0
$\text{rn}_2(x)$ in RN-Coding	0	$\bar{1}$	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 there is a tie-situation as here. \square

The example shows that there is a very compact encoding of RN-Coded numbers derived directly from the 2'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 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 have previously been pursued in [NMLE00] in a floating-point setting, denoted “packet-forwarding”.

Definition 4 (Canonical encoding)

Let the number x be given in sign-extended 2's complement representation as the bit string $b_{m+1}b_m \cdots b_{\ell+1}b_\ell$, with $b_{m+1} = b_m$, such that $x = \sum_{i=\ell}^m b_i 2^i - b_{m+1} 2^{m+1}$. Then the canonical encoding of the RN-Coded representation of x is defined as the pair

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

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

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

Note that this encoding is amenable to a straightforward way of performing arithmetic processing, since the round-bit may very often be taken into account at no additional cost, simply by using it as a carry-in to an adder together with the 2's complement component. The signed-digit interpretation is available in the encoding by pairing bits, (d_{i-1}, d_i) for $i > k$ and (r, d_k) , when truncated at position k . Obviously, the encoding then allows another rounding by truncation.

There are other equally compact encodings of RN-Coded 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) non-zero digit. Due to the alternating signs of the non-zero 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 RN-Coding to 2's Complement

The example of converting $0000000\bar{1}$ into its 2's complement equivalent 11111111 shows that it is not possible in to perform this conversion in constant time, 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-Coding is a special case of the (redundant) signed-digit representation, obviously this conversion is fundamentally equivalent to an addition.

If an RN-Coded number is in canonical encoding, conversion into 2's complement may require a non-zero round-bit to be added in, it simply consists in an incrementation for which traditional methods exists. But to complete the picture let us develop a parallel prefix type algorithm for adding in such a round-bit. Let the operand in canonical encoding be

$$x \sim (x_{m+1}, x_m, \dots, x_\ell, r),$$

then the propagate- and generate-bits for input to the trees are

$$p_i = x_i \text{ and } g_i = 0 \text{ for } i = \ell, \ell + 1, \dots, m, m + 1.$$

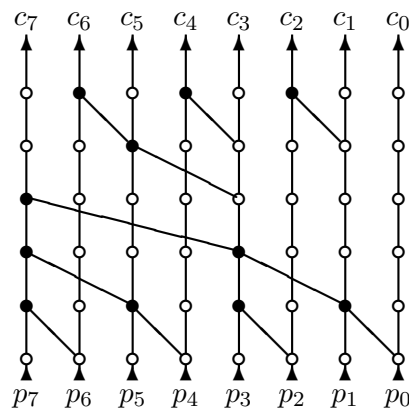
By the composition rule $(g, p) \circ (g', p') = (g + pg', pp')$, so all the generate-bits will be zero, and need not be calculated. Thus the nodes of the parallel prefix tree for calculating the carries will consist of AND-operators, with input at the leaves $p_i = x_i$. Let the output of the i^{th} tree then be the combined propagate-bits

$$P_i = \bigwedge_{k=\ell}^i p_k \text{ for } i = \ell, \ell + 1, \dots, m.$$

The actual carries can then be found as $c_\ell = r$ and $c_i = r P_{i-1}$ for $i > \ell$. However, this will require a broadcast of the round-bit to all positions. This can be avoided by changing the input p_ℓ to $p_\ell = r$ and shifting the rest over so that they are defined as $p_i = x_{i-1}$. The final output of the conversion is then found as

$$e_i = x_i \oplus c_i \text{ for } i = \ell, \ell + 1, \dots, m, m + 1.$$

The figure below shows parallel prefix trees following the structure suggested by Brent and Kung in [BK82] for calculating the carries of an 8 digit conversion. With $p_i = x_{i-1}$ for $i > 0$ and $p_0 = r$ as input at the bottom, and an AND operation in each of the black nodes, the carries appears at the top, to be XOR'ed with the input bits x_i .



Comparing the complexity of these parallel prefix trees with those of carry look-ahead trees for normal adders, it is found that the lengths of the longest paths in count of nodes through the trees are the same, but the nodes of the adder trees are more complex and slower than the nodes here.

4 Some Applications

We shall here start by looking at situations where the result of a rounding by truncation is to be utilized as an intermediate value subject to further processing.

The most obvious situation is where the result is to be added to something else, in which case the round-bit of a canonical encoding can be used as carry-in to a normal 2's complement adder, whether this is a redundant or a non-redundant adder. Note that adding two operands in canonical encoding, with result in the same encoding, can be performed by any kind of carry-completing 2's complement adder, taking one of the round-bits as carry-in to the addition, and leaving the other as the round-bit of the result.

For accumulation of many addends in canonical encoding, redundant adders as say carry-save adders directly apply. For each addition one round bit can be absorbed as carry-in, and the other kept as round bit of the result. At the end a final non-redundant result can be obtained by a carry-completing adder, absorbing the last round-bit.

In the case of an operand is to be subtracted, the inverted bit-pattern of the operand is fed to the adder together with the inverted round-bit as carry-in. Hence there is no need to convert from RN-Coding to regular 2's complement before any further additive processing.

The same applies if an RN-Coded value is to be used as the multiplier factor in a multiplication, and it is to be recoded to a higher radix, say 4 or 8, to reduce the height of the multiplier tree, or the number of cycles in an iterative multiplier. Here an RN-Coded operand directly recodes into such higher radices simply by grouping digits (which then also form RN-Codings).

4.1 Applications in Signal Processing

Although RN-Coding applies equally well to floating-point representations incorporating non-absorbed round-bits, as also suggested in [NMLE00], let us here think of use in high-speed fixed-point digital signal processing applications.

Two particular applications needing frequent rounding comes to mind here, calculation of inner products for filtering, and polynomial evaluations for approximation of standard functions. For the latter application, the most 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 most 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. 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 again 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.

4.2 An Extension to CORDIC algorithms

Although not based on RN-Codings, but along the same idea based on terms of alternating signs, we wish to decompose a number t as a sum

$$t = \sum_{i=0}^{\infty} d_i \arctan 2^{-i} \quad \text{for } d_i \in \{-1, 0, 1\}$$

(i.e., as with the conventional CORDIC with double rotation – to handle the zeros), but with the additional constraint that:

- $d_n = -1 \Rightarrow$ the smallest $k > n$ such that $d_k \neq 0$ satisfies $d_k = +1$;
- $d_n = +1 \Rightarrow$ the smallest $k > n$ such that $d_k \neq 0$ satisfies $d_k = -1$.

Let us denote

$$t_n = \sum_{i=0}^{n-1} d_i \arctan 2^{-i}.$$

Thanks to the additional constraint, the error bound when we approximate t by t_n is $\arctan 2^{-n}$ instead of the usual, approximately twice larger bound $\sum_{k=n}^{\infty} \arctan 2^{-k}$. This is the equivalent, with the “base” ($\arctan 2^{-i}$) of the radix-2 (i.e., “base 2^{-i} ”) RN-coding. To perform rotations, we can store the angle decomposed in that “base”, this will lead to more accurate rotations, provided we can easily compute the decomposition.

Lemma 5 For any $x \geq 0$,

$$\arctan x \leq 2 \arctan(x/2)$$

Proof: The result is immediate using the power series for $\arctan x$. □

Here is an algorithm for generating the decompositions. We assume that

$$|t| \leq \arctan 2^0 = \pi/4,$$

and assume we have already computed d_0, d_1, \dots, d_{n-1} , and $t_n = \sum_{i=0}^{n-1} d_i \arctan 2^{-i}$.

First case: $t_n \leq t$

Assume $t - t_n \leq \arctan 2^{-n}$, we will show by induction that this will be satisfied. Let k be the largest integer $\geq n$ such that

$$t_n + \arctan 2^{-k} \geq t.$$

we choose $d_k = 1$ and (if $n < k$) $d_n = d_{n+1} = \dots = d_{k-1} = 0$. We easily get

1. $t_{k+1} = t_n + \arctan 2^{-k} \geq t$
which implies that the next nonzero “digit” d_j , if any, will be -1 ;
2. $t_n + \arctan 2^{-k-1} \leq t$,
which implies $-\arctan 2^{-k} + \arctan 2^{-k-1} \leq t - t_{k+1}$.

Hence, from the lemma $-\arctan 2^{-k-1} \leq t - t_{k+1} \leq 0$. which corresponds to the induction hypothesis.

Second case: $t_n \geq t$

This case is symmetrical to the previous one. We assume

$$t_n - \arctan 2^{-n} \leq t$$

(induction hypothesis). Let k be the largest integer $\geq n$ such that this condition is satisfied. Choose $d_k = -1$ and (if $n < k$) $d_n = d_{n+1} = \dots = d_{k-1} = 0$, we then get as above:

1. $t_{k+1} \leq t$ (which implies that the next nonzero “digit” d_j , if any, will be +1);
2. $t_n - \arctan 2^{-k-1} \geq t$, which implies $0 \leq t - t_{k+1} \leq \arctan 2^{-k-1}$

which corresponds to the induction hypothesis.

5 Conclusions

Concentrating on binary RN-Coded operands we have shown how a simple encoding, based on the ordinary 2’s complement representation, allows trivial conversion from 2’s complement representation to RN-Coding, and a simple parallel prefix algorithm for conversion the other way. We have demonstrated how operands in RN-Coding can be used with hardly any penalty in many standard calculations, while allowing rounding by a simple truncation. Thus in applications where many roundings are needed, it is possible to avoid the penalty of many intermediate log-time roundings, at the expense of possibly only a single log-time rounding at the end. As a small divergence we have demonstrated how the idea of exploiting coefficients of alternating signs in CORDIC algorithms may be utilized, possibly an idea to be further explored.

References

- [BK82] R.P. Brent and H.T. Kung. A Regular Layout for Parallel Adders. *IEEE Transactions on Computers*, C-31(3):260–264, March 1982.
- [Boo51] A.D. Booth. A Signed Binary Multiplication Technique. *Q. J. Mech. Appl. Math.*, 4:236–240, 1951. Reprinted in [Swa80].
- [KM05] P. Kornerup and J.-M. Muller. RN-Coding of Numbers: Definition and some Properties. In *Proc. IMACS’2005*, July 2005. Paris.
- [NMLE00] A. Munk Nielsen, D.W. Matula, C.N. Lyu, and G. Even. An IEEE Compliant Floating-Point Adder that Conforms with the Pipelined Packet-Forwarding Paradigm. *IEEE Transactions on Computers*, 49(1):33–47, January 2000.
- [Vol59] J.E. Volder. The CORDIC Trigonometric Computing Technique. *IRE Transactions on Electronic Computers*, EC-8:330–334, 1959. Reprinted in [Swa80].
- [Swa80] Earl E. Swartzlander, editor. *Computer Arithmetic, Vol I*. Dowden, Hutchinson and Ross, Inc., 1980. Reprinted by IEEE Computer Society Press, 1990.