

Properties of two's complement floating point notations

Sylvie Boldo, Marc Daumas

Laboratoire de l'Informatique du Parallélisme UMR 5668 CNRS ENS de Lyon, INRIA France
e-mail: {Sylvie.Boldo, Marc.Daumas}@ENS-Lyon.Fr

Published online: 9 December 2003 – © Springer-Verlag 2003

Abstract. Few designs, mostly those of Texas Instruments, continue to use two's complement floating point units. Such units are simpler to build and to validate, but they do not comply to the dominant IEEE standard for floating point arithmetic. We compare some properties of the two systems in this text. Some features are lost, but others remain unchanged. One strong example is the case of Sterbenz's theorem and our recent extension. We show in the paper that the theorem and its extension hold for the two's complement architecture. Still, users should ensure that results are large enough on circuits that do not implement gradual underflow. Theorems have been proven and validated using the Coq automatic proof checker.

Keywords: Digital signal processing – Avionics – Formal proof

1 Introduction

Floating point arithmetic is dominated by the ANSI-IEEE 754 [35] and 854 [6] standards that use signed-magnitude notations. On the one hand, implementing common two's complement notations would save circuit area. On the other hand, most studies of floating point arithmetic have been done using the IEEE standard arithmetic. Moreover, signed-magnitude notations allow better treatment of exceptions with signed zeros. No general-purpose processor trades these advantages for a limited number of transistors.

However, DSP (digital signal processor) designers do care about area savings as their circuits are intended to be produced in huge amounts and should be priced as tightly as possible. This is the case for Texas Instruments' TMS 320C3x [36], which implements two's complement floating point arithmetic. Using two's complement is clearly

advantageous for hardware designers, and we will see in this work how the decision to use two's complement changes properties of floating point numbers from a user's point of view compared to signed-magnitude notations.

Because DSPs are simple and inexpensive to produce, they are a natural choice for mass production of consumer goods. But DSPs can also be integrated into very expensive jetliners. Military-grade processor SMJ 320C3x circuits should be preferred for avionics and military applications. In this case, designers expect that, since DSP architecture is simple, they will continue working correctly in situations where a general-purpose processor may fail.

They may be used as redundant devices of more elaborate processors or to disable the devices when suspect. This situation occurs in the flight control primary and secondary computers (FCPC/FCSC) [23]. From a user's point of view, it is critical that the intended hardware differences do not make too many changes to the implemented arithmetic.

This paper presents properties and proofs of two different options: (i) the dominant IEEE standard floating point arithmetic and (ii) the uncommon two's complement arithmetic implemented in DSPs. We have shown in [3] that a few very natural conditions of implemented floating point arithmetic are sufficient to narrow our focus down to these two options.

We were especially interested in the well-used Sterbenz's theorem [34]: given two floating point numbers x and y such that

$$\frac{y}{2} \leq x \leq 2y,$$

the implemented subtraction $x \ominus y$ is exact provided the system carries at least one guard bit. We have shown that we first have to check that the exact difference $x - y$ can be represented in the given floating point format. On IEEE standard floating point systems, Sterbenz's theo-

rem in its usual form is a straightforward consequence of our result. On other systems such as the Texas Instruments TMS 320C3x, we have to verify that the implementation actually computes the expected result.

As we compare properties of a very common model against a rarely studied one, it seems unavoidable to use automatic proof checking. Even careful proofs may erroneously refer to properties of the very common IEEE model since authors, reviewers, and readers are almost uniquely trained in writing proofs in this model. Past studies have shown that automatic proof checking should also be used to validate safety-critical systems [20, 30]. This also allows us to state very general results, especially concerning the radix used.

Section 2 explains the actual technology used to represent signed integers and how the Texas Instruments TMS 320C3x works. We relate our theorems to real hardware implementations with the flowchart of Texas Instruments' TMS 320C3x addition. Section 3 describes our formal approach to floating point numbers. Section 4 recalls a few key properties shared by both options. Section 5 extends conditions of Sterbenz's theorem. Section 6 concludes and proposes perspectives for further developments.

2 Two's complement implemented floating point arithmetic

2.1 Representing signed integers

Signed integers can be represented in a variety of ways. We immediately recall signed-magnitude and two's complement, as mantissas of floating point numbers are signed integers (or more exactly signed fixed point numbers).

For integer arithmetic, hardware designers and users have long agreed on two's complement arithmetic. A $p + 1$ bit signed number is represented by a bit s and a p bit unsigned binary component n . It is valued to

$$-s \times 2^p + n,$$

as opposed to

$$(-1)^s \times n$$

for the signed-magnitude interpretation. Such a choice remains open for floating point arithmetic.

With $p + 1$ bits, two's complement notation is able to represent all the integer numbers in the range

$$\{-2^p, \dots, 2^p - 1\}.$$

We divide the integer interpretation by 2^{p-1} to get the fixed point interpretation of manuals. Bits of n are numbered b_0 down to b_{1-p} . The two's complement notation means

$$-2 \times s + (b_0 . b_{-1} b_{-2} \cdots b_{2-p} b_{1-p})_2$$

and the signed-magnitude notation means

$$(-1)^s \times (b_0 . b_{-1} b_{-2} \cdots b_{2-p} b_{1-p})_2.$$

The discrete intervals of represented values are

$$\{-2, \dots, 2 - 2^{1-p}\}$$

and

$$\{-2 + 2^{1-p}, \dots, 2 - 2^{1-p}\}$$

with a step of 2^{1-p} .

In the signed-magnitude notation, a normalized mantissa is defined by $b_0 = 1$ and a denormal one by $b_0 = 0$. For the two's complement notation, a normalized mantissa is defined by $b_0 = \neg s$ (Boolean negation) and a denormal one by $b_0 = s$. The range of possible denormal mantissas is

$$\{-1, \dots, 1 - 2^{1-p}\}$$

and

$$\{-1 + 2^{1-p}, \dots, 1 - 2^{1-p}\}.$$

We deduce that a mantissa m is normalized if and only if $2 \times m$ exceeds the admissible range.

2.2 Texas Instruments' implementation of the subtraction

The formal conclusion of Theorem 6 of Sect. 5 states that under mild assumptions and provided

$$\frac{y}{2} \leq x \leq 2y \tag{1}$$

there exists a representable number that is the exact result of $x - y$. We have to look at the way the addition/subtraction is performed by TMS 320C3x to be sure that this exact result is really returned by the unit.

Figure 1 presents a simplified version of the addition flowchart. This operation is first performed on extended precision and then rounded.

The mantissa of one input is shifted down, and the length of the shift is given by the difference of exponents. Following (1), we know from Theorem 3 of Sect. 4.2 that this length is at most 1.

The result of the addition of the mantissas may use up to 32 bits. That means that 8 additional bits are available for the intermediate result. No bit is discarded during the shift, and the exact result is computed by the addition of the mantissa. These steps do not introduce any rounding error since, following Theorem 6 of Sect. 5 under the condition satisfied by Theorem 1 of Sect. 4.1, the result fits the 24-bit format.

We deduce immediately from (1) that

$$x - y \in [-y/2; y],$$

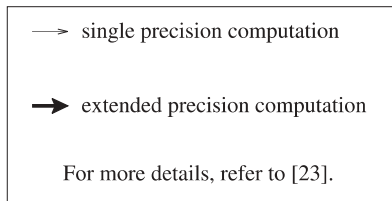
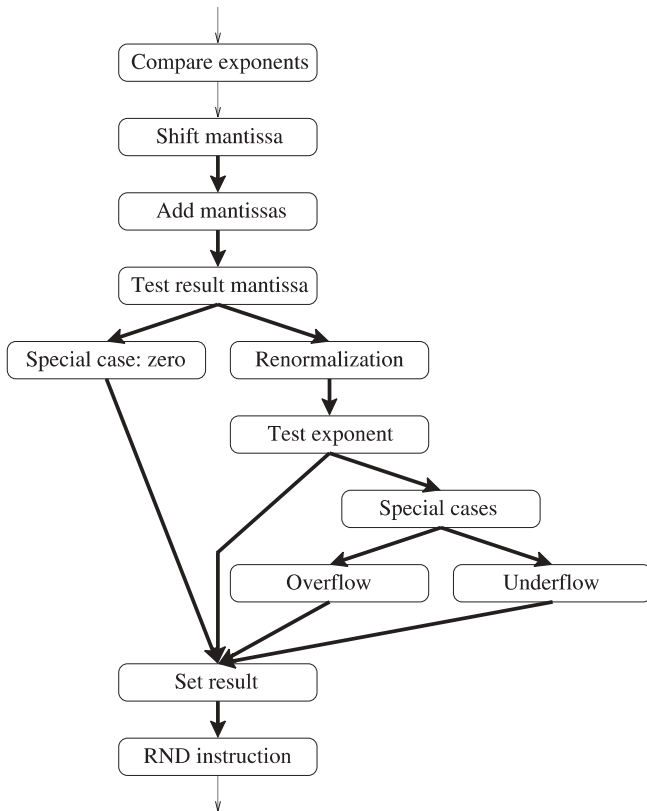


Fig. 1. Flowchart for floating point addition followed by a rounding instruction

and thus no overflow can occur. If the exact result is a denormal number, TMS 320C3x returns 0, as this processor does not handle such numbers. This ends the proof.

The subtraction is also correct when

$$2y \leq x \leq \frac{y}{2}.$$

The proof may abundantly use Theorem 1 of Sect. 4.1 and the above-mentioned theorems or we may replace Theorem 6 of Sect. 5 by Theorem 8 of Sect. 5.2, where $N_i = N_i^*$, $N_s = N_s^*$, and $E_i = E_i^*$ (hypotheses of Theorem 8).

Adding a new case to the result of [17], we see that one guard bit is sufficient for Sterbenz's theorem to hold using a different notation than the IEEE-like signed-magnitude notation for the mantissa. On the contrary, Sterbenz's theorem does not hold if users manipulate extended numbers rather than single precision numbers. In this case,

the operation is performed without any guard bit and the result is not necessarily found by the floating point unit.

3 Formal approach to floating point numbers

3.1 Fallacies and pitfalls

Setting up a formal specification of floating point arithmetic is difficult, as it should be correct, complete, and useful. The literature shows that none of these pitfalls should be regarded as easily avoided.

The first formal specification on floating point arithmetic can be dated back to 1989 [2]. It focuses on the IEEE 754 standard for binary floating point arithmetic and tries to illustrate the advantages to be gained by using a nonalgorithmic formalism such as Z in the specification. The author later uses the formalization to prove the correctness of Ocaml procedures to perform the nonexceptional arithmetic functions and mimic the INMOS T800 transputer's floating point hardware.

There was a transcription error in the formal specification recalled by Professor William Kahan at the July 2002 meeting of the IEEE 754R group in Cupertino, CA.¹ Somewhere in the specification, the most negative binary integer was not correct, so the routing in add/subtract operations went the wrong way when signed zeros were used.

Because this first specification did not fully implement separate overflow, divide-by-zero, and invalid exceptions, it is not complete. Academic specifications such as ours tend to remain incomplete as they are used by specialists to explore properties. Nevertheless, only fully complete specifications should be used to validate implementations. The flag error of the integer conversion first detected by a user with a Pentium Pro [7] had not been detected earlier because specifications used at Intel were not complete [5, 27].

Theorem provers have long been used to mechanically check the correctness of floating point algorithms with PVS [19, 26], HOL [16], or ACL2 [31], and one specification has been motivated by avionic application [4]. Guaranteeing the completeness of a specification with respect to a text in a natural language is probably beyond the scope of formal proof. A high-level specification using higher-order logic tools is easier to review than specifications limited to more elementary logic and objects.

For these reasons, all our results have been developed and validated using Coq proof assistant [18]. Coq is a theorem checking system based on the Curry–Howard isomorphism. Systems like Coq allow users to define new objects and to derive consequences of these definitions formally while checking every detail. Coq is based on higher-order logic; therefore, it is possible to state properties in their most general form. For example, universal

¹ <http://grouper.ieee.org/groups/754/>.

quantification has been used to state properties that are true for any rounding mode. Higher logic is needed as rounding modes are defined to be arbitrary properties between real values and floating point numbers.

Proofs are built interactively using high-level tactics that may solve some “easy” subgoals. We used `pcocq` [1], a working environment for Coq with a nice graphical interface and a “pretty” printer.

At the end of each proof, Coq records a proof object that contains all the details of derivation and ensures that the theorem is valid. These objects can be double-checked for life-critical applications by tools such as `BindLib`, a program designed independently of Coq developers.

All our proofs can be downloaded on the Internet at

<http://perso.ens-lyon.fr/sylvie.boldo/coq>.

They are based on early developments of the floating point library [12] available at

<http://www-sop.inria.fr/lemme/AOC/>.

We describe next our abstract data type for floating point arithmetic. As stated earlier, a specification should be as concise as possible to remain trusted. The representations are abstracted to generic integers to work with simple definitions. Parameters are later tuned exactly to fit the different number representations presented above.

3.2 Abstract definition of floating point numbers

A generic floating point number is a pair of integers (n, e) . It is mapped from \mathbb{Z}^2 onto \mathbb{R} by

$$(n, e) \mapsto n \times \beta^e,$$

where β , the **radix** of the floating point system, is a constant integer strictly greater than one.² Later n will be called the mantissa and e the exponent.

`Float` is defined below in Coq:

```
Record float : Set := Float {
  Fnum: Z;
  Fexp: Z }.
```

The value of one element is obtained by using the constant parameter β and the `FtoR` function corresponding to \mapsto :

```
Definition FtoR := [x : float]
  (Rmult (Fnum x)
    (powerRZ (IZR radix) (Fexp x))).
```

Two pairs are **equivalent** if they are mapped to the same real value. This equality will be denoted by $=_{\mathbb{R}}$. Coq files are barely understandable to nonusers. Theorems and definitions will be presented using a “pretty” printer. Below is an example obtained from the previous definition:

² The floating point radix is not necessarily related to the integer radix of Sect. 2.1.

Definition 1.

$$FtoR := x : float \mapsto Fnum(x) \times \beta^{Fexp(x)}.$$

Quantities treated by computers must fit into finite fields; we focus our interest on pairs (n, e) such that n and e are bounded. For practical reasons, we do not use an upper bound on the exponent and overflow is not yet defined in our formalism. A **representable** floating point pair is such that

$$n \in \{-N_i, \dots, N_s\} \quad \text{and} \quad e \geq -E_i,$$

where N_i and N_s are set according to the description of Sect. 2.1. So $N_i = N_s = \gamma^p - 1$ corresponds to IEEE 754-like behavior, i.e., signed-magnitude p -digit mantissa. The case where $N_i = N_s + 1 = 2^p$ corresponds to two's complement implementation. The IEEE standards set

$$E_i = 2^{l-1} + p - 3$$

for an l bit exponent field, that is, $E_i = 149$ for single precision and $E_i = 1074$ for double precision. Texas Instruments' TMS 320C3x uses $E_i = 148$ for single precision and $E_i = 156$ for extended precision.

The definition of representable pairs in Coq is given next.

Definition 2. $FboundedI := b : FboundI, d : float \mapsto$

$$\begin{aligned} &(-vNumInf(b) \leq Fnum(d)) \\ &\wedge (Fnum(d) \leq vNumSup(b)) \\ &\wedge (-dIExp(b) \leq Fexp(d)). \end{aligned}$$

Unless explicitly specified, the following properties hold for any radix and any bounds on the mantissa and the exponent.

3.3 Multiple representations

Like the IEEE standard on extended formats, our proposed library possibly defines many representable pairs with the same value. For example, the three radix two pairs

$$(1100_2, 4)_2, (110_2, 5)_2, \text{ and } (11_2, 6)_2$$

share the same real value $3 \times 2^6 = 192$. This fact can be disturbing as one real value can be associated with many different representable pairs that do not share the same properties.

To retain common floating point behavior, we define a canonical pair for each representable pair. This pair is meant to represent the actual fields stored in a computer that are associated with the number. A pair is **normal** if it is representable and its exponent cannot be reduced by multiplying the mantissa by the radix, that is to say

$$n \times \beta \notin \{-N_i, \dots, N_s\}.$$

A pair is **denormal** if it is representable and the exponent reduction is impossible because the minimal accepted exponent is already used, despite the mantissa being small enough to be multiplied by the radix. That is,

$$n \times \beta \in \{-N_i, \dots, N_s\} \quad \text{and} \quad e = -E_i.$$

Any representable pair is equivalent to one unique pair, either normal or denormal. The unique pair is called the **machine** pair. Few theorems are needed to arrive at this conclusion.

The first one, `FcanonicalUnique`, states that if p and q are two machine pairs such that $p =_{\mathbb{R}} q$, then p and q are syntactically equal (Leibniz's equality). Other theorems prove the correctness of `FnormalizeI`, the function defined below, in constructing the machine representation from any representable pair:

```
Fixpoint FNIAux [v, N, q : nat] : nat :=
Cases q of
0 => 0
| (S q') =>
Cases
(Zcompare (Zmult (Zpower_nat radix q') v)
(Zmult radix N)) of
INFERIEUR => q'
| EGAL => q'
| _ => (FNIAux v N q')
end
end.
```

Definition FNI :=
[q, N : nat] (pred (FNIAux q N (S (S N)))).

```
Definition FnormalizeI :=
[b : FboundI] [p : float]
Cases (Zcompare ZERO (Fnum p)) of
EGAL => (Float ZERO (Zopp (dIExp b)))
| INFERIEUR => (Fshift radix (min
(FNI (absolu (Fnum p)) (vNumSup b))
(absolu (Zplus (Fexp p) (dIExp b)))) p)
| SUPERIEUR => (Fshift radix (min
(FNI (absolu (Fnum p)) (vNumInf b))
(absolu (Zplus (Fexp p) (dIExp b)))) p)
end.
```

Expressing that the function is correct means that (i) if p is representable, then the result `FnormalizeI(p)` is representable (`FnormalizeIBounded`). It also means that (ii) the result is a machine pair (`FnormalizeIFcanonicalI`) such that (iii) the input pair p and the result pair are mapped to the same real value, which is to say $p =_{\mathbb{R}} \text{FnormalizeI}(p)$ (`FnormalizeICorrect`). We omit these proofs as they are quite cumbersome, though not difficult.

The number system that we have just defined handles denormal numbers (gradual underflow) as this helps write more robust codes [13]. Sterbenz's theorem, which will be presented in Sect. 5, cannot be true if denormal numbers are not allowed. Let λ be the lowest positive normal num-

ber. Its value is

$$\lambda = \left(\left\lfloor \frac{N_s}{\beta} \right\rfloor + 1 \right) \times \beta^{-E_i}$$

and the following floating point number is

$$\lambda^+ = \left(\left\lfloor \frac{N_s}{\beta} \right\rfloor + 2 \right) \times \beta^{-E_i}.$$

The quantities λ and λ^+ satisfy $\lambda^+/2 \leq \lambda \leq 2\lambda^+$ and

$$\lambda^+ - \lambda = \beta^{-E_i},$$

which is a denormal number. This example shows that, without allowing denormal numbers, the subtraction of x and y under the conditions of Sterbenz's theorem may not be represented.

4 Key properties

In this section, we address “basic” questions on number systems with radix-independent theorems.

4.1 Negating a number

A representable pair p can be negated if there exists another representable pair q such that $q =_{\mathbb{R}} -p$. On IEEE-compliant computers, the mantissa is stored with separate sign and magnitude. It corresponds to $N_i = N_s$, and so it follows that any number can be easily negated by flipping the sign bit.

This fact is not necessarily true on all floating point systems. The following theorem, checked with Coq (`FoppBoundedI`, `FoppBoundedI2`, and `FoppBoundedIInv`), answers the question for two's complement notation.

Theorem 1. *On a floating point system bounded by N_i , N_s , and E_i with $N_i \neq N_s$, any representable pair can be negated to a representable pair if and only if*

$$|N_i - N_s| = 1 \quad \text{and} \quad \beta \mid \max(N_i, N_s).$$

Proof. Without loss of generality, we assume that $N_s > N_i$. As a consequence, any pair (n, e) with

$$n \in \{-N_i, \dots, N_i\}$$

can be easily negated into a representable pair by negating its mantissa. The pairs (n, e) with

$$n \in \{N_i + 1, \dots, N_s\}$$

can only be negated by manipulating the exponent. Therefore, β should divide all the $n \in \{N_i + 1, \dots, N_s\}$. That is possible only for $N_i + 1 = N_s$ if β divides N_s .

On the contrary, if N_s is a multiple of β and $N_i = N_s - 1$, any representable pair can be negated to another representable pair. \square

The case studies for a system that does not handle denormal numbers and for the upper bound on the exponent are treated separately with theorem `FoppBoundedIExp`, omitted here. Only two numbers cannot be negated with TMS 320C3x two's complement notation. Negating these two numbers causes either

overflow: the opposite of the formal machine pair just above the positive overflow threshold can be represented, but its negation causes an overflow; or

underflow: the positive underflow threshold is representable but it cannot be negated as denormals are not implemented on TMS 320C3x.

The next theorem is used to prove that the negation is the only opposite on a system that handles denormal numbers, i.e., if the rounding of $x + y$ is 0, then y is the negation of x . Rephrasing [22], we prove in `OppositeIUnique` that the distance between two different floating point numbers is at least β^{-E_i} .

Theorem 2. *On a floating point system bounded by N_i , N_s , and E_i with an arbitrary rounding mode, let x and y be two representable pairs. If $x + y \neq_{\mathbb{R}} 0$ and z is a rounded result of $x + y$, then $|z| \geq \beta^{-E_i}$.*

Proof. We first restrict the proof to the case where $-x < y$, that is, $x + y > 0$. As x and y are representable, there exist two integers m_x and m_y such that

$$x = m_x \times \beta^{-E_i} \quad \text{and} \quad y = m_y \times \beta^{-E_i}.$$

It follows that $x + y = (m_x + m_y) \times \beta^{-E_i}$. As $x + y > 0$, we deduce that $m_x + m_y > 0$. Since $m_x + m_y$ is an integer, $m_x + m_y \geq 1$ and $x + y \geq \beta^{-E_i}$.

The active rounding mode is monotonic. Since $x + y \geq \beta^{-E_i}$, the rounded value of $x + y$ is greater than the rounded value of β^{-E_i} . The first quantity is z , and the second one is β^{-E_i} , as a representable pair is not changed by rounding. We conclude that $|z| = z \geq \beta^{-E_i}$.

The symmetric case where $y < -x$ is handled in the same way. \square

We conclude, as in [3], that we prefer to use signed-magnitude or two's complement as opposed to an arbitrary notation.

4.2 Lexicographic order

Many authors, including [8], have recognized that it is a nice feature to have lexicographic order of positive machine representations coincide with the order of represented real values. As this fact is not necessarily trivial in a generic floating point system, we establish the two following theorems based on `LexicoPosCanI`, `LexicoNegCanI`, and `LexicoCanI`. The first one will be used in the proof of our first generalization of Sterbenz's theorem.

Theorem 3. *On a floating point system bounded by N_i , N_s , and E_i , for any **machine** pair (n_x, e_x) representing x*

and any representable pair (n_y, e_y) representing y ,

$$|x| \leq |y| \quad \text{and} \quad xy \geq 0 \quad \text{implies} \quad e_x \leq e_y.$$

Proof. We assume that $x \geq 0$. The other case is handled similarly. As (n_x, e_x) is a machine pair, it is either normal or subnormal. If it is subnormal, then $e_x = -E_i$ and, as (n_y, e_y) is representable, $e_x = -E_i \leq e_y$.

We can now consider that (n_x, e_x) is normal. This means that $n_x \times \beta \notin \{-N_i, \dots, N_s\}$. As $x \geq 0$, $n_x \geq 0$, and thus $n_x \times \beta > N_s$.

Let us assume that $e_x > e_y$, thus $e_x - 1 \geq e_y$. We will deduce that $y < x$, which is absurd: $y =_{\mathbb{R}} n_y \times \beta^{e_y} \leq N_s \times \beta^{e_x - 1} < n_x \times \beta \times \beta^{e_x - 1} =_{\mathbb{R}} x$. \square

Theorem 4. *On a floating point system bounded by N_i , N_s , and E_i with $|N_i - N_s| \leq 1$, for any **machine** pair (n_x, e_x) representing x and any representable pair (n_y, e_y) representing y ,*

$$|x| < |y| \quad \text{implies} \quad e_x \leq e_y.$$

The difference between the preceding theorems and the usual IEEE-like situation arises from the fact that the magnitude of a floating point number may not be represented or may use another exponent. We establish the following corollary.

Corollary 1. *On a floating point system bounded by N_i , N_s , and E_i with $|N_i - N_s| \leq 1$, for any machine pair (n_x, e_x) representing x and any representable pair (n_y, e_y) representing y ,*

$$e_x < e_y \quad \text{implies} \quad |x| \leq |y|.$$

When $|N_i - N_s| > 1$, we have a very different behavior. Here is an example that also shows that the bound on the difference $N_i - N_s$ is tight. We define a binary notation with mantissas between -1001_2 and 111_2 . The pairs $(100_2, 1)_2$ and $(-1001_2, 0)_2$ are machine pairs, yet their magnitudes and their exponents are not in the same order. This cannot happen in IEEE-compliant systems or on the TMS 320C3x.

5 Sterbenz's theorem

In 1974, Sterbenz [34] presented a theorem about the exact subtraction of two floating point numbers x and y when they are sufficiently close to one another, that is,

$$\frac{y}{2} \leq x \leq 2y.$$

The theorem stating that $x \ominus y$ is exact under the preceding condition was presented for any radix provided the hardware was accurate enough. Later authors [14, 17] presented similar results with an emphasis on didactic aspects. We have seen in Sect. 2.2 that Sterbenz's theorem

can also be used with systems that provide an extended precision for intermediate results such as the Texas Instruments TMS 320C3x.

More recently [15, 24], some authors have proposed to extend Sterbenz's theorem to machines with a fused multiply and accumulate operation such as Intel IA 64 that can be used to implement accurate approximations to elementary functions [25]. Following the proof of Sect. 2.2 this extension can also be used with the Texas Instruments TMS 320C3x as long as one guard bit is still available in the extended precision.

It was recognized in [12, 29] that Sterbenz's theorem is not a property of hardware but rather a property of the floating point representation. Given x and y , the question is to know whether or not $x - y$ can be represented. This is clearly a key condition, necessary for the implemented floating point subtraction $x \ominus y$ to return the exact result $x - y$.

With IEEE-like behavior, any floating point operation is reduced to two steps. An intermediate result is first computed to sufficient accuracy and then rounded. Designers must guarantee that the system always returns the result as if the infinitely precise mathematical operations were rounded. For example, the subtraction is implemented as the composition of two mathematical functions, namely, the subtraction ($-$) and the user-specified rounding function (\circ):

$$x \ominus y = \circ(x - y).$$

The details of the implementation are not relevant to users since knowing the rounding function is sufficient to deduce the value returned by any operation. Users usually expect the rounding function to be a monotonic (nondecreasing) projection of the real numbers over the set of the machine floating point pairs. The latest property implies that for any floating point number v

$$\circ(v) = v.$$

Therefore, establishing Sterbenz's equality does not require any knowledge about the rounding function provided it is a projection.

We present in this section the behavior of generic floating point systems in regard to Sterbenz's theorems. Unfortunately, the theorem is not true for all floating point notations. For example, let the radix be 2 and the format such that mantissas are between -1100_2 and 1111_2 . Let x be $(1111_2, 0)$ and y be $(1110_2, 1)$. Both x and y are representable such that $\frac{y}{2} \leq x \leq 2 \times y$ but $x - y$ is -1101_2 , and this value cannot be represented exactly. We will later give a list of additional conditions for the assertion to be true.

5.1 Extending Sterbenz's theorem to other notations

It is amazing to realize that the following theorem is true whatever the radix and the bounds N_i and N_s . Moreover, the proof has been upgraded automatically by the

Coq proof checker from the previous **SterbenzAux** proof presented in [12] that was supposed to work only when $N_i = N_s$.

Theorem 5. *On a floating point system bounded by N_i , N_s , and E_i with no assumption of a relation between N_i and N_s , for any representable pairs (n_x, e_x) and (n_y, e_y) representing x and y such that*

$$y \leq x \leq 2y,$$

the difference $x - y$ is representable. Furthermore, $x - y$ can be represented by (n, e) defined as

$$\begin{aligned} n &= n_x \beta^{e_x - \min(e_x, e_y)} - n_y \beta^{e_y - \min(e_x, e_y)} \\ e &= \min(e_x, e_y). \end{aligned}$$

The proof is omitted here. It is very close to the first case of the proof of Theorem 8 presented later.

On a floating point system where any representable pair can be negated without rounding such as presented in Sect. 4.1, Sterbenz's theorem stated below (**SterbenzOppI**) is proved by applying Theorem 5 twice.

Theorem 6. *On a floating point system bounded by N_i , N_s , and E_i where any representable pair can be negated into another representable pair, for any representable pairs x and y such that*

$$\frac{y}{2} \leq x \leq 2y,$$

the difference $x - y$ can be represented.

Proof. We prove the theorem correct when $y/2 \leq x \leq y$ by applying Theorem 5 to $X = y$ and $Y = x$ so that $X - Y = -(y - x)$ is a representable pair. \square

Although we demonstrated in Sect. 4 that it is most desirable to use a number system with a few natural properties including the fact that every representable pair can be negated without rounding, we present now a generic theorem (**SterbenzI**) valid for any representation. The details of the proof are available on the Internet.

Theorem 7. *On a floating point system bounded by N_i , N_s , and E_i where $|N_i - N_s| \leq \delta$, for any machine pair (n_x, e_x) representing x and any representable pair (n_y, e_y) representing y such that*

$$\frac{y + \delta \beta^{\min(e_x, e_y)}}{2} \leq x \leq 2y,$$

the difference $x - y$ can be represented.

5.2 Sterbenz's theorem on catastrophic cancellations

All processors and many DSPs provide different floating point precisions. For example, the IEEE 754 standard provides 32-bit **single** and 64-bit **double** precisions. Intel IA32 architectures also provide 80-bit **double**

extended precision, and some workstations implement 128-bit quad precision. The Texas Instruments TMS 320C3x provides 16-bit short, 32-bit single, and 40-bit extended precisions.

Extended precision can be used to compute two quantities x and y close to each other that will be subtracted. The accumulated relative errors of x and y are magnified by each digit cancelled in $x \ominus y$ [11]. If the subtraction cancels l bits, we need an accuracy of $p+l$ bits on x and y to produce a p bit accurate result.

Theorem 8 (**SterbenzApproxI**) states that if x and y are sufficiently close, $x - y$ can be stored to a reduced p bit precision format without introducing a new rounding error. This theorem continues work from published theorems [15, 24].

Theorem 8. *Let the working precision be bounded by N_i , N_s , and E_i and the alternate precision be bounded by N_i^* , N_s^* , and E_i^* . We assume that $-E_i \leq -E_i^*$ and that for each precision any representable pair can be negated without rounding with the respective bounds. Let*

$$\gamma = \frac{\max(N_i, N_s) + 1}{\min(N_i^*, N_s^*) + 1}.$$

For all representable alternate precision pairs x and y such that

$$\frac{1}{1+\gamma}|y| \leq |x| \leq (1+\gamma)|y| \quad \text{and} \quad xy \geq 0,$$

the difference $x - y$ can be represented with the working precision.

Proof. Parameter γ can be expanded to

$$\gamma = \max\left(\frac{N_i+1}{N_i^*+1}, \frac{N_s+1}{N_s^*+1}, \frac{N_s+1}{N_i^*+1}, \frac{N_i+1}{N_s^*+1}\right).$$

Each value considered for γ corresponds to one case of the theorem. We will present here the proof for the case where

$$\gamma = \frac{N_s+1}{N_s^*+1}$$

with the additional condition that x and y are two alternate precision representable pairs such that

$$0 \leq y \leq x \leq (1+\gamma)y.$$

Let y' be the machine representation of y . We define the pair (n_u, e_u) representing $u =_{\mathbb{R}} x - y' =_{\mathbb{R}} x - y$ from its components:

$$e_u = \min(e_x, e_{y'}) \\ n_u = n_x \beta^{e_x - \min(e_x, e_{y'})} - n_{y'} \beta^{e_{y'} - \min(e_x, e_{y'})}.$$

As $0 \leq y' \leq x$ and y' is a machine representation, we are sure that $e_{y'} \leq e_x$ from Theorem 3 and $e_u = e_{y'}$. As y' is representable with alternate precision, we know from

hypotheses that $e_u \geq -E_i^* \geq -E_i$. As $y' \leq x$, we have $u \geq 0$ and $n_u \geq 0$, thus we easily check that $-N_i \leq 0 \leq n_u$.

The last assertion to prove is that $n_u \leq N_s$. As they are both integers, it is equivalent to proving that $n_u < N_s + 1$:

$$\begin{aligned} n_u &= u \times \beta^{-e_u} \\ &= (x - y') \times \beta^{-e_{y'}} \\ &\leq ((1+\gamma)y - y') \times \beta^{-e_{y'}} \\ &= y' \times \beta^{-e_{y'}} \times \gamma \\ &= n_{y'} \times \gamma \\ &< (N_s^* + 1) \times \gamma = N_s + 1. \end{aligned}$$

To conclude the proof, the four subcases must be completed separately, and then sign symmetries are used between the cases to finish the theorem. \square

The preceding theorem can be used in two ways. It gives a condition for switching from alternate precision to working precision without introducing any rounding error. It also can be used to bound the numbers of digits of $x - y$ given the ratio between numbers x and y .

Nowhere in the theorem or in the proof is it mentioned that alternate precision should be extended compared to working precision. We can use γ larger than one to bound the precision of $x - y$ without error where target precision is extended from the precision of x and y . Using our theorem would reduce the number of additional digits by one compared to the usual technique.

5.3 Concluding on representable pairs

Texas Instruments uses in its TMS 320C3x the two's complement notation for mantissas. This notation was also in use in the Honeywell 6080N computer [32]. A different notation with the same mantissa range is studied in an exercise of [21]. This number system is well suited as all the natural properties (stability through negation, existence and uniqueness of an opposite, and lexicographic order of the pairs) are still true and Sterbenz's theorem holds. We are unaware of any working floating point unit that does not use either signed-magnitude or the two's complement notation for the mantissa encoding.

The following theorem (**ReductRange** and **ReductRangeInv**) can be used to deduce that the set of the represented numbers is almost identical to an IEEE-compatible unit and to TMS 320C3x. Should Texas Instruments decide to implement denormal pairs and precise rounding, the unit could be almost functionally IEEE compliant.

Theorem 9. *The set of real numbers represented on a floating point system bounded by $N_i > 1$, $N_s > 1$, and E_i is identical to the set of numbers represented on a system bounded by N_i , $N_s - 1$, and E_i (resp. $N_i - 1$, N_s , and E_i) if and only if*

$$\beta \mid N_s \quad (\text{resp.} \quad \beta \mid N_i).$$

A side effect of this work is the latest extension of Sterbenz's theorem on signed-magnitude implementations. It uses the preceding theorem and is validated in Coq (`SterbenzApprox`).

Theorem 10. *Let the working precision use p digits for the mantissa and the exponent be bounded by E_i , and let the alternate precision use $p+l$ digits for the mantissa and the exponent be bounded by E_i^* . We assume that $-E_i \leq -E_i^*$ and that both precisions use signed-magnitude notation. For all representable alternate precision pairs x and y such that*

$$\frac{1}{1 + \beta^{-l} + \beta^{-p-l}} |y| \leq |x| \leq (1 + \beta^{-l} + \beta^{-p-l}) |y|$$

and $xy \geq 0$, the difference $x - y$ can be represented with the working precision.

NB: l does not have to be positive. It is a signed integer.

6 Conclusion and perspectives

Most general-purpose widely-available processors use a signed-magnitude representation. Some books [17, 28] even present the signed-magnitude notation as the natural floating point notation. This notation is in use in well-studied IEEE-754 compliant hardware. Some designs use radix 10 [9], and several of them retain a radix 16 compatibility mode [33]. Yet most systems use radix 2. The “basic” properties and Sterbenz's theorem were scattered in the literature, as most of them had been published over time. These properties hold for all these notations, and proofs have been checked in a radix generic formalism.

We have shown that the floating point number system used for TMS 320C3x is very close to the one defined by an IEEE-compliant processor with a very different interpretation for the mantissa field. We have also shown that gradual underflow and correct rounding would be very sensible in such a system, although neither was implemented. Finally, we have adapted some very useful results such as Sterbenz's theorem, provided no underflow occurs, to two's complement and to the TMS 320C3x.

The main motivation of this work was to compare in a generic formalism (i) the most common behavior, namely, binary IEEE-754, with other implementations such as (ii) radix-independent IEEE 854-compatible circuits, (iii) two's complement almost-IEEE 854 behavior, and (iv) non-IEEE behavior. We succeeded and even contributed new results for IEEE 754 architectures in Theorem 10.

Without a strong incentive for formal analysis of the TMS 320C3x, such work would probably not have been carried out. It has been made possible by the very formal and generic development of the proofs used in Coq. Our conclusions would likely not be trusted had they not been

checked by an automatic proof checker since the proofs are very technical and prone to many small mistakes that would not have been ruled out by experimental knowledge.

We will continue to investigate natural properties of floating point number systems as they lead us to necessary conditions on number systems. For example, Sterbenz's theorem and the possibility of negating a number are also key in the analysis of numerical software behavior such as [10, 24].

As mentioned Sect. 3, our formal development handles neither exceptions nor nonnumeric quantities such as infinity (possibly signed), Not-A-Number (possibly quiet or signalling), and negative zero. An abstract definition focusing on represented values should not include signed zeros as they represent the same value but have different implemented arithmetic properties. In our view of the IEEE standard, negative zero is clearly a nonnumeric value with specific rules for production and treatment.

Nonnumeric quantities will certainly be part of a future superset of `float`, but setting up a system that does not need them helped us reduce the number of peculiar cases in our proofs. These questions will be treated in our formal connection with low- and high-level programming languages.

Acknowledgements. We wish to thank Laurence Rideau and Laurent Théry for their work on the initial development of the Coq library. The authors are also indebted to the anonymous reviewers for their numerous comments and suggestions on both the details and the overall organization of the submitted work.

References

1. Amerkad A, Bertot Y, Rideau L, Pottier L (2001) Mathematics and proof presentation in Pcoq. In: Proceedings of the conference on proof transformation and presentation and proof complexities, Siena, Italy. Available at: <http://www-sop.inria.fr/lemme/Laurence.Rideau/proof-pcoq.ps.gz>
2. Barrett G (1989) Practical algorithm for selection on coarse grained parallel computers. IEEE Trans Softw Eng 15:611–621. <http://dlib.computer.org/ts/books/ts1989/pdf/e0611.pdf>
3. Boldo S, Daumas M (2002) Properties of the subtraction valid for any floating point system. In: Proceedings of the 7th international workshop on formal methods for industrial critical systems, Málaga, Spain, pp 137–149
4. Carreño VA, Miner PS (1995) Specification of the IEEE-854 floating-point standard in HOL and PVS. In: Proceedings of the 1995 international workshop on higher order logic theorem proving and its applications, Aspen Grove, UT, supplemental proceedings. Available at: <http://shemesh.larc.nasa.gov/fm/ftp/larc/vac/hug95.ps>
5. Chen YA, Clarke E, Ho PH, Hoskote Y, Kam T, Khaira M, O'Leary J, Zhao X (1996) Verification of all circuits in a floating point unit using word level model checking. In: Srivas M, Camilleri A (eds) Proceedings of the 1st international conference on formal methods in computer-aided design, Palo Alto, CA, pp 19–33
6. Cody WJ, Karpinski R et al (1984) A proposed radix and word-length independent standard for floating point arithmetic. IEEE Micro 4:86–100
7. Collins RR (1997) Inside the Pentium II math bug. Dr. Dobbs's Journal 22(8):52, 55–57. Available at: <http://www.ddj.com/articles/1997/9708/9708f/9708f.htm>

8. Coonen JT (1978) Specification for a proposed standard for floating point arithmetic. Memorandum ERL M78/72, University of California, Berkeley
9. Cowlishaw MF (2003) Decimal floating point: algorithm for computers. In: Bajard JC, Schulte M (eds) Proceedings of the 16th symposium on computer arithmetic, Santiago de Compostela, Spain, 2003, pp 104–111. Available at: <http://csdl.computer.org/comp/proceedings/arith/2003/1894/00/1894toc.htm>
10. Daumas M, Langlois P (2003) Additive symmetries: the non-negative case. *Theor Comput Sci* 291:143–157. Available at: [http://dx.doi.org/10.1016/S0304-3975\(02\)00223-2](http://dx.doi.org/10.1016/S0304-3975(02)00223-2)
11. Daumas M, Matula DW (1997) Validated roundings of dot products by sticky accumulation. *IEEE Trans Comput* 46:623–629.
12. Daumas M, Rideau L, Théry L (2001) A generic library of floating-point numbers and its application to exact computing. In: Proceedings of the 14th international conference on theorem proving in higher order logics. Edinburgh, UK, pp 169–184.
13. Demmel J (1984) Underflow and the reliability of numerical software. *SIAM J Sci Stat Comput* 5:887–919
14. Goldberg D (1991) What every computer scientist should know about floating point arithmetic. *ACM Comput Surv* 23:5–47. Available at: <http://www.acm.org/pubs/articles/journals/surveys/1991-23-1/p5-goldberg/p5-goldberg.pdf>
15. Harrison J (1999) A machine-checked theory of floating point arithmetic. In: Bertot Y, Dowek G, Hirschowitz A, Paulin C, Théry L (eds) Proceedings of the 12th international conference on theorem proving in higher order logics, Nice, France, pp 113–130. Available at: <http://www.cl.cam.ac.uk/users/jrh/papers/fparith.ps.gz>
16. Harrison J (2000) Formal verification of floating point trigonometric functions. In: Hunt WA, Johnson SD (eds) Proceedings of the 3rd international conference on formal methods in computer-aided design, Austin, TX, pp 217–233. Available at: <http://www.link.springer.de/link/service/series/0558/papers/1954/19540217.pdf>
17. Higham NJ (1996) Accuracy and stability of numerical algorithms. SIAM, Philadelphia. Available at: <http://www.ma.man.ac.uk/~higham/asna.html>. Also at: <http://www.maths.man.ac.uk/~higham/asna/>
18. Huet G, Kahn G, Paulin-Mohring C (1997) The Coq proof assistant: a tutorial: version 6.1. Technical Report 204, Institut National de Recherche en Informatique et en Automatique, Le Chesnay, France. Available at: <ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RT/RT-0204.pdf>
19. Jacobi C (2001) Formal verification of a theory of IEEE rounding. In: Proceedings of the 14th international conference on theorem proving in higher order logics, Edinburgh, UK, pp 239–254, supplemental proceedings. Available at: <http://www.informatics.ed.ac.uk/publications/online/0046/b239.pdf>
20. Kern C, Greenstreet MR (1999) Formal verification in hardware design: a survey. *ACM Trans Des Automat Electron Sys* 4:123–193. Available at: <http://delivery.acm.org/10.1145/310000/307989/p123-kern.pdf>
21. Knuth DE (1997) The art of computer programming: seminumerical algorithms, 3rd edn. Addison-Wesley, Reading, MA
22. Kulisch U (2000) Rounding near zero. In: Proceedings of the 4th real numbers and computers conference, Dagstuhl, Germany, pp 23–29
23. Laurent O, Michel P, Wiels V (2001) Using formal verification techniques to reduce simulation and test effort. In: Proceedings of the international symposium of formal methods Europe, Berlin, Germany, pp 465–477. Available at: <http://link.springer.de/link/service/series/0558/papers/2021/20210465.pdf>
24. Li RC, Boldo S, Daumas M (2003) Theorems on efficient argument reductions. In: Bajard JC, Schulte M (eds) Proceedings of the 16th symposium on computer arithmetic, Santiago de Compostela, Spain, pp 129–136.
25. Markstein P (2000) IA-64 and elementary functions: speed and precision. Prentice Hall, Upper Saddle River, NJ. Available at: <http://www.markstein.org/>
26. Miner PS, Leathrum JF (1996) Verification of IEEE compliant subtractive division algorithms. In: Proceedings of the 1st international conference on formal methods in computer-aided design, pp 64–78. Available at: http://www.ece.ou.edu/~leathrum/Formal_Methods/computer_arithmetic/fmcd.ps
27. O’Leary J, Zhao X, Gerth R, Seger CJH (1999) Formally verifying IEEE compliance of floating point hardware. *Intel Technol J*, vol 3. Available at: http://developer.intel.com/technology/itj/q11999/pdf/floating_point.pdf
28. Overton MJ (2001) Numerical computing with IEEE floating point arithmetic. SIAM, Philadelphia. Available at: <http://www.siam.org/catalog/mcc07/ot76.htm>
29. Priest DM (1992) On properties of floating point arithmetics: numerical stability and the cost of accurate computations. PhD thesis, University of California at Berkeley, Berkeley, CA. Available at: <ftp://ftp.icsi.berkeley.edu/pub/theory/priest-thesis.ps.Z>
30. Rushby J, von Henke F (1991) Formal verification of algorithms for critical systems. In: Proceedings of the conference on software for critical systems, New Orleans, pp 1–15. Available at: <http://www.acm.org/pubs/articles/proceedings/soft/125083/p1-rushby/p1-rushby.pdf>
31. Russinoff DM (1998) A mechanically checked proof of IEEE compliance of the floating point multiplication, division and square root algorithms of the AMD-K7 processor. *LMS J Comput Math* 1:148–200. <http://www.onr.com/user/russ/david/k7-div-sqrt.ps>
32. Schryer NL (1981) A test of computer’s floating-point arithmetic unit. Technical report 89, AT&T Bell Laboratories. Available at: <http://cm.bell-labs.com/cm/cs/cstr/89.ps.gz>
33. Schwarz EM, Smith RM, Krygowski CA (1999) The S/390 G5 floating point unit supporting hex and binary architectures. In: Koren I, Kornerup P (eds) Proceedings of the 14th symposium on computer arithmetic, Adelaide, Australia, pp 258–265. Available at: <http://computer.org/proceedings/arith/0116/0116toc.htm>
34. Sterbenz PH (1974) Floating point computation. Prentice-Hall, Upper Saddle River, NJ
35. Stevenson D et al (1987) An American national standard: IEEE standard for binary floating point arithmetic. *ACM SIGPLAN Notices* 22:9–25
36. Texas Instruments (1997) TMS320C3x User’s guide. Available at: <http://www-s.ti.com/sc/psheets/spru031e/spru031e.pdf>