

Newton-Raphson Algorithms for Floating-Point Division Using an FMA

Nicolas Louvet, Jean-Michel Muller, Adrien Panhaleux

▶ **To cite this version:**

Nicolas Louvet, Jean-Michel Muller, Adrien Panhaleux. Newton-Raphson Algorithms for Floating-Point Division Using an FMA. 21st IEEE International Conference on Application-specific Systems Architectures and Processors (ASAP), 2010, Jul 2010, Rennes, France. pp.200-207. ensl-00549027

HAL Id: ensl-00549027

https://ens-lyon.hal.science/ensl-00549027

Submitted on 21 Dec 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Newton-Raphson Algorithms for Floating-Point Division Using an FMA

Nicolas Louvet, Jean-Michel Muller, Adrien Panhaleux

Abstract

Since the introduction of the Fused Multiply and Add (FMA) in the IEEE-754-2008 standard [6] for floating-point arithmetic, division based on Newton-Raphson's iterations becomes a viable alternative to SRT-based divisions. The Newton-Raphson iterations were already used in some architecture prior to the revision of the IEEE-754 norm. For example, Itanium architecture already used this kind of iterations [8]. Unfortunately, the proofs of the correctness of binary algorithms do not extend to the case of decimal floating-point arithmetic. In this paper, we present general methods to prove the correct rounding of division algorithms using Newton-Raphson's iterations in software, for radix 2 and radix 10 floating-point arithmetic.

Keywords

floating-point arithmetic; decimal floating-point arithmetic; division algorithm; Newton-Raphson iterations

1. Introduction

When a floating-point *Fused-Multiply and Add* (FMA) instruction is available in hardware, a common method is to implement the division operation in software using Newton-Raphson's iterations. In binary floating-point arithmetic, this is already the case for example on the Itanium architecture. The FMA instruction allows to efficiently compute a correctly rounded quotient, even when the working precision used to perform the iterations is the same as the precision of the quotient [2], [8]. Moreover, the new IEEE-754-2008 standard [6] for floating-point arithmetic standardize both binary and decimal floating-point arithmetic, and introduce a correctly rounded FMA operation. As a consequence, software implementation of binary and decimal division may become a common practice in the near future.

In this paper, we present the techniques we developed for proving correct rounding for division algorithms based on Newton-Raphson's iterations performed with an FMA. While the previous works on this topic (see [8, chap. 8] for an overview) only dealt with binary floating-point arithmetic, the results we propose can be used to prove also the correctness of decimal division algorithms. For clarity, we focus here on rounding to the nearest, but the methods described can also be used for the directed rounding attributes of the IEEE-754-2008 standard.

Starting from previous results on exclusion intervals for division by Markstein and Harrison [8], [5] in binary floating-point arithmetic, we give a bound on the radius of the exclusion intervals applicable in any radix. To prove the correct rounding of the reciprocal operation using an extension of the exclusion interval, we also adapt the worst cases analysis by Harrison and Cornea [5], [3] for the reciprocal operation to the case of radix 10.

In a division algorithm, the Newton-Raphson's iterations are usually performed in a higher precision than the precision of the operand and of the quotient. When computing a quotient of floating-point numbers in the highest available precision, the proofs for radix 2 [9], [8, chap. 5] does not extend to radix 10: We also propose here a new method to ensure correct rounding in this case.

We mainly focus in this paper on software aspects, but the results presented may also be useful for the implementation of decimal Newton-Raphson's division algorithms in hardware [11].

1.1. Notations

In this article, we assume that no overflow nor underflow occurs, and that the inputs are normal numbers. We note $\mathbb{F}_{\beta,p}$ the set of radix-β, precision-p floating-point

- *Nicolas Louvet is with UCBL, LIP (CNRS, ENS de Lyon, INRIA, UCBL), Université de Lyon. E-mail: nicolas.louvet@ens-lyon.fr*
- *Jean-Michel Muller is with CNRS, LIP (CNRS, ENS de Lyon, INRIA, UCBL), Université de Lyon. E-mail: jean-michel.muller@ens-lyon.fr*
- *Adrien Panhaleux is with École Normale Supérieure de Lyon, Université de Lyon, LIP (CNRS, ENS de Lyon, INRIA, UCBL). E-mail: adrien.panhaleux@ens-lyon.fr*

numbers. We call *betade* an interval of the form $[\beta^e, \beta^{e+1})$. For any $z \neq 0$ in $\mathbb{R}$, if $z \in [\beta^{e_z}, \beta^{e_z+1})$ with $e_z \in \mathbb{Z}$, then e_z denotes the *exponent* of z, and $\mathrm{ulp}(z) := \beta^{e+1-p}$ its *unit in the last place*.

The middle of two consecutive floating-point numbers in $\mathbb{F}_{\beta,p}$ is called *midpoint* in precision p : every midpoint m in precision p can be written as $m = \pm(s_m + 1/2 \cdot \beta^{1-p})\beta^{e_m}$, with s_m a significand of precision p in $[1, \beta)$. Given $z \in \mathbb{R}$, we denote z rounded to the nearest floating-point value in $\mathbb{F}_{\beta,p}$ by $\mathrm{RN}_{\beta,p}(z)$, or more shortly by $\mathrm{RN}(z)$. We assume that the usual round to nearest even tie-breaking rule is applied when z is a midpoint [6]. Let us also recall that the FMA operation computes $\mathrm{RN}_{\beta,p}(a \times b + c)$ for any $a, b, c \in \mathbb{F}_{\beta,p}$.

Since we do not consider overflow or underflow, computing the quotient a/b is equivalent to computing the quotient of their significand. We then assume without loss of generality that both a and b lie in the betade $[1, \beta)$.

In the sequel of the paper, we consider three different precisions: p_i is the *precision of the input operands*, p_w is the *working precision* in which the intermediate computations are performed, and p_o is the *output precision*. Hence, given $a, b \in \mathbb{F}_{\beta,p_i}$, the division algorithm considered is intended to compute $\mathrm{RN}_{\beta,p_o}(a/b)$. We only consider the cases $p_w \geq p_o$ and $p_w \geq p_i$. The computation of a multiprecision quotient is not the goal of this paper.

Given $x \in \mathbb{R}$, we also use the following notation to distinguish between two kinds of approximation of x: $\hat{x}$ denotes any real number regarded as an approximation of x, and $\tilde{x}$ is the floating-point number obtained after rounding $\hat{x}$ to the nearest.

1.2. Newton-Raphson's Iterations

To compute a/b, an initial approximation $\hat{x}_0$ to $1/b$ is obtained from a lookup table addressed by the first digits of b. One next refines the approximation to $1/b$ using iteration (1) below:

$$\hat{x}_{n+1} = \hat{x}_n + \hat{x}_n\,(1 - b\hat{x}_n)\,. \tag{1}$$

Then $\hat{y}_n = a\hat{x}_n$ is taken as an initial approximation to a/b that can be improved using

$$\hat{y}_{n+1} = \hat{y}_n + \hat{x}_n\,(a - b\hat{y}_n)\,. \tag{2}$$

There are several ways of using the FMA to perform Newton-Raphson iterations. To compute the reciprocal $1/b$ using Equation (1), we have the following two iterations:

$$\text{Markstein} \quad \begin{cases} \tilde{r}_{n+1} = \mathrm{RN}(1 - b\tilde{x}_n) \\ \tilde{x}_{n+1} = \mathrm{RN}(\tilde{x}_n + \tilde{r}_{n+1}\tilde{x}_n) \end{cases} \tag{3}$$

$$\text{Goldschmidt} \quad \begin{cases} \tilde{r}_1 = \mathrm{RN}(1 - b\tilde{x}_0) \\ \tilde{r}_{n+2} = \mathrm{RN}(\tilde{r}_{n+1}^2) \\ \tilde{x}_{n+1} = \mathrm{RN}(\tilde{x}_n + \tilde{r}_{n+1}\tilde{x}_n) \end{cases} \tag{4}$$

The Markstein iteration [7], [8] immediately derives from Equation (1). The Goldschmidt iteration [4] is obtained from the Markstein iteration (3), by substituting r_{n+1} with r_n^2. Even if both iterations are mathematically equivalent, when we use them in floating-point arithmetic, they behave differently, as Example 1 shows.

Example 1. In binary16 ($p_w = 11, \beta = 2$):

$$b = 1.1001011001$$
$$1/b = 0.\underbrace{10100001010}_{11}100011...$$

Markstein's iteration	Goldschmidt's iteration
$\tilde{x}_0 = \ 0.10101010101$	$\tilde{x}_0 = \ 0.10101010101$
$\tilde{r}_1 = -1.1101100010 \cdot 2^{-5}$	$\tilde{r}_1 = -1.11011000100 \cdot 2^{-5}$
$\tilde{x}_1 = \ 0.10100000110$	$\tilde{x}_1 = \ 0.10100000110$ // $\tilde{r}_2$...
$\tilde{r}_2 = \ 1.1100111010 \cdot 2^{-9}$	$\tilde{x}_2 = \ 0.10100001010$ // $\tilde{r}_3$...
$\tilde{x}_2 = \ 0.10100001011$	($\tilde{x}_n$ remains the same)

In the Goldschmidt iteration, $\tilde{r}_{n+2}$ and $\tilde{x}_{n+1}$ can be computed concurrently. Hence, this iteration is faster due to its parallelism. However, in this example, only the Markstein iteration yields the correct rounding. A common method [8] is to use Goldschmidt's iterations at the beginning, when accuracy is not an issue, and next to switch to Markstein's iterations if needed on the last iterations to get the correctly rounded result.

Concerning the division, one may consider several iterations derived from Equation (2). We only consider here the following ones:

$$\text{Markstein} \quad \begin{cases} \tilde{r}_{n+1} = \mathrm{RN}(a - b\tilde{y}_n) \\ \tilde{y}_{n+1} = \mathrm{RN}(\tilde{y}_n + \tilde{r}_{n+1}\tilde{x}_n) \end{cases} \tag{5}$$

$$\text{Goldschmidt} \quad \begin{cases} \tilde{r}_0 = \mathrm{RN}(a - b\tilde{y}_0) \\ \tilde{r}_{n+2} = \mathrm{RN}(\tilde{r}_{n+1}^2) \\ \tilde{y}_{n+1} = \mathrm{RN}(\tilde{y}_n + \tilde{r}_{n+1}\tilde{x}_n) \end{cases} \tag{6}$$

1.3. Outline

Section 2 shows how to prove a faithful rounding, and how the information of faithful rounding can be used in the Newton-Raphson division algorithms. Section 3 then introduces necessary conditions that prove the correct rounding of these algorithms. Section 4 gives error bounds on the different variations of Newton-Raphson's iterations. Finally, Section 5 shows an example on how to prove a correct rounding of a Newton-Raphson based algorithm.

2. Faithful rounding

In some cases explained in Section 3, a faithful rounding is required in order to guarantee correct rounding of the quotient a/b. One may also only need a faithful rounding of the quotient or the reciprocal. This section provides a sufficient condition to ensure a faithful rounding of the

quotient. We then remind the exact residual theorem, that will be used for proving the correct rounding in Section 3.

2.1. Ensuring a faithful rounding

To prove that the last iteration yields a correct rounding, we use the fact that a faithful rounding has been computed. To prove that at some point, a computed approximation $\tilde{y}_n$ is a faithful rounding of the exact quotient a/b, we use a theorem similar to the one proposed by Rump [10], adapted here to the general case of radix β.

Theorem 1. *Let $\hat{x} \in \mathbb{R}$ be an approximation to $z \in \mathbb{R}$. Let $\tilde{x} \in \mathbb{F}_{\beta,p}$ be such that $\tilde{x} = \mathrm{RN}(\hat{x})$. If*

$$|\hat{x} - z| < \frac{1}{2\beta}\,\mathrm{ulp}(z), \tag{7}$$

then $\tilde{x}$ is a faithful rounding of z.

The condition of Theorem 1 is tight: Assuming β is even, if $z = \beta^k$, then $\hat{x} = z - \frac{1}{2\beta}\,\mathrm{ulp}(z)$ will round to a value that is not a faithful rounding of z, as illustrated on Figure 1.

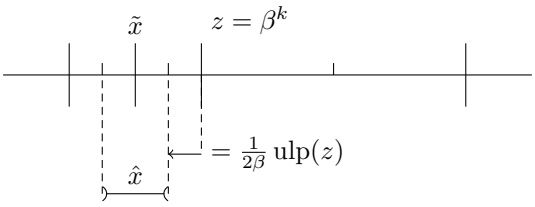


Figure 1. Tightness of the condition on $|\hat{x} - z|$

2.2. Exact residual theorem

When $\tilde{y}_n$ is a faithful rounding of a/b, The residual $\mathrm{RN}(a - b\tilde{y}_n)$ is computed exact. The theorem was first stated by Markstein [7] and has been more recently proved by John Harrison [5] and Boldo and Daumas [1] using formal provers.

Theorem 2 (Exact residual for the division)**.** *Let a, b be two floating-point numbers in $\mathbb{F}_{\beta,p}$, and assume $\tilde{y}_n$ is a faithful rounding of a/b. For any rounding mode $\circ$, $\tilde{r}_{n+1} = \circ(a - b\tilde{y}_n)$ is computed exactly (without any rounding), provided there is no overflow or underflow.*

3. Round-to-nearest

In this section, we present several methods to ensure correct rounding. We first present a general method of exclusion intervals that only applies if the quotient a/b is not a midpoint, and how to extend the exclusion intervals in the case of reciprocal. We then show how to handle the midpoint cases separately.

3.1. Exclusion intervals

A common way of proving correct rounding for a given function in floating-point arithmetic is to study its exclusion intervals (see [5], [8, chap. 8] or [9, chap. 5]).

Given $a, b \in \mathbb{F}_{\beta,p_i}$, either a/b is a midpoint at precision p_o, or there is a certain distance between a/b and the closest midpoint. Hence, if we assume that a/b is not a midpoint, then for any midpoint m, there exists a small interval centered at m that cannot contain a/b. Those intervals are called the *exclusion intervals*.

More formally, let us define $\mu_{p_i,p_o} > 0$ as the smallest value such that there exist $a, b \in \mathbb{F}_{\beta,p_i}$ and a midpoint m in precision p_o with $|a/b - m| = \beta^{e_{a/b}+1}\mu_{p_i,p_o}$. If a lower bound on μ_{p_i,p_o} is known, next Theorem 3 can be used to ensure correct rounding, as illustrated by Figure 2 (see [5] or [9, chap. 12] for a proof).

Theorem 3. *Let a, b in $\mathbb{F}_{\beta,p_i}$ be such that a/b is not midpoint in precision p_o for the division, and $\hat{y}$ be in $\mathbb{R}$. If $|\hat{y} - a/b| < \beta^{e_{a/b}+1}\mu_{p_i,p_o}$, then $\mathrm{RN}(\hat{y}) = \mathrm{RN}(a/b)$.*

To bound the radius of the exclusion intervals, we generalize the method used by Harrison [5] and Marius Cornea [3] to the case of radix β.

Theorem 4. *Assuming $p_o \geq 2$ and $p_i \geq 1$, a lower bound on μ_{p_i,p_o} is given by*

$$\mu_{p_i,p_o} \geq \frac{1}{2}\beta^{-p_i-p_o}. \tag{8}$$

Proof: By definition of μ_{p_i,p_o}, it can be proved that a/b is not a midpoint. Let m be the closest midpoint to a/b, and note $\delta\beta^{e_{a/b}+1}$ the distance between a/b and m:

$$\frac{a}{b} = m + \delta\beta^{e_{a/b}+1}. \tag{9}$$

By definition, μ_{p_i,p_o} is the smallest possible value of $|\delta|$. As we excluded the case when $a/b = m$, we have $\delta \neq 0$. We write $a = A\beta^{1-p_i}$, $b = B\beta^{1-p_i}$ and $m = (M + 1/2)\beta^{1-p_o+e_{a/b}}$, with A, B, M integers and $\beta^{p_i-1} \leq A, B, M < \beta^{p_i}$. Equation (9) becomes

$$2B\beta^{p_o}\delta = 2A\beta^{p_o-1-e_{a/b}} - 2BM - B. \tag{10}$$

Since $2A\beta^{p_o-1-e_{a/b}} - 2BM - B$ is an integer and $\delta \neq 0$, we have $|2B\beta^{p_o}\delta| \geq 1$. Since $\beta^{p_i-1} \leq B < \beta^{p_i}$, the conclusion follows. □

In radix 2, the following example illustrates the sharpness of the result of Theorem 4.

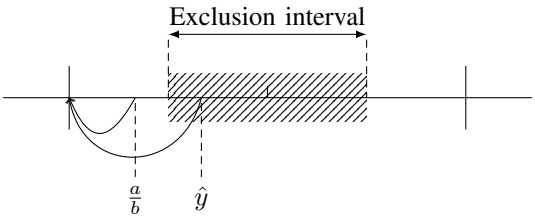


Figure 2. Use of exclusion intervals for proving the correct rounding

Example 2. In radix 2, for any precision $p_i = p_o$, $b = 1.11\ldots1 = 2 - 2^{1-p_i}$ gives

$$\frac{1}{b} = \frac{1}{2} + 2^{-1-p_i} + \underbrace{\frac{2^{-1-2p_i}}{1-2^{-p_i}}}_{\delta\beta^{e_{a/b}+1}} = 0.\underbrace{100\ldots0}_{p_i \text{ bits}}\underbrace{100\ldots0}_{p_i \text{ bits}}1\ldots$$

From this example, an upper bound on μ_{p_i,p_i} can be deduced, and for any precision $p_i \geq 1$ one has

$$2^{-1-2p_i} \leq \mu_{p_i,p_i} \leq \frac{2^{-1-2p_i}}{1-2^{-p_i}}.$$

The following result can also be seen as a consequence of Theorem 3 (see [8, chap. 8] or [9, p.163] for a proof).

Theorem 5. *In binary arithmetic, when $p_w = p_o$, if $\tilde{x}$ is a correct rounding of $1/b$ and $\tilde{y}$ is a faithful rounding of a/b, then an extra Markstein's iteration yields $RN_{p_o}(a/b)$.*

3.2. Extending the exclusion intervals

When $p_w = p_o = p_i$, the error bounds of Section 4 might remain larger than the bound on the radius of the exclusion intervals of Section 3.1. A way to prove correct rounding is then by extending the exclusion intervals.

In this subsection, we describe a method to determine all the inputs $(a, b) \in \mathbb{F}^2_{\beta,p_i}$ such that the quotient a/b is not a midpoint and lies within a distance $\beta^{e_{a/b}+1}\mu$ from the closest midpoint m. Once all such worst cases are determined, correct rounding can be guaranteed considering two cases:

- If a/b corresponds to one of the worst cases, we then run the Newton-Raphson algorithm on the input (a, b) and check that the result is correct.
- If (a, b) is not one of those worst cases and $\hat{y}$ is an approximation to a/b that satisfies $|\hat{y}-a/b| < \beta^{e_{a/b}+1}\mu$, then $\mathrm{RN}_{\beta,p_o}(\hat{y}) = \mathrm{RN}_{\beta,p_o}(a/b)$.

Unfortunately, there are too many worst cases for the division, but one can apply this method for the reciprocal. Starting from Equation (10) of Section 3.1, one has:

$$2\beta^{p_i+p_o} - \underbrace{2B\beta^{p_o}\delta}_{\Delta} = B(2M+1). \qquad (11)$$

Factorizing $2\beta^{p_i+p_o} - \Delta$, with $|\Delta| \in \{1, 2, \ldots\}$ into $B(2M+1)$ with respect to the range of these integral significands isolates the worst cases. After finding all the worst cases such that $|\Delta| < n$, the extended radius is such that $\mu \geq \beta^{-p_i-p_o}n/2$. Table 1 shows how many values of b have to be checked to extend the exclusion interval.

n	2	3	4	5	6	7
binary64	2	68	68	86	86	86
decimal128	1	1	3	3	19	22

Table 1. Number of b to check separately according to the extended radius of the exclusion interval $\mu \geq \beta^{-p_i-p_o}n/2$

There is a particular worst case that is worth mentioning: When $b = \beta - 1/2\ \mathrm{ulp}(\beta)$, the correctly rounded reciprocal is $1/\beta+\mathrm{ulp}(1/\beta)$, but if the starting point given by the lookup table is not $\mathrm{RN}_{\beta,p_w}(1/b)$, even Markstein's iterations cannot give a correct rounding, as shown in Example 3.

Example 3. In binary16, ($p_w = p_i = p_o = 11, \beta = 2$):

$$\begin{aligned} b &= 1.1111111111 \\ \tilde{x} &= 0.1 = \text{Table-lookup}(b) \\ \tilde{r} &= 2^{-11} \text{ (exact)} \\ \tilde{x} &= 0.1 \end{aligned}$$

Hence, $\tilde{x}$ will always equals 0.1, which is not the correct rounding of $\mathrm{RN}(1/b)$.

A common method [8] to deal with this case is to tweak the lookup table for this value. If the lookup table is addressed by the first k digits of b, then the output corresponding to the address $\beta - \beta^{1-k}$ should be $1/\beta + \beta^{-p_w}$.

3.3. The midpoint case

Theorem 3 can only be used to ensure correct rounding when a/b cannot be a midpoint. In this subsection, we summarize our results about the midpoints for division and reciprocal.

3.3.1. Midpoints in radix 2. Markstein already proved for radix 2 that for any a, b in $\mathbb{F}_{2,p_i}$, a/b cannot be a floating-point number of precision p with $p > p_i$ [8]. This means that a/b cannot be a midpoint in a precision greater or equal to p_i. However, Example 4 shows that when $p_i > p_o$, a/b can be a midpoint.

Example 4. Inputs: binary32, output: binary16.

$$\begin{aligned} a&=1.00000001011001010011111 \\ b&=1.00101100101000000000000 \\ a/b&=0.\underbrace{11011011001}_{p_o=11}1 \end{aligned}$$

When computing reciprocals, we have the following.

Theorem 6. *In radix 2, and for any precisions p_i and p_o, the reciprocal of a floating-point number in $\mathbb{F}_{2,p_i}$ cannot be a midpoint in precision p_o.*

Proof: Given a floating-point number $b \in \mathbb{F}_{2,p_i}$ in $[1,2)$, we write $b = B2^{1-p_i}$, where B is an integer. If $1/b$ is a midpoint in precision p_o, then $1/b = (2Q+1)2^{-p_o-1}$ with Q an integer. This gives $B(2Q+1) = 2^{p_i+p_o}$. Since B and Q are integers, this equality can hold only if $Q = 0$. This implies $b = 2^{1+p_o}$, which contradicts $b \in [1,2)$. □

3.3.2. Midpoints in radix 10. In decimal floating-point arithmetic, the situation is quite different. As in radix 2, there are cases where a/b is a midpoint in precision p_o, but they can occur even when $p_i = p_o$, as shown in Example 5. Contrarily to the binary case, there are also midpoints for the reciprocal function, characterized by Theorem 7.

Example 5. In decimal32 ($p_i = p_o = 7, \beta = 10$):

$$a = 2.000005, \quad b = 2.000000, \quad a/b = 1.000002\,5$$

Theorem 7. *In radix 10, for any precisions p_i and p_o, there are at most two floating-point numbers in a single betade of $\mathbb{F}_{10,p_i}$ whose reciprocal is a midpoint in precision p_o. Their integral significands are $B_1 = 2^{p_i+p_o}5^{z_1}$ and $B_2 = 2^{p_i+p_o}5^{z_2}$, with*

$$z_1 = \left\lceil p_i - p_o\frac{\ln(2)}{\ln(5)} - \frac{\ln(10)}{\ln(5)} \right\rceil, z_2 = \left\lfloor p_i - p_o\frac{\ln(2)}{\ln(5)} \right\rfloor.$$

Proof of Theorem 7.: Given a floating-point number $b \in \mathbb{F}_{10,p_i}$ in the betade $[1,10)$, we rewrite it $b = B10^{1-p_i}$. If $1/b$ is a midpoint, we then have $1/b = (10Q + 5)10^{-p_o-1}$, with $Q \in \mathbb{Z}$, which gives $B(2Q+1) = 2^{p_i+p_o}\,5^{p_i+p_o-1}$. Since $2Q+1$ is odd, we know that $2^{p_i+p_o}$ divides B. Therefore we have $B = 5^z 2^{p_i+p_o}$ with z an integer. Moreover, we know that $1 \le b < 10$, which gives

$$p_i - p_o\frac{\ln 2}{\ln 5} - \frac{\ln 10}{\ln 5} \le z \le p_i - p_o\frac{\ln 2}{\ln 5}.$$

The difference between the two bounds is $\ln 10/\ln 5 \approx 1.43$. Therefore, there can be at most two integers z between the two bounds. □

Using Theorem 7, we isolate the at most two values of b whose reciprocal is a midpoint. These values are checked separately when proving the correct rounding of the reciprocal. Table 2 gives the corresponding b when $p_i = p_o$, for the IEEE 754-2008 decimal formats.

3.4. Correctly handling midpoint cases

Let us recall that the midpoints cases for reciprocal can be handled as explained in §3.3.1 §3.3.2. Hence, we only focus here on division.

	decimal32	**decimal64**	**decimal128**
p	7	16	34
b_1	2.048000	1.67772160…0	1.12589990684262400…0
b_2	2.048000	8.38860800…0	5.62949953421312000…0

Table 2. Decimal floating-point numbers whose reciprocal is a midpoint in the same precision

When p_i, p_o and the radix are such that division admits midpoints, the last Newton-Raphson iteration must be adapted to handle the case where a/b is a midpoint. We propose two methods, depending whether $p_w = p_o$. Both methods rely on the exact residual theorem 2 of section 2.2, so it is necessary to use a Markstein iteration (5) for the last iteration.

3.4.1. When $p_w > p_o$. The exclusion interval theorem of Section 3.1 does not apply, since there are several cases where a/b is a midpoint in precision p_o. In that case, we use the following Theorem 8 instead of Theorem 3.

Theorem 8. *We assume β is even and $p_w > p_o$, and we perform a Markstein iteration:*

$$\begin{cases} \tilde{r} = \mathrm{RN}_{\beta,p_w}(a - b\tilde{y}), \\ \tilde{y}' = \mathrm{RN}_{\beta,p_o}(\tilde{y} + \tilde{r}\tilde{x}). \end{cases}$$

If $\tilde{y}$ is a faithful rounding of a/b in precision p_w, and $|\hat{y}' - a/b| < \beta^{e_{a/b}+1}\mu_{p_i,p_o}$, then $\tilde{y}' = \mathrm{RN}_{\beta,p_o}(a/b)$.

Proof of theorem 8: If a/b is not a midpoint in precision p_o, Theorem 3 proves that $\tilde{y}'$ is the correct rounding of a/b. Now, we assume that a/b is a midpoint in precision p_o. Since β is even and $p_w > p_o$, a/b is a floating-point number in precision p_w. Since $\tilde{y}$ is a faithful rounding of a/b in precision p_w, we have $\tilde{y} = a/b$. Using Theorem 2, we know that $\tilde{r} = 0$, which gives $\hat{y}' = a/b$, hence $\tilde{y}' = \mathrm{RN}_{\beta,p_o}(\hat{y}') = \mathrm{RN}_{\beta,p_o}(a/b)$. □

Example 6 shows why it is important to round directly in precision p_o in the last iteration.

Example 6. inputs: binary32, output: binary16.

a=1, b = 1.01010011001111000011011
$\tilde{y}$=0.11000001001011111111111 (faithful)
$\tilde{r}$=1.01010011000111000011011 · 2^{-24} (exact)
$\tilde{y}'$=0.$\underbrace{11000001001}_{\text{11 bits}}$1000000000000 = $\mathrm{RN}_{24}(\tilde{y} + \tilde{r}\tilde{y})$
$\tilde{y}''$=0.11000001010 = $\mathrm{RN}_{11}(\tilde{y}')$

Due to the double rounding, $\tilde{y}''$ is not $\mathrm{RN}_{11}(a/b)$.

3.4.2. When $p_w = p_o$. The quotient a/b cannot be a midpoint in radix 2. For decimal arithmetic, Example 7 suggests that it is not possible in this case to round correctly using only Markstein's iterations.

Example 7. In decimal32 ($p_i = p_o = p_w = 7, \beta = 10$):

$$a = 6.000015,\ b = 6.000000$$
$$\tilde{x} = \mathrm{RN}(1/b) = 0.1666667,\ a/b = 1.000002\,5$$

$$\begin{aligned}
\tilde{y}_0 &= \mathrm{RN}(a\tilde{x}) &&= 1.000003\\
\tilde{r}_1 &= \mathrm{RN}(a - b\tilde{y}_0) &&= -0.000003\\
\tilde{y}_1 &= \mathrm{RN}(\tilde{y}_0 + \tilde{r}_1\tilde{x}) &&= 1.000002\\
\tilde{r}_2 &= \mathrm{RN}(a - b\tilde{y}_1) &&= 0.000003\\
\tilde{y}_2 &= \mathrm{RN}(\tilde{y}_1 + \tilde{r}_2\tilde{x}) &&= 1.000003
\end{aligned}$$

Algorithm 1 can be used in this case to determine the correct rounding of a/b from a faithfully rounded approximation.

```
b_s = b ulp(a/b) ;                    /* b_s ∈ F_{β,p_w} */
/* Assume ỹ faithful */
r̃ = a − bỹ ;     /* r̃ exactly computed. */
if r̃ > 0 then
    c = RN(2r̃ − b_s);
    if c = 0 then return RN(ỹ + 1/2 ulp(a/b));
    if c < 0 then return ỹ;
    if c > 0 then return ỹ + ulp(a/b);
else /* r̃ ≤ 0 */
    c = RN(2r̃ + b_s);
    if c = 0 then return RN(ỹ − 1/2 ulp(a/b));
    if c < 0 then return ỹ − ulp(a/b);
    if c > 0 then return ỹ;
end
```

Algorithm 1: Returning the correct rounding in decimal arithmetic when $p_w = p_o$.

Theorem 9. *Let us assume that $\beta = 10$ and $p_w = p_o$ and that $\tilde{y}$ is a faithful rounding of a/b. Then, Algorithm 1 yields the correct rounding of a/b.*

Proof: By assumption, $\tilde{y}$ is a faithful rounding of a/b. Thus, there exists ϵ such that $-\mathrm{ulp}(a/b) < \epsilon < \mathrm{ulp}(a/b)$ and $\tilde{y} = a/b + \epsilon$. Also, according to Theorem 2, $\tilde{r} = -b\epsilon$. Six cases, depending on the signs of $\tilde{r}$ and c, have to be considered for the whole proof. We only present here two cases, the others being similar.

• Case $\tilde{r} \geq 0$ and $2\tilde{r} - b\ \mathrm{ulp}(a/b) < 0$: Since $\tilde{r}$ is positive, $-\epsilon \leq 0$. Moreover, since $2\tilde{r} - b\ \mathrm{ulp}(a/b) < 0$ we have $-1/2\ \mathrm{ulp}(a/b) < \epsilon < 0$. Hence, the correct rounding of a/b is $\tilde{y}$.

• Case $\tilde{r} < 0$ and $2\tilde{r} + b\ \mathrm{ulp}(a/b) = 0$: From $2\tilde{r} + b\ \mathrm{ulp}(a/b) = 0$, we deduce that a/b is a midpoint and $\mathrm{RN}(a/b) = \mathrm{RN}(\tilde{y} - 1/2\ \mathrm{ulp}(a/b))$. □

4. Error bounds

In this section, we present the techniques we used to bound the error in the approximation to the reciprocal $1/b$ or to the quotient a/b obtained after a series of Newton-Raphson iterations. As our aim is to analyze any reasonable sequence combining both Markstein's or Goldschmidt's iterations, we only give the basic results needed to analyze one step of these iterations. The analysis of a whole sequence of iterations can be obtained by combining the induction relations proposed here: This is a kind of *running error analysis* (see [9, chap. 6]) that can be used together with the results of Sections 3.1 and 3.2 to ensure correct rounding.

All the arithmetic operations are assumed to be performed at precision p_w, which is the precision used for intermediate computations. Let us denote by ϵ the unit roundoff: In round-to-nearest rounding mode, one has $\epsilon = \frac{1}{2}\beta^{1-p_w}$. In the following, we note

$$\begin{aligned}
&\hat{\phi}_n := |\hat{x}_n - 1/b|, && \tilde{\phi}_n := |\tilde{x}_n - 1/b|,\\
&\hat{\psi}_n := |\hat{y}_n - a/b|, && \tilde{\psi}_n := |\tilde{y}_n - a/b|,\\
&\tilde{\rho}_n := |\tilde{r}_n - (1 - b\tilde{x}_{n-1})|, && \tilde{\sigma}_n := |\tilde{r}_n - (a - b\tilde{y}_{n-1})|.
\end{aligned}$$

4.1. Reciprocal iterations

Both for Markstein's iteration (3) and for Goldschmidt's iteration (4), the absolute error $\hat{\phi}_n$ in the approximation $\hat{x}_n$ is bounded as

$$\hat{\phi}_{n+1} \leq (\tilde{\phi}_n + |1/b|)\tilde{\rho}_{n+1} + |b|\tilde{\phi}_n^2, \tag{12}$$

$$\tilde{\phi}_{n+1} \leq (1 + \epsilon)\hat{\phi}_{n+1} + |\epsilon/b|. \tag{13}$$

Hence it just remains to obtain induction inequalities for bounding $\tilde{\rho}_{n+1}$.

4.1.1. Reciprocal with the Markstein iteration (3). One has $\tilde{r}_{n+1} = \mathrm{RN}(1 - b\tilde{x}_n)$, hence

$$\tilde{\rho}_{n+1} \leq |\epsilon||b|\tilde{\phi}_n. \tag{14}$$

The initial value of the recurrence depends on the lookup-table used for the first approximation to $1/b$. Inequality (12) together with (14) can then be used to ensure either faithful or correct rounding for all values of b in $[1, \beta)$, using Theorems 1 or 3.

At iteration n, if $\tilde{x}_n$ is a faithful rounding of $1/b$, then Theorem 2 implies $\tilde{\rho}_{n+1} = 0$. Hence in this case one has $\hat{\phi}_{n+1} \leq \tilde{\phi}_n^2$, which means that no more accuracy improvement can be expected with Newton-Raphson iterations. Moreover, if we exclude the case $b = 1$, since b belongs to $[1, \beta)$ by hypothesis, it follows that $1/b$ is in $(\beta^{-1}, 1)$. Since $\tilde{x}_n$ is assumed to be a faithful rounding of $1/b$, one has $\mathrm{ulp}(\tilde{x}_n) = \mathrm{ulp}(1/b)$, and we deduce

$$\tilde{\phi}_{n+1} \leq |b|\tilde{\phi}_n^2 + 1/2\ \mathrm{ulp}(\tilde{x}_n), \tag{15}$$

which gives a sharper error bound on $\tilde{\phi}_{n+1}$ than (12) when $\tilde{x}_n$ is a faithful rounding of $1/b$.

4.1.2. Reciprocal with the Goldschmidt iteration (4). For the Goldschmidt iteration, one has

$$\tilde{\rho}_{n+1} \leq (1+\epsilon)\left(\tilde{\rho}_n + |b|\tilde{\phi}_{n-1} + 1\right)\tilde{\rho}_n + \epsilon. \quad (16)$$

Combining (16) into (13), one can easily deduce a bound on the error $\hat{\phi}_{n+1}$.

4.2. Division iterations

Both for Markstein's iteration (5) and for Goldschmidt's iteration (6), one may check that

$$\hat{\psi}_{n+1} \leq |b|\tilde{\psi}_n\tilde{\phi}_m + (\tilde{\phi}_m + |1/b|)\tilde{\sigma}_{n+1}, \quad (17)$$

$$\tilde{\psi}_{n+1} \leq (1+\epsilon)\hat{\psi}_{n+1} + \epsilon|a/b|. \quad (18)$$

Now let us bound $\tilde{\sigma}_{n+1}$.

4.2.1. Division with the Markstein iteration (5). In this case, one has

$$\tilde{\sigma}_{n+1} \leq \epsilon|b|\tilde{\psi}_n. \quad (19)$$

Again, if $\tilde{y}_n$ is a faithful rounding of a/b, due to the exact residual theorem 2, one has $\hat{\psi}_{n+1} \leq |b|\tilde{\psi}_n\tilde{\phi}_m$, which is the best accuracy improvement that can be expected from one Newton-Raphson iteration.

4.2.2. Division with the Goldschmidt iteration (5). Using the same method as in §4.1.2, we now bound $\tilde{\sigma}_{n+1}$:

$$\begin{aligned}\tilde{\sigma}_{n+1} \quad \leq \quad & (1+\epsilon)(\tilde{\sigma}_n + |b|\tilde{\psi}_{n-1})(|b|\tilde{\psi}_{n-1} + |b|\tilde{\phi}_m + \tilde{\sigma}_n) \\ & +(1+\epsilon)\tilde{\sigma}_n + \epsilon|a|. \end{aligned} \quad (20)$$

Then, from (17), a bound on $\hat{\psi}_{n+1}$ can be obtained.

5. Experiments

Using the induction relations of Section 4, one can bound the error on the approximations to a/b for a given series of Newton-Raphson iterations, and use it with the sufficient conditions presented in Section 3 to ensure correct rounding. Let us consider three examples : Algorithms 2, 3 and 4 below. The certified error on $\hat{x}$ and $\hat{y}$ for those algorithms is displayed on Figure 3.

Algorithm 2 computes the quotient of two binary128 ($p_i = 113$) numbers, the output being correctly rounded to binary64 ($p_o = 53$). The internal format used for the computations is also binary128 ($p_w = 113$). Since $p_i > p_o$, there are midpoints for division, as stated in Section 3.3. After the MD_1 iteration, we know from Theorem 1 that $\tilde{y}$ is a faithful rounding of a/b, as shown in Figure 3(a). An extra Markstein's iteration gives an error on $\hat{y}$ that is smaller than the radius of the exclusion interval $\beta^{e_{a/b}+1}\mu_{113,53}$, as illustrated by Figure 3(a). Hence, Theorem 8 of Section 3.4.1 applies and guarantees that Algorithm 2 yields a correct rounding of the division, even for the midpoint cases.

$$\begin{array}{lll}
\tilde{x} = \text{Table-lookup}(b); & \{\text{Error less than } 2^{-8}\} & \\
\tilde{r} = \mathrm{RN}_{113}(1 - b\tilde{x}); & & \\
\tilde{x} = \mathrm{RN}_{113}(\tilde{x} + \tilde{r}\tilde{x}); & \{\mathrm{MR}_1\} \;||\; & \tilde{r} = \mathrm{RN}_{113}(\tilde{r}^2); \\
\tilde{x} = \mathrm{RN}_{113}(\tilde{x} + \tilde{r}\tilde{x}); & \{\mathrm{GR}_2\} \;||\; & \tilde{r} = \mathrm{RN}_{113}(\tilde{r}^2); \\
\tilde{x} = \mathrm{RN}_{113}(\tilde{x} + \tilde{r}\tilde{x}); & \{\mathrm{GR}_3\} & \\
\tilde{y} = \mathrm{RN}_{113}(a\tilde{x}); & \{y_0\} & \\
\tilde{r} = \mathrm{RN}_{113}(a - b\tilde{y}); & & \\
\tilde{y} = \mathrm{RN}_{113}(\tilde{y} + \tilde{r}\tilde{x}); & \{\mathrm{MD}_1\} & \\
\tilde{r} = \mathrm{RN}_{113}(a - b\tilde{y}); & & \\
\tilde{y} = \mathrm{RN}_{53}(\tilde{y} + \tilde{r}\tilde{x}); & \{\mathrm{MD}_2\} &
\end{array}$$

Algorithm 2: Computing the quotient of two binary128 numbers, output in binary64.

$$\begin{array}{lll}
\tilde{x} = \text{Table-lookup}(b); & \{\text{Error less than } 2^{-8}\} & \\
\tilde{r} = \mathrm{RN}_{53}(1 - b\tilde{x}); & & \\
\tilde{x} = \mathrm{RN}_{53}(\tilde{x} + \tilde{r}\tilde{x}); & \{\mathrm{MR}_1\} \;||\; & \tilde{r} = \mathrm{RN}_{53}(\tilde{r}^2); \\
\tilde{x} = \mathrm{RN}_{53}(\tilde{x} + \tilde{r}\tilde{x}); & \{\mathrm{GR}_2\} & \\
\tilde{r} = \mathrm{RN}_{53}(1 - b\tilde{x}); & & \\
\tilde{x} = \mathrm{RN}_{53}(\tilde{x} + \tilde{r}\tilde{x}); & \{\mathrm{MR}_3\} & \\
\tilde{r} = \mathrm{RN}_{53}(1 - b\tilde{x}); & & \\
\tilde{x} = \mathrm{RN}_{53}(\tilde{x} + \tilde{r}\tilde{x}); & \{\mathrm{MR}_4\} & \\
\tilde{y} = \mathrm{RN}_{53}(a\tilde{x}); & \{y_0\} & \\
\tilde{r} = \mathrm{RN}_{53}(a - b\tilde{y}); & & \\
\tilde{y} = \mathrm{RN}_{53}(\tilde{y} + \tilde{r}\tilde{x}); & \{\mathrm{MD}_1\} & \\
\tilde{r} = \mathrm{RN}_{53}(a - b\tilde{y}); & & \\
\tilde{y} = \mathrm{RN}_{53}(\tilde{y} + \tilde{r}\tilde{x}); & \{\mathrm{MD}_2\} &
\end{array}$$

Algorithm 3: Computing the quotient of two binary64 numbers, output in binary64.

Algorithm 3 computes the quotient of two binary64 numbers, with $p_i = p_w = p_o = 53$. Since binary arithmetic is used and $p_w = p_o$, there are no midpoints for division. After the MR_4 iteration, $\tilde{x}$ is less than $2 \cdot \beta^{e_{1/b}+1}\mu_{53,53}$. Hence, by excluding two worst cases as explained in Section 3.2, and checking thoses cases, we ensure a correct rounding of the reciprocal using Theorem 3. Since a faithful rounding of a/b at iteration MD_1 is ensured by the error bounds of Section 4, Theorem 5 proves that the next Markstein's iteration outputs a correct rounding.

Algorithm 4 computes the quotient of two decimal128 numbers, with $p_i = p_w = p_o = 34$. The starting error given by the lookup table is less than $5 \cdot 10^{-5}$. Since $p_w = p_o$, Algorithm 1 is needed to ensure the correct rounding of the division. Notice that to improve the latency, b_s in Algorithm 1 can be computed concurrently with the first Newton-Raphson iterations. As shown in Figure 3(c),

$\tilde{x} = \text{Table-lookup}(b);$ || $b_s = b\ \text{ulp}(\frac{a}{b});$
$\tilde{r} = \text{RN}_{34}(1 - b\tilde{x});$
$\tilde{x} = \text{RN}_{34}(\tilde{x} + \tilde{r}\tilde{x});\ \{\text{MR}_1\}$ || $\tilde{r} = \text{RN}_{34}(\tilde{r}^2);$
$\tilde{x} = \text{RN}_{34}(\tilde{x} + \tilde{r}\tilde{x});\ \{\text{GR}_2\}$ || $\tilde{r} = \text{RN}_{34}(\tilde{r}^2);$
$\tilde{x} = \text{RN}_{34}(\tilde{x} + \tilde{r}\tilde{x});\ \{\text{GR}_3\}$
$\tilde{y} = \text{RN}_{34}(a\tilde{x});\ \{y_0\}$
$\tilde{r} = \text{RN}_{34}(a - b\tilde{y});$
$\tilde{y} = \text{RN}_{34}(\tilde{y} + \tilde{r}\tilde{x});\ \{\text{MD}_1\}$
$\tilde{r} = \text{RN}_{34}(a - b\tilde{y});$
Call Algorithm 1.

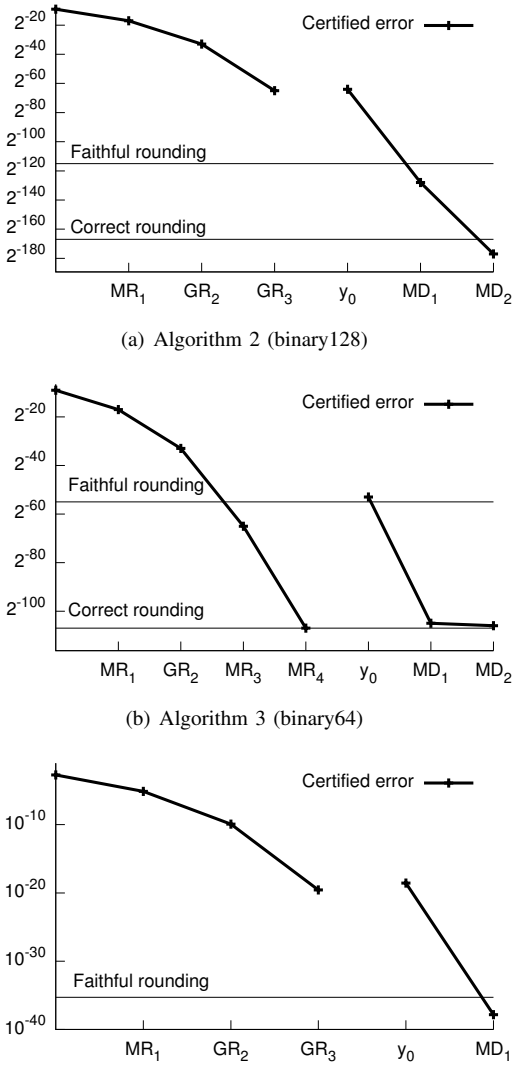
Algorithm 4: Computing the quotient of two decimal128 numbers, output in decimal128.

$\tilde{y}$ is a faithful rounding after the MD_1 iteration. Hence, Theorem 9 ensures correct rounding for Algorithm 4.

Conclusion

In this paper, we gave general methods of proving correct rounding for division algorithms based on Newton-Raphson's iterations, for both binary and decimal arithmetic. Performing the division in decimal arithmetic of two floating-point numbers in the working precision seems to be costly, and we recommend to always use a higher internal precision than the precision of inputs.

We only considered the round-to-nearest rounding mode in this paper. To achieve correct rounding in other rounding modes, only the last iteration of the Newton-Raphson algorithm has to be changed, whereas all the previous computations should be done in the round-to-nearest mode.



(a) Algorithm 2 (binary128)

(b) Algorithm 3 (binary64)

(c) Algorithm 4 (decimal128)

Figure 3. Absolute error before rounding for each algorithm considered. (M/G: Markstein/Goldschmidt, R/D: reciprocal/division)

References

[1] Sylvie Boldo and Marc Daumas. Representable correcting terms for possibly underflowing floating point operations. In Jean-Claude Bajard and Michael Schulte, editors, *Proceedings of the 16th Symposium on Computer Arithmetic*, pages 79–86, Santiago de Compostela, Spain, 2003.

[2] Marius Cornea, John Harrison, and Ping Tak Peter Tang. *Scientific Computing on Itanium-Based Systems*. Intel Press, 2002.

[3] Marius Cornea-Hasegan. Proving the IEEE correctness of iterative floating-point square root, divide, and remainder algorithms. *Intel Technology Journal*, (Q2):11, 1998.

[4] R. E. Goldschmidt. Applications of division by convergence. Master's thesis, Dept. of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, MA, USA, June 1964.

[5] John Harrison. Formal verification of IA-64 division algorithms. In M. Aagaard and J. Harrison, editors, *Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000*, volume 1869 of *Lecture Notes in Computer Science*, pages 234–251. Springer-Verlag, 2000.

[6] IEEE Computer Society. *IEEE Standard for Floating-Point Arithmetic*. IEEE Standard 754-2008, August 2008. available at http://ieeexplore.ieee.org/servlet/opac?punumber=4610933.

[7] Peter Markstein. Computation of elementary functions on the IBM RISC System/6000 processor. *IBM J. Res. Dev.*, 34(1):111–119, 1990.

[8] Peter Markstein. *IA-64 and elementary functions: speed and precision*. Hewlett-Packard professional books. 2000.

[9] Jean-Michel Muller, Nicolas Brisebarre, Florent de Dinechin, Claude-Pierre Jeannerod, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, and Serge Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010. ACM G.1.0; G.1.2; G.4; B.2.0; B.2.4; F.2.1., ISBN 978-0-8176-4704-9.

[10] Siegfried M. Rump, Takeshi Ogita, and Shin'ichi Oishi. Accurate floating-point summation part I: Faithful rounding. *SIAM Journal on Scientific Computing*, 31(1):189–224, 2008.

[11] Liang-Kai Wang and Michael J. Schulte. A decimal floating-point divider using Newton-Raphson iteration. *J. VLSI Signal Process. Syst.*, 49(1):3–18, 2007.