The Journal of Supercomputing (2024) 80:9742–9761
https://doi.org/10.1007/s11227-023-05814-y

Advancements in number representation for high-precision computing

H. Mora[1] · M. T. Signes-Pont[1] · F. A. Pujol López[1] · J. Mora-Pascual[1] · J. M. García Chamizo[1]

Accepted: 11 November 2023 / Published online: 14 December 2023

Abstract
Efficient representation of data is a fundamental prerequisite for addressing computational problems effectively using computers. The continual improvement in methods for representing numbers in computers serves as a critical step in expanding the scope and capabilities of computing systems. In this research, we conduct a comprehensive review of both fundamental and advanced techniques for representing numbers in computers. Additionally, we propose a novel model capable of representing rational numbers with absolute precision, catering to specific high precision applications. Specifically, we adopt fractional positional notation coupled with explicit codification of the periodic parts, thereby accommodating the entire rational number set without any loss of accuracy. We elucidate the properties and hardware representation of this proposed format and provide the results of extensive experiments to demonstrate its expressiveness and minimal codification error when compared to other real number representation formats. This research contributes to the advancement of numerical representation in computer systems, empowering them to handle complex computations with heightened accuracy, making them more reliable and versatile in a wide range of applications.

Keywords High-performance computing · Architecture and systems · Computer arithmetic · Number representation

✉ H. Mora
hmora@ua.es

M. T. Signes-Pont
teresa@ua.es

J. Mora-Pascual
jeronimo@ua.es

[1] Department of Computer Science Technology and Computation, University of Alicante, Alicante, Spain

1 Introduction and motivation

Consider the following problem:

With the first-order equation

$$3x = 1,$$

, the result is well known and easy to calculate, and the value for x is found as

$$x = 0.\hat{3}_{10}$$

That number expressed in fractional positional notation indicates, with absolute precision, the value of the variable x. However, as with most currently available algorithmic technologies, it is not possible to obtain an exact result, but only an approximation.

This result occurs due to the nature of the positional representation of the periodic number, which comprises infinite fractional digits.

This issue raises several questions regarding the characteristics of numerical representation. How many significant digits does a number have? To what extent is this influenced by the number representation base? What numeric sets cause this situation? An analysis of these questions will enable us to clarify whether there are alternative numerical codifications for these numbers or whether it is necessary to relinquish the expression of their exact value through a positional representation. The most frequent is to undertake approximations to the correct value using a specific number of digits.

Number processing plays a vital role in resolving various computational problems that are challenging to address with global solutions due to factors such as the complexity of operations, functional domains, data nature, and numerical ranges of operands and results. Certain applications, in particular, demand high-performance computing for intricate mathematical calculations, while simultaneously imposing stringent constraints on accuracy. For instance, scenarios like calculating trajectories for moving bodies in space or over significant distances, guidance and positioning systems, quantum theory calculations, intensive financial transactions, climate modeling, and more [1], all require utmost precision in computations. In these applications, the magnitude of the error could be difficult to delimit, and the incorrect results could propagate by other subsequent operations. Thus, even a minor imprecision during operations can lead to substantial deviations in the final results, especially in algorithms composed by highly iterative and complex numerical methods [2, 3].

In such computation-intensive scenarios, the representation of data is a fundamental concern for achieving accurate computational resolutions. Key questions arise: What numerical set is best suited for data codification? How many significant digits are necessary to ensure accurate calculations? Which representational format should be employed? Addressing these critical questions leads us to an appropriate numerical codification that strikes a delicate balance between accuracy and computational complexity.

In this paper, we propose a computation model that satisfactorily solves the accuracy requirements of a set of problems from the lowest level of the computer architecture. More specifically, the main contribution of this work is a new method for representing error-free rational numeric data, based on floating-point schemes. This method allows the development of operators for arithmetic functions in order to build a rational arithmetic unit. A typical arithmetic unit supports a small set of basic operators from which a large set of mathematical functions can be computed by function composition. For example, from the basic operators of addition and multiplication, more complex functions such as reciprocation, square root, exponentiation, or trigonometric functions can be obtained using the Newton–Raphson, Goldschmidt, or Taylor algorithms [4].

In spite real number set is more generic than rational one, real irrational numbers can be approximated by rational numbers as much as it is needed, and in addition, rational numbers can be represented in positional way with finite word length.

The proposed rational computational model constitutes an algebraic field with addition and multiplication operations where every nonzero rational number has multiplicative inverse and can be represented with the format.

This paper is structured as follows: Sect. 2 provides a review of the current state of knowledge on numerical representation and analyses the most relevant proposals of this topic, Sect. 3 develops the formal framework in which the model is constructed and describes the proposed numerical representation of rational numbers and their implementation, Sect. 4 shows a comparison with other formats, Sect. 5 develops an empirical evaluation, next, Sect. 6 describes an application example and finally, Sect. 7 summarizes the conclusions of this work.

2 Literature survey of number representation in a computer

2.1 Number representation

The relevance of the topic at hand is underscored by the multitude of proposals and works dedicated to resolving specific aspects of the problem. For applications that demand greater accuracy, the prevalent approach involves employing variable precision software tools for reasons of portability. These tools act as a layer of abstraction, furnishing structures and operations with adjustable lengths tailored to each application's requirements. Such tools encompass function libraries, various data types, arithmetic packages, and extensions for common programming languages like C, Pascal, Python, Fortran, etc. [5–7]. While they extend the range of representation formats to accommodate more significant digits and achieve greater accuracy, the computational complexity of their operations often limits their general use. Consequently, these software solutions increase computation runtimes compared to native machine numerical formats, and their final codification remains constrained by the formats and numerical representations supported by the underlying hardware, which adds its own restrictions.

The pursuit of high-precision calculations has led to the development of algorithms that work with data of considerable length, compensating for the limitations of standard representation formats. In this context, proposals for numerical decimal processing prove particularly interesting as they sidestep errors introduced by binary conversion of data [8–11]. Notable methods for calculating results with a large number of digits include the well-known algorithms of Newton–Raphson, Goldschmidt, CORDIC, and Taylor [11–13], as well as arithmetic on-line operation methods [14, 15], which perform operations from the most significant to the least significant digits. While some of these algorithms are included in the aforementioned software tools, they inherit the same limitations. Some hardware implementations have been proposed and developed, offering limited solutions, including those mentioned below.

In general, numerical codification formats are predominantly conditioned by the characteristics of the numerical sets they represent, with distinct formats for natural, integer, and real numbers. Fractional positional notation, used to represent real numbers, directly expresses the value of the number with an integer and a fractional part.

The simplest fractional representation is the fixed-point format, which sets a specific number of digits for the integer and fractional parts, leading to rounded fractional point representation, while advantageous for simplicity in operator design, fixed-point formats can only precisely represent numbers that align with discrete codifiable elements, approximating the remainder.

For a broader representation range, floating-point systems are particularly relevant. In floating-point systems, the integer part is minimized, and the fractional part is extensively developed, akin to scientific notation. This format is structured into sign, exponent, and mantissa fields. While it extends the expression capacity of fixed-point formats, it still establishes a discrete range of representation for real numbers in a computer. The widely adopted standard for floating-point number representation is IEEE-754 [16], encompassing various representation formats, rounding methods, and exceptions. However, finite precision sets employed in floating-point schemes do not allow representation of fractional values beyond the significant digits allowed by the format, leading to calculation errors [17–20] that prevents a satisfactory resolution for applications requiring high precision results. While these formats include methods to refine or round the codification to reduce errors, modifying the least significant digits of the representation compromises exact representation.

As the demand for computing capacities surpasses present systems, research is inclined toward designs dedicated to addressing hardware deficiencies. This leads to specific computer architectures with new functionalities tailored for resolving such problems. Numerical representation is closely related to this situation, with numerous valid solutions for specific environments. Alternative methods of numerical expression and arithmetic calculations implemented in specialized processor designs have been explored, and the progress made, along with notable deficiencies, are reviewed in the subsequent sections. These proposals are classified based on their expressiveness and capacity for number representation.

The method with the greatest expressivity is symbolic computation, characterized by exact representation using algebraic expressions for numerical values, forming an exact arithmetic scheme [21, 22]. Continuous fractions also offer an exact method of representation for rational numbers, encoding numbers through successive fractions of integers [23].

Another group of proposals relies on the arithmetic model of intervals, which defines the inaccuracy introduced when providing a numerical result. Numbers are expressed by the interval extremes in which they are found, codified using floating-point notation [24, 25]. Arithmetic operations are performed on the interval extremes, retaining results within the limits [26]. While this technique does not yield a single exact value, it maintains the error of the numerical result, delimited by the interval limits. Interval arithmetic provides guaranteed results, but it is not well suited for the validation of high precision. Instead, the round-off error propagation could be estimated using stochastic arithmetic and then increasing number certainty [27]. Some proposals combine interval arithmetic and symbolic notation, with lazy arithmetic [28] expressing results through symbolic mathematical expressions while providing numeric results as intervals. For higher precision requirements, new calculations can be performed to obtain improved approximations.

Other proposals aim to increase the precision of numerical values by using a greater number of significant digits to represent operands and results, attempting to achieve accurate approximations for specific problem requirements. However, such systems cannot encode numbers with an infinite number of fractional digits, leading to the loss of exact representation of rational values and processing errors. This group includes staggered arithmetic and on-line arithmetic: Staggered arithmetic [29] represents each number using a variable list of non-overlapping floating-point values that provide the value of the number they encode; On-line arithmetic [14, 15] processes operands serially, performing calculations digit-by-digit based on partial knowledge of the input data. This property allows for the design of segmented calculation methods, facilitating operations with a variable number of digits and establishing regular structures for arithmetic units.

Lastly, other numerical representations focus on improving computer performance through the simplification of basic operators and increasing the arithmetic unit's throughput. These proposals are useful for high-throughput arithmetic processes, such as in DSP applications, but might not prioritize accuracy. Examples include Residue Number System (RNS) [30], and Redundant Binary Representations (RBR) [31].

2.2 Findings

The exploration of these state-of-the-art representation methods demonstrates that conventional techniques and standard formats encounter challenges when representing numbers with infinite fractional digits. While certain specific proposals offer valid and widely accepted solutions, enhancing the expression capacity of new methods and formats often results in increased complexity. Moreover, for many applications, having a single number that represents the result's value is crucial.

Software-based representation proposals, though versatile and compatible with most systems for constructing specialized applications, fall short of meeting the performance expectations of some applications. In contrast, hardware low-level solutions, based on interval arithmetic and other variable significant digit methods, offer intriguing alternatives to improve result precision. However, they may not provide exact values, presenting approximations whose quality is challenging to gauge.

In conclusion, applications requiring high-performance computing and substantial calculation accuracy emphasize the need for an appropriate expression of operands and the development of corresponding low-level algorithms. This paves the way for conceiving an exact method of operand codification based on numerical representation, representing an improvement over current proposals. Such a method should provide knowledge of operand values and generated partial results, with effective adjustment or approximation policies according to the application's requirements. The subsequent section addresses the establishment of an appropriate representation format.

3 Rational exact representation format

3.1 Specification of the double mantissa format

Fixed and floating-point representation schemes show that although their number-set representation is contained in the rational set $\mathbb{Q}$, there are infinite rational numbers that cannot be expressed in these formats, although arbitrary lengths of the binary word are permitted. That is, as much as the amount of digits represented is increased, it is impossible to codify such values exactly [8, 17, 18]. Nevertheless, the codification of rational numbers is of particular interest because it is the largest subset of $\mathbb{R}$, where the exact value of their elements can be written in a positional representation format. The inherent characteristics of the rational numbers in $\mathbb{Q}$ suggest the possibility of obtaining a representation model that fulfils the following objectives:

1. To contain the exact positional notation of the numbers representing them in a direct way.
2. To allow an indeterminate number of exact fractional digits to be obtained according to the requirements of each application.
3. To reach very high or very low extreme values.

The proposed number representation is based on an extension of the floating-point scheme, where it is considered both the fixed and the periodic expansion of the positional representation of rational numbers. This proposal develops a representation format for high precision computing proposed in our previous researches [32, 33].

The number's mantissa is obtained by concatenation of the fixed mantissa (m_f) and the periodic mantissa (m_p) for an indefinite number of times. Expression 1 illustrates its construction.

$$\text{Mantissa } (M) = m_\mathrm{f} m_\mathrm{p} m_\mathrm{p} m_\mathrm{p} m_\mathrm{p} m_\mathrm{p} \ldots = m_\mathrm{f} \hat{m}_\mathrm{p} \tag{1}$$

The value of the rational number is now obtained according to the same expression as with the floating-point format:

$$x \in \mathbb{Q}, x = (-1)^{s} \cdot M \cdot B^{E}, \tag{2}$$

where B is the numerical base of the representation, M is the complete mantissa formed by the concatenation of the fixed mantissa and the periodic mantissa for an infinite number of times, and E is the exponent. The exact value of the complete mantissa can be calculated by means of the following expression:

$$M = \frac{m_f m_p - m_f}{\underbrace{(B-1)\cdots(B-1)}_{M_P\mathrm{WL}}\ \underbrace{0\cdots 0}_{M_F\mathrm{WL}}} \tag{3}$$

where M_PWL is the length of the periodic mantissa, $M_F$wl is the length of the fixed mantissa, $m_f m_p$ is the one-time concatenation of the fixed mantissa and the periodic mantissa, and the denominator is formed by the concatenation of the digits as often as indicated.

To avoid multiple representations of the numbers, the format imposes the following normalization conditions:

1. The mantissa of the number is normalized by positioning the first significant digit to the right of the fractional point. If the fixed mantissa does not exist, the periodic mantissa is then normalized by rotating its digits to the left.

$$M = 0, m_f \hat{m}_p / M \in [B^{-1}, 1[\tag{4}$$

2. When the fixed mantissa or periodic mantissa does not exist, their corresponding field in the codification will be left empty.

Please, note that the cases when the periodic mantissa is 0 or $(B-1)$ are the same. The periodic mantissa equal to 0 is the general case when the number has not period. And the case when the periodic mantissa is equal to the base representation minus one (‘1’ in binary representation base), it is achieved just by adding ‘one’ to lest significant bit of the fixed mantissa.

3. The fixed mantissa cannot contain groups of digits that match with the periodic mantissa in its least significant part:

$$\begin{aligned} m_f &= \alpha_{M_f\mathrm{WL}-1} \cdots \alpha_1 \alpha_0 \\ m_p &= \gamma_{M_p\mathrm{WL}-1} \cdots \gamma_1 \gamma_0 \end{aligned} \tag{5}$$

Then,

$$\forall i \in \left[0..M_p\mathrm{WL} - 1\right], \quad \alpha_{i-1} \ldots \alpha_1 \alpha_0 \neq \gamma_{i-1} \cdots \gamma_1 \gamma_0$$

4. The periodic mantissa must have the minimum number of digits, that is, it should not be composed of smaller periodic sub-mantissas.
$$m_{\mathrm{p}} = \gamma_{M_{\mathrm{p}}\mathrm{WL}-1} \cdots \gamma_1\gamma_0$$

Then,

$$\begin{aligned}
&\forall i \in [1..M_{\mathrm{p}}\mathrm{WL}-1]\,/M_{\mathrm{p}}\mathrm{WL} \bmod i = 0 \Rightarrow \\
&\Rightarrow \gamma_{i-1}\cdots\gamma_1\gamma_0 \neq \gamma_{2i-1}\cdots\gamma_1\gamma_i\wedge \\
&\wedge\gamma_{i-1}\cdots\gamma_1\gamma_0 \neq \gamma_{3i-1}\cdots\gamma_{2i+1}\gamma_{2i}\wedge \\
&\cdots \\
&\wedge\gamma_{i-1}\cdots\gamma_1\gamma_0 \neq \gamma_{M_{\mathrm{p}}\mathrm{WL}-1}\cdots\gamma_{M_{\mathrm{p}}\mathrm{WL}-i+2}\gamma_{M_{\mathrm{p}}\mathrm{WL}-i+1}
\end{aligned} \tag{6}$$

The number of digits, $M_{\mathrm{p}}\mathrm{WL}$, that forms the period of a rational number is closely related to the denominator of the irreducible fraction that represents the rational number according to the following expression:

$$\begin{aligned}
&\forall x = \alpha_{M_{\mathrm{f}}\mathrm{WL}}\alpha_{M_{\mathrm{f}}\mathrm{WL}-1}\ldots\alpha_i,\alpha_{i-1}\ldots\alpha_1\alpha_0\widehat{\gamma_{M_{\mathrm{p}}\mathrm{WL}-1}\cdots\gamma_1\gamma_0} \in \mathbb{Q}, \exists \mathrm{a},\mathrm{b} \in \mathbb{Z}/ \\
&/x = \frac{a}{b} \wedge \mathrm{b} \neq 0 \wedge \mathrm{mcd}(a,b) = 1 \Rightarrow M_{\mathrm{p}}\mathrm{WL} = \Im(b) < b
\end{aligned} \tag{7}$$

The number of periodic fractional digits is always smaller than b. In particular, the definition of the function $\Im$ that determines $M_{\mathrm{p}}\mathrm{WL}$ consists of the following congruence relations:

$$\begin{aligned}
&\text{if} \quad \mathrm{mcd}(B,b) = 1 \Rightarrow B^{M_{\mathrm{p}}\mathrm{WL}} \equiv 1 \bmod (b) \\
&\text{else} \quad \exists b', m \in \mathbb{Z}/b = \mathrm{m}\cdot b'\wedge \\
&\mathrm{mcd}\big(B,b'\big)b = 1 \wedge \mathrm{m} = \prod_{i=1}^{p} f_i^j
\end{aligned} \tag{8}$$

where $f_1, \ldots, f_{\mathrm{p}}$ id the list of prime factors of the representation base B, and $j \in \mathbb{N}$. Thus:

$$\frac{a}{b} = \frac{1}{m}\cdot\frac{a}{b} \wedge \mathrm{mcd}\big(B,b'\big) = 1 \wedge B^{M_{\mathrm{p}}\mathrm{WL}} \equiv 1 \bmod \big(b'\big) \tag{9}$$

Thus, the nonzero period of the periodic numbers is related to the representation base. This observation is of great relevance for numerical representation and computer arithmetic, since in a binary domain there are fractional numbers of zero period in decimal that will produce periodic expansion in binary representation. Hence, the importance of defining a model that allows to represent and operate with these numbers accurately.

3.2 Properties of the representation format

The proposed format covers the whole $\mathbb{Q}$ set as each number can be represented exactly. In this way, the proposed representation function, $\Gamma_{\mathrm{identity}}$, is a bijective application among the rational number set.

$$\Gamma_{\text{identity}} : \mathbb{Q} \to \mathbb{Q} \tag{10}$$

This function complies with the following properties:

Injective: Each rational value has a different codification. The amount of fractional digits of rational numbers is finite or it is periodically infinite within a finite period.

$$\forall x_1, x_2 \in \mathbb{Q}, \Gamma_{\text{identity}}(x_1) \neq \Gamma_{\text{identity}}(x_2) \tag{11}$$

Surjective: Any rational value can be represented with the proposed format.

$$\forall x \in \mathbb{Q}, \exists\ s, E, m_{\text{p}}, m_{\text{f}} \in \mathbb{Z}/\Gamma_{\text{identity}}(x) = x \tag{12}$$

Existence of inverse: Due to the conjunction of the two other properties, (11) and (12), the representation function is provided with the inverse of the $\mathbb{Q}$ set, that is, it is possible to construct a function that obtains the initial rational value for each number codification.

$$\exists \Gamma^{-1}_{\text{identity}} : \mathbb{Q} \to \mathbb{Q}/\forall x \in \mathbb{Q}, x = \Gamma^{-1}_{\text{identity}}\left(\Gamma_{\text{identity}}(x)\right) \tag{13}$$

As a result of these properties, any rational number that is normalized according to the format will have a characteristic expression comprising a sign, an exponent, a fixed mantissa, and a periodic mantissa. Therefore, this implementation constitutes an exact evaluation of the identity function.

The format must provide special codifications that represent special situations or exceptional cases, e.g., codification of zero, overflow, underflow, and error result. Special codes of the exponent are allowed to express these situations, similar to the conventional floating-point formats.

3.3 Hardware implementation

In this subsection, we describe a specific structure of the registers comprising the flexible storage space that can store the representation format data. The proposed implementation is one of several that are possible for this format. The binary base is adopted for encoding number representations for performance reasons.

The proposed format distributes the significant digits of the representation of the number into four parts: sign (1 bit), exponent (EWL bits), fixed mantissa (M_{f}WL bits), and periodic mantissa (M_{p}WL bits), where WL is the word length. The fixed mantissa (m_{f}) constitutes the fractional part of the non-periodic rational number, whereas the periodic mantissa (m_{p}) represents the digits that form the repetitive part. The exponent, as in the floating-point format, expresses the order of magnitude of the number. With this technique, numbers with infinite fractional digits are obtained from a finite codification and their exact representation can be obtained by the ALU. The following figure shows a diagram of the representation scheme (Fig. 1).

The proposed design for hardware implementation is based on the layout of the fields that make up the number in a finite length word and that provide flexibility in the distribution of fractional numbers between the fixed and periodic mantissa.

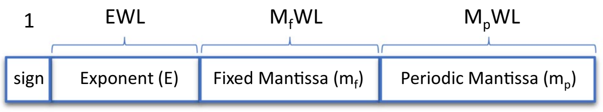
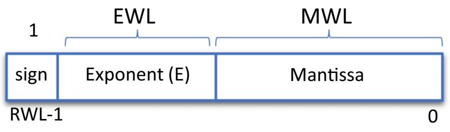


Fig. 1 General scheme of the format of the double mantissa representation

Fig. 2 Distribution of the digits of the number fields



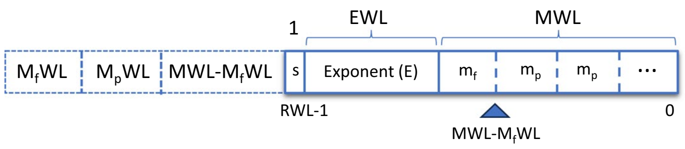


Fig. 3 Structure of the implementation of the double mantissa format

Therefore, the parts of the number are placed consecutively in a fixed-RWL register. This length must be previously set, depending on the application requirements. The sign (1), the exponent (EWL), and the mantissa (MWL) are provided with a number of positions determined for their representation.

The following figure shows a diagram of the register that contains the information with the distribution of the lengths in their fields. The mantissa is formed by the fixed and periodic digits of the number (Fig. 2):

As can be seen from the previous figure, the representation of the exponent is made in the assigned space. Any integer number codification format will be used to complete all of the reserved positions.

The flexible division of the mantissa digits into fixed and periodic parts requires a pointer, which marks the separation between the two parts and which enables separate processing. To complete all of the digits assigned to the mantissa field, the fractional digits of the period are concatenated forming a cycle, and their lengths are stored with the previous pointer. These three segments of the data are associated with the register that contains the number and are placed adjacent to it, as can be seen in the structure illustrated in the following figure (Fig. 3):

The mantissa of the numbers without a fixed part is constructed only by means of concatenation of the digits of the periodic mantissa. In this case, the pointer that marks the separation between the mantissas takes the value MWL.

When the length of any part of the number exceeds the register length, its exact expression is not possible and it will be necessary to adjust the codification to the closest representable value. The proposed criterion is intended to completely represent the exponent and the sign and to apply the cut in the mantissa. When the mantissa is affected by the limitation of MWL digits, its approximate expression is

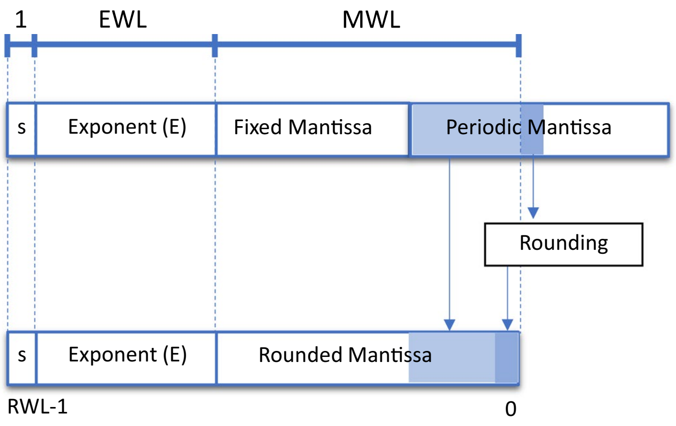


Fig. 4 Transference of digits between period and fixed mantissas and round off

considered, and the last digits are ruled out following the order of magnitude of the digits. The situations that might arise are as follows:

- If the amount of < EWL, MWL > digits is enough for the complete codification of the exponent and the mantissa overall, its exact representation may be made.
- If the amount of MWL digits is not enough for the complete representation of the mantissa of the number, an approximate representation is obtained. Two scenarios are proposed in the codification according to the number field affecting the reduction of digits. The first is shown in Fig. 4.
- *Case I* If full codification of the periodic mantissa is not possible, then it is eliminated. A zero value is assigned to the indicator of the period length, and its digits are used to construct an extended fixed mantissa, which fulfils the total length available. Then, the approximate value of the mantissa is rounded. As the following figure shows, the rounding is made with the period digits remaining outside the limit. Any rounding method referred to in the bibliography may be applied using the necessary amount of digits [34].
- *Case II* If the complete codification of the fixed mantissa is not possible, then the periodic mantissa is directly rejected by assigning zero to its length indicator, and the fixed mantissa is then adjusted by rounding off the remaining digits.

In both cases, the position of the start of the periodic mantissa, which is marked by the associated pointer, is irrelevant.

- Finally, if the amount of available EWL digits does not even accommodate the representation of the order of magnitude, the number is not representable with the conditions imposed. In this case, the result will show an error.

The characteristics of the representation format ensure that there exists a register size that is suitable for the exact representation of each set of rational numbers.

Table 1 Format's field size

Representation format	Mantissa	Exponent
IEEE simple precision [16]	23 (binary)	8
IEEE double precision [16]	52 (binary)	11
IEEE quadruple precision [16]	113 (binary)	15
IEEE octuple precision [16]	237 (binary)	19
IEEE dec64 [16]	16 (decimal)	10
IEEE dec128 [16]	34 (decimal)	14
CADAC [36]	6..106 (binary)	10
M.J. Schulte [26]	64..4160(binary)	16
Proposed	M_fWL + M_pWL (binary)	EWL

However, it is crucial to find a length that strikes a balance between representation complexity and the required expressive capacity for each application. Moreover, the placement of these registers will have an impact on the system's performance. It is advisable to position the register bank within the arithmetic unit and in close proximity to the operation hardware.

4 Comparison with others floating-point formats

The implementation features described in the previous section establish the conversion rules of the proposed format to any floating-point format, including the IEEE formats. In these cases, the field lengths and the representation base will specify the representation scope of the format.

The following table shows a comparison between the field lengths used in some floating-point formats Table 1.

We have observed the tendency of the formats to use more and more bits in the codification of the numbers to obtain more precise results. However, the requirements of each application will mark the necessary precision, and the hardware restrictions will limit the number of bits.

Considering the whole mantissa, both exact and approximate representations, is placed on the same level as the proposed double mantissa representation format and the classic floating-point format. Moreover, it has the additional capacity of coding a set of periodic rational numbers without error. Specifically, if the record structure is set according to the size of the fields according to the IEEE-754 standard, a compatible representation is returned. This fact allows numbers to be processed with existing algorithms and complete the rational computing arithmetic model.

However, advantages are obtained if suitable methods are used for the new format to take advantage of their ability to express and produce accurate results.

Table 2 shows some codification examples made by the representation format in comparison with the standard representation in floating point. The sample aims to be as heterogeneous as possible and covers periodic and non-periodic rational numbers

Table 2 Rational number representation

Decimal number	Binary number expansion	IEEE 754 simple precision codification			Proposed format		
		Exp	Mantissa	Error	Exp	Fixed mantissa	Periodic mantissa
0.125	0.001	01111100	00000000000000000000000	0	110	1	–
0.2	0.0011001100110011…	01111100	10,011,001,100,110,011,001,101	$1.19{\cdot}10^{-8}$	110	–	1100
0.6	0.1001100110011001…	01111110	00110011001100110011010	$7.15{\cdot}10^{-8}$	0	–	1001
2.18	10.0010111000010100011111…	10,000,000	00010111000010100011111	$1.72{\cdot}10^{-7}$	010	100	0101110000 1,010,001,111
$0.\hat{3}$	0.1010101010101010…	01111101	01010101010101010101011	$1.67{\cdot}10^{-8}$	0	–	10
$7.\hat{5}$	111.1000111000111…	10,000,001	11,100,011,100,011,100,011,100	$2.11{\cdot}10^{-7}$	011	1	111,000
$37.\widehat{21}$	100,101.0011011001…	10,000,100	00101001101100100110110	$1.16{\cdot}10^{-6}$	0110	1001	0100110110
$64,01\widehat{369863}$	1,000,000.000000111	10,000,101	00000000000011100000100	$3.86{\cdot}10^{-6}$	0111	1,000,000	000000111

both in decimal and binary. The exponent of the proposed method is represented in sign/magnitude format.

The previous table shows IEEE 754 Simple Precision codification for simplicity, but accuracy problems would be the same as in the new IEEE release [16] and extended binary formats. From the results of this Table 2, the following facts can be derived:

- The proposed method is able to represent numbers with an infinite number of digits within a discrete representation range.
- Errors produced in the decimal–binary–decimal conversion are avoided.
- The proposed method allows the exact representation of numbers for which conventional techniques produce errors.

5 Empirical evaluation

In this work, we performed a series of empirical tests to analyze the functional behavior of the proposed format. The experiments were based on a study of the codification of the significant part of the mantissa, assuming that both the exponent and the sign were represented correctly in all cases. The experiments were conducted by means of a simulation in a C programming environment.[1] They are divided into two sets.

Experiment Set I Studying the number of digits necessary for codification of the rational values.

The tests were performed studying the relationship that exists between the area complexity and the numerator and the denominator of the fraction that produces the rational value (*a/b*). The results obtained are shown in Figs. 5 and 6. Figure 5 shows the increase in the number of digits necessary to contain the fixed mantissa with regard to the fraction's numerator. It grows according to the integer logarithm of the numerator in the same way as in conventional formats. The denominator does not influence the growth of this mantissa.

Figure 6 shows a linear growth of the number of digits of the periodic mantissa with regard to the fraction denominator. This means that in the worst case, the number of digits of the period depends directly on its magnitude. When there are timing or area restrictions, this growth can be a disadvantage, which should be taken into account when designing the solution.

Experiment Set II Comparing the proposed method and the conventional methods of representation as used in most general-purpose computers, i.e., the standard IEEE 754.

Conclusions were drawn from this test about the expressiveness of this format and about the deviations that take place in rational numerical representation.

The profile of this test is defined as the codification of 10^7 random rational numbers (non-periodic and periodic) in the IEEE 754 representation format in

[1] C++ for Windows in Microsoft Visual Studio.

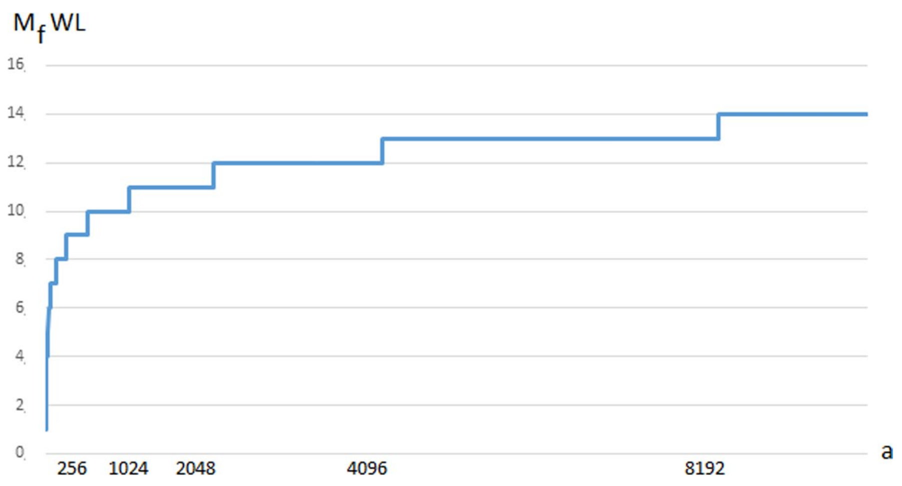


Fig. 5 Growth of digit of fixed mantissa

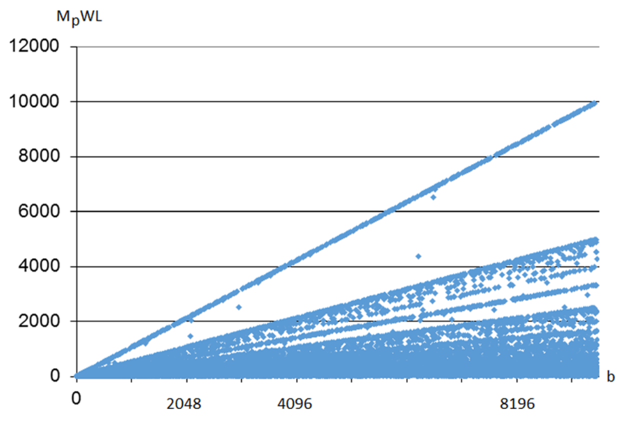


Fig. 6 Growth of digit of periodic mantissa

simple precision (32 bits) and double precision (64 bits). Each non-periodic number belongs to the interval [0, 1] and consists of 128 random significant fractional bits. Each periodic number belongs to the interval [0, 1] and is built by means of a fraction 1/b, where b is a 16-bit random integer value that is not a power of 2.

In all the cases, the proposed format can represent all the generated numbers without error by means of the double mantissa representation as shown in Table 2. Thus, the aim of these tests is to check the IEEE 754 format representations while taking the proposed method as a valid reference in the error-free codification of rational numbers. From this comparison, an absolute measurement of the error committed by the IEEE 754 simple and double precision formats are

Table 3 First incorrect bit average

IEEE format	Non-periodic numbers		Periodic numbers	
	Position	σ	Position	σ
IEEE 754 simple precision	23.36	8.97	23.08	8.34
IEEE 754 double precision	52.01	15.05	51.82	14.88

Table 4 Average error in IEEE 754 representation

IEEE format	Non-periodic numbers		Periodic numbers	
	Position	σ	Position	σ
IEEE 754 simple precision	$2.97{\cdot}10^{-8}$	$2.05{\cdot}10^{-13}$	$2.97{\cdot}10^{-8}$	$1.18{\cdot}10^{-13}$
IEEE 754 double precision	$5.55{\cdot}10^{-17}$	$1.85{\cdot}10^{-33}$	$5.54{\cdot}10^{-17}$	$1.03{\cdot}10^{-33}$

obtained as well as the deviation on the correct codification in terms of the position of the first incorrect bit.

Table 3 shows the average of the first incorrect position from the mantissa. In all of the cases, its value is very near the length of the mantissa. It represents a measurement of the similarity between the exact value and the true value represented, although it is not indicative of the error committed in the representation.

Table 4 shows the average error produced in the mantissa's magnitude. These errors should be taken into account as much as they are influenced by an exponent that is able to amplify their magnitude. Such imprecision makes us reflect on the suitability of using floating-point formats in the data representation.

The calculation methods with this representation format should have the capability to deliver the precise result of the operation within a finite word size, thereby bypassing the need for a final rounding stage; the length of the exact result, represented in a positional notation system, is directly proportional to the initial size of the operands; and the development of strategies to fine-tune precision and the result's length is achievable by iterative calculation methods of the mantissas. The utilization of iterative structures and pre-calculated data can serve as a means of achieving flexibility in designs and adjusting the result according to the unique requirements of each application [32, 35].

6 Application example and discussion

In this section we describe an application example in which the proposed computational model has advantages over other number representation formats. The working scenario could be to a stock market which perform a lot of daily operations. The exactness of number representation is essential to maintain the accuracy of prices, ensure correct assessment, and determine the evolution of the value of the stocks. In

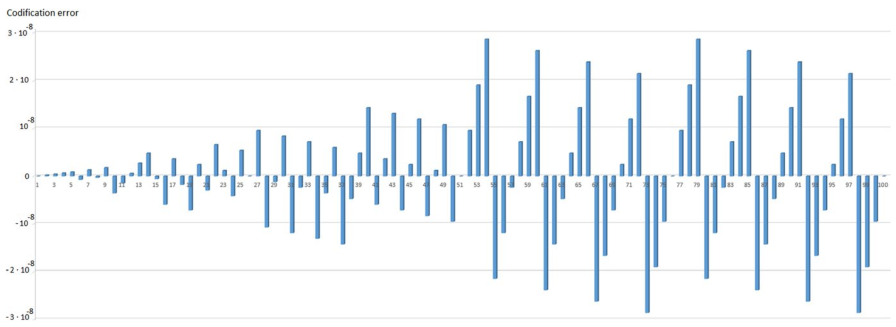


Fig. 7 Representation error for all multiples of euro cent with IEEE SP format

this case, it is considered that the rational domain of the numbers is euro cents, that is, two decimal fractional digits.

The rational representation of these numbers by IEEE binary formats is not exact for the majority of the numbers with this precision. Figure 7 shows the representation error for all multiples of euro cent with IEEE Simple Precision format (32 bits length).

The average error of the previous representation is $\sim 10^{-8}$ and the accumulated error, if addition of absolute values is considered, it is $\sim 9.9 \cdot 10^{-7}$. These results show, once again, the insufficiency of binary positional numeric formats to represent rational numbers exactly, regardless of the number of digits used to it.

The representation of cents has the following positional numeric expression[2]:

$$0.01€ = 0.00\ 0000101\widehat{000111}1010111_2$$

By means the proposed method, encoding these numbers required only 22 significant digits with the following distribution between fixed and periodic mantissas:

$$0.01€ = 0.00_{m_f}00001010001111010111_{m_p}$$

All fractional numbers required in pricing will be a multiple of the previous one, and therefore, our method does not need more fractional digits for coding them exactly.

Although there are other radix-10 encoding formats in the latest revision of the IEEE 754 standard able to represent numbers of these characteristics, they cannot resolve the problem discussed in this example for all cases, as it can produce the results with decimal periodic digits that cannot be encoded in these formats. Such as the rational number (1/3) outlined in the introduction to this paper.

[2] The numerical positional expansion of the fractions [1/*a*] where a∈[1..1000] is provided as supplementary material.

7 Conclusions

In this study, we introduced a binary representation format for rational numbers. The core concept revolves around leveraging the mathematical characteristics of their fractional representation and separately codifying the fixed and periodic digits. As a result, rational numbers are furnished with a positional representation of their value, featuring a finite number of significant fractional digits, which allows for their codification within a finite material space representation. This floating-point coding surpasses classic fixed or floating-point representations, which can be seen as specific cases thereof, by accurately codifying a broader numeric set without errors.

One of the primary advantages of the proposed format is its ability to circumvent errors arising from the human user's introduction of data into a computer, where the base codification changes from decimal to binary. When applied to systems with area restrictions, the method behaves similarly to other floating-point formats, incorporating a rounding process that grants it variable precision properties. Simulation tests convincingly demonstrate its superior expression capacity when compared to conventional binary methods, and the application example showcases a real case of number codification. For these reasons, our format presents a viable alternative to symbolic calculation for exact processing.

In the future, we will work on the design of basic arithmetic operators in order to build a minimum operative set of instructions to compute high-precision demanding applications. We specially will focus on the multiplicator operator, since this function is the key for developing other advanced elementary functions.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s11227-023-05814-y.

Acknowledgements This work was supported by the Spanish Research Agency (AEI) (https://doi.org/10.13039/501100011033) under project HPC4Industry PID2020-120213RB-I00.

Author contributions All authors have contributed to all sections of the document. H.M. conduct the experiments. All authors reviewed and accepted the manuscript.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Data availability All data has been generated through the methods described in this paper.

Declarations

Competing interests The authors declare no competing interests.

Conflict of interest The authors declare that they have no conflict of interest.

Consent to participate The authors declare that they agree to participate.

Consent for publication The authors declare that they agree to publish.

References

1. Bailey DH (2005) High-precision floating-point arithmetic in scientific computation. Comput Sci Eng 7(3):54–61
2. Javidi M, Saedshoar Heris M (2023) New numerical methods for solving the partial fractional differential equations with uniform and non-uniform meshes. J Supercomput 79:14457–14488. https://doi.org/10.1007/s11227-023-05198-z
3. Rico-Garcia H, Sanchez-Romero JL, Jimeno-Morenilla A, Migallon-Gomis H, Mora-Mora H, Rao RV (2019) Comparison of high-performance parallel implementations of tlbo and jaya optimization methods on manycore GPU. IEEE Access. https://doi.org/10.1109/ACCESS.2019.2941086
4. Ercegovac MD, Lang T, Muller J-M, Tisserand A (2000) Reciprocation, square root, inverse square root, and some elementary functions using small multipliers. IEEE Trans Comput 49(7):628–637. https://doi.org/10.1109/12.863031
5. Fousse L, Hanrot G, Lefèvre V, Pélissier P, Zimmermann P (2007) MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Trans Math Softw 33(2):13. https://doi.org/10.1145/1236463.1236468
6. Bailey DH, Li XS, Hida Y, Thompson B (2002) ARPREC: an arbitrary precision computation package. Software available online: http://crd-legacy.lbl.gov/~dhbailey/mpdist, Accessed on 30 Sept 2023
7. Dhawale PG, Kamboj VK, Bath SK (2023) A levy flight based strategy to improve the exploitation capability of arithmetic optimization algorithm for engineering global optimization problems. Trans Emerg Telecommun Technol. https://doi.org/10.1002/ett.4739
8. Bohlender G (1990) What do we need beyond IEEE arithmetic? Computer arithmetic and self-validating numerical methods. Academic Press, Boston
9. Muhamed FM (2023) Exact versus inexact decimal floating-point numbers and arithmetic. IEEE Access 11:17891–17905. https://doi.org/10.1109/ACCESS.2023.3244891
10. Chu Z, Li Z, Xia Y, Wang L, Liu W (2021) BCD adder designs based on three-input XOR and majority gates. IEEE Trans Circuits and Syst II: Express Br. https://doi.org/10.1109/TCSII.2020.3047393
11. Sanchez-Romero JL, Mora H, Mora-Pascual J, Jimeno-Morenilla A (2011) Function approximation on decimal operands. Digit Signal Proces 21(2):354–366. https://doi.org/10.1016/j.dsp.2010.06.013
12. Changela A, Zaveri M, Verma D (2023) A comparative study on CORDIC algorithms and applications. J Circuits, Syst Comput 32(05):2330002. https://doi.org/10.1142/S0218126623300027
13. Sanchez-Romero JL, Mora H, Mora-Pascual J, Jimeno-Morenilla A (2008) Architecture Implementation of an Improved Decimal CORDIC Method. In: IEEE International Conference on Computer Design. Lake Tahoe, CA USA. https://doi.org/10.1109/ICCD.2008.4751846
14. Usman M, Ercegovac M, Lee JA (2023) Low-latency online multiplier with reduced activities and minimized interconnect for inner product arrays. J Signal Process Syst. https://doi.org/10.1007/s11265-023-01856-w
15. Ercegovac MD (2020) On Reducing Module Activities in Online Arithmetic Operations. In: 2020 54th Asilomar Conference on Signals, Systems, and Computers. IEEE. Pacific Grove, CA, USA. p 524–528. https://doi.org/10.1109/IEEECONF51394.2020.9443576
16. IEEE Std 754-2019 (2019) Institute of Electrical and Electronic Engineers Standard for Floating-Point Arithmetic, Institute of Electrical and Electronics Engineers, New York, NY, USA. https://doi.org/10.1109/IEEESTD.2019.8766229

17. Goldberg D (1991) What every computer scientist should know about floating-point arithmetic. Comput Surv 23(1):5–48
18. Lafage V (2020) Revisiting what every computer scientist should know about floating-point arithmetic. arXiv:2012.02492. https://doi.org/10.48550/arXiv.2012.02492
19. Kneusel RT (2017) Arbitrary precision floating-point. Numbers and computers. Springer, Cham. https://doi.org/10.1007/978-3-319-50508-4_9
20. Zhang Z, Xu J, Hao J et al (2023) Hierarchical search algorithm for error detection in floating-point arithmetic expressions. J Supercomput. https://doi.org/10.1007/s11227-023-05523-6
21. Meurer A et al (2017) SymPy: symbolic computing in python. PeerJ Comput Sci. 3:e103. https://doi.org/10.7717/peerj-cs.103
22. Campbell JM, Cantarini M, D'Aurizio J (2022) Symbolic computations via fourier-legendre expansions and fractional operators. Integral Transform Spec Funct 33(2):157–175. https://doi.org/10.1080/10652469.2021.1919103
23. Ibran Z, Aljatlawi E, Awin A (2022) On continued fractions and their applications. J Appl Math Phys 10:142–159. https://doi.org/10.4236/jamp.2022.101011
24. Ganesan K, Veeramani P (2005) On arithmetic operations of interval numbers. Int J Uncertain, Fuzziness Knowl-Based Syst 13:619–631
25. Revol N, Benet L, Ferranti L, Zhilin S (2023) Testing interval arithmetic libraries, including their IEEE-1788 compliance. In: Wyrzykowski R, Dongarra J, Deelman E, Karczewski K (eds) Parallel processing and applied mathematics. Springer, Cham. https://doi.org/10.1007/978-3-031-30445-3_36
26. Schulte MJ (2000) A family of variable-precision interval arithmetic processors. IEEE Trans Comput 49(5):1–11. https://doi.org/10.1109/12.859535
27. Graillat S, Jézéquel F, Wang S et al (2011) Stochastic arithmetic in multiprecision. Math Comput Sci 5:359–375. https://doi.org/10.1007/s11786-011-0103-4
28. Gianantonio PD, Lanzi PL (2004) Lazy algorithms for exact real arithmetic. Electron Notes Theor Comput Sci 104:113–128. https://doi.org/10.1016/j.entcs.2004.08.021
29. Priest DM (1991) Algorithms for Arbitrary Precision Floating Point Arithmetic. In: Symposium of Computer Arithmetic. p. 132–143 https://doi.org/10.1109/ARITH.1991.145549
30. Rubia JJ, Shibi CS, Balajishanmugam V, Lincy RB (2023) High-performance computing based on residue number system: a review. Int Conf Adv Comput Commun Syst (ICACCS). https://doi.org/10.1109/ICACCS57279.2023.10112959
31. Li B, Wang J, Ding G, Fu H, Lei B, Yang H, Bi J, Lei S (2021) A high-performance and low-cost montgomery modular multiplication based on redundant binary representation. IEEE Trans Circuits Syst II: Express Br. https://doi.org/10.1109/TCSII.2021.3053630
32. Mora H, Mora-Pascual J, García-Chamizo JM, Signes-Pont MT (2017) Mathematical model and implementation of rational processing. J Comput Appl Math 309:575–586. https://doi.org/10.1016/j.cam.2016.05.001
33. Mora-Mora H, Mora-Pascual J, García-Chamizo JM, Jimeno-Morenilla A (2006) Real-time arithmetic unit. Real-Time Syst 34:53–79. https://doi.org/10.1007/s11241-006-8753-z
34. Thompson SR, Stine JE (2020) A Novel rounding algorithm for a high performance IEEE 754 double-precision floating-point multiplier. IEEE Int Conf Comput Des (ICCD). https://doi.org/10.1109/ICCD50377.2020.00081
35. Mora H, Mora-Pascual J, Signes-Pont MT, Sánchez-Romero JL (2010) Mathematical model of stored logic based computation. Math Comput Model 52(7):1243–1250. https://doi.org/10.1016/j.mcm.2010.02.034
36. Cohen MS, Hull TE, Hamacher VC (1983) CADAC: a controlled-precision decimal arithmetic unit. IEEE Trans Comput C–32:370–377. https://doi.org/10.1109/TC.1983.1676238