Mathematical model and implementation of rational processing

H. Mora, J. Mora-Pascual, J.M. García-Chamizo, M.T. Signes-Pont

PII: S0377-0427(16)30218-7
DOI: http://dx.doi.org/10.1016/j.cam.2016.05.001
Reference: CAM 10631

To appear in: *Journal of Computational and Applied Mathematics*

Received date: 10 November 2015
Revised date: 27 April 2016

1. It is introduced a formal framework for processing rational numbers.
2. A representation system based on positional notation system is described.
3. A method for calculating the addition funcion is detailed.
4. Experiments and application example have been made to validate the model.

Mathematical Model and Implementation of Rational Processing

H. Mora[b,*], J. Mora-Pascual[b], J.M. García-Chamizo[b], M.T. Signes-Pont[b]

(b) Specialized Processor Architectures Laboratory, Department of Computer Technology, University of Alicante, San Vicente del Raspeig, Alicante (E-03080). Spain

Abstract.-

Precision in computations is a considerable challenge to adequately addressing many current scientific or engineering problems. The way in which the numbers are represented constitutes the first step to compute them and determines the validity of the results. The aim of this research is to provide a formal framework and a set of computational primitives to address high precision problems of mathematical calculation in engineering and numerical simulation. The main contribution of this research is a mathematical model to build an exact arithmetical unit able to represent without error rational numbers in positional notation system. The functions under consideration are addition and multiplication because they form an algebraic commutative ring which contains a multiplicative inverse for every non-zero element. This paper reviews other specialized arithmetic units based on existing formats to show ways to make high precision computing. It is proposed a formal framework of the whole arithmetic architecture in which the operators are based. Then, the design of the addition operation is detailed and its hardware implementation is described. Finally, extensive evaluation of this operator is performed to prove its ability for exact processing.

Keywords.- computer arithmetic, computational methods, exact computation, rational processing, mathematical model, computational efficiency

1 Introduction

1.1. Need for precision in calculations

High precision computing is a very active research area due to the number of interesting applications that need it. For example, research about the nature of matter in the LHC or the search for extrasolar planets requires complex numerical calculations on the edge of the range of representation formats. Without going so far, other much more common everyday operations, such as financial calculations, also require a good precision to avoid inadmissible computing and rounding errors. The following table (Table 1) shows some examples of the representation error by floating point binary format codification and operations with simple decimal data. It shows how the same number can have multiple different binary representations and how the accumulative operations increase the error.

[*] Corresponding autor
Email addresses: hmora@ua.es (H. Mora[*]), jeronimo@ua.es (J. Mora Pascual), juanma@ua.es (J.M. García Chamizo), teresa@ua.es (M.T. Signes Pont)

Table 1: Error representation of decimal numbers coded in simple precision binary floating point format

	Decimal Number	Binary floating point representation	Error
a	0.6	0.600000023841857	2.3E-8
b	0.1	0.100000001490116	1.4E-9
c	0.7	0.699999988079071	1.1E-8
d	0.4	0.400000005960464	5.9E-9
e	0.04	0.039999999105930	1.0E-10
a + b	0.7	0.700000025331973	2.5E-8
a + a	1.2	1.200000047683715	4.7E-8
d + d + d	1.2	1.200000017881392	1.7E-8
b + c + d	1.2	1.199999995529651	4.5E-9
e + ... + e (10 times)	0.4	0.399999991059300	8.9E-9

The introduction of decimal representation formats has significantly improved the accuracy of applications for processing numerical data introduced by humans through a terminal [1]. The programming languages and database systems include among their types the new data money or decimal to represent 10-base numeric data [2-4]. New processors support these formats and offer a wide instruction set for native execution [5-9]. Although the computational cost is superior to binary, for certain applications, most precision compensates this reduced performance.

However, in other areas the problem of precision remains unresolved since their numeric range of the operands and results are found into the periodic rational or irrational real domain. In these cases, the numbers do not correspond with representable values in binary format nor decimal. For example, next table (Table 2) shows some rational numbers represented in binary and decimal floating point format. These cases make clear the codification and operation error by both standard formats.

Table 2: Error representation of rational numbers coded in simple precision binary and decimal floating point format

	Rational Number	Binary floating point representation	Error	Decimal floating point representation	Error
a	1/3	0.33333334326744	1.0E-8	0.3333333	3.3E-8
b	1/7	0.14285714924335	6.4E-9	0.1428571	4.2E-8
c	11/21	0.52380955219268	2.8E-8	0.5238095	2.3E-8
d	1/6	0.16666667163372	4.9E-9	0.1666667	3.3E-8
a+b+c	1	1.00000004470348	4.4E-8	0.9999999	1.0E-7
d+d+d	1/2	0.50000001490116	1.5E-8	0.5000001	1.0E-7

These systematic errors make it necessary to have a computation model able to represent these numbers and to operate with them without error.

1.2. Challenges and Objectives of the work

This work aims to propose a mathematical model to represent and compute rational numbers. This model constitutes the formal framework of an arithmetic architecture where computational techniques are defined to build the operators with rational numbers and perform exact processing.

The key idea of this research is based on representing explicitly the non-zero periodic part of the rational numbers expressed by the positional number system. The challenges of this notation are in developing computational techniques to process the numbers, especially if they are also coded in floating point. So that, this work introduces the calculation method of the addition operator for the proposed rational representation scheme. Its specification for hardware implementation will be detailed in deep. The multiplication function can be designed based on the same principles.

The novelty of this work lies in proposing calculation methods for rational numbers represented in positional number system where periodic numbers can be represented in a direct way without error. The experiments show that this approach is an alternative to the decimal formats for exact coding rational numbers.

This paper is structured as follows: section 2 provides a review of the current state of knowledge on specialized arithmetic processing. The most relevant proposals and works of this issue are described and some findings about them are stated; section 3 describes the formal framework and rational functions on which the architectural model is constructed, section 4 explains the implementation of the rational processing. The overall architecture is introduced, and the representation format and the addition operator are detailed; section 5 shows an empirical evaluation of the results of processing the double mantissa numbers and an example of application on intensive calculus, finally, section 7 summarizes the conclusions of this work.

2 Related Work

2.1. High precision computing proposals

This section is not intended to contain an exhaustive and detailed review of state-of-the-art, but only introducing the more representative proposals and results that show the progress in the issue of high precision computing.

In first place, the last floating point standard [1] is the most used for number codification and arithmetic computing [10, 11]. There are research works which shows the limitations of these formats [39] and the error produced by the processing of the floating point arithmetic [12, 13].

On the way of search for improving precision in calculations, software solutions are the first stage. There are a great variety of math libraries for numerical calculation with greater precision than conventional standard formats [2-4]. However, these proposals are executed from the application level of the architecture and do not offer a right performance for processing-intensive applications. In addition, in some problems, they are not able to provide the required numerical accuracy. So that, with the aim of improving computer performance while maintaining high precision, several arithmetic units and specialized processor designs have been proposed.

The method with greater capacity for expression is based on symbolic representing system. By means symbolic representation, it is possible to express any number exactly. For example: π, e, ⅓, etc. Based on these principles, a rational arithmetic unit has been proposed [14] in which the numbers are represented symbolically by means of fractions. The basic operations such as addition, subtraction, multiplication and division are implemented, operating on numerators and denominators of fractions with integer arithmetic. Nevertheless, processing with these symbolic expressions is costly in computational terms, particularly when there are no easy ways of simplification. In addition, in order to provide the final result in positional number system by means of a single numerical value, it is necessary to carry out a division operation which may cause approximation errors.

Other methods aims to represent real computable numbers by means numerical sequences: linear fractional transformations, Cauchy sequences and continued fractional notations. Linear Fractional Transformation is a method for transforming the real number to a computable sequence of digits [15, 16]; Cauchy sequences consist on a sequence of rational numbers approximating the real number [17] and continued fractions is a representation based on an iterative process of fractional decomposition of the real number [18, 19]. There are hardware architecture designs for processing numbers coded in this last fashion [20, 21]. However, they reveal highly complex arithmetical operations and, as symbolic computation, when a positional numerical result is required, it is needed additional operations to transform the sequences and an approximation error arises.

There are other alternatives for representing and computing rational numbers based on *Stern-Brocot tree* [33] and *Möbius number systems* [34, 35], but the algorithmic methods are also costly and the problem of transforming the results to positional number system persist.

In order to delimit the representation error, mathematical models based on interval arithmetic have been proposed. This method consist on expressing the numbers within an interval. In that way, the resulting error can be limited to the width of the interval [22-24]. This technique is a very popular way for estimate the numerical error of calculations. So that, there are several programming tools and software packages available for high-precision calculation using interval arithmetic for many languages [25]. In addition, interval arithmetic units and specialized processors have been designed to improve the time performance of computations. The following are among the most representative examples:

- The *interval arithmetic VP processor* [26] consists of a floating point representation structure in which the field which stores the mantissa can have a variable length. Although the accuracy of the representation of the intervals bounds is increased, the level of approximation is limited by the amount of significant digits to be represented.
- Multimedia processor [27]: this proposal uses the multimedia extensions of Intel and Motorola processors to implement the interval arithmetic and allows a variety of rounding policies to minimize the error bounds.
- CORDIC processor [28]: this design provide a set of functions computed by means the CORDIC algorithm. The proposal allow specify the precision to perform the operations.
- Variable Precision Processor [29]: this work proposes a real-time system composed by several word-length operators. The unit uses interval arithmetic to determine the accuracy of the results.

Nevertheless, this method has some drawbacks for exact computing: in first place, it does not represent the number exactly, but only with an interval; the arithmetic operations must be performed on two bounds of the interval; and finally, this interval can grow up according to the operations performed.

Other processor designs provide a result consisting of a single number by means of using a lot of significant digits such as *staggered* and *on-line arithmetic* processors. The *processor for staggered interval arithmetic* [30] is able to use conventional floating point units for numbers coded in standard IEEE format. Its main disadvantage is the complicated and costly computation process. For example, the comparison between two numbers becomes a complex algorithm due to the fact that for the same number there are multiple different representations. Regarding on-line arithmetic, there are several hardware processors based on this method. For example:

- The *JANUS coprocessor* [31] considers a maximum accuracy of 600 digits. This proposal implements only the multiplication/addition operation.
- The *VLP coprocessor* [32] consists of an arithmetical coprocessor developed with FPGA platform with capacity for reconfiguration depending on the operations to be carried out.
- *Online Processor* [36] is an arithmetic array processor for solving tridiagonal systems of linear equations. This architecture allows parallel operator designs and produces result of arbitrary length.
- *Decimal on-line arithmetic* [40, 41] proposes on-line addition units for decimal number representation.

Finally, with the same objective of increasing the amount of significant digits of the numbers there are hardware proposals able to work with data having a variable number of digits:

- The *CADAC processor* [37] codifies the data by means of variable word length. Each word contains sign, exponent, mantissa, mantissa length fields and pointers to them. Disadvantages include the additional complexity of its arithmetic unit.
- The VP *coprocessor for FPGA* [38] is an evolution with respect to the previous design. This design does not limit the amount of digits of the number's significant mantissa. In this case it uses a structure based on a variable amount of 64-bit words. The representation format enables concatenation of various words until the codification of the number is complete.

2.2. Findings

This review finds that there are several alternatives to represent numbers in a computer and even some proposals allow to represent rational numbers without error. The standard representation formats are not among them.

The main drawbacks of the more accurate ways of representing numbers are the complexity of their arithmetic methods and the lack of precision when the numbers are transformed into positional number system expressions.

The most recent proposals based on the interval arithmetic and on-line methods offer alternatives of interest which improve the precision of the results. However, they are unable to provide an exact value but only an approximation for irrational or periodic rational numbers.

The hardware support for computing them is a clever alternative to provide greater performance and avoid the software overhead due to specialized math libraries, especially when trying to build embedded devices or solving high performance computing applications.

3 Formal Framework for High Precision Computing

The exact computation problem can be defined by means the following definitions.

Let f be a general mathematical function. Any computable function whose result approximates to f according to a particular implementation is defined *implementation function* Γ *of* f. In this way:

$$\mathrm{codomain}(\Gamma_f(\vec{x})) \subseteq \mathrm{codomain}(f(\vec{x})) \qquad (1)$$

and then,

$$\forall\, \vec{x} \in \mathrm{domain}(\Gamma_f),\ |\Gamma_f(\vec{x}) - f(\vec{x})| \leq \varepsilon \qquad (2)$$

where,

$\vec{x}$: Function's arguments.
$\varepsilon : \varepsilon \in \mathbb{R}^+ \cup \{0\}$. Approximation of f by Γ.

The general work of an arithmetic unit is to process mathematical functions. The calculation of these functions is its main purpose.

An *architecture* Λ is characterised both by the set of functions that it provides and by the way in which they are implemented. Let the following set of functions be:

$$\Phi = \{f_1, f_2, \ldots, f_n\} \tag{3}$$

An architecture Λ_Φ that provides these functions will be made up of:

$$\Lambda_\Phi = \{\Gamma_{f1}, \Gamma_{f2}, \ldots, \Gamma_{fn}\} \tag{4}$$

That is, Λ_Φ will contain the specific implementation of each function of the Φ set, where each Γ_{fi} produces an approximation to $f_{i.}$ In this way, f_i is the objective of the arithmetic unit whereas Γ_{fi} corresponds to the function that is finally provided. The implementation of those functions does not have to be unique. Thus, several implementations of the same function that represent different approaches to f, with different values of $\boldsymbol{\varepsilon}$ could exist. For example, the different arithmetic adder implementations for each operand size of different representation format.

It is defined a function Γ_f *as having an exact evaluation of f* if the result provided by Γ_f is equal to the result of f, that is, in accordance with the expression (2), $\varepsilon = 0$.

$$\forall\, \vec{x} \in \mathrm{domain}(\Gamma_f), \Gamma_f\,(\vec{x}) = f(\vec{x}) \tag{5}$$

In this case, the Λ architecture implements that function effectively. An exact arithmetic architecture requires all its functions to be implemented in a totally effective way.

The framework for high precision computing works into a rational domain. The set of functions provided is the following:

$$\Phi_{\mathbb{Q}} = \{\mathrm{identity}, \mathrm{addition}, \mathrm{multiplication}\} \tag{6}$$

The mathematical identity function is defined by the following expressions:

$$\begin{aligned} &\mathrm{identity}: \mathbb{Q} \rightarrow \mathbb{Q} \\ &\forall\, x \in \mathbb{Q}.\ \mathrm{identity}(x) = x \end{aligned} \tag{7}$$

The implementation of the identity function corresponds to the codification of the rational numbers in a floating point representation format based on the positional notation system. The main idea consist in representing the fractional part of the rational number by means a double mantissa codification where fixed and periodic mantissas are explicitly represented. The fixed mantissa is the fractional part of the non-periodic rational number, whereas the periodic mantissa represents the digits that form the repetitive part. In this way, the whole digit string of the number can be easily obtained by concatenating the fixed mantissa with the periodic mantissa for an indefinite number of times.

Along with this method that provides exact representation of the numbers, the arithmetic functions of $\Phi_{\mathbb{Q}}$ define a 'commutative ring' whose nonzero elements form an 'abelian group' under multiplication and addition operations:

$(\mathbb{Q}, +)$ and $(\mathbb{Q}, \cdot)$ are abelian groups.
$(\mathbb{Q}, +, \cdot)$ is a commutative ring.

The formal framework is characterized by the exact representation of the numbers and by these mathematical properties. Only with exact representation, the inverse functions can be achieved by the abelian groups and the commutative ring can be built. These properties let the formal correctness of the problem formulation and allow to the unit perform the calculations of scientific or engineering problems from a numerical point of view.

However, the main challenges of this mathematical model are to conceive calculation methods to implement the arithmetic functions where the numbers are coded in the proposed method. The next section describes the implementation method of the identity and addition functions for rational processing.

4 Arithmetic Unit Architecture

In general terms, our proposal consists of developing an arithmetic architecture which contains a set of operators which achieve the exact result for rational operands. The arithmetic unit architecture provides the *implementation function* Γ of each of the functions of $\Phi_{\mathbb{Q}}$ as it is defined in expression (8).

$$\Lambda_{\Phi\mathbb{Q}} = \{\Gamma_{\text{identity}}, \Gamma_{\text{addition}}, \Gamma_{\text{multiplication}}\} \quad (8)$$

These operators produce the exact results of the functions. Figure 2 shows a diagram summarising the unit design.

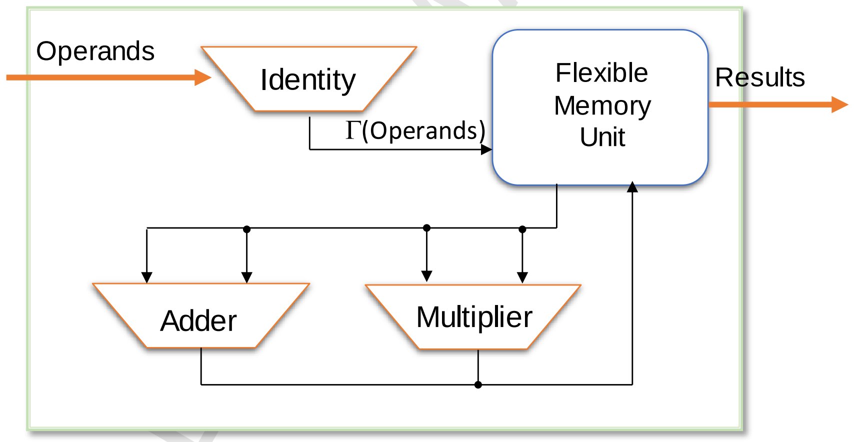


Fig. 1. General overview of the Rational Arithmetic Architecture

4.1. Identity operator

The first task is to implement the identity function able to express any element of the set of rational numbers. The proposed implementation, Γ_{identity}, is a bijective application with which any rational number can be represented in a finite representation space.

$$\Gamma_{\text{identity}}: \mathbb{Q} \rightarrow \mathbb{Q} \quad (9)$$

According to previous idea, the proposed implementation lets coding both fixed and periodic mantissa of the rational numbers. In addition, the number can be represented in floating point in order to provide higher expressive capacity. The next figure shows a scheme of the representation, where EWL, M_fWL and M_pWL are the field lengths.

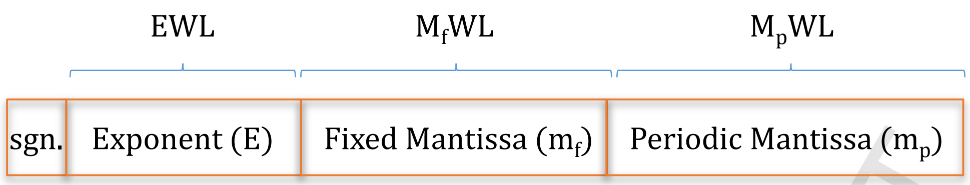


Fig. 2. General scheme of the double mantissa representation

The value of the number is built by means of the standard floating point expression:

$$A \in \mathbb{Q}, A = (-1)^s \cdot M \cdot B^E \tag{10}$$

where B is the base of the representation, M is the complete mantissa of the number, and E is the exponent.

The codification of the numbers in that format by the arithmetic unit is an implementation of the identity function Γ_{identity}. The Word Length (WL) of the fields of the representation format can be adjusted to the required precision of each problem. Thus, the operands, intermediate numbers and results can be stored in a 'Flexible Memory Unit' with indirect addressing of variable word lengths [42].

Nevertheless, the representation can also be made into conventional fixed registers with a Register Word Length (RWL) established for the architecture. In this way, the length of the fields of the exponent (EWL) and the mantissa (MWL) are fixed too. In this case, it is required a pointer which marks the separation between the fixed and periodic parts of the mantissa and which enables a separate processing. To complete all of the digits assigned to the mantissa field, the periodic mantissa is placed forming a cycle, and the lengths of fixed and periodic mantissas are stored with the previous pointer. This data joins the register that contains the number as can be seen in the structure illustrated in the following figure:

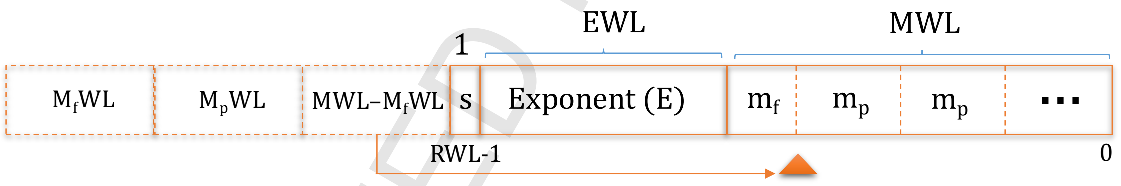


Fig. 3. General scheme of the double mantissa representation in fixed register of RWL length.

4.2. Addition operator

The importance of the addition operation in numerical calculation has motivated the work of many researchers aiming to improve it [43-45]. Most of the methods are designed for operands represented according to IEEE-754 standard [1] (both binary and decimal base). They show how to perform the operation by manipulating the fields that make up the numbers: displacement, addition of mantissas, exponent's treatment, normalization and rounding.

The implementation of the addition function is named as Γ_{addition}. The next expression formalizes this implementation on rational operands:

$$\begin{gathered}\Gamma_{\text{addition}}: \mathbb{Q} \times \mathbb{Q} \rightarrow \mathbb{Q} \\ \forall\, x_A, x_B \in \mathbb{Q}.\ \Gamma_{\text{addition}}(x_A, x_B) = x_A + x_B\end{gathered} \tag{11}$$

The arithmetic algorithm of addition operator is based on the following design principles: the standard stages of floating-point operation formats are taken as a starting-point for the new method; the use of iterative methods to process the data successively and; the computation of each significant mantissa separately to facilitate parallel designs.

Hence, the calculation methods have the following characteristics: they can provide the exact result of the operation in a finite word size and then, the final rounding stage is avoided; the length of the exact result expressed in positional notation system is proportional to the initial size of the operands; the design of strategies to adjust the precision and the result's length is feasible by acting on the iterative methods of calculation of the mantissas. Using iterative structures and pre-calculated data could be a way of achieving flexibility on the designs and can be used to adjust the result to the needs of each application [46,47].

In first place, we present the general algorithm for exact addition without taking account of any memory restrictions about the length of the fields. Those variable-length register set make up the flexible memory unit. Next, we will propose the architecture of the operator for low level hardware implementation (Γ_{addition}) based on the previous algorithm. Each stage of the addition is composed by elementary operators such as comparisons, n-bits adders, shifts and rotations. These primitives, are implemented according to well-known designs for them. In this approach, it is considering the required precision and the available size of the words into the arithmetic unit design.

The algorithm introduced consists of an extension of the traditional floating-point addition method where operators and results are expressed according to the double mantissa representation (sign, exponent and fixed and periodic mantissa) depicted in figures 2 and 3. Therefore, the proposed method for the addition consists of the stages shown in figure 4:

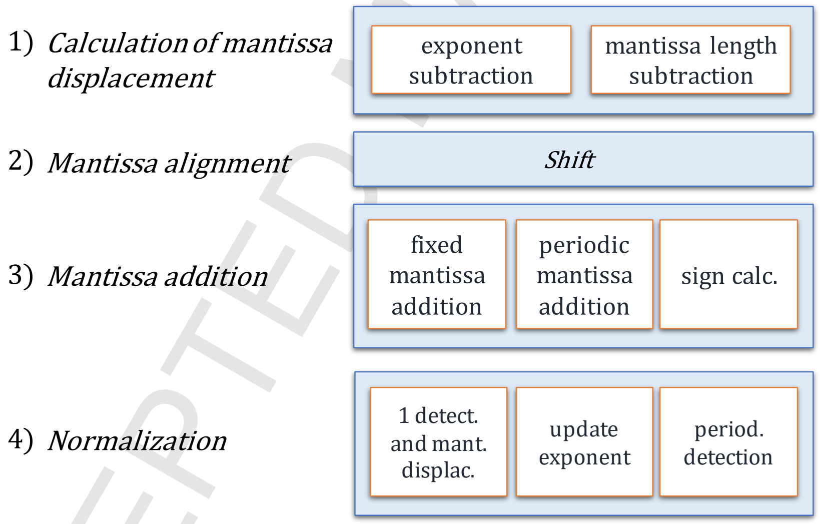


Fig. 4. Rational addition double mantissa stages.

The next paragraphs describe each stage of the method. Along this exposition, it is considered two operands A and B represented into this format as inputs of the operation.

1. Calculation of mantissa displacement: The digits of the same order of magnitude must be aligned to perform the addition of the mantissas. The displacement of digits must consider the different exponents and the difference of fixed mantissa's digits of both numbers. The calculation procedure consists in subtracting from the exponents' comparison, the fixed mantissa lengths' comparison of both numbers as it is described in expression (12) and it illustrated in the following figure (Fig. 5).

$$\text{Mantissa displacement} = (\text{Exponent}_A - \text{Exponent}_B) - (M_fWL_A - M_fWL_B) \qquad (12)$$

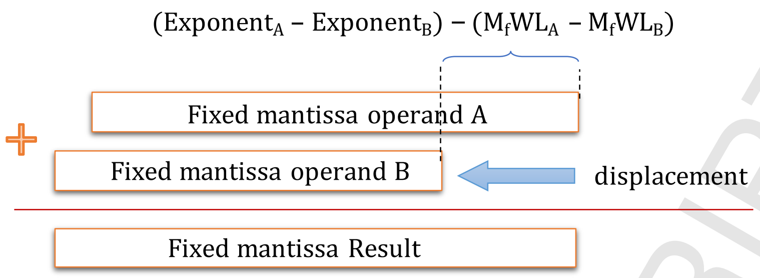


Fig. 5. Displacement of mantissas in a variable length scheme

The next figure (Fig. 6) shows the flow design of that computing where the primitive operators (complement, mux and n-bit adders) take part.

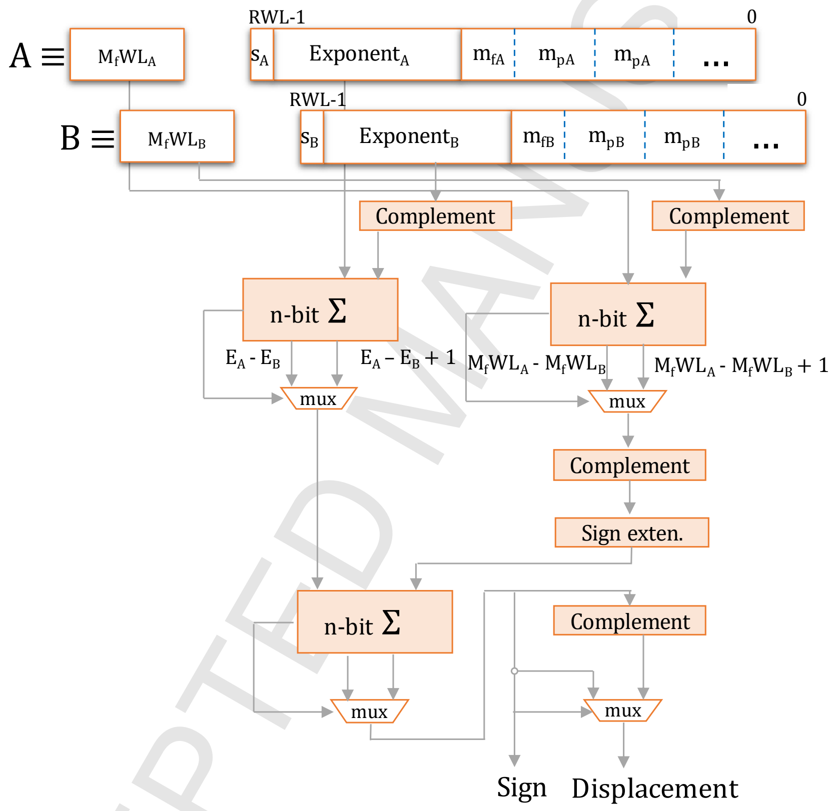
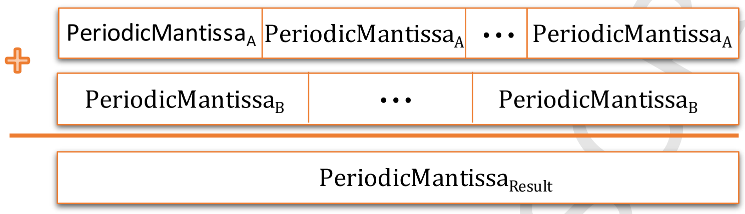


Fig. 6. Calculation of mantissa displacement stage implementation

The computational cost of this stage comes from the calculation of three addition operations. Since the M_fWL is a fixed field, the computational cost of this stage is lineal with the exponent lengths. So that: $T_{1)} \in O(EWL)$.

2. *Mantissa alignment*: there are two types of transformation of the mantissas: fixed mantissa is shifted to the left whereas the periodic mantissa is rotated to the same direction. Periodic digits are taken when necessary to complete the fixed mantissa. With these alignment movements, the fixed mantissa length is increasing by the calculated displacement while the periodic mantissa length remains constant. The implementation of these two transformations consist only in updating the periodic-fixed separator pointer by the addition of the calculated displacement. The computational cost of this stage is lineal with the exponent size: $T_{2)} \in O(EWL)$.

3. Mantissa addition: At this stage, the result of the addition of fixed and periodic mantissas are the fixed and periodic mantissas of the result respectively. Both operations have different computational procedures which can be done in parallel with iterative addition methods. The negative numbers are processed in complement. These complement operations, if necessary, are implicit in Fig. 7 and 8. Each type of mantissa requires a different procedure: fixed mantissas can be added directly, while periodic mantissas need to be length-equalled by repetition before making the addition, as shown in figure 7. It is necessary control logic in order to not exceed registers limits, and in that case, to produce an approximate result.



$\text{PeriodicMantissa}_A$
$\text{PeriodicMantissa}_A$
• • •
$\text{PeriodicMantissa}_A$
+
$\text{PeriodicMantissa}_B$
• • •
$\text{PeriodicMantissa}_B$
$\text{PeriodicMantissa}_{Result}$

Fig. 7. Periodic Mantissa addition

The carry bit of periodic addition must be added to the periodic result and propagated to the fixed addition. Figure 8 shows the relations between the addition of the mantissas and the double propagation of the carry to fixed and periodic operations.

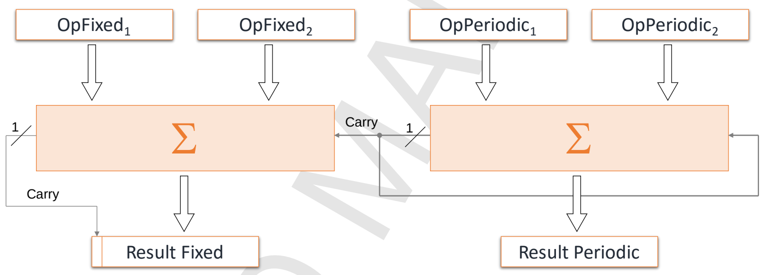
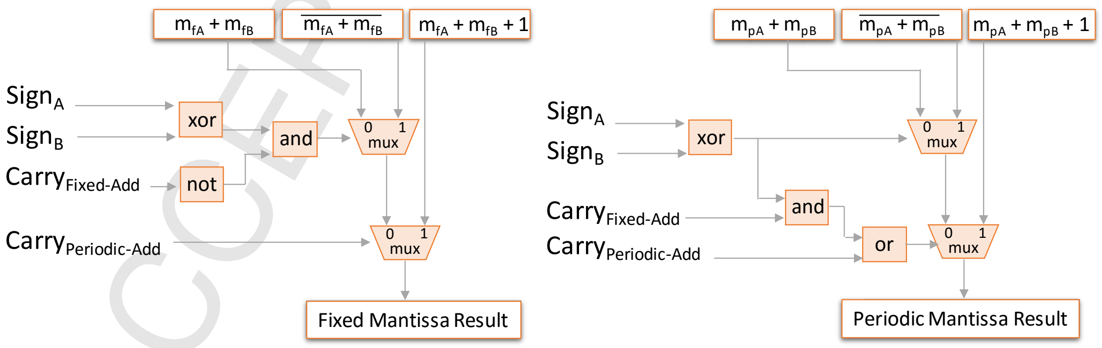


Fig 8. Carry propagated adder between periodic and fixed mantissa

To perform these operations in parallel, three results are obtained: the addition, the addition complemented and the addition plus one. The right result is then selected according the numbers' sign and the carry bits of both operations. The selection functions are implemented as shown in the next figure (Fig. 9).



(a) (b)

Fig. 9. (a) Fixed mantissa selection logic; (b) Periodic mantissa selection logic

The sign of the result is a combinational function of the operands and the carry produced in fixed mantissa addition according to the standard procedure for complement addition. The next figure (Fig. 10) depicts this logic function.

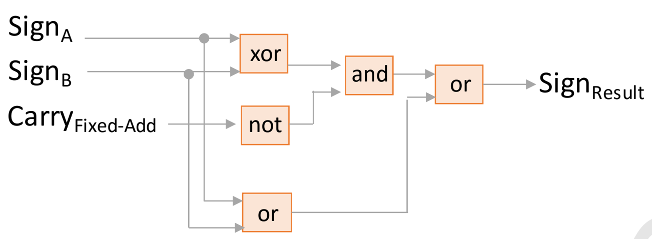


Fig. 10. Sing of the result calculation

The computational cost of this stage is produced mainly by the addition operations of the mantissas. The periodic mantissa additions have a variable length depending on the least common multiple of the periodic mantissas' length, but in any case, the complexity is lineal with whole mantissa length MWL. $T_{3)} \in O(MWL)$.

4. Normalization: The results are normalized as in the standard floating point format [1]. However, the addition operation can give other illegal configurations in the fixed and periodic mantissas under the proposed representation format. These cases are produced when the periodic mantissa is composed of a group of digits constituting a subperiod itself or when fixed mantissa ends with a set of digits contained in the periodic mantissa. In those cases, the result must be normalized to produce the simpler expression of each number. The method to check the existence of subperiods in the periodic mantissa consist in rotating & comparing on. If the method finds a match, the number of rotations indicates the final size of the period. This normalization stage needs control logic to manage the process and adjust the result to the registers' length.

The described normalization procedures need to go through the mantissas to check for illegal configurations. So that, this stage has a complexity lineal with mantissa lengths. $T_{4)} \in O(MWL)$.

All the previous stages are computed sequentially. The temporal cost is obtained by the contribution of all of them. So that, the overall computational complexity is lineal with operand lengths: $T \in O(RWL)$. This complexity is similar to standard floating point additions methods.

5 Empirical evaluation & Application example

This section analyses the representation scheme and the quality of the results of the addition method to validate the proposal for computing high precision problems.

5.1. Empirical evaluation of the result

In first place, it is analysed how the format works in representing the rational number of the result. The experiments are focused on studying when the normalization stage is needed and what is the growth of the resulting mantissas of massive chained addition operations. This research is aimed to estimate the frequency of the extra delay produced by the normalization step and the memory needs to store the results on complex calculations.

The experimental set up is the following:

- It has been made series of 10^6 addition operations of rational numbers represented in the proposed double mantissa format. The average of the results of each series has been obtained to draw the results shown in table 1.
- The series corresponds with numbers generated in a growing range in order to deduce behaviors related with the range width. So that, for each addition, two

rational numbers have been built by means of a pair of fractions whose numerators and denominator have been randomly generated into a power of two range. That is, a/b where: a, b ∈ [1..2^i] and i ∈ [4..8].

The next table (table 1) shows the results obtained:

	series				
Normalization stage	16	32	64	128	256
Simplifying the periodic mantissa	2,6%	2,2%	1,6%	1,3%	1,1%
Sub-mantissas in fixed mantissa	6,6%	3,1%	1,2%	0,6%	0,1%

Table 1: Normalization needs of the addition results

The frequency where the normalization stage reduces the length of the result of the addition is small and it is reduced with increasing the generation range of the values to add. That is, the wider generation range, the lower effect has the normalization stage on the result.

The proposed operation method can work with numbers denormalized. Therefore, in problems with large variable domains, the normalization stage can be avoided in order to improve the temporal cost. In addition, this stage can be performed at the end of a sequence of chained operations where the result of the operation is the input of the next operation. In this way, the normalization is computed only on the resulting number.

In second place, it is studied the size of the results of the additions and how it grows with successive calculations. The experiments consist on compute series of chained operations where the rational numbers have been generated according the same criteria as the previous set up. The experiments show that the fixed mantissa has a logarithmic growth with the amount of additions and it's independent from the generation range of the fractions, whereas the periodic mantissa has a growth related to the generation range. It is due to the process in which the periodic mantissa is built in the described addition method. The length before normalizing is equal to least common multiple (LCM) of the periodic mantissas' size of the operands. However, this periodic part has an upper limit. There is a maximum length for each range given by the LCM of the lengths of the periods of all numbers of the range. Thus, the maximum size of the periodic part will be conditioned by the operator's domain. For example, in the case of processing numbers with two decimal fractional digits, the maximum size of the periodic part will be 20 binary digits. That is,

$$1/100_{10} = 00\overset{\frown}{00001010001111010111}_2$$

where:

m_f = 00
m_p = 00001010001111010111

In this way, the result of any chained additions of operands with two decimal fractional digits has a periodic part of up to 20 bits length. Other ranges have different lengths which can be obtained experimentally. This finding is useful to design fixed registers to store the numbers and results.

5.2. Application example

This section shows a calculation example of rational numbers in a context of intensive addition/subtraction operations. The aims of this experiment is to test the accuracy and reliability of the proposed model in comparison with the representation and computation of the numbers with standard methods.

The working scenario could be to a stock market which perform a lot of daily operations. Most of these operations typically consist of arithmetic addition/subtraction on the prices of the stocks processed very intensively. For example, the European *Eurex* is one of the major international derivatives markets which exceeded 11 million contracts daily [28]. The accuracy of these operations is essential to maintain the correctness of prices, ensure exact assessment of stocks and determine the evolution of the value of the assets.

The application example consists in simulating a session day on the market and calculating the evolution in the price of a stock. The addition/subtraction operations are updating the prices of the stocks and accumulating the result.

In this experiment, the operations have been performed using the proposed method for rational processing and the IEEE 754 standard floating point format for binary and decimal representation of the numbers. This decimal representation allows encoding monetary values correctly, however, many of the transactions of the stock market get their value as a result of previous division operations. It produces representation errors of numbers which contain periodic part (i.e. 1/3). This errors in the input data are then propagated [12].

Typically, the rational domain of the stocks is euro cents, that is, two decimal fractional digits. Thus, the stock prices are rational numbers on the form: $\alpha.\beta_1\beta_2$ where a $\alpha \in \mathbb{N}$ and $\beta_i \in [0..9]$.

The experimental set up is the following:

- Calculation of series of chained addition operations of rational numbers where the output of an addition is the input for the next.
- Each rational number has been randomly generated by means a fraction a/b where:

  $a \in [1..100]$ and $b = 100_{10}$
- The numbers have been represented and computed in two ways: according the proposed method, and in standard binary IEEE 754. The final stock prices are compared after the calculations.
- The exact value of each series has been obtained by means symbolic fractional calculation. That is, calculating the common denominator, updating the numerators and adding them. For this purpose, a software library has been developed.

The experiments verify that the proposed method for high precision computing of rational numbers produces exact results regardless of the number of chained operations executed, whereas the standard binary IEEE 754 format representation causes an error which increases with the amount of operations. These errors can be magnified especially when involving a large volume of operations. What is normal in this type of scenarios.

The error rate for each series of chained operations of the calculations made by standard binary IEEE 754 arithmetic is drawn in the graph below (Fig. 11).

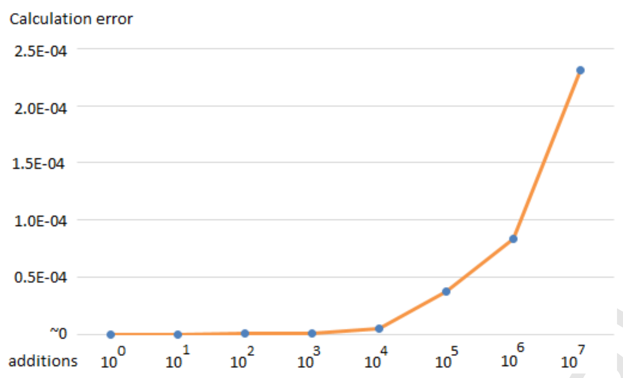


Fig. 11: Error evolution of binary IEEE 754 simple precision

In terms of computational cost, in our example there is no performance problem, since the key aspect is the precision. In other compute-intensive scenarios where response time is critical may be necessary to consider what type of operation has the best cost/accuracy relationship.

6 Conclusions

The research presented in this paper describes a mathematical model for representing and processing rational numbers. The numbers are expressed in the positional notation system where the periodic part of the numbers is explicitly represented. The proposed operators of the model are the addition and multiplication. These functions are a ring algebra which allow building a set of computational primitive operators.

The arithmetic addition is described in depth detailing the stages involved and the procedures for implementing them. The calculation method is based on the standard floating point addition method and uses iterative strategies which allow processing fields of variable length. The multiplication operator can be designed following the same principles.

The experiments carried out analyse the representation and the addition method from two points of view: first, it is evaluated the behaviour of the double mantissa in the representation of the result and its evolution when multiple consecutive operations are computed; and secondly it is described a realistic application example where a lot of chained additions are needed. As a result, it was found that the addition operation provides exact results, whereas the standard representation produces increasing errors in the results, even using the decimal system.

The proposed model, consisting of the representation format and the operators, is an alternative calculation model when the accuracy in storage and processing of rational numbers is a key aspect to be considered.

We are currently working on developing the multiplication operator to complete the model. In addition, the methods will be implemented in reprogrammable logic in order to build a prototype of specialized processor on exact rational processing.

References

[1] American National Standards Institute and Institute of Electrical and Electronic Engineers, *IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Standard 754-2008*, 2008.

[2] M. Cowlishaw, The decNumber C library, *IBM Corporation Report*. 2010.

[3] M. Cowlishaw, J. Bloch and J. D. Darcy, Fixed, Floating, and Exact Computation with Java's BigDecimal, *Dr. Dobb's Journal, The Word of software development*, 2004, Available online:

http://collaboration.cmc.ec.gc.ca/science/rpn/biblio/ddj/Website/articles/DDJ/2004/0407/0407b/0407b.html#rs1, (Accessed on 24 september 2015).

[4] Intel® Decimal Floating-Point Math Library, 2011, Available online: https://software.intel.com/en-us/articles/intel-decimal-floating-point-math-library, (Accessed on 24 september 2015).

[5] H.A.H. Fahmi et al., Decimal Floating Point for future processors, *International Conference on Microelectronics (ICM)*, pp. 443 – 446, 2010.

[6] E. M. Schwarz, J. S. Kapernick, and M. F. Cowlishaw, Decimal floating-point support on the IBM System z10 processor, *IBM Journal of Research and Development*. Vol. 53 (1), pp. 1-10, 2009.

[7] Fujitsu, *Fujitsu's new generation SPARC64 processor unveiled at Hot Chips 24*, 2013, Available online: http://www.fujitsu.com/global/services/computing/server/sparcenterprise/key-reports/featurestory/sparce-feature1209.html. (Accessed on 24 september 2015).

[8] P. Bergner et al., *Performance Optimization and Tuning Techniques for IBM Power Systems Processors Including IBM POWER8*, IBM RedBooks, 2nd ed, ISBN: 0738440922, 2015.

[9] JL. Sanchez et al., An iterative method for improving decimal calculations on computers, *Mathematical and Computer Modelling,* Vol. 50 (5), pp. 869-878, 2009.

[10] David H. Bailey, High-precision floating-point arithmetic in scientific computation, *Computing in Science and Engineering,* 2005, Vol. 7 (3), pp. 54-61, 2005.

[11] Yuanwu Lei , Yong Dou , Lei Guo , Jinbo Xu , Jie Zhou , Yazhuo Dong , Hongjian Li, VLIW coprocessor for IEEE-754 quadruple-precision elementary functions, *ACM Transactions on Architecture and Code Optimization (TACO)*, v.10 n.3, p.1-22, 2013.

[12] T. Lang, J. D Bruguera, A hardware error estimate for floating-point computations, *Proc. SPIE Conference. Advanced Signal Processing Algorithms, Architectures and Implementations*, 2008.

[13] D. Piso and J.D. Bruguera, Obtaining Accurate Error Expressions and Bounds for Floating-Point Multiplicative Algorithms, *The Computer Journal*, vol. 2, no. 57, pp. 319-331, 2014.

[14] D. Matula, P. Kornerup, Finite Precision Rational Arithmetic: An Arithmetic Unit, *IEEE Transactions on Computers*, Vol. C-32 pp. 378-387, 1983.

[15] A. Edalat, R. Heckmann, Computing with real numbers: (i) LFT approach to real computation, (ii) Domain-theoretic model of computational geometry, *Lecture Notes in Computer Science*, vol.2395,193-267, 2002.

[16] V. Ménissier-Morain, Arbitrary precision real arithmetic: design and algorithms, *The Journal of Logic and Algebraic Programming*, Vol. 64, pp. 13–39, 2005.

[17] H.J. Boehm, Constructive real interpretation of numerical programs, *ACM Conference on Interpreters and Interpretives Techniques*, ACM, 1987.

[18] D. Lester, Effective Continued Fractions, *IEEE Symposium on Computer Arithmetic*, pp. 163-170, 2001.

[19] J. Vuillemin, Exact real computer arithmetic with continued fractions, *IEEE Transactions on Computers*, Vol. 39 (8), pp. 1087-1105, 1990.

[20] T. Brabec, R.Lórencz. Arithmetic Unit Based on Continued Fractions. *International Scientific Conference on Electronic Computers and Informatics, pp. 225-230*, 2006.

[21] O. Mencer. *Rational Arithmetic Units in Computer Systems*, PhD Thesis, Stanford University, 2000.

[22] H. Dawood, *Theories of Interval Arithmetic: Mathematical Foundations and Applications*. Saarbrücken: LAP LAMBERT Academic Publishing. ISBN 978-3-8465-0154-2, 2011.

[23] P. Krka, Exact real arithmetic for interval number systems, *Theoretical Computer Science*, Vol. 542, p.32-43, 2014

[24] J. D. Bruguera, Optimizing the representation of intervals, *Science of Computer Programming*, Vol. 90 (A), pp. 21–33, 2014.

[25] J. Žilinskas, Comparison of Packages for Interval Arithmetic, *Informatica*, Vol. 16 (1), pp. 145–154, 2005.

[26] M.J. Schulte, A Family of Variable-Precision Interval Arithmetic Processors, *IEEE Transactions on Computers*, Vol. 49, no. 5, pp. 1-11, 2000.

[27] J.W. Von Gudenberg, Interval Arithmetic on Multimedia Architectures, *Reliable Computing*, Vol. 8 (4), pp 307-312, 2002.

[28] J. Hormigo, J. Villalba, E. L. Zapata, CORDIC Processor for Variable-Precision Interval Arithmetic, *Journal of VLSI Signal Processing*, Vol. 37, pp. 21–39, 2004.

[29] Sukemi, A G.P. RatnaAgung Putri; H. Sudibyo, A.P. Anak; H. Sudibyo, Incorporating Different Bitspaces to Create a Variable Precision Processor, *Advanced Science Letters*, Vol. 21 (1), pp. 78-82, 2015.

[30] M.J. Schulte, E.E. Swartzlander, Jr., A Processor for Staggered Interval Arithmetic, *International Conference on Application Specific Array Processors*, pp. 104-112, 1995.

[31] A. Guyot, Y. Herreros, J.M. Muller, JANUS, an On-line Multiplier/divider for manipulating large numbers, *IEEE Symposium on Computer Arithmetic*, pp. 106-111, 1989.

[32] A.F. Tenca, M.D. Ercegovac, A variable long-precision arithmetic unit design for reconfigurable coprocessor architectures, *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 216 – 225, 1998.

[33] M. Niqui. Exact real arithmetic on the Stern-Brocot tree. *J. Discrete Algorithms*, Vol. 5(2), pp.356-379, 2007.

[34] A. Kazda, P. Kůrka. Representing real numbers in Möbius number systems. *Actes des rencontres du CIRM*, 1 no. 1, p. 35-39, 2009.

[35] P. Kůrka. Stern-Brocot graph in Möbius number systems, *Nonlinearity*, Vol. 25, pp. 57-72, 2012.

[36] M.D. Ercegovac, J.M. Muller, Arithmetic Processor for Solving Tridiagonal Systems of Linear Equations, *Asilomar Conference on Signals, Systems and Computers*, pp. 337 – 340, 2006.

[37] M.S. Cohen, T.E. Hull, V.C. Hamacher, CADAC: A Controlled-Precision Decimal Aritmetic Unit, *IEEE Transactions on Computers*, Vol C-32, pp 370-377, 1983.

[38] C.-Y. Hsu, *Variable Precision Arithmetic Processor in FPGAs*, Master's Thesis, University of Toronto, 1996.

[39] P. Kornerup, D. W. Matula, Finite Precision Number Systems and Arithmetic, Cambridge University Press, ISBN: 9781139643559, 2010.

[40] C. García, S. Gonzalez, J. Villalba, E.L. Zapata, On-line Decimal Adder with RBCD Representation, *IEEE International Conference on Application-Specific Systems*, *Architectures and Processors, pp.* 53 – 60, 2012.

[41] C. Garcia, S. Gonzalez, J. Villalba, E.L. Zapata, Decimal online multioperand addition, *Asilomar Conference on Signals, Systems and Computers, pp.* 350 – 354, 2012.

[42] F.L. Steven et al., Addressing Mechanisms for VLIW and Superscalar Processors, *Microprocessing and Microprogramming*, Vol. 39. pp. 75-78, 1993.

[43] J. D. Bruguera, T. Lang, Floating-Point Fused Multiply-Add: Reduced Latency for Floating-Point Addition. *IEEE Symposium on Computer Arithmetic*, pp. 42-51, 2005.

[44] A. Vázquez, E. Antelo, A High-Performance Significand BCD Adder with IEEE 754-2008 Decimal Rounding, *IEEE Symposium on Computer Arithmetic, (ARITH 2009)*, pp. 135 – 144, 2009.

[45] K. Yehia, H.A.H. Fahmy, M. Hassan, A redundant decimal floating-point adder, *Asilomar Conference on Signals, Systems and Computers (ASILOMAR), pp.* 1144 – 1147, 2010.

[46] J. M. García-Chamizo, J. Mora-Pascual, H. Mora-Mora, M. T. Signes-Pont, Calculation methodology for flexible arithmetic processing, IFIP International Conference on Very Large Scale Integration (VLSI-SOC), 2003.

[47] H Mora-Mora, J Mora-Pascual, MT Signes-Pont, JL. Sánchez-Romero, Mathematical model of stored logic based computation, *Mathematical and Computer Modelling* Vol. 52 (7), pp. 1243-1250, 2010.

[48] Eurex Group, *Press releases*, 2012, Available online: http://www.eurexgroup.com/group-en/newsroom/press-releases/186248/, (accessed on November 2015)