UDC 004.7

Heorhii Loutskii,
Andrii Dolgolenko, Oleksandr Dolgolenko

METHOD OF SIMPLIFICATION OF COMPUTATIONS WITH A FLOATING POINT IN THE SUPERSCALAR PROCESSOR

Георгій Луцький,
Андрій Долголенко, Олександр Долголенко

СПОСІБ СПРОЩЕННЯ ОБЧИСЛЕНЬ З ПЛАВАЮЧОЮ КРАПКОЮ В СУПЕРСКАЛЯРНОМУ ПРОЦЕСОРІ

This article describes results of development of the approach to building fast operational device of adding-subtracting a long sequence of floating-point numbers with dynamic branching of work at the level of RISCs, which without additional software complications, will ensure the law of associativity when performing addition of sequence of positive numbers. This paper describes the functional circuit of such operational device which does not require for its work elements of firmware control. The operational device can be implemented for SF and F formats of floating-point numbers. For other formats such implementation of the operational device is more reasonable to base on an algorithm similar to the Kahan 's algorithm.

Keywords: Operational Device, RISC Operations, Floating-Point, Law of Associativity, Kahan's Algorithm.

Fig.: 1. Tabl.: 1. Bibl.: 8.

У статті розглядається підхід до побудови швидкого операційного пристрою додавання-віднімання довгої послідовності чисел з плаваючою крапкою, що без додаткових програмних ускладнень забезпечить виконання закону асоціативності при складанні послідовності додатних чисел. Описана функціональна схема такого пристрою, котра не потребує для своєї роботи елементів мікропрограмного керування. Показано, що операційний пристрій за цією схемою може бути реалізованим для половинного та одинарного форматів представлення чисел з плаваючою крапкою. Для старших форматів представлення чисел з плаваючою крапкою реалізація подібного операційного пристрою виглядає більш розумною на основі алгоритму, подібного доалгоритму Кехена.

Ключові слова: операційний пристрій, скорочений набір операцій, плаваюча крапка, закон асоціативності, алгоритм Кехена.

Рис.: 1. Табл.: 1. Бібл.: 8.

Target setting. When constructing cores of most of the modern microprocessors with the x86-64 architecture, OoOE (Out-of-Order Execution) technology, based on the implementation of a Restricted Data Flow (RDF) [1,2], is used. Such microprocessors are

called superscalar microprocessors [3], or microprocessors with the CISC-RISC (CISC-outside RISC-inside) architecture, where: CISC - Complex Instruction Set Computing (full set of operations of x86-64 architecture), RISC - Reduced Instruction Set Computing (short set of operations implemented by a number of microprocessor operating devices). RISC is also called: uop, micro-ops, μops, or similar terms.

In the process of operating the cores of such microprocessors, a number of CISC commands, currently active in the flow of commands, are simultaneously decoded into a plurality of RISC operations. RISC implementation planning is performed according to the RDF architecture, based on the readiness to execute RISC operations operands. Prior to RISCs obtaining the ready to execute status, they are placed in the reserve station cells [4-5]. RISC, which became ready to execute, can be transmitted from cell reservation station to a free operating device that can execute it. Thus, in modern microprocessors, dynamic parallelism is organized at the level of RISCs.

When forming CISC flows – commands that operate with floating point operands, both programmers and developers of optimizing compilers must take into account features of specific implementations of arithmetic with floating-point [6]. As a consequence of these features, for example, floating-point arithmetic does not perform standard mathematical laws such as commutative and associative [7] and it is possible to do so that the calculated answers almost entirely consist of "noise" [7].

For example, multiplication and division operations do not greatly increase the relative error, but subtracting almost equal quantities can significantly increase it. One of the consequences of the possible unreliability of the addition operation sequence of floating-point numbers is a violation of the law of associativity: $(u + v) + w \neq u + (v + w)$ for some u, v, w. The distributive law that binds the operations × and +: $u \times (v + w) \neq (u \times v) + (u \times w)$ may also be violated. Performing addition and subtraction operations sequence numbers even with fixed-point is known as [7] left-associative. This means that operations in such an arithmetic expression must be strictly executed from left to right. Even guidelines for programmers are developed, that contain recommendations for the organization of computing with fewer errors [7]. For example, if you want to add a long sequence of positive numbers [7], you should first sort them out and perform operations starting with the smallest numbers.

Analysis of these guidelines shows that it is often difficult to carry out such rules for the programmers, for example, because unknown values of variables, because the need for pre-sorting of numbers by size, etc. The implementation of such guidelines by the compiler at the stages of preparation for computing is also difficult for the same reasons. In addition, the implementation of these rules, such as the need to change the order of filing operands when performing addition sequence of positive numbers with floating-point, creates the complexity for organizing parallel calculations.

To reduce the error of adding-subtracting a long sequence of floating-point numbers, Kahan 's algorithm, which is also known as compensatory summation, is used [8]. Reducing the error is achieved by introducing an additional variable to store a

growing amount of error. With compensating summation, the worst error does not depend on the number of operands, so a large number of operand values can be combined with an error that depends only on the accuracy of the floating-point representation format. But according to this algorithm, each operation of addition-subtraction is transformed into 4 operations of addition-subtraction type and 4 assignment operations.

The research objective. The purpose of this project is to develop the approach to building fast operational device of adding-subtracting a long sequence of floating-point numbers for dynamic branching of work at the level of RISCs, which without additional software complications, will ensure the law of associativity when performing addition sequence of positive numbers.

The approach to creation of the operational device. Ensure the implementation of the associativity rule for the addition operation over a sequence of positive of floating-point numbers without additional software complications is possible if you perform computation of all intermediate results without losing significant bits.

Consider the possibility of constructing an operational device (OD) for adding-subtracting a sequence of floating-point numbers, which performs the operation $o = o \pm x$ at each step of the calculation, where: x is the next operand of the sequence that can be taken at each step of the calculation on OD inputs for processing; o - an intermediate result of addition-subtraction of a sequence of numbers. To increase the accuracy of the o in the OD will be calculated with an accuracy that is limited only by the range of order change and the accuracy of the representation of the mantissa of the processed format [6]. At the beginning of computing a new sequence of numbers o will be zeroed. Simultaneously with the calculation of the new value of o, its previous value will be converted to the processed numeric format with the rounding to the nearest [6] and output to the OD outputs. Suppose that at the OD input, according to standard [6], each floating-point number x has the form: $x = f_x \cdot 2^{ex}$, where: f_x is a n-bit normalized ($1 \le |f_x| < 2$, with $x \neq 0$) fractional part of the number x (mantissa); e_x - number order (unsigned integer from interval $[e_{max}, 0]$); f_x and e_x are represented by a direct binary code. The floating-point number has two characters: a sign number ***(sign)***, displayed in a separate bit; the order sign, is displayed by the ***bias*** of the order [6].

The schematic design of OD addition-subtraction operations over a sequence of floating-point numbers with increased accuracy of execution is shown in Figure 1. His work is as follows.

Filing of the operands is one at a ***clock*** (see fig.1) of work of the OD. So, on the i-th clock of the work on the inputs of the OD, a regular operand (number x) is given. In this case, the control node 1 is its transformation from a processed floating-point number format [6] to the internal range of processing numbers r, where $r = e_{max}+1+n$ and is the number of binary digits in the representation of a fixed-point number. Namely, the input f_x is given by a normalized mantissa of the operand, the input e_x is the order of the operand, the input $sign_x$ - sign of the operand. With the arrival of the front edge of the ***clock*** pulse on the input ***clock*** data are recorded, respectively, in the

registers of the mantissa ***RG*** f_x, the order ***RG*** e_x and the trigger control ***Tg*** $sign_x$. At the same time, if the number x is the initial operand of the new sequence of numbers, the value of ***RG*** f_o is reset using the ***reset*** signal.

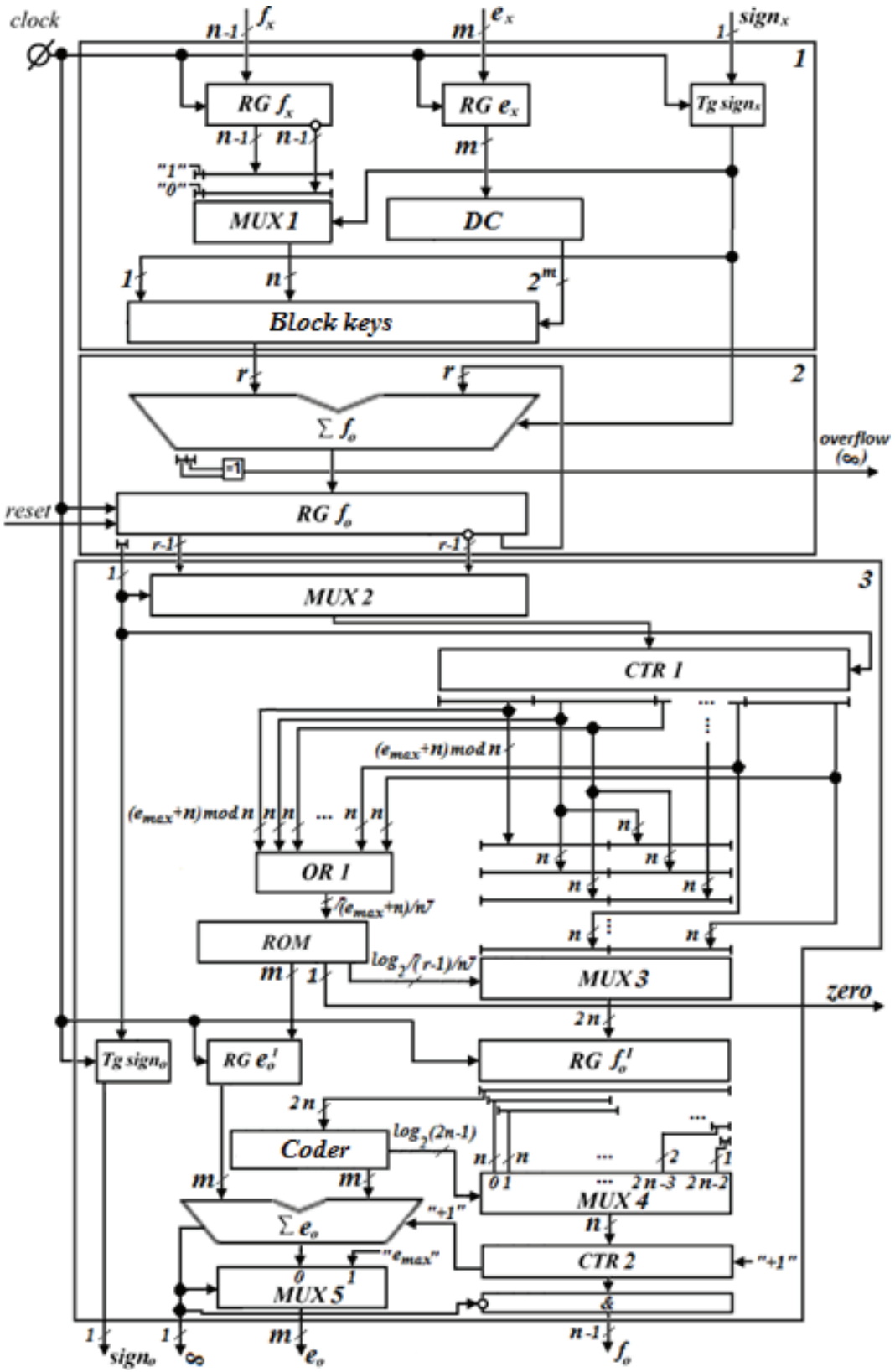

Fig. 1. Functional operating device (OD) addition-subtraction operations scheme

With the ***MUX 1*** multiplexer and depending on the $sign_x$ value, the transfer of ***RG*** f_x from the mantissa f_x code to the ***Block keys*** input with the recovery of the hidden bit ***MSB*** [6] and the $sign_x$ is carried out. Depending on the value of the order e_x, on one of the outputs of the ***DC*** decoder, a signal is generated that provides the input of the ***Block keys*** to the first input of the adder $\sum f_o$. As a result of this transfer, the arithmetic left shift of the inverse code f_x is carried out with ***MSB*** and $sign_x$ on e_x bits, and all other lower bits of the first input of the adder $\sum f_o$ are filled with the value of the $sign_x$, which, together with the submission of the $sign_x$ also to the input of the junior carry input adder $\sum f_o$ provides formation supplementary code x, brought to the r range.

On the $(i+1)$-th step of the OD's work on its inputs, a regular operand of a computable sequence can be taken. At the same time, in the register ***RG*** f_o in the node 2 summation from the outputs $\sum f_o$, the result of the summation of the previous operand will be recorded, and in the trigger ***Tg*** $sign_o$ and registers: ***RG*** e_o^I and ***RG*** f_o^I in the node 3, the result will be written, respectively: the sign, order and mantissa are partially normalized the previous value of o (from the summation of the operand that could be taken at the inputs of the OD on the $(i-1)$-th step of his work).

Thus, OD represents a conveyor information converter consisting of three segments. At each step of the OD's work on its inputs, a regular value of the number x of the executed operation $o = o + x$, in the general case of different microprocessor cores (from the core that captured the OD clock cycle) can be taken. If any of the cores need to perform the operation $o = o - x$, the $sign_x$ value applied to the OD inputs must be changed to the opposite.

Simultaneously with the summation on $\sum f_o$ of the number x with the accumulated amount f_o, which flows from the register of ***RG*** f_o to the second input $\sum f_o$, an ***overflow*** signal is generated (as the sum of modulo 2 of the two highest bits $\sum f_o$), which indicates about the output of a new value of f_o from the range r. In the node 3 forming result transfers the previous accumulated amount of f_o to an intermediate representation in a floating-point form with a $2n$-digit binary code of the mantissa of the f_o^I result in the forward code and the m-bit value of the order corresponding to the f_o^I mantissa. To do this, first, the f_o value from ***RG*** f_o, using the ***MUX 2*** multiplexer and the ***CTR 1*** counter, is translated into a straight-line code, and from its representation the $sign_o$ is deleted, which is written to the ***Tg*** $sign_o$ at the next cycle of the work of the OD. Then with the help of a group of ***OR1*** formed address entry to permanent memory ***ROM*** (actually the older nonzero group of binary digits is found in f_o). The ***OR1*** group consisting of $\lceil (e_{max}+n)/n \rceil$ n-inputs elements ***"OR"*** (the senior element in the group has $(e_{max}+n) mod n$ inputs. At this address, from the first ***ROM*** outputs, the value of the order e_o^I is read, which corresponds to the finding of the higher significant digit f_o on the lower-rank position of the older non-zero binary digits in f_o. This read-out value of the order e_o^I is written to ***RG*** e_o^I at the next time the OD works. From the second output ***ROM*** the zero address reads a ***zero*** signal, which indicates that $f_o = 0$. In this case, from the first ***ROM*** outputs, the zero value of the

order of e_o^I is read [6]. From the third ***ROM*** outputs at the address formed in ***OR 1***, the control information is read out to the ***MUX 3*** multiplexer. This information provides the passage from the outputs ***CTR 1*** through ***MUX 3*** to the ***RG*** f_o^I inputs only of the highest non-zero binary digits group f_o and the discharges group that is next after by her. At the next cycle OU work, these two groups of digits are written to ***RG*** f_o^I.

In the next work of the OD normalizing the mantissa of the result f_o is achieved by removing the hidden bit, its rounding to the nearest, and making the corresponding correction to the value e_o. To do this, using the encoder, depending on the number of zeros to the first significant bit in f_o, the control information is generated by the ***MUX 4*** multiplexer. This information provides the passage from the outputs of ***RG*** f_o^I through ***MUX 4*** to the inputs of the ***CTR 2*** counter ***n*** of the highest significant bits f_o, shifted left to ***k*** bits, where ***k*** is the number of zeros to the first significant bit in f_o. In this case, the first significant bit in f_o on the inputs of the ***CTR 2*** counter is not transmitted, which provides for the removal of the hidden bit. At the same time, at other encoder outputs, generates correction an order of e_o equal to $(e_{max}+n) mod n-k$ is formed, if the first significant bit in the direct code f_o was found in the older group of digits, or $n-k$ in all other groups of digits. With the help of the $\sum e_o$ adder, this correction is added to the value e_o^I that arrives at the other inputs of the adder from ***RG*** e_o^I and is summed up with the value of the carry input entry of the adder coming from the counter ***CTR 2*** and indicates the ***overflow*** of the mantissa f_o as a result of its rounding to the nearest [6]. The result of this so that the value of the mantissa f_o, as well as without the ***CTR 2*** overflow signal, will be normalized, rounded and deleted with a hidden bit and is output from the outputs of ***CTR 2*** through a group of ***(n-1)*** 2-inputs elements ***"AND"*** to output f_o OD. Thus, from the outputs $\sum e_o$ through the input ***0*** multiplexer ***MUX 5*** output e_o OD issued value of the order of the intermediate result. When the ***carry*** signal from most significant bit $\sum e_o$ appearing, indicating the overflow e_o i.e. $e_o > e_{max}$, according to [6], the output of f_o OD gives a zero value, and the output e_o OD, through the input 1 of the multiplexer ***MUX 5***, gives the value of e_{max}.

Table 1 summarizes the values of the bits of the main OD blocks for different formats of representation of floating-point numbers, according to [6].

Conclusions. The paper describes the possibility of constructing an operational device for addition-subtraction of a sequence of floating-point numbers with an accuracy that is limited only to the range of order change and the accuracy of representation of the mantissa of the processed format.

This approach to constructing the operational device of addition-subtraction of a sequence of floating-point numbers looks very promising due to the simplification of the computational process from the programmer's point of view, because it ensures implementation of the associativity law when performing addition of sequences of positive numbers, without additional complications required on the software level.

The paper describes the functional circuits of such an operational device, which does not require elements of firmware control for its work. As it can be seen from the

table 1, the operational device, under the current technological level of components development, can be implemented for SF and F formats of floating-point numbers [6]. For other formats such implementation of operational device is more reasonable to base on an algorithm similar to the Kahan 's algorithm.

Table 1

The values of the bits of the main OD blocks for different formats of representation of floating-point numbers

Main OD blocks	***SF***	***F***	***DF***	***DEF***	***QF***
RG f_x	10	23	52	64	112
RG e_x, RG e_o^I, $\sum e_o$, MUX 5	5	8	11	15	15
MUX 1, MUX 4, CTR 2	11	24	53	65	113
RG f_o , $\sum f_o$	43	280	2101	32833	32881
MUX 2, CTR 1	42	279	2100	32832	32880
OR 1	4	12	40	506	291
ROM	16 x 8	4K x 13	T x 18	2^{506} x 25	2^{291} x 25
MUX 3, RG f_o^I	22	48	106	130	226

References

1. Y. Patt, W. Hwu, et al, Experiments with HPS, a Restricted Data Flow Micro architecture for High Performance Computers, Digest of Papers, COMPCON 86, (March 1986), pp. 254-258.

2. M. Simone, A. Essen, A. Ike, A. Krishnamoorthy, T. Maruyama, N. Patkar, M. Ramaswami, M. Shebanow, V. Thirumalaiswamy, D. Tovey (1995). Implementation trade-offs in using a restricted data flow architecture in a high performance RISC microprocessor. New York. pp. 151-162.

3. John L. Hennessy, David A. Patterson. Computer Architecture. A Cuantitative Approach, USA, Morgan Kaufmann, 2012 – 497 p.+ add-ins.

4. Kanter, David (November 13, 2012). "Intel's Haswell CPU Microarchitecture" (http://www.realworldtech.com/ haswell-cpu/).

5. J.Shen, M. Lipasti. Modern Processor Design: Fundamentals of Superscalar Processors. Waveland Press, 2013- 642 p.

6. IEEE 754: Standard for Binary Floating-Point Arithmetic / 3 april 2014. – URL: http://grouper.ieee.org/groups/754/.

7. What Every Computer Scientist Should Know About Floating-Point Arithmetic. – URL: https://ece.uwaterloo.ca/~dwharder/NumericalAnalysis/ 02Numerics/Double/paper.pdf.

8. Higham, Nicholas J. Accuracy and Stability of Numerical Algorithms. SIAM, 2002, pp. 110–123.

Autors

Heorhii Loutskii – professor, Department of Computer Engineering, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute".

E-mail: georgijluckij80@gmail.com

Луцький Георгій Михайлович – професор, кафедра обчислювальної техніки, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського».

Andrii Dolgolenko – PhD student, Department of Computer Engineering, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute".

E-mail: andrew@ncube.co.uk

Долголенко Андрій Олександрович – аспірант, кафедра обчислювальної техніки, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського».

Oleksandr Dolgolenko – associate professor, Department of Computer Engineering, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute".

E-mail: aleks.dolgolenko@gmail.com

Долголенко Олександр Миколайович – доцент, кафедра обчислювальної техніки, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського».

РОЗШИРЕНА АНОТАЦІЯ

Георгій Луцький,
Андрій Долголенко, Олександр Долголенко

СПОСІБ СПРОЩЕННЯ ОБЧИСЛЕНЬ З ПЛАВАЮЧОЮ КРАПКОЮ В СУПЕРСКАЛЯРНОМУ ПРОЦЕСОРІ

Актуальність теми дослідження. При побудові ядер більшості сучасних мікропроцесорів з архітектурою *x86-64* використовується технологія *OoOE (Out-of-Order Execution)*, що заснована на реалізації обмеженої архітектури потоку даних (*Restricted Data Flow (RDF*)). Такі мікропроцесори отримали назву суперскалярних мікропроцесорів, або мікропроцесорів з архітектурою *CISC-RISC («CISC-outside RISC-inside»),* де: *CISC - Complex Instruction Set Computing*, *RISC - Reduced instruction set computing* (скорочений набір команд, що реалізується множиною операційних пристроїв ядра мікропроцесора).

Постановка проблеми. При формуванні потоків *CISC* – команд, що оперують операндами з плаваючою крапкою, як програмістам так і розробникам оптимізуючих компіляторів доводиться враховувати особливості реалізації арифметики з плаваючою крапкою. В наслідок цих особливостей, наприклад, для арифметики з плаваючою крапкою не виконуються стандартні математичні закони, такі як комутативний та асоціативний і неважко так невдало провести обчислення, щоб їх відповіді майже цілком складалися із "шуму".

Аналіз останніх досліджень і публікацій. Протягом останніх років, побутова ядер суперскалярних мікропроцесорів ґрунтується на тому, що деяка кількість *CISC* – команд, активного в даний момент потоку команд, одночасно декодується на множину *RISC*– операцій. Планування виконання *RISC* – операцій здійснюється відповідно до архітектури *RDF*, на підставі готовності до виконання операндів *RISC* – операцій. До набування *RISC* – операціями стану готовності до виконання, вони розміщуються в комірках станції резервування. *RIS*C – операція, котра набула стану готовності, може бути переданою з комірки станції резервування в вільний операційний пристрій, що може її виконати. Таким чином в ядрах сучасних мікропроцесорах організовується динамічний паралелізм на рівні *RISC* – операцій (їх також називають: *uop, micro-ops, μops*, або подібними термінами).

Виділення недосліджених частин загальної проблеми. Дана стаття присвячена вивченню та аналізу підходу до побудови більш точного операційного пристрою суматора-віднімача послідовності чисел з плаваючою крапкою.

Постановка завдання. Завданням є розробка швидкодіючого операційного пристрою суматора-віднімача з плаваючою крапкою який може бути використаними для динамічного розгалуження роботи суперскалярного ядра на рівні *RISC* – операцій і при цьому, без додаткових програмних ускладнень, забезпечить виконання закону комутативності для довгої послідовності додатних чисел.

Викладення основного матеріалу. Розглянуто підхід до побудови швидкого операційного пристрою додавання-віднімання довгої послідовності чисел з плаваючою крапкою, що без додаткових програмних ускладнень забезпечить виконання закону асоціативності при складанні послідовності додатних чисел. Описана функціональна схема такого пристрою, котра не потребує для своєї роботи елементів мікропрограмного керування.

Висновки. Операційний пристрій за розглянутою схемою може бути реалізованим для половинного та одинарного форматів представлення чисел з плаваючою крапкою. Для старших форматів представлення чисел з плаваючою крапкою реалізація подібного операційного пристрою виглядає більш доцільною на основі використання алгоритму, подібного до алгоритму Кехена.

Ключові слова: операційний пристрій, скорочений набір операцій, плаваюча крапка, закон асоціативності, алгоритм Кехена.