Parallel Section RT. IoT, Real Time Systems.

УДК: 004.896

Andrii Shapran, Oleksandr Dolholenko

DIVISION USING THE BASE RADIX16 NUMBER SYSTEM TO FORM FRACTION DIGITS

The operation of dividing numbers in floating-point form is the most complex operation performed in a microprocessor core. To speed it up, the Intel company, starting with the Sandy Bridge architecture, uses a division algorithm using to represent the fraction of the number system with a base of r=16 (Radix 16).

The article analyzes the requirements of the IEEE Std 754™-2008 standard for floating-point arithmetic. A basic structural scheme for the implementation of floating-point division operations has been developed, that has similar features in many specific implementations of microprocessor cores. To reduce the calculation time of the floating-point division operation, the implementation of the mantissa divider block using the Radix base 16 calculation system to form the quotient digits has been considered. Separate blocks of the divider are designed to the level of the functional scheme.

Key words: mantissa, order, division algorithm, normalization, number system with base Radix 16.

Introduction. The floating-point representation of numbers is similar to the commonly used form of number representation in scientific computing and consists of two parts: the significant part of the number (or mantissa) f and the exponent (exponent, or order) e. A floating-point number x is represented as $\pm,e_x$, f_x and has a value*:* $x = \pm f_x r^{ex}$, where r – base of the number system (base of the exponent, or base of the degree).

Some reductions and abbreviations (acronyms) adopted in the standard *IEEE 754* [1]:

LSB – least significant bit;

MSB – the most significant bit;

NaN **–** not a number.

The absolute value of the mantissa of a normalized number is in the range [1, 2]. During normalization, it shifts so that in $|f|$ $MSB = 1$. An operation is performed on each left shift $e = e - 1$, with each right shift $e = e + 1$. Thus, the two-digit dot in the representation of f is always located after the bit *MSB.*

The MSB bit of the mantissa of a normalized number, which is always equal to 1, is removed from the representation of the number and only a small part of the mantissa is stored in memory. In an arithmetic device, this hidden bit is restored and thereby contributes to increasing the accuracy of the representation of operands without taking up memory space.

A floating-point number has two signs: the sign of the number (sign) is displayed by a separate bit; the sign of the order is displayed by the order bias (bias).

The standard requires using several data types (formats) for floating-point calculations [1].

A *denormalized number* is a floating-point number, exponent of wich is zero (0 is e_{min}) and non-zero mantissa: $\boldsymbol{e + bias} = e_{min}$, $f \neq 0$, $sign = 0\ V\ 1$. When performing an arithmetic operation with a denormalized operand, the hidden bit is not restored in it. The use of denormalized operands makes the effect of loss of significance less drastic. Without using of such operands, some small values that cannot be represented as normalized numbers would have to be rounded to 0. Such solution would degrade the accuracy of floating-point calculations in some cases. For example, a number $(0.1)_2 \times 2^{-126}$ does not have a normalized representation in the IEEE single format. If we use 0 instead of this number in further calculations, then the "gradual loss of accuracy of the result" will occure. At this time, the implementation of the ability to process denormalized numbers in a computing device can cause unwanted hardware and time costs. In this regard, the standard does not require mandatory implementation of denormalized number processing.

The purpose of the article. The purpose of this article is to develop a device for performing the operation of dividing floating-point numbers using the Radix base 16 numbering system for the intermediate representation of the quotient digits.

Presenting main material. As a result of the research, a structural scheme for the implementation of division operations has been developed, which has similar features for many specific implementations of microprocessor cores. Based on this, the developed scheme can be considered as basic, commonly used, in its general features, when developing specific options for implementation of floating-point division operations. This divider scheme is shown in the fig. 1.

The divider usually consists of: an operand unpacking block, a mantissa divider, a number of additional circuits used for processing: orders, exceptions (± 0, $\pm\infty$, *NaN*), normalization and rounding of the result, as well as its packaging.

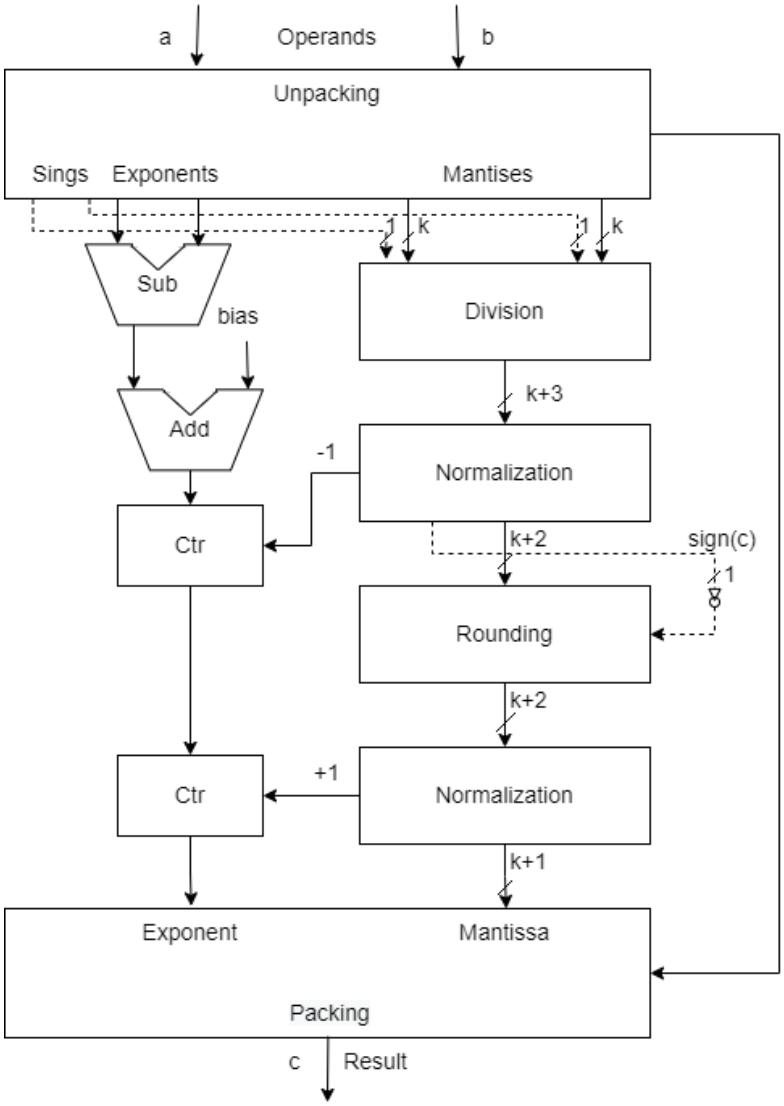

Fig. 1. Floating point divider

Performing a division operation on floating-point operands $c = a/b$ is reduced to the following operations:

$$(\pm f_a \text{ x } 2^{(ea+bias)})/(\pm f_b \text{ x } 2^{(eb+bias)}) = \pm f_a / f_b \text{ x } 2^{(ea - eb+bias)} = f_c \text{ x } 2^{(ec+bias)}.$$

When constructing a floating-point divisor, care must be taken to ensure correctness and avoid unjustified loss of accuracy. In addition, the possibility of handling any exceptions should be implemented.

Unpacking includes the selection of: the sign of the mantissa, as well as the recovery of the hidden MSB bit for each of the operands. During unpacking, the format of the operands is also converted to the internal format of the arithmetic device (for example, to the format of quadruple precision). The unpacked operands are tested for the presence of exceptions among them: 0, *NaN*, ±∞. If there is an exception, the

result of the operation is formed in accordance with the relations given in [1] and sent to the package, bypassing the division of the mantissa.

The preliminary order of the quotient is calculated by subtracting the two shifted orders of the operands and adding shift sum to the resulting:

$$(e_a+bias) - (e_b+bias)+bias= (e_a - e_b)+bias= e_c+bias.$$

Division of mantissas with signs represented by an additional code is carried out with the help of a matrix divider, or with the help of a sequential divider with a high base, for example, *Radix*-16 [2]. After dividing two normalized mantissas, each of which is in the range [1, 2], the mantissa of the quotient can be in the range (1/2, 2). In this regard, for its normalization it may be necessary to shift by one digit to the left with a decrease of 1 in the preliminary value of the order of the quotient. After the first normalization, the k+3 digit mantissa of the quotient is truncated to k+2 digits.

When rounding the mantissa of the quotient f_c loss of its normalization may occur again. At the same time $| f_c |$ may be equal to 2, and for its normalization, a shift of one digit to the right with an increase of 1 value may again be required e_c+bias.

To increase the speed, it is possible to pre-calculate the value of ec+bias increased by 1 at each normalization step and choose the correct value after it becomes clear whether a shift is required after rounding. Since mantissa division is the most complicated part of floating-point division, there is enough time for such calculations. In addition, rounding should not be a separate step at the end of the operation. It can be combined with mantissa division equipment.

The speed of the divisor based on the algorithm of recurrent calculation of numbers depends mainly on the delay of the function that generates the digit of the quotient.

The basis of the development of the mantissa division block is the SRT algorithm of mantissa division in the number system with the base r = 16, which is described in [4]. The article presents an analytical approach that extends the well-known theory [5-8] for performing standard SRT division and allows to implement the function of predicting the number of the part more easily.

In fig. 2 the block diagram of the mantis division unit performing the operation $q = x/d$ is shown and can process all floating-point number formats provided by the standard [1]. There are five such formats: half precision (SF) - 16 bits, single precision (F) - 32 bits, double precision (DF) - 64 bits, double extended precision (DEF) - 80 bits and quadruple precision (QF) (128 discharges). To achieve this goal, the mantissa

division block must be able to handle, according to the format: 12, 25, 54, 66 and 114 bit binary mantissas d and x (together with the sign and hidden bits) in positive code.

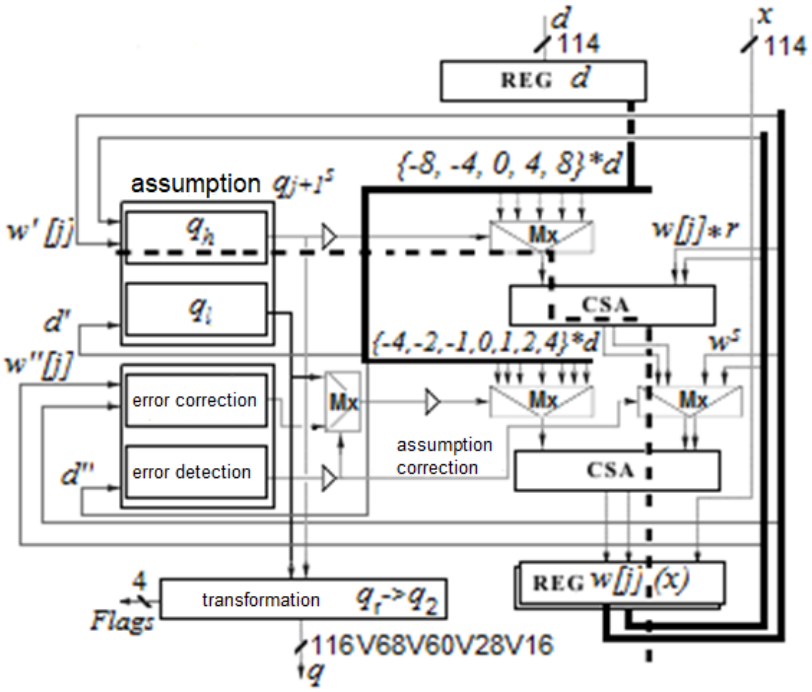

Fig. 2. Structural diagram of the mantissa division block in the counting system with r=16 and memorization of transfers during additions.

At each cycle of this scheme, the prediction of the next digit of the share is carried out q_{j+1}. The mantissa of the fraction q is calculated in the redundant number system with the base r = 16 and is immediately converted into a binary positive code, as described in [8]. Before conversion, each of the provided digits of the share q_{j+1} is a signed number $|q_{j+1}| \le a$, for which the redundancy factor is fulfilled $p = a/(r-1) = 12/15$, or in other words: each digit of the fraction in the device is assumed from the range $q_{j+1} \in \{-12,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,12\}$ and consists of two components, according to [4]. Each digit of the fraction from the redundant sixteen-year numbering system, in the process of calculation, is converted into four digits of the binary system. As a result, the necessary number of calculation cycles for predicting the digits of the fraction is reduced by four times, compared to the calculation of the digits of the fraction directly in the binary number system. The usage of a redundant digital set for fractional prediction significantly increases the speed of each individual calculation cycle by using carry-preserving adders.

The duration of one calculation cycle in the operation of the mantissa division block can be calculated by the signal propagation delay in the critical (longest) chain of the circuit from Fig. 2, where it is shown by a dashed line.

In order to reduce the number of CSAs (carry-saved single-bit adder lines), forming *d* multiples of q_{js} *€{-11, 11},* are not used in the device under development, as it would require three CSAs. Instead of it, the correction function is used. As a result, the required number of calculation cycles for predicting the quotient digits can be reduced by less than four times. In order to increase the speed of the scheme, the prediction of the fraction number at the jth step of the calculation is carried out simultaneously with the detection of an error at the (j -1)th step.

Each digit q_{j+1s} of parts is calculated as the result of of two component parts sum calculated on two different combination schemes:

$$q_{j+1s} = q_h + q_l, \text{ where: } q_h \text{ €}\{\pm 8, \pm 4, 0\}, q_l \text{ €}\{\pm 4, \pm 2, \pm 1, 0\}.$$

In fig. 3 the logic of the assumption from fig. 4 in detail is shown. In this figure: CPA - Carry Propagate Adder (adder with propagated transfers); CS - combination formation schemes q_h and q_l, in accordance.

Older grades of the partial remainder *w'[j]*, calculated in the form with stored hyphens, are pre-processed using CPA. The outputs of the CPA are connected to the inputs of two different combinational circuits as shown in Fig. 3. The second inputs of the combinational circuits are connected to the two higher digits of the divider *d*. The first combinational circuit calculates q_h and requires only the five most significant bits of the value. The second calculates q_l and needs the entire evaluation of the partial balance.

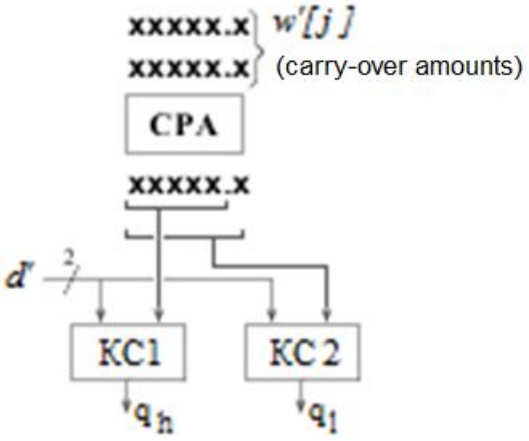

Fig. 3. Details of the implementation of the assumption logic.

The scheme for converting the mantissa of the fraction *q* from the redundant number system into a binary positive code, according to [8], is built as an accumulating adder/subtractor.

Conclusions. In this work, a reconfigurable floating-point divider has been developed, that can dynamically reconfigure to divide operands of all five operand formats required by the standard for floating-point arithmetic *IEEE Std 754™-2008.*

The calculation of the mantissa of the quotient is carried out using the redundant number system with a base to predict the digits of the quotient *r =16* and numbers *{-12,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,12}.* In the process of calculating the numbers, the fraction is converted into a normal binary positive code.

Using such a redundant numbering system for temporarily representation of the quotient's mantissa digits reduces the required number of iterative calculation steps by 4 times. In fact, it is achieved by four binary digits of the mantissa calculation at each iteration, which can be adjusted at the next iteration.

The performed development will be useful for designing fixed and floating point division operation device, for a microprocessor core with a superscalar architecture compatible with the ***x86-64*** family.

References:

1. IEEE 754: Standard for Binary Floating-Point Arithmetic [Электронный ресурс] / 3 апрель 2014. – URL: http://grouper.ieee.org/groups/754/.

2. Behrooz Parhami. Computer Arithmetic. Algorithms and Hardware Designs, New York, Oxford University Press, 2000 – 491 p.

3. Pippenger, N., "The Complexity of Computations by Networks," IBM J. Research and Development, Vol. 31, No. 2, pp. 235-243, March 1987.

4. C VLSI, 1999. Proceedings. Ninth Great Lakes Symposium on Year: 1999, Pages: 74 - 77, DOI: 10.1109/GLSV.1999.757380

5. L. Benini, E. Macii, and M. Poncino. Telescopic Units: Increasing the Average Throughput of Pipelined Designs by Adaptative Latency Control. In 34th Design Automation Conference, 1997.

6. J. Cortadella and T. Lang. Division with Speculation of Quotient Digits. In 11th Symposium on Computer Arithmetic, pages 87–94, 1993.

7. J. Cortadella and T. Lang. High-Radix Division and Square Root with Speculation. IEEE Transaction on Computers,C-43(8):919–931, August 1994.

8. M.D. Ercegovac and T. Lang. Division and Square Root. Digit-Recurrence Algorithms and Implementations. Kluwer Academic Publishers, Norwell, MA, 1994.

AUTHORS

Oleksandr Dolholenko - associate professor, candidate of technical sciences, Senior Research Fellow, Department of Computer Engineering, National Technical University of Ukraine "Ihor Sikorskyi Kyiv Polytechnic Institute".

Andrii Shapran – student, Department of Computing, National Technical University of Ukraine "Ihor Sikorskyi Kyiv Polytechnic Institute".

E-mail: andriyito.ti99@gmail.com