respectively. Operands and results have the same format of n bits. The result of the left normalization of the operands gives also the number N (N=k-l+1), which presents the number of unknown digits of the quotient.

As shown in Table 1.2, when the quotient is not correct, it is adjusted by adding one to its least significant bit. This necessitates recognition of the relevant situations defined in the table. Recognizing (or decoding) the need for correction performs a logical function that depends on the signs of the operands, and in the latter two cases, whether or not the division is exact. If Table 1.2 is treated as a truth table, it can be considered that the logical function expressing the need for quotient correction is a disjunction of three logical terms

$$\mathrm{COR} = \mathrm{cor1} \cup \mathrm{cor2} \cup \mathrm{cor3}\ , \qquad (1.4)$$

where cor means correction. The first term expresses the correction condition according to the second row of the table

$$\mathrm{cor1} = \overline{(X[n-1])} \cap (Y[n-1])\ . \qquad (1.5)$$

The second and the third term (cor2 and cor3) are correction functions related to the third and fourth case respectively. These functions depend on the signs of the operands, as well as on the value of the last partial remainder, i.e. whether the division is exact or not. So we get the following expressions for them:

$$\mathrm{cor2} = \left[(X[n-1]) \cap \overline{(Y[n-1])}\right] \cap \overline{\mathrm{EQ(R)}}\ , \qquad (1.6)$$

and therefore

$$\mathrm{cor3} = \left[(X[n-1]) \cap (Y[n-1])\right] \cap \mathrm{EQ(R)}\ . \qquad (1.7)$$

In the above expressions EQ(R) denotes the logical value of a function that decodes a zero partial remainder in an arbitrary iteration (arbitrary level) during the division. Since this fact has to be decoded at each iteration, this function should have the following cumulative look:

$$\mathrm{EQ(R)} = \bigcup_{m=0}^{n-2} \mathrm{EQ}(R_m)\ . \qquad (1.8)$$

In other words, function (1.8) expresses the possibility of prematurely exact division, which can be observed at each iteration.

We want to draw attention to the fact that, according to the understanding of the traditional division algorithm, the last partial remainder has an extremely complex definition of being the last one. For example, it is the last one if all the unknown digits of the quotient, i.e. if all N digits are obtained. However, there are cases of an exact premature division, after a different number of levels, before the control number is reached. In the sense of the problem solved here, i.e. in the sense of the hardware implementation of the division process, determining the remainder as the last one is extremely inconvenient as it can be obtained at any arbitrary level of the logic scheme of the divider. Therefore, the statement that the division is exact, if the quotient does not have a fractional part, is a very convenient interpretation of whether or not a correction is needed in the last two cases of the table.

Based on the above explanation, we can present the synthesized part of the structure of the hardware divider.

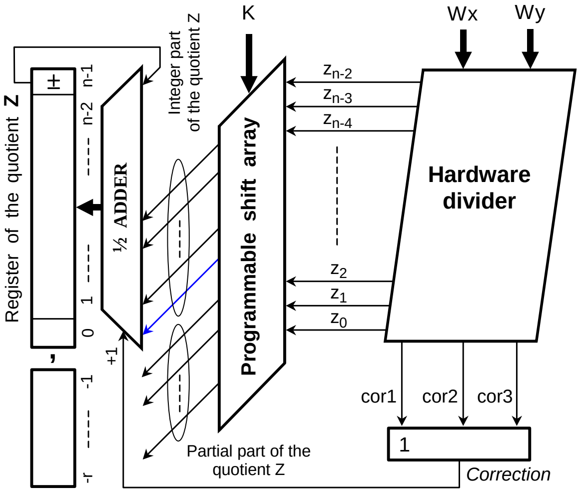


Fig. 1.2. Structural scheme for the stage of the actual division

The two left normalized operand which signs define the initial content of the quotient register RGZ are fed to the inputs of the hardware divider. (n-1) digits of the quotient (z_{n-2}, z_{n-3}, …, z_1, z_0), as well as the three terms of the correction function (cor1, cor2, cor3), come out of the scheme. The quotient is fed to a right-shift programmable array. The shifting is in the direction of the radix point and it is an arithmetic shift, i.e. with the so-called sign extension. As can be seen from the drawing, this scheme is programmed to shift by parameter K. The number of shifts is defined as:

$$K = (n-1) - N\ . \qquad (1.9)$$

The value of parameter K can be calculated at the left normalization stage when calculating the value of N (number of unknown digits in the quotient). Then, this value must be stored in a different registry RGK until the end of the operation.

As far as the correction is concerned, the above scheme illustrates how the result, which is shifted and positioned according to the right position of the radix point is adding with the value of the COR (0 or 1) function fed to the least significant bit №0 of the half-adder ½ADDER. The fractional part of the quotient in the structural scheme is shown for illustration only. It is untrue without correction and should be discarded. If we still want to keep it, then the correction (+1) must be applied not in the least significant bit of the integer (b_0), but in the least significant fixed bit of the real quotient (b_{-r}).

Figure 1.3. bellow presents the synthesized logical scheme of a hardware divider, which always calculates 7 most significant digits of the quotient (z_6 to z_0).

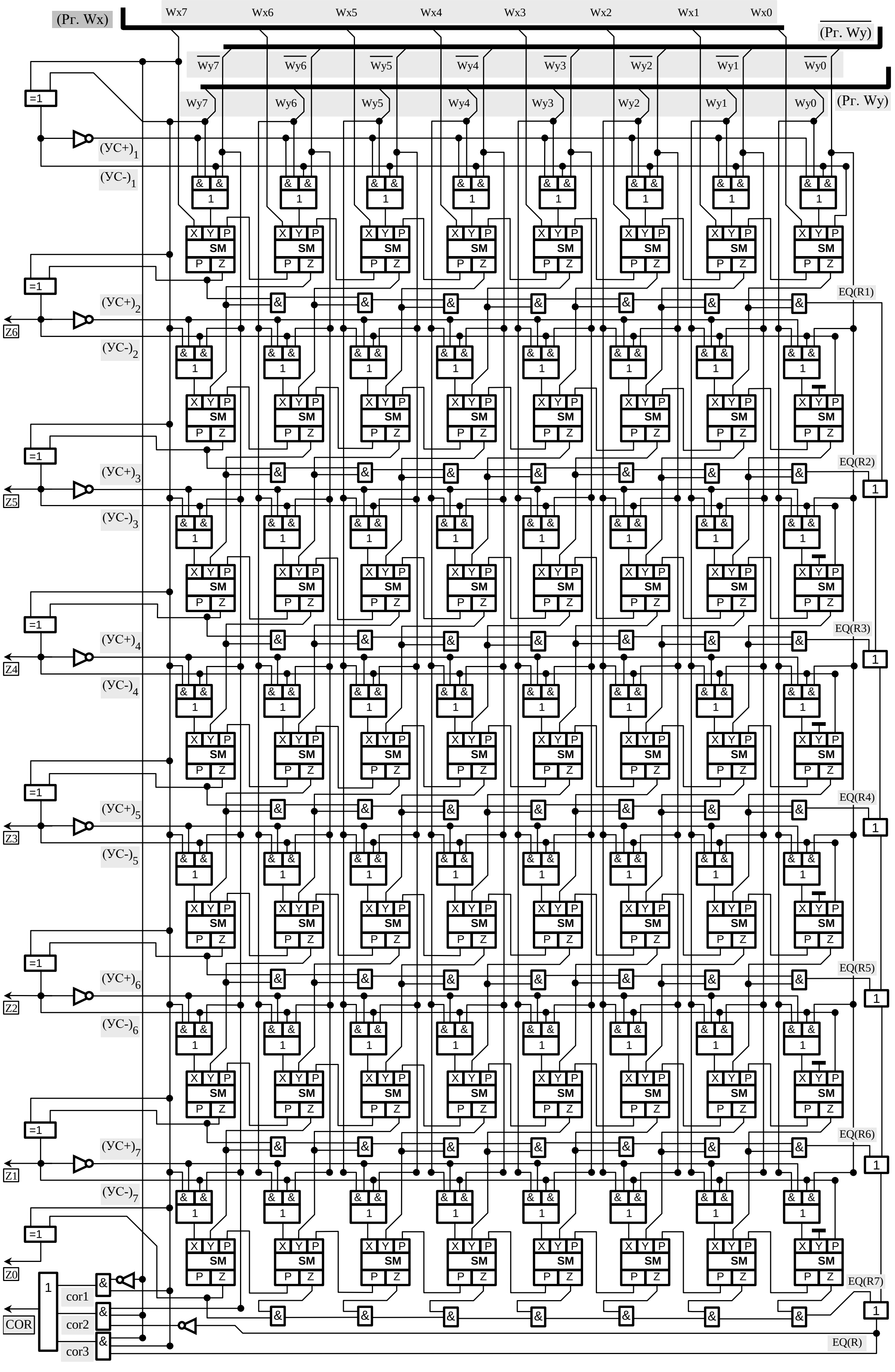


Fig. 1.3. Principal logic scheme of a hardware divider 8x8

After shifting of K-bits to the right, as shown in the examples and in the logical structure presented in Figure 1.2, the most significant N digits of the integer remain in the register of the quotient. As is known, the left-most bit contains the determined initial sign of the quotient z7, and the subsequent digits repeat it, completing at left the shifted N-bit quotient according to the rule for the sign extension.

As noted in the analysis, it is not possible for the hardware solution of the actual division to achieve that dynamics that is inherent in the algorithm. We mean the actual length of the quotient, which varies and it is expressed by the parameter N. The actual length of the quotient is adapted to the scheme by the parameter K, which initializes the programmable shift array. However, the functionality of the solution favors its use in the case of floating point division. As is known, the mantissas of the operands are always left normalized numbers, from which it follows that the quotient always gets the same length. In other words, when divide numbers with a left-fixed point, the hardware divider does not depend in its synthesis on the parameter N and its structure does not require a programmable shift array. In order to align the two cases, the shift array can be further manipulated to be transparent when divide left-fixed point numbers.

Part 2 – Calculation of the Remainder

Theoretical Ground

Based on the synthesis outlined above in the first part, here we present its continuation, referring to the hardware calculation of the second result - the remainder. In conclusion, we describe our idea of the complete structure of the combinational logic scheme of the divider, which makes the division operation identical to the multiplication operation in both structure and performance. In order to optimize the combinational scheme of the divider and minimize its latency, the same methods that are possible on the hardware multiplier [4] can be used.

We have already pointed out that division Z=X/Y is the most complex arithmetic operation, unlike any other and generates two independent results required in the computational algorithms - quotient Z and remainder R. The following definition of operation division can be given: the quotient and the remainder are such numbers that Z.Y+R = X.

We also recall that Y is called a module for comparing number X with other numbers having the same remainder R as well as that the quotient Z is defined as a multiplication factor of the module Y in this conception. Exactly in this interpretation, number X can be expressed by the multiplication factor Z, the comparison module Y and the remainder, or the image R, so X = Z.Y + R.

The iterative algorithm which is applied to obtain the digits of the quotient is preceded with a left normalization of the operands. In the process of this normalization, the integer N, indicating the number of unknown digits of the quotient, is determined. The number N is defined by the equation N = k-l+1.

Presenting the algorithm in [4], it is shown that the partial remainder, which determines the last digit in the quotient, can be used to derive the following equation

$$2^{-(k-l)}.R_{k-l+1} = X - Z.Y \quad .$$

According to the definition given at the beginning, this equality can be written as

$$2^{-(k-l)}.R_{k-l+1} = R \ . \qquad (2.1)$$

Equality (2.1) is remarkable in that it expresses how the second result of a division operation can be obtained, namely the remainder R of the division. The conclusion is that the remainder R is contained in the last partial remainder R_{k-l+1}, which should be shifted (k-1)-bit to the right to be represented correctly as an integer. This is a signed number and will be automatically received in its twos’ complement representation.

There is one further explanation. If the last subtraction yielding the last partial remainder R_{k-l+1}, has been successful, it contains the desired remainder R. However, if the subtraction has been unsuccessful, then the last partial remainder R_{k-l+1} should be restored from the preceding partial remainder R_{k-l}. The desired remainder R is to be contained in R_{k-l}. Restoration of the previous partial remainder is achieved by adding or subtracting the divisor Wy, an operation selected according to the rule as follows

$$\begin{cases} \textit{if} \ \ (R_{k-l+1}[n-1]) = Wy[n-1]\,, \\ \qquad \textit{then} \ \ R_{k-l} = R_{k-l+1} - Wy; \\ \textit{if} \ \ (R_{k-l+1}[n-1]) \neq Wy[n-1]\,, \\ \qquad \textit{then} \ \ R_{k-l} = R_{k-l+1} + Wy. \end{cases} \qquad (2.2)$$

Finally, the remainder R can be calculated applying the following algorithm:

- If the obtained quotient Z is an odd number, the remainder R is contained in the last difference.
- If the obtained quotient Z is an even number, the remainder R is contained in the preceding difference. In this case, for the determination of the remainder, it is necessary to restore the previous difference.
- The final remainder is obtained after an arithmetic shift to the right of the (k-l) bits of the corresponding partial remainder.

Two numerical examples illustrating the explained algorithm are presented below. Both examples illustrate an inexact division, i.e. dvision with remainder. The first example illustrates a division of two positive integers and the quotient being an even number. The latter fact leads the algorithm to the case when the remainder is contained in the preceding difference (000100). It is restored, then (k-l)-bit shifting to the right follows and the remainder is formed.

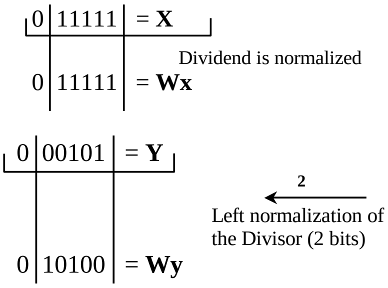
Examples

Example 1. Perform a division operation Z=X/Y of the numbers X=31 and Y=5, which are presented in a bitset of 6 bits (n = 6 [b]).

We should get the following results: quotient Z=6 and remainder R=1, i.e. 31=5.6+1.

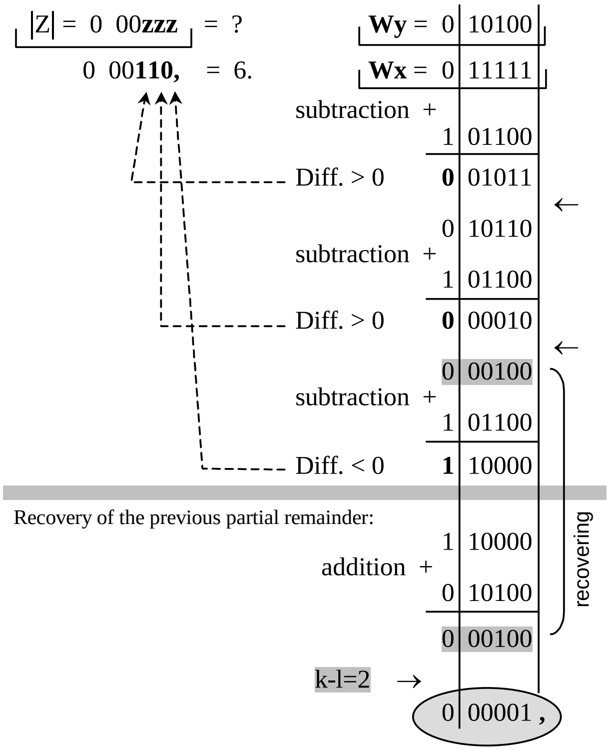
|X| = 0 11111 ; |Y| = 0 00101 .

Normalization of the operands X and Y.



N = 2-0+1 = 3 (3 unknown digits of the quotient)

k-1 = 2-0 = 2 - shifted to 2[b] to form the remainder.



R = 1 .

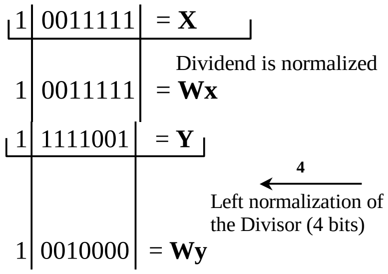
In this case, the quotient does not need a correction: Z=+6.

The second example illustrates the division of two negative numbers, and the quotient is being an odd number. The latter fact leads the algorithm to the case when the remainder is contained in the last difference (10100000). This is the case where the last difference contains the remainder that is formed after the required shifting.

Example 2. Perform a division operation Z=X/Y of the numbers X= -97 and Y= -7, which are presented as twos’ complement numbers in a bitset of 8 bits (n=8[b]). We should get this answer: quotient Z=+13 and remainder R= -6, i.e. (-97) = (-7).13-6.

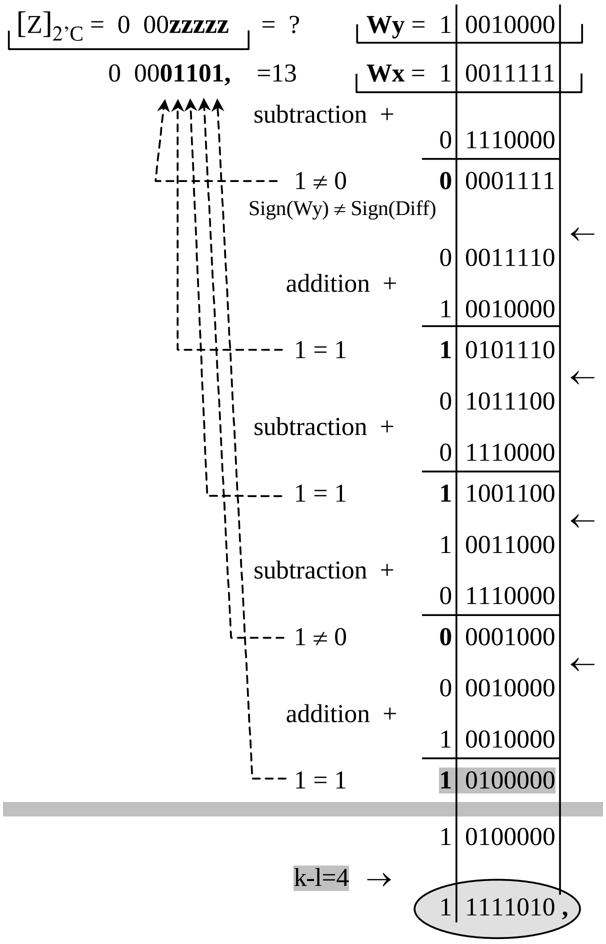
$[X]_{2'sC}$ = 1 0011111 ; $[Y]_{2'sC}$ = 1 1111001 .

Normalization of the operands X and Y.



N=4-0+1 = 5 (5 unknown digits of the quotient)

k-1=4-0 = 4 - shifted to 4 [b] to form the remainder.



= $[\mathbf{R}]_{2'C}$; **R** = -6 .

In this case, the quotient does not need an adjustment: Z=+13.

Operation division is the most complex of all operations on integer numbers. Various situations in various combinations can occur during their execution. Such situations are indefiniteness (Y=0), overflow, exact division, and prematurely exact division. To illustrate all these cases, a number of numerical examples should be performed. Such examples, which are useful for real synthesis, can be seen in [8].

Synthesis of the Logical Structure

The exposed theoretical grounds, algorithms and numerical examples allow us to synthesize that part of a logical structure that complements the one presented in the first part of the explanation to its final appearance.

We consider the presentation of our project to be completed with what we have said here. There are, of course, a few more details, such as the case of undefined operation (when Y=0), or the case of overflow when divide integer numbers, etc. The reflection of these subtleties in the synthesized scheme is quite possible, requiring only an excellent knowledge of the algorithm.

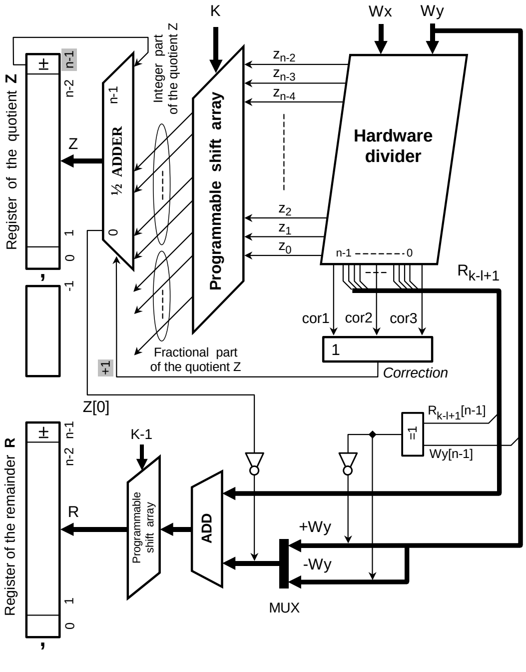


Fig. 2.1. Logical structure of the hardware divider, supplemented by the elements needed to calculate the remainder

The complete implementation of the hardware divider in the form of a single combinational scheme is entirely possible. This translates the division into a single cycle operation, making it analogous to multiplication operation, as well as to some structural elements in the Floating Point Units (FPU).

Conclusion

We briefly describe the entire composition of this sophisticated combinational circuit. Only 4 registers are needed - 2 input registers for the input operands X and Y and two output registers for both results - quotient Z and remainder R. Between these two pairs of registers, the following composite combinational circuits are sequentially arranged. First of all, on the input registers, there are schemes for determining the number of the leftmost insignificant digits of the operands. These are combinational schemes that are synthesized in our project, published in [7].

The numbers formed by these two schemes are fed to an adder that calculates the above mentioned number N. As a result of this adder, there may be also the numbers K and (K-1) required to initialize the next functional elements of the scheme. The operands are still at the input of the hardware divider, and two combinations come at its outputs. The first one, passing through the shift array and the half-adder ½ ADDER (Figure 1.1), is loaded in the register RGZ as a first result - quotient Z. The second one, representing the last difference R_{n-1}, passes sequentially through the adder ADD and the right shifting array and is loaded in the output register RGR as a second result - remainder R. If the scheme is used to divide floating-point numbers, the remainder loses its meaning and the number K takes a value of 0 that makes the shifting array transparent.

The two numerical examples presented in the article illustrate the functioning of the algorithm and the synthesized logical structure. More numerical examples can be seen in [8].

The sequential elements in the described combinational scheme can be organized into a micro-pipeline shown in Figure 2.2. The pipeline organization will allow to increase the performance of the calculation unit in cases where the mathematical calculations contain multiple consecutive division operations. The micro-pipeline control of the hardware divider can be achieved with the methods and means, which are described in detailes in the monograph [9].

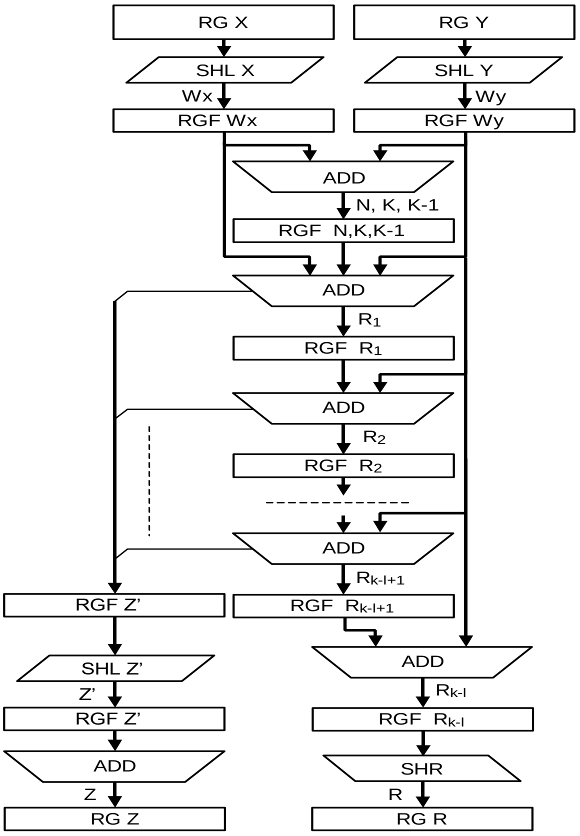


Fig. 2.2. Examplary logical structure for micro-pipeline organization of the hardware divider

As can be seen from the structure of the pipeline, it includes RGF registers, combinational schemes – adders ADD, programmable arrays shifting to the left – SHL, and right - SHR. The structure also includes finite state machines required to control the individual stages of the pipeline, which are depicted on the diagram.

References

[1]. Richard, P., Zimmermann, B., Zimmermann, P., *(2010), Modern Computer Arithmetic*, Cambridge Monographs on Computational and Applied Mathematics (No. 18), Cambridge University Press, November 2010, 236 pages. ISBN-13: 978-0521194693.

[2]. Cavagnino, D., Werbrouck, A., *(2008), Efficient Algorithms for Integer Division by Constants Using Multiplication*, The Computer Journal, Volume 51, Issue 4, 1 July 2008, Pages 470–480.

[3]. Kumar, D., Saha, P., Dandapat, A., *(2017), Hardware Implementation of Methodologies of Fixed Point Division Algorithms*, International Journal of Smart Sensing and Inteligent Systems, Vol. 10, No.3, September 2017.

[4]. Tyanev, D. S., *(2008), Computer Organization. Tom1, Tom 2,* Technical University of Varna, ISBN: 978-954-20-0412-7, ISBN 978-954-20-0413-4.
Accessible from: http://www.tyanev.com/home.php?lang=bg&mid=18&mod=1&b=12 .

[5]. Trummer, R., Zinterhof, P., Trobec, R., (2005), A High-Performance Data-Dependent Hardware Divider, Parallel Numerics'05, 193-206 M. Vajterˇsic, R. Trobec, P. Zinterhof, A. Uhl (Eds.) Chapter 7: Systems and Simulation, ISBN: 961-6303-67-8.

[6]. Takagi N., Kadowaki, S., Takagi, K., (2005), A hardware algorithm for integer division. Computer Arithmetic. ARITH-17 2005, 17th IEEE Symposium, ISSN: 1063-6889.

[7]. Tyanev, D. S., Petkova, Y. P., (2015), Logic *scheme for determining the number of leftmost insignificant digits in a bit-set of any length*. SciTechnol: Journal of Computer Engineering & Information Technology, USA, ISSN: 2324-9307, 2015, Vol. 4, Issue 1. doi: 10.4172/2324-9307.1000123.
Accessible from: https://www.scitechnol.com/logic-scheme-for-determining-the-number-of-leftmost-insignificant-digits-in-a-bitset-of-any-length-hRjK.php?article_id=3259

[8]. Tyanev, D. S., *(2007), Computer organization. Digital arithmetic's – exercises*, Technical University of Varna, ISBN: 954-20-0258-0.
Accessible from: http://www.tyanev.com/home.php?lang=bg&mid=18&mod=1&b=7 .

[9]. Tyanev, D. S., *(2016), Asynchronous pipeline systems with common structures (Synthesis Methodology),* Technical University of Varna.
Accessible from: http://www.tyanev.com/home.php?lang=bg&mid=18&mod=1&b=14

[10]. Lemire, D., Kaser, O., Kurz, N., *(2019), Faster Remainder by Direct Computation Applications to Compilers and Software Libraries.* Accessible from: https://arxiv.org/abs/1902.01961.pdf .

Received 20.05.2019

Д.С. ТЯНЕВ, Ю.П. ПЕТКОВА
Технический университет Варны, Болгария.

АРИФМЕТИЧЕСКОЕ ДЕЛЕНИЕ. ЧАСТНОЕ И ОСТАТОК. ЛОГИЧЕСКИЕ СТРУКТУРЫ И ОПЕРАЦИОНЫЕ СХЕМЫ

Представлен проект вычислителя для быстрого выполнения операции деления целых чисел со знаком. Конечным результатом синтеза является полная и уникальная комбинационная схема. Операнды и результаты операции являются числами, представленными в дополнительном коде. Приведены синтез логической структуры и комбинированной схемы для расчета первого результата – частного и синтезированный алгоритм и логическая схема для вычисления второго результата - остатка. Операция выполняется в течение времени переключения схемы комбинации, то есть вычисление двух результатов происходит максимально быстро.

Ключевые слова: операция деление, частное, остаток, алгоритм, логическая схема.

Д. С. ТЯНЕВ, Ю.П. ПЕТКОВА
Технічний університет Варни, Болгарія.

АРИФМЕТИЧНЕ ДІЛЕННЯ. ЧАСТКА І ЗАЛИШОК. ЛОГІЧНІ СТРУКТУРИ І ОПЕРАЦІЙНІ СХЕМИ

Представлений проект обчислювача для швидкого виконання операції ділення цілих чисел зі знаком. Кінцевим результатом синтезу є повна і унікальна комбінаційна схема. Синтез вимагав представлення теоретичного обгрунтування для операції ділення і отриманих алгоритмів для обчислення частки і залишку. Операнди і результати операції є числами, представленими в додатковому коді. У статті наведено синтез логічної структури і комбінованої схеми для розрахунку першого результату – частки, а також синтезований алгоритм і логічну схему для обчислення другого результату – залишку. Операція виконується протягом часу перемикання комбінаційної схеми, таким чином, обчислення двох результатів відбувається максимально швидко.

Ключові слова: операція ділення, приватне, залишок, алгоритм, логічна схема.