JPRS L/10111

10 November 1981

Translation

COMPUTING SYSTEMS AND SYNCHRONOUS ARITHMETIC

By

M.A. Kartsev and V.A. Brik

FBIS FOREIGN BROADCAST INFORMATION SERVICE

FOR OFFICIAL USE ONLY

APPROVED FOR RELEASE: 2007/02/09: CIA-RDP82-00850R000400070026-2

NOTE

JPRS publications contain information primarily from foreign newspapers, periodicals and books, but also from news agency transmissions and broadcasts. Materials from foreign-language sources are translated; those from English-language sources are transcribed or reprinted, with the original phrasing and other characteristics retained.

Headlines, editorial reports, and material enclosed in brackets [] are supplied by JPRS. Processing indicators such as [Text] or [Excerpt] in the first line of each item, or following the last line of a brief, indicate how the original information was processed. Where no processing indicator is given, the information was summarized or extracted.

Unfamiliar names rendered phonetically or transliterated are enclosed in parentheses. Words or names preceded by a question mark and enclosed in parentheses were not clear in the original but have been supplied as appropriate in context. Other unattributed parenthetical notes within the body of an item originate with the source. Times within items are as given by source.

The contents of this publication in no way represent the policies, views or attitudes of the U.S. Government.

APPROVED FOR RELEASE: 2007/02/09: CIA-RDP82-00850R000400070026-2

JPRS L/10111

10 November 1981

COMPUTING SYSTEMS AND SYNCHRONOUS ARITHMETIC

Moscow VYCHISLITEL'NYYE SISTEMY I SINKHRONNAYA ARIFMETIKA in Russian 1981 (signed to press 26 Mar 81) pp 2-5, 162-275, 348-359

[Annotation, table of contents, preface, chapters 4 and 5, bibliography and index from book "Computing Systems and Synchronous Arithmetic", by Mikhail Aleksandrovich Kartsev and Vladimir Arkad'yevich Brik, Izdatel'stvo "Radio i svyaz'", 10,000 copies, 360 pages, UDC 681.32]

CONTENTS

[Text] Annotation

The authors investigate, on the one hand, the organization of structures of multimachine, multiprocessor and conveyor computing systems and the organization of computations in them, and one the other hand, the majority of known methods for building high-speed synchronous adders, multipliers and devices for division used in computing systems and machines.

Preface

When we started on this book, we realized there is currently no shortage of material on computing systems. On the contrary, the flow of books and articles on this subject is essentially outstripping the development of technology and methods of applying computing systems. The number of titles of just monographs on this subject probably exceeds the number of computing systems that have been implemented and are operating in the world. Meanwhile, there are still too many points not yet clear in this field.

Even the terminology has not stabilized. In this book, by the phrase "computing system," used in the title, we mean computing resources designed to execute parallel computations; a precise definition of this concept is in section 1.1.1.

But the question least analyzed, in our view, is throughput of computing systems. Too often the data given on the throughput of a computing system in design or production are obtained by simply adding the throughput of the individual resources that make up the system. Meanwhile, the situations users of the actual computing system encounter may be considerably different; accordingly, the data on system throughput that he needs are also different.

If the system is used within a major computer center, where a large number of users solve their relatively small problems, total system throughput is of little interest to each individual user. It is important to him only to the extent that it affects

the time he gets for operating in the interactive mode or the jobs accepted from him for processing in the batch mode.

We deal with a very similar situation, as a rule, in using a computing system within an automated production management system (on the scale of an enterprise, sector and the national economy as a whole) and in various information retrieval systems. Here too there is usually a number of small jobs with few links to each other, but operating with a common data bank. But the designer of an ASU [automated management system] or information retrieval system must, of course, be interested in the total throughput of computing resources since the specific set of jobs in his system must be executed within specific time intervals (sometimes--within several hours, or several days or several weeks).

We encounter a fundamentally different situation in automated process control systems, in solving major scientific problems and in other cases. High throughput of compyting resources is needed in this case to obtain within a brief time a solution to one, but rather massive, problem. It seems to us in many cases neither the computer system designers, nor those expecting to use the systems for the purposes indicated clearly understand that the same computer system cannot achieve identical throughput in solving major problems of different classes and that there has to be a specific correspondence between the specific problem properties and the computing system structure for computing system capabilities to be used reasonably efficiently.

In this book, when we speak of high-throughput computing systems, basically we have in mind precisely the latter situation (precise definitions of actual user throughput and user efficiency of a computing system are given in section 1.1.2.).

Chapters 1 and 2 are devoted to a detailed consideration of precisely these points. Singled out as a result is the type of synchronous computing systems which potentially can achieve the maximum in actual user throughput. Naturally, this throughput can be implemented when a certain class of problems is run (i.e. problems having certain properties), but this class is rather broad and includes rather major problems.

Chapters 3-6 are devoted to an examination of the most complex technical questions that arise in building these systems--development of synchronous methods of executing arithmetic and logic operations, i.e. methods that provide for minimal time in executing an operation irrespective of the operands on which it is executed.

The importance of these chapters is considerably broader than could be deduced from the preceding, since the use of synchronous methods for executing operations is necessary even when building conventional (single-processor) control machines designed to operate in real time. In many cases, these synchronous methods of executing operations achieve higher speed than the well known asynchronous methods, and it is expedient to apply them in developing any high-speed digital computer in general.

Chapter 7 is intended for the reader who needs more background and who would like to understand fully the content of the whole book. He should start reading the book precisely from this chapter which contains elementary information on the principles of computer technology.

Chapters 1 and 2 of this book were written by M. A. Kartsev and chapters 3-7 by V. A. Brik.

The authors will be grateful to readers for comments made on the book's content.

V. A. Brik, M. A. Kartsev

4. High-Speed Synchronous Multipliers

Multiplication has to be performed to solve the majority of computing problems. Problems that include division can often be solved by using multiplication, addition and subtraction. Division can be performed, for example, by using multiplication and tables of inverse values stored in memory. There are many methods for "getting around" division, i.e. accomplishing it without special-purpose devices designed for this.

Multiplication cannot be "avoided." The only way to avoid direct calculation of a product is to use multiplication tables. However, employing this method in machines with parallel operation requires a large amount of storage. Any software method of multiplication requires repeated addition which cannot be performed as quickly as multiplication is performed by using high-speed hardware methods. Therefore, the rapid hardware method for performing multiplication is a very common approach in designing the arithmetic unit.

All known synchronous methods for speeding up multiplication reduce essentially to the separate or complex use of the following three methods:

1. Reducing the time needed for performing addition of partial products, i.e. the products of the multiplicand by the individual digits or groups of digits of the multiplier.

2. Reducing the number of partial products by using multiplicand multiples formed in advance.

3. Using subtraction in multiplication which permits reducing the number of additional adders needed to form multiples of the multiplicand.

This classification is based on logical and mathematical ideas, the implementation of which leads to raising speed. Each of these directions is embodied in the group of methods for speeding up multiplication, the hardware solutions of which can differ considerably from each other.

In this chapter, the basic synchronous methods for speeding up execution of multiplication are discussed.

4.1. Decreasing the Time for Adding Partial Products

The time for adding all the partial products can be reduced by three methods: speeding up the procedure for adding the next partial product to the sum of the preceding partial products; starting the addition of the following partial product prior to completion of the addition of the preceding partial product; and finally, building circuits that add the running sum of partial products at once to several successive partial products.

The main implementation of the first method is the use of a fast adder for adding the next partial product to the sum of the preceding. If one desires, this group of methods may also include the use of the well known multiplying circuits that overlap addition with a shift of the multiplicand (fig. 7.7.1, b and d [not reproduced]).

The second method is implemented in the method of multiplying in which the so-called carry save adder is used [1 and section 4.3.2.]. The idea of this method consists in forming the running sum of the next partial product with the sum of the preceding partial products in the form of a two-digit code, i.e. in the form of two numbers. In the extreme case, one of these two numbers is formed from the step-by-step sums s, and the other from the step-by-step carries e. The principle of operation of this device is illustrated by the simplified structural diagram shown in fig. 4.1.1.

Key:
1. Rg1 -- register 1
2. Rg2 -- register 2
3. Rg3 -- register 3
4. Rg4 -- register 4
5. P -- next partial product
6. Sm -- coincidence-type adder
7. e -- step-by-step carries
8. s -- step-by-step sums

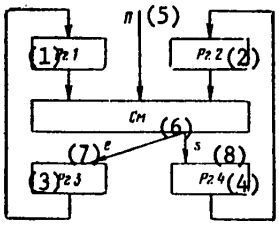


Fig. 4.1.1

Key:
1. Rg1 -- register 1
2. Rg2 -- register 2
3. Rg3 -- register 3
4. Rg4 -- register 4
5. P1 -- partial product 1
6. Sm1 -- adder 1
7. e -- step-by-step carries
8. s -- step-by-step sums
9. P2 -- partial product 2
10. Sm2 -- adder 2
11. Sm3 -- adder 3

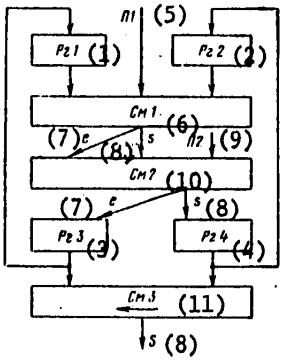


Fig. 4.1.2

During each cycle, the next partial product P is added to the sum of the preceding partial products in the coincidence-type adder Sm. At the end of the cycle, the values of the signals e and s are stored in registers Rg3 and Rg4, and at the start of the next cycle are sent to registers Rg1 and Rg2. A gain in speed is achieved because there is no carry propagation process in the adder Sm; its operating time in this case is reduced, obviously, to the operating time of one one-digit adder.

With more economical (but also slower) versions, the entire adder is subdivided not into individual digits, but into relatively quick q-digit ($1 < q < n$) adders. In doing so, the first of the two indicated numbers is formed from the outputs of the sums of these adders, and the second number, containing q-fold fewer significant digits, from the carry signals generated in the q-digit adders (by one signal for each such adder).

In any of these versions, after the two-digit code of the final product is obtained, the two-digit code is translated into one-digit, i.e. conventional addition. This may be done in a conventional parallel, or even better--in a fast parallel, adder. In the device shown in fig. 4.1.1, final addition may be performed in the adder Sm, which must in this case, after running through all the cycles of addition with the partial products, be "retuned" from the mode for translating three-digit code (i.e. three numbers) into two-digit--to the mode for translating two-digit code into one-digit. Instead of this, final addition may be performed in a separate adder, which should be installed after registers Rg3 and Rg4. This version of the device will be shown in fig. 4.1.2.

The third method for speeding up the process of adding partial products assumes the introduction of additional adders into the arithmetic unit. The number m of partial products that are simultaneously added to the running sum of partial products may vary. A multiplication operation, in the process, consists of a series of cycles, during each of which m new partial products are added. In doing so, the carry storage method is also usually used and this speeds up the process further. In fig. 4.1.2, as an example, it is shown how another two (m = 2) partial products P1 and P2 can be added in each cycle to the running sum of partial products and with that the sum formed in the form of two-digit code. Final addition in this scheme is done by adder Sm3 which it is advisable to make fast.

In the ultimate version, the multiplier composes at once all partial products. Multiplication in this case is performed in one cycle. Multipliers of this type are called simultaneous (array, synchronous, pyramids of adders etc.). An analysis and description of these devices are given in sections 4.4. and 6.3.

4.2. Preliminary Formation of Multiples of the Multiplicand

In this method of multiplication, the multiplier is subdivided into groups of q digits each and may have yet another group containing less than q digits. Each group of digits is decoded independently of the others as a conventional binary number. Considering by convention that the point is located at the right of the group, in decoding there is generated one of the numbers (signals)

$$0, 1, 2, 3, \ldots, 2^q - 1$$

and used as the next partial product is respectively one of the 2^q multiples of the multiplicand C, shifted the necessary way relative to the running sum of partial products:

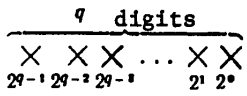
$$0, 1C, 2C, 3C, \ldots, (2^q - 1)C.$$

The rule for decoding one group for the case q = 3 is shown in table 4.2.1.

Table 4.2.1.

Digits of group	Multiple	Digits of group	Multiple	Digits of group	Multiple	Digits of group	Multiple
000	0	010	2C	100	4C	110	6C
001	1C	011	3C	101	5C	111	7C

Thus, the individual digits of the group when decoded correspond to the natural weights:

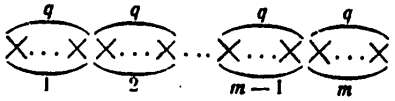


Let us designate the number of groups into which the multiplier is subdivided (or, which is the same, the number of partial products) by m. The relationship between n, q and m has the following form:

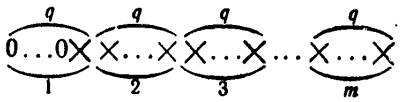
$$m=\left]\frac{n}{q}\right[^{*}. \qquad (4.2.1)$$

*)]d[is the smallest integer not less than d.

Extreme cases of subdividing of the multiplier into groups are the case when all groups are "full":



(lengths of the groups are indicated at the top, their numbers at the bottom), and the case when a group not full contains only one digit:



(the group not full is shown at the left, but it can be in any other position of the multiplier). The remaining cases are intermediate between these two. What has been said is illustrated by the double inequality

$$(m-1)q+1\leqslant n\leqslant mq,$$

the right and left boundaries of which correspond to the two indicated extreme cases of subdivision. The relationship (4.2.]) is valid in all cases.

Those multiples of KC, for which K is odd and greater than one, are formed in advance by using auxiliary adders, the number of which, evidently, is equal to $2^{q-1}-1$. The remaining multiples of the multiplicand are derived through additional shifts from C and from the multiples generated by the auxiliary adders. Thus, when q = 2, an adder for forming 3C is required; when q = 3, adders of 3C, 5C and 7C are needed; when q = 4, adders of 3C, 5C, 7C, 9C, 11C, 13C and 15C, etc.

One group of auxiliary adders may compute the multiples of KC, using as source numbers the values of $\pm C2^t$ (t is an integer) obtained by a simple shift of the multiplicand. For example: 3C = C + 2C, 5C = 4C + C, 7C = 8C - C, 9C = 8C + C, 15C = 16C - C, etc. The other auxiliary adders have to use the outputs of the adders of the first group. For example: 13C = 8C + 5C or 13C = 16C - 3C, etc.

It is evident that when this method is used, decoding of the groups may begin (i.e. multiplication may begin) at either end of the multiplier. It is also possible to decode simultaneously several groups of digits of the multiplier; this necessity occurs when several partial products are added simultaneously in the multiplier, as was shown, for example, in fig. 4.1.2.

It is believed that when this method is used, multiplication time is reduced about q-fold (we assume that the partial products in turn are added in a parallel accumulating adder, do not consider the possibility of the presence of one not full group of digits of the multiplier and ignore the time spent by the auxiliary adders on forming the multiples of the multiplicand in the beginning of the operation). This comment also pertains to the methods discussed in section 4.3.

4.3. Use of Negative Partial Products

This method is usually applied in combination with the preceding. Discussed below are two alternatives for using subtraction when multiplying. In both cases, each partial product corresponds to multiplication of the multiplicand by a group of q digits of the multiplier. The main difference is that with the first alternative, decoding may begin at either end of the multiplier, while in the second, multiplication is done from the low-order digits of the multiplier. There are other differences too.

4.3.1. Method of Multiplication by a Group of q Digits of the Multiplier with Decoding of q + 1 Digits of the Multiplier

When this method of speeding up multiplication is used, the next group of q digits of the multiplier is decoded together with the high-order digit of the adjacent low-order group, which is considered an additional low-order digit. All digits of the group, except the high-order and additional, are assigned the same natural weights as in the method described in section 4.2. The high-order digit of the group is assigned the weight -2^{q-1}, and the additional digit the weight 1:

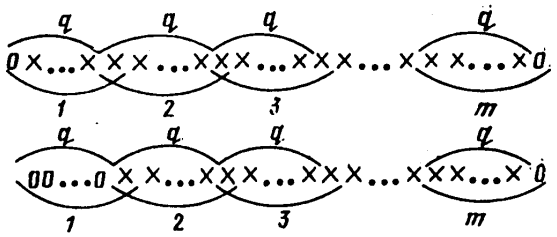
$$\overbrace{\times\ \times\ \times \ldots \times\ \times}^{q+1\ \text{digits}}$$
$$-2^{q-1}\ 2^{q-2}\ 2^{q-3} \quad 2^1\ 2^0\ 2^0$$

Thus, the high-order digit of each group is decoded twice: once as the high-order digit of this group, and again as the additional digit of the adjacent high-order group.

The high-order digit of the highest-order group and the additional low-order digit of the lowest-order group must always be zeroes. Therefore, there must be satisfied the double inequality

$$(m-1)q \leqslant n \leqslant mq-1,$$

the boundaries of which are explained the following way (each lower arc indicates the simultaneously decoded digits):



The number of groups of digits of the multiplier (number of partial products) with this method is equal to

$$m = \left]\frac{n+1}{q}\right[.$$

The sum of the weights of the nonzero digits of the group determines the factor K in the multiple K_C selected as the next partial product. The value of K in the process assumes one of the values

$$0, \pm 1, \pm 2, \pm 3, \ldots, \pm 2^{q-1},$$

i.e. the number of multiples is increased by one in comparison to the preceding method, but in return the number of additional adders for forming the multiples of the multiplicand is reduced to $2^{q-2} - 1$.

The decoding rules are illustrated by table 4.3.1 (q = 3).

Table 4.3.1.

(1) Разряды группы	K	(1) Разряды группы	K	(1) Разряды группы	K	(1) Разряды группы	K
0000	0	0100	2	1000	—4	1100	—2
0001	1	0101	3	1001	—3	1101	—1
0010	1	0110	3	1010	—3	1110	—1
0011	2	0111	4	1011	—2	1111	0

Key:
1. Digits of group

4.3.2 Method of Multiplication from Low-Order Digits of the Multiplier

In decoding the next group of digits of the multiplier by this method, an analysis is made of q next digits and one binary digit of the "carry" from the preceding (adjacent low-order) group into the given one. Let us designate this digit by the letter e. The weights assigned to the individual digits in the process have the following values:

$$\overbrace{\times\ \times\ \times \cdots \times\ \times}^{q\ \text{digits}}\ e$$
$$2^{q-1}\ 2^{q-2}\ 2^{q-3} \qquad 2^1\ 2^0\ \ 2^0$$

In other words, the digits of the multiplier have natural weights, and the weight of the digit e is equal to one. The factor K in the multiple of the multiplicand KC, used as the next partial product, is selected from the 2^q values

$$0, \pm 1, \pm 2, \pm 3, \ldots, \pm (2^{q-1}-1), 2^{q-1}$$

in such a way that there is produced the relationship

$$K+2^q e'=S,$$

where e' is the digit, selected simultaneously with K, of the "carry" into the next (adjacent high-order) group of digits of the multiplier (e' = 0 or e' = 1), and S is the sum of the weights of the nonzero digits of the decoded group of digits of the multiplier together with the weight of the nonzero digit e. The number of additional adders for forming the multiples of the multiplicand in the process, as in the preceding method, is equal to $2^{q-2}-1$.

One of the possible rules for decoding one group of digits of the multiplier for the case of q = 3 is shown in table 4.3.2.

Another alternative for decoding for the case of q = 3 was offered in work [2] (see table 4.3.3).

Table 4.3.2

Digits of group	e	K	e'	Digits of group	e	K	e'
000	0	0	0	000	1	1	0
001	0	1	0	001	1	2	0
010	0	2	0	010	1	3	0
011	0	3	0	011	1	4	0
100	0	4	0	100	1	−3	1
101	0	−3	1	101	1	−2	1
110	0	−2	1	110	1	−1	1
111	0	−1	1	111	1	0	1

Table 4.3.3.

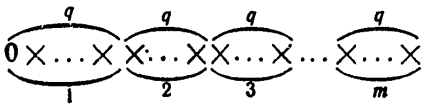
Digits of group	e	K	e'	Digits of group	e	K	e'
000	0	0	0	000	1	1	0
001	0	1	0	001	1	2	0
010	0	2	0	110	1	3	0
011	0	3	0	011	1	4	0
100	0	4	0	100	1	−3	1
101	0	−3	1	101	1	6	0
110	0	6	0	110	1	−1	1
111	0	−1	1	111	1	0	1

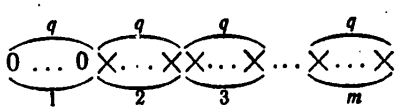
As can be seen, the number of multiples used in both cases is 8, but the multiple 6C emerges in place of the multiple -2C. Other versions of decoding rules can be suggested too.

Among the possible alternatives, a certain advantage is possessed by those in which $e' = 0$ is produced when the high-order digit of the group being decoded equals zero. Such, in particular, are both versions shown in tables 4.3.2 and 4.3.3. In the process, when the groups are formed in a way such that there is always a zero on the left in the high-order group, it is thereby possible to avoid a carry from this group, i.e. avoid the necessity of forming yet another partial product. Therefore, the parameters n, m and q must be associated by the relationship

$$(m-1)q \leqslant n \leqslant mq-1.$$

The right and left boundaries of the inequality correspond to the following versions of subdividing the multiplier into groups of digits:





As in the preceding method, the number of groups of digits of the multiplier (number of partial products) is equal to

$$m = \left\lceil \frac{n+1}{q} \right\rceil.$$

Table 4.3.4.

Characteristics of method	I	II	III
Number of multiplicand multiples required	2^q	2^q+1	2^q
Number of additional adders required	$2^{q-1}-1$	$2^{q-1}-1$	$2^{q-1}-1$
Relationship between n, m and q	$m=\left]\frac{n}{q}\right[$	$m=\left]\frac{n+1}{q}\right[$	$m=\left]\frac{n+1}{q}\right[$

Shown in table 4.3.4 is a comparison of the methods just discussed for speeding up multiplication, in the use of which the multiplicand is multiplied at once by a group consisting of q digits of the multiplier. The numbers I, II and III designate: I -- the method using only positive multiples of the multiplicand (section 4.2); II and III -- the methods using not only positive, but also negative multiples (sections 4.3.1 and 4.3.2 respectively).

To illustrate these methods, let us show how multiplication of some multiplicand C by multiplier A, equal to 0.101100011 (n = 9), is performed by all three methods when q = 3.

In the first case, subdivision into groups and decoding would be such:

$$0,\underbrace{101}_{5}\underbrace{100}_{4}\underbrace{011}_{3}$$

and the multiplication would actually be performed in three cycles the following way:

$$CA=3C2^{-3}+4C2^{-6}+5C2^{-9}.$$

In the second case, the subdivision might be such:

$$\underbrace{000}_{1}\underbrace{,101}_{-2}\underbrace{100}_{-4}\underbrace{011}_{3}0$$

and there would be four cycles:

$$CA=3C2^{-3}-4C2^{-6}-2C2^{-9}+1C2^{0}.$$

In the third case, let us subdivide the multiplier the following way:

$$\underbrace{000}_{1},\underbrace{101}_{-3}\underbrace{100}_{4}\underbrace{011}_{3}$$

and there would again be four cycles

$$CA=3C2^{-3}+4C2^{-6}-3C2^{-9}+1C2^{0}.$$

In comparing the merits and shortcomings of all three methods, it should be noted that

with n, a multiple of q, multiplication by methods II and III requires execution of one more cycle than is the case in using method I;

to implement methods II and III, substantially fewer additional adders are required than that for implementation of method I;

the number of multiplicand multiples for methods I and III is one less than for method II; and

in contrast to method III, methods I and II are suitable for application in array multipliers (see section 4.4), since all groups of digits of the multiplier can be decoded simultaneously when they are used.

Thus, each of the three methods has certain advantages that favorably distinguish it compared to at least one of the other two methods.

As another illustration of this group of methods, let us consider a possible hardware implementation for multiplication with the parameters n = 8 and q = 3 which implements method III (fig. 4.3.1). Version "a" of the circuit for non-fast

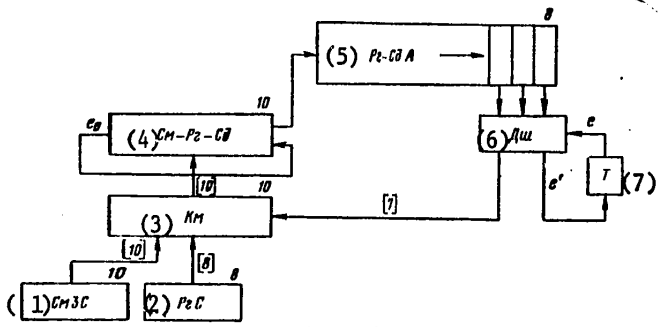


Fig. 4.3.1.

Key:

1. Sm3C -- adder 3C
2. RgC -- register C
3. Km -- switch
4. Sm-Rg-Sd -- adder-register-shifter
5. Rg-SdA -- register-shifter A
6. Dsh -- decoder
7. T -- flip-flop

multiplication was chosen as the "prototype" of the device (see fig. 7.7.1, a, page 337 [not reproduced]). In our case (fig. 4.3.1), the first cycle of multiplication begins with the decoding of the three low-order digits of the multiplier A located in the register-shifter. If among these three digits there is at least one "one," then the decoder Dsh produces one of seven control signals +1, -1, +2, -2, +3, -3 or +4 (the number of signals is indicated in the corresponding positions in fig. 4.3.1 in brackets) and the carry signal e' (the signal e in the first cycle equals zero). The control signal generated passes through the switch Km into the 10-digit accumulating adder-shifter as the first partial product the corresponding multiple of the multiplicand C from the register RgC or the adder Sm3C, which even before the start of the first cycle calculates and stores the 10-digit triple multiple 3C (let us number the digits of this number from 1 to 10). If, for example, the three

low-order digits of the multiplier equal 101, then the signal e' = 1 is generated and stored in flip-flop T and there is generated the control signal -3 which passes through the switch Km the number 3C in inverted code (one of the 10 digits of the switch is shown in fig. 4.3.2). The partial product is added to the contents of the adder (when the first cycle is executed, the contents equal zero) and the sum is stored. At the end of the cycle, there is a shift right three positions in the adder and in the register for the multiplier. In the process, the three low-order digits of the product move from the adder into the freed high-order positions of the register for the multiplier. The second and third cycles are executed similarly, but there are no shifts at the end of the third cycle. The cyclic carry circuit e_0 is used in the adder during addition. Among the other details, one could note that the circuit in fig. 4.3.2 changes slightly for the extreme digits of the switch, since it is necessary to consider that $c_i = 0$ when i = 9 or 10; $c_{i-1} = 0$ when i = 1 or 10; and $c_{i-2} = 0$ when i = 1 or 2.

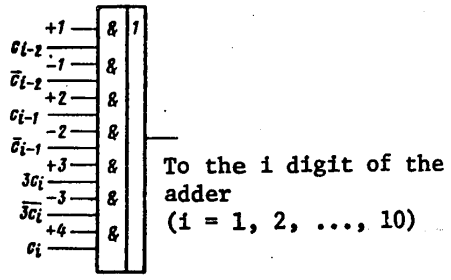


Fig. 4.3.2.

4.4. Analysis and Technique of Building Simultaneous Multipliers

4.4.1. Classification of Simultaneous Multipliers

It was already mentioned at the beginning of this chapter that the modern computer spends a considerable portion of its time on performing multiplication. In a number of developments of new machines, the arithmetic unit is being freed from many trivial operations, which leads to an increase in the percentage of time spent on multiplication and division. The arithmetic unit of such a machine spends about half of its time on these two operations. Nevertheless, the capacity of the apparatus in the machine for these operations is rarely great. In major computing systems and digital computers, the situation often occurs in which to improve the overall economic indicators of the system with large storage and high-level peripherals and control, it is advantageous to increase the outlays for multiplication and division even higher than the level in which the outlay increment yields an equal increment in the speed of multiplication and division. Simultaneous multipliers (OU) are among the means of this type of speeding up.

The main feature of simultaneous multipliers is that execution of multiplication in them is a unified, continuous, complex, transient process of simultaneous addition of all partial products, which is not divided into smaller time intervals by clock signals as is done in multipliers of any other type.

Assume that it is necessary to form the final 2n-digit product of two n-digit factors and that additional multiples of the multiplicand are not used in the multiplier (use of these multiples was covered in sections 4.2 and 4.3). Then the total

number of digits in all addends (partial products) equals n^2. We shall see later that with the variation of the value of n, the quantity of apparatus in a simultaneous multiplier of a given type varies approximately in proportion to n^2. Therefore, the application of simultaneous multipliers has been called in work [3] the method of the second order of speeding up multiplication.

It has been suggested [4] that the classification of simultaneous multipliers be based on dividing them into array simultaneous multipliers and compound simultaneous multipliers (fig. 4.4.1). Array simultaneous multipliers [4-29] contain about n^2 elements of the same type (for example, one-digit binary adders), connected to each other by a given method, which add up all the partial products. If additional multiples of the multiplicand (see sections 4.2 and 4.3) are used, the number of adding elements is reduced accordingly and a decoder for the multiplier and assembly for forming the necessary multiples of the multiplicand are incorporated into the device.

Simultaneous array multipliers may in turn be divided into homogeneous and heterogeneous. Simultaneous homogeneous array multipliers belong to the class of devices often called iterative arrays and are a set of identical elements that form regular network. Because of the homogeneity of the structure and adherence to the principle of "shorter-range interaction" of the elements, the application of these devices is very promising in high-throughput large and medium-size computers, developed on the base of the modern technology of manufacture of integrated circuits. A large number of iterative arrays has now been developed that are multiplying, dividing [22-24, 31-34], adding and other calculating assemblies; among the latter can be noted circuits for calculating the square root [34-36], for multiplying and dividing a binary number by a constant [37], for squaring [38], for converting a number from the binary system to the decimal [37], etc. Homogeneous array simultaneous multipliers [1, 4-16, 22-24] also contain elements of the same type connected in a regular way. Heterogeneous array simultaneous multipliers [1, 4, 13-21] lack this regularity.

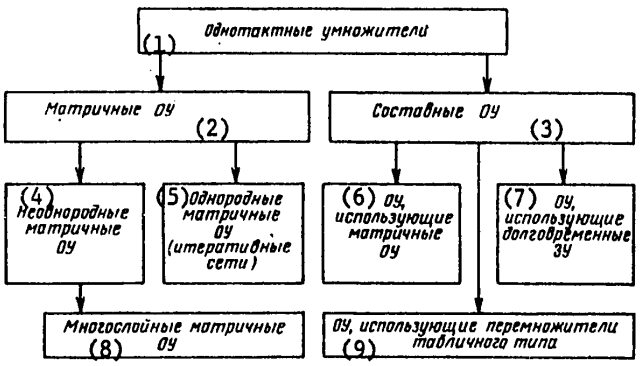


Fig. 4.4.1.

Key:

1. Simultaneous multipliers
2. Array simultaneous multipliers
3. Compound simultaneous multipliers
4. Heterogeneous array simultaneous multipliers
5. Homogeneous array simultaneous multipliers (iterative arrays)
6. Simultaneous multipliers using array simultaneous multipliers
7. Simultaneous multipliers using read-only memory
8. Multilayer array simultaneous multipliers
9. Simultaneous multipliers using tabular type multipliers

A large group of heterogeneous array simultaneous multipliers is made up of the so-called multilayer structures [1, 4, 13-21] that are distinguished by the short time needed to form the two-digit code of the product (see section 4.4.4).

Compound simultaneous multipliers are synthesized from simultaneous multipliers with smaller digit length: array [4, 27], tabular or are small read-only memory devices [25, 26] that calculate the products of the separate parts of the multiplicand and the multiplier. The interconnections between the "elementary" simultaneous multipliers in the compound simultaneous multiplier can be made by one or another method and, in particular, as the single-digit adders are connected in the homogeneous or multilayer simultaneous multipliers; the simultaneous multipliers thus obtained can be called macro-homogeneous, macro-multilayer, etc.

Simultaneous multipliers are now ever more often finding application in the arithmetic units of digital machines. In a number of cases, the same array unit is used not only for multiplication, but also division [22-24]. As large computing systems are developed in which the proportion of the arithmetic unit is not large compared to the rest of the apparatus, the application of simultaneous multipliers will probably increase. Also used in many cases are non-full arrays that simultaneously add several, but not all, partial products [27, 39, 40]. These devices were discussed earlier (section 4.1).

Simultaneous multipliers can be built, generally speaking, from various elementary cells. Simultaneous multipliers in which the main type cell is a single-digit binary adder with potential elements have become widespread. In what follows, we will have in mind the simultaneous multipliers containing such adders.

A large number of circuits for single-digit adders are now well known. In the subsequent points in this chapter, it will be shown that the speed of the entire simultaneous multiplier may essentially depend on the choice of circuit for the one-digit adder. Therefore, it is advisable to consider the possible structures of an adder from the viewpoint of their effect on the properties of the simultaneous multiplier.

In arithmetic units not using array circuits, signal delay time in carry circuits of one-digit adders has a far greater effect on device speed than delay time in adding circuits. In devices using array adders, the total time for the multiplication operation (and other operations in the performance of which the array takes part) essentially depends also on the speed of the adding circuits of the one-digit adders. For this work, it is convenient to introduce in the discussion the ratio of maximal delay time in an adding circuit to the maximal delay time in the carry circuit of the one-digit adder.

Some circuits of one-digit, coincidence-type adders are shown in fig. 4.4.2. One can see from an examination of them that the maximal delay in the carry circuit τ_E, generally speaking, is not equal to the maximal delay in the adding circuit τ_{add}. It can be assumed that in the general case, $\tau_{add} = k\,\tau_E$, where k is some constant. However, for the majority of practical applications, we can limit the examination to two basic cases: $k = 1$ or $k = 2$. In the first of these, we will call the adder "simultaneous," and in the second case--"two-cycle."

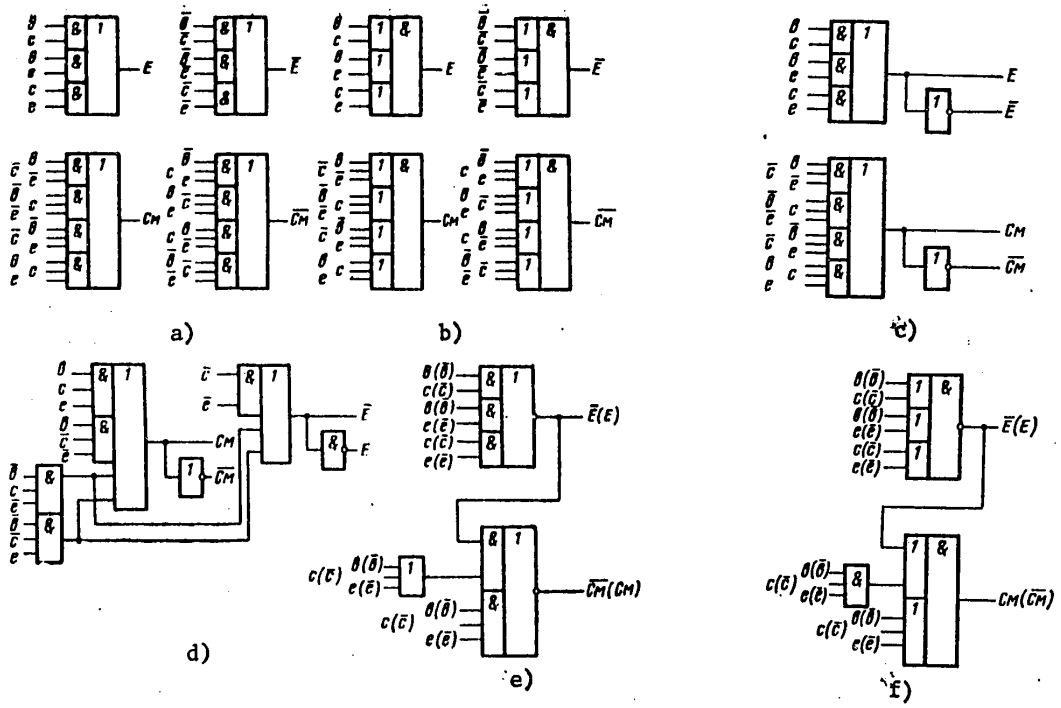


Fig. 4.4.2. One-digit, coincidence-type adder circuits

Key:
E = carry circuit Cm = Sm = add circuit

Let us call the τ_E delay a "time step" and assume that it is identical for all adders (in this case, the term "time step" [takt] has no relation to the phrase "simultaneous multipliers" [odnotaktnyye umnozhiteli]). The value k = 1, as a rule, pertains to the adders (fig. 4.4.2, a-d) in which the add and carry signals are generated independently of each other, although the input part of these two circuits may be common (fig. 4.4.2, d). Such adders require paraphase input signals, and consequently, if we want to use them in an array, then the outputs of these adders must also be paraphase. The case k = 2 pertains to adders in which the add signal is generated by using the carry signal (fig. 4.4.2, e and f). Such adders do not require paraphase input signals and are therefore, as a rule, more economical. The carry signal code in these circuits is inverse to the code of the addends, while in circuit 4.4.2, e, the add signal is also inverse in relation to the inputs. The circuit for a binary, one-digit adder has the so-called property of self-duality which means the inversion of all input signals leads to inversion of output signals. This property is reflected in fig. 4.4.2, e and f, by the superscripts in parentheses and will be used from here on.

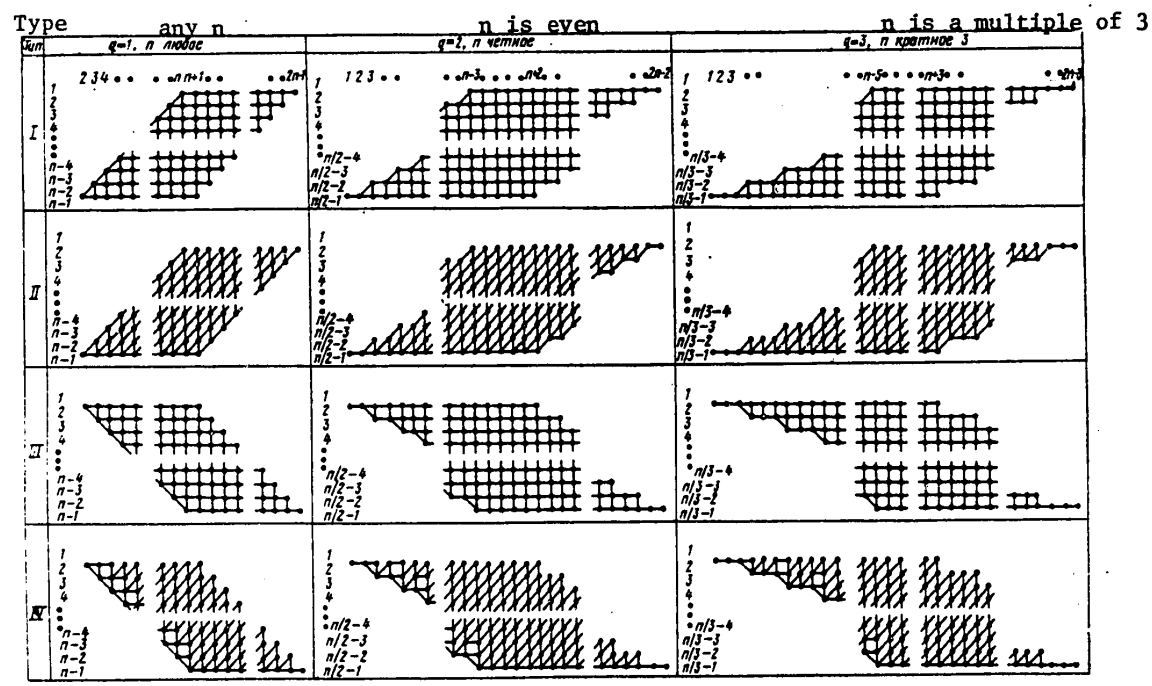


Fig. 4.4.3

The main merits of simultaneous multipliers are high speed, simplicty of control circuits and ease of adjustment and finding defects. The progress of microelectronics and, in particular, the evolution of LSI circuit technology has permitted countering the main shortcoming of these devices—large amount of apparatus. The experience of the evolution of Soviet and foreign computer technology in recent years shows that the use of simultaneous multipliers is becoming an important means for raising the throughput of computing resources.

4.4.2. Analysis of Processes of Adding Partial Products in Homogeneous Simultaneous Array Multipliers

For types of circuits for 12-array simultaneous multipliers are shown in simplified form in fig. 4.4.3. Each small dot in the figure represents a one-digit, binary adder; the add signals are directed from the top downwards; the carry signals are directed from right to left, from right upwards to left and from right downwards to the left. A common principle of structure has been kept in each of these circuits: the adders pertaining to one digit form a vertical chain and the connections between adjacent positions have a regular nature. Thus, the structures shown are iterative arrays.

Often combined to reduce the quantity of apparatus are the application of the method for speeding up multiplication that is being investigated and the method for

speeding it up, described earlier, that consists in decoding groups containing q digits of the multiplier each, and in using preliminarily computed multiples of the multiplicand. Therefore, in fig. 4.4.3 for each of the four types of arrays are shown three circuits corresponding to the versions of multiplication in which each partial product is obtained through multiplication of the multiplicand by q digits of the multiplier (q = 1, 2 or 3). In doing so, it is assumed that when q = 2, the multiples 0, 1C, 2C and 3C are used (i.e. there is an additional adder to calculate the triple multiplicand in the arithmetic unit), but when q = 3, the multiples 0, 1C, 2C, 3C, 4C, 5C, 6C and 7C are used (i.e. there are adders to compute the values of 3C, 5C and 7C). When q = 2 or q = 3, other multiples could be used too: For example, in multiplying by two digits, one could use the multiples 0, 1C, 2C, -1C and -2C, and in multiplying by three digits—the multiples 0, 1C, 2C, 3C, 4C, -1C, -2C, -3C and -4C. In the process, the number of additional adders is reduced, but there emerges the apparatus necessary for input of the negative partial products. But basically the structure of the array and its properties are preserved.

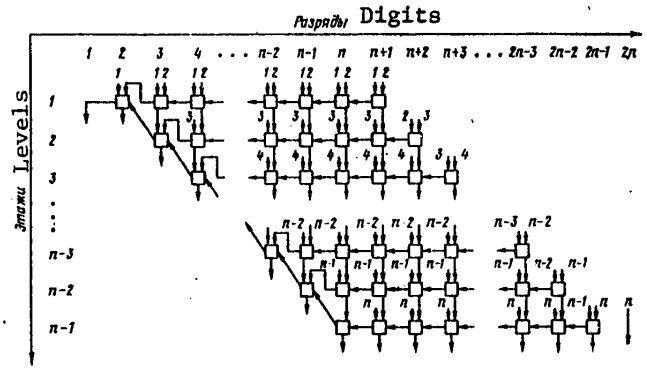


Fig. 4.4.4

The array multipliers in fig. 4.4.3 were presented, as noted, in a somewhat simplified form--not shown in particular are the circuits that form the partial products; also not shown are the inputs through which the partial products go in; both the adders and the half-adders are depicted in identical fashion, etc. The nature of the simplifications can be seen by comparing the type III circuit shown in the first column in fig. 4.4.3 with the same circuit shown in more detail in fig. 4.4.4. In the device shown in fig. 4.4.4, n partial products with n digits in each are added together. The numbers of the partial products are entered near the arrows indicating the position of input of the individual digits of the partial products. The one-digit adders and half-adders are shown as squares.

The circuits in the second and third columns in fig. 4.4.3 are shown for n, a multiple to two and three, respectively. If this condition is not met, then the number of levels of the adders turns out equal, respectively, to the values n/2-1 and n/3-1 rounded to the next larger integer. The structure of the device and its properties in the process are essentially unchanged.

The assembly for forming the partial products is not shown in either fig. 4.4.3 or fig. 4.4.4. In the simplest case (q = 1), this assembly contains n^2 two-input AND circuits that form n^2 signals $a_i c_j$ (i = 1, 2, ..., n; j = 1, 2, ..., n) where $a_i c_j$ are the digits of the cofactors. The signals generated (simultaneously) are binary numbers which are in the digits of the partial products (n products for n digits). Arranged under one another in accordance with the weights of the digits, these partial products form an array almost coincident in form with the arrays in fig. 4.4.3 in column q = 1. Now each dot represents one digit of one partial product. (One can say that with that in the figure there is not shown only the low-order partial product in the digit positions from (n + 1) through 2n inclusive.) For the simultaneous multiplier to "start working," there remains only the superimposition of the array of partial products on the array of adders, i.e. the connection of the outputs of the AND circuits with the corresponding free inputs of the adders. In the process, it is evident that the digits in one column of the array of numbers can be fed to the inputs of the corresponding column of adders in any order, i.e the digits in the column can change positions.

If q is greater than 1, the number of partial products is reduced q-fold (when n is a multiple of q), and each of them is lengthened from n to n + q digits. When q is equal to 2 or 3, the simultaneous multiplier assumes the form shown in the corresponding position in fig. 4.4.3. An example of a complete type II simultaneous multiplying circuit (it can also be considered a type IV circuit) for the case n = 6 and q = 2 is shown in fig. 4.4.5, in which besides the array of adders itself, also shown are the additional adder to generate the triple multiplicand, and the circuits for forming the partial products. The blank rectangles in this figure represent one-digit adders and half-adders. A type I device (q = 2) is described in work [1], page 451. There too are complete simultaneous multiplying circuits of type I, q = 1 (page 438); type III, q = 1 (page 445); type IV, q = 1 (page 448) and others.

It can be seen from figs. 4.4.3 - 4.4.5 that a final 2n-position product is formed at the outputs of the simultaneous multiplier.

Not shown in these figures are the additional inverters needed to "phase" the signals when using the adders shown in fig. 4.4.2, e and f, since in arrays of this type there may be encountered not only normal, but also inverted signals.

Let us discuss the conditions under which such a necessity occurs. For this, let us note first of all that any of the structures shown in fig. 4.4.3 may be divided into parts, each of which consists of three or four one-digit adders connected to each other "in a ring" (fig. 4.4.6). Let us first discuss a ring of four adders. It is easy to be persuaded that, as shown in fig. 4.4.6, a and b, the two upper adders of this ring must be built with the same circuit conforming to either fig. 4.4.2, e, or fig. 4.4.2, f (for simplicity, the adders in fig. 4.4.6 have been designated with the letters e or f), and each of the two lower adders may be built with either of these two circuits. Thus, such a ring may be built with adders of either type e, or type f, or with both.

However, a ring of three adders may be built without additional inverters only if type f (see fig. 4.4.6, c) circuits are available to the developer. But if only type e adders are on hand, an additional inverter (fig. 4.4.6, d) must be installed

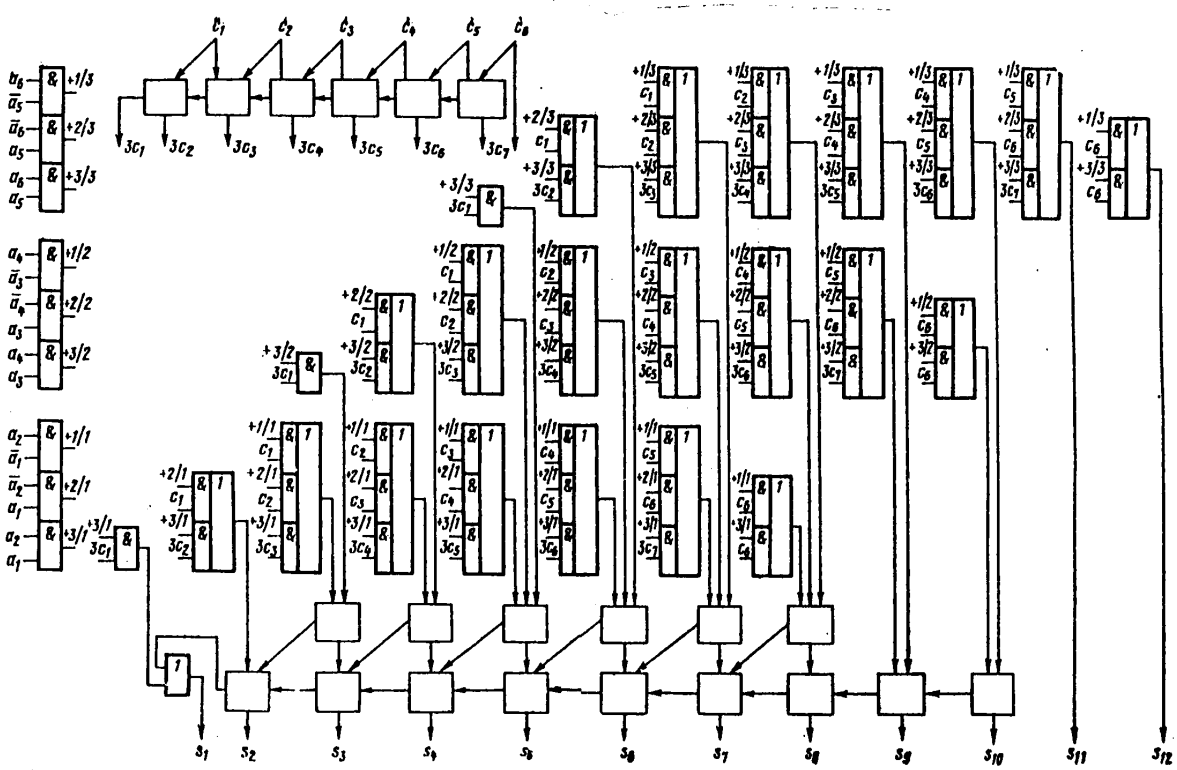


Fig. 4.4.5. Example of complete type II simultaneous multiplying circuit for the case n = 6 and q = 2.

in the ring for "phasing." In the figure, this inverter is included in the add circuit, but obviously instead of this, it can be included in either of the two carry circuits forming a "triangle" together with the indicated add signal.

It should be noted that the consideration relative to the impossibility of building an array with type e adders without using additional inverters pertains not only to the structures shown in fig. 4.4.3, but also to any array in the circuit of which there can be singled out at least one closed circuit, formed by one-digit adders and the add and carry signals that connect them, in which the number of adders would be odd. It is also evident that such circuits exist practically in any array. However, their number is different in different types of arrays. It can be deduced from fig. 4.4.3 that type I and III arrays contain the minimal number of such circuits ("triangles"). For the case of q = 2, this number equals (n - 4)/2, and for arrays II and IV—respectively, (3n-6)/2 and (5n-20)/2; when q = 3, arrays I and III contain (n-6)/3 "triangles" each compared to (5n-18)/7 and (7n-36)/3 for arrays II and IV. When q = 1, arrays I, II, III and IV contain, respectively, n - 2, n - 1, n - 2 and 3n - 8 "triangles."

It is evident that the additive inverters must be installed in such a way that the "phasing" of the signals in the adjacent circuits is not disrupted and wherever possible, the speed of the array is not reduced. Direct analysis shows that the first of these two requirements can easily be met for all the circuits shown in fig. 4.4.3 by having the number of additive inverters equal the number of "triangles." However, it will be shown later that to meet the second requirement, the number of additive inverters may have to be increased.

Let us move on to an analysis of the processes for determining the solution in homogeneous array simultaneous multipliers [9].

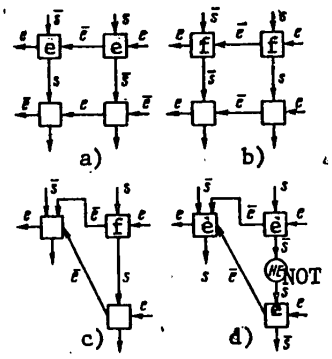


Fig. 4.4.6

It was mentioned earlier that summing the partial products in a simultaneous multiplier in the general case is a complex transient process. During the flow of this process, the signal at some point in the device may be switched several times from one level to another before the final, correct value is obtained. The presence of intersecting flows of signals limits the possibility for additional speeding up. In some arithmetic units, we also have to reckon with the fact that besides the loss of time on the transient process that occurs after the feeding of the partial products, there are also losses of time on decay of the processes in the array after all input signals are removed. The decay time is less than the addition time, but it also can be, as will be shown in section 4.4.3, rather long.

The time for performing multiplication in an array multiplier depends on the array circuit, the parameters of the elements used and on the values of the cofactors. There are at present no simple methods for precisely estimating the maximal operating time in the various simultaneous multipliers. Therefore, we will estimate the speed of arrays "from above," assuming the transient process flows the "worst" way. The speeds of the arrays shown in fig. 4.4.3 may be compared by a technique based on the following two assumptions.

1) Let us assume that all the partial products are fed into the array simultaneously at a moment in time t = 0. (In reality, this condition is not met due to scattering of the parameters of the elements of the preceding assemblies, delays of signals in the conductors, etc. Also, for arrays using additional multiples of the multiplicand that are generated by additional adders, this condition cannot be met since the very generation of multiples takes some time and the low-order digits of the multiples emerge before the high-order ones. In an actual device, it is advisable to make these additional adders fast, but only to the extent that the full time of operation of the arrays depends only on its structure, but not on the additional adders.)

2) Let us also assume that from each one-digit "two-clock-period" (simultaneous) adder, the correct carry signal is emitted in a "clock period," and the sum signal within two "clock periods" (within one "clock period") after all three terms are fed to the inputs of the adder.

By this technique, one can build 24 timing diagrams corresponding to 12 structures of arrays shown in fig. 4.4.3 and to 2 values of magnitude k (k = 1 or k = 2). General expressions for the coordinates of the characteristic points of these diagrams were derived on the basis of the diagrams.

Five of these diagrams are shown in figs. 4.4.7 - 4.4.11. The dots indicate the moments of establishment of correct signals at outputs of sums of all adders of the arrays. For clarity, diagrams corresponding to specific values of n are shown; however, the exact formulas given in the figures for the coordinates of the characteristic points of the diagrams correspond to any value of n (however, with the restriction that n is considered a multiple of q).

The diagrams obtained correspond to the hypothetical "worst" transient process, from the viewpoint of speed, and make it possible to estimate "from the top" the actual processes occurring in homogeneous array simultaneous multipliers.

Thus, it can be seen from fig. 4.4.10 that

1) the carry is propagated "slowly" along each level until the (n + 1)-th bit (the passage of the carries is delayed by the sum signals coming in from above), after which the passage continues "quickly";

2) the full time of operation of this array is 3n - 3 "clock periods";

3) after 2n - 3 "clock periods," the transient process decays in the entire circuit, with the exception of the adders located at the 2 ... n digits along the lower edge. This means that having built the circuit for fast passage of carries along the indicated adders, one can reduce the full time of operation of the device from the magnitude 3n - 3 to the magnitude of the order 2n - 2 "clock periods." Building fast circuits in other places of the array to further reduce the full time would require hardware investments

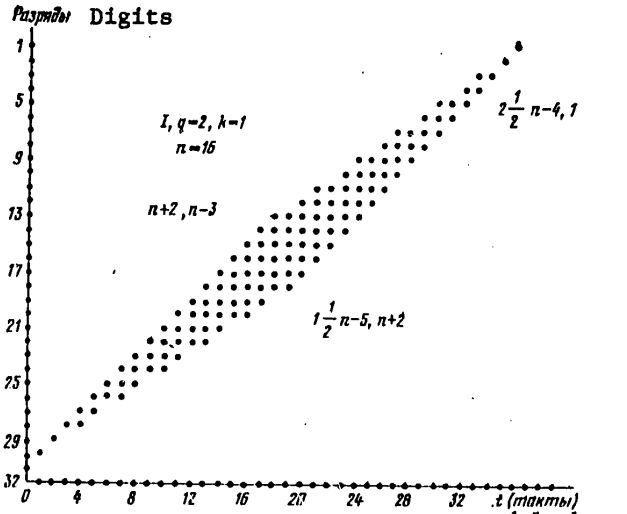


Fig. 4.4.7

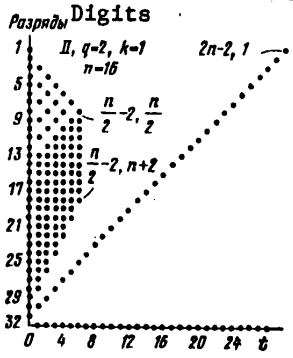


Fig. 4.4.8

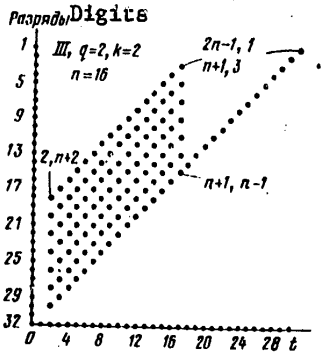


Fig. 4.4.9

that are too great. It can be seen from the figure that t = 2n - 3 is the time for forming the two-row code of the product, i.e. the time for forming the two numbers whose sum is the product sought.

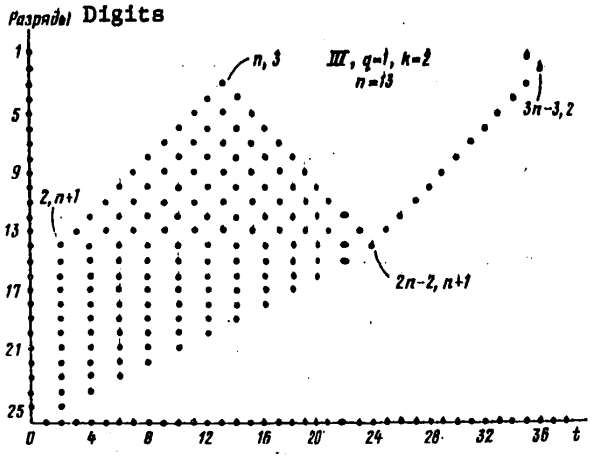


Fig. 4.4.10

All 24 timing diagrams are consolidated in the table shown in fig. 4.4.12. Some simplifications were made to pick out the main features of the diagrams: sections of the diagrams that correspond to the process of establishing the signals in the main mass of th adders are shown in the form of circuits; the processes of carry passage along the lower edge of the array are represented by a separate line. On the coordinate axes are shown (with a precision to several "clock periods" and bits) the coordinates (time and number of bit) of the characteristic points of the circuit (it is assumed the coordinates of the characteristic points are indicated for $n \to \infty$) and the full array operating time T is indicated precisely.

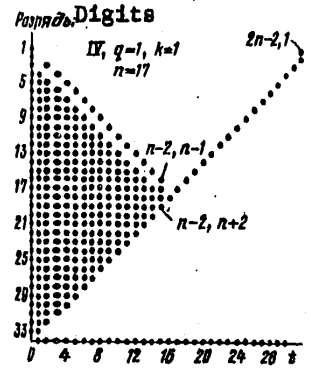


Fig. 4.4.11

Examination of these timing diagrams allows making the following conclusions on the properties and features of the homogeneous array simultaneous multipliers in question.

1) Changing the array structure type, the type of one-digit adder or the parameter q may considerably change the timing diagram.

2) In a number of cases, passage of the carries can be speeded up along the entire lower edge of the array (for example, in circuits II and IV when q = 3), and in other cases, this can be done only in the high-order n digits (in arrays II, III and IV when q = 1); and finally, it is practically impossible to organize any speeding up for structure I.

3) Under otherwise equal conditions, type I arrays are slower than any others. It is practically impossible to speed up type I arrays since passages at all levels would have to be speeded up for this.

These diagrams allow selecting the optimal variant of a homogeneous array multiplier and the circuit for a one-digit adder. However, in the process a number of other circumstances must also be considered; these can not be analyzed in general form because of the large number of possible situations. The basic circumstances are the following:

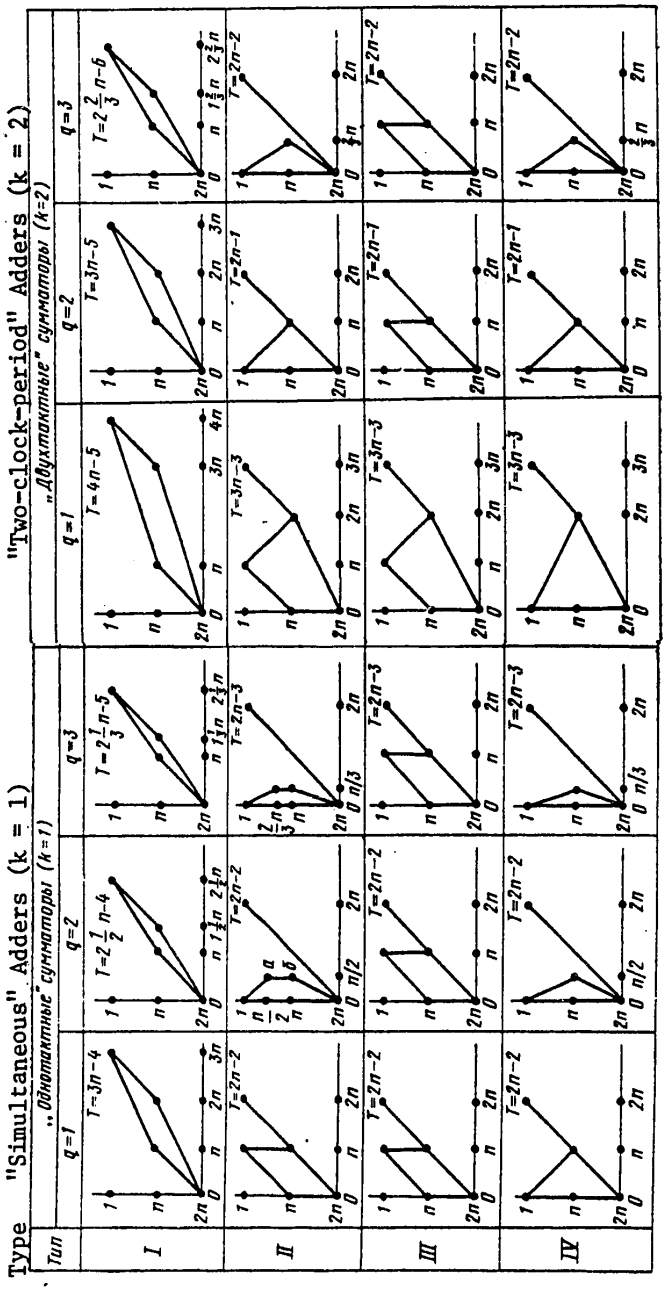


Fig. 4.4.12

Fig. 4.4.12

1) System of elements used. The type, speed, reliability, load capacity, dimensions, cost of various elements available to the designer, and the very fact of availability or lack of a particular element may affect the choice of an array circuit.

2) Standardization of building blocks, e.g. LSI circuits into which the array and the entire arithmetic unit are "subdivided."

3) The necessity in certain cases of incorporating additional inverters for "phasing" of signals into the array circuit.

4) Use of the array during other operations. This may be especially important. If, for example, nonrestoring division has to be performed in the array, type I and II arrays cannot be used since they do not provide the capability of shifting the next remainder. A type III or IV array can be used to implement this method of division. The first of these is best suited when the next remainder has to be obtained in a one-row code, and the second--if the remainder is formed in a two-row code. If division is to be implemented by using one of the methods using multiplication (section 5.3), then generally speaking, any type of array is suitable.

These timing diagrams of homogeneous array simultaneous multipliers make it possible to select positions for installing additional inverters (needed for "phasing" signals when type e one-bit adders are used) so that the speed of the array is not changed. A comparison of the circuits and diagrams allows concluding that in all circuits except three (II, q = 2; III, q = 2 or 3), one can install inverters in a way such that their quantity equals the quantity of "triangular" circuits and in the process, the delays introduced by the inverters are not reflected in the value of T. In the three cases mentioned, the additional inverters should be installed as shown in simplified form by the X's in fig. 4.4.13. In doing so, the quantity of inverters must exceed the number of "triangles" (see table 4.4.1) so that the inverters are not located in "critical" paths of signal propagation.

It can be seen from table 4.4.1 and the timing diagrams in particular that when q = 2 and type e adders are used, array III, with the same speed as arrays II and IV, and better speed than circuit I, contains considerably fewer additional inverters. Type III arrays were first described, apparently, in work [10].

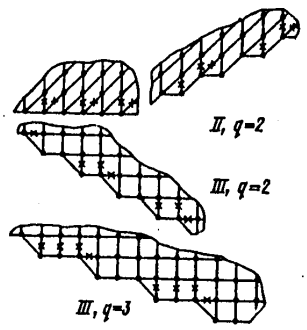


Fig. 4.4.13

4.4.3. Analysis of "Decay" Processes in Homogeneous Array Simultaneous Multipliers

If another--simpler and quicker--operation (addition, for example) is performed in an array after multiplication, and if in the process the pause between the end of the multiplication and the beginning of the next operation is less than the signal "decay" time in the array, which occurs after setting all input signals in the array to the zero state, then the decay time must be taken into account when calculating the time for performance of the next operation. When necessary, hardware means for protection form this effect have to be used. For example, using a special signal, one can "disconnect" the outputs of the upper levels of the array from the lower level, in which an addition-subtraction operation can be performed.

Table 4.4.1.

Type of Array	Number of "Triangular" Circuits	Number of Additional Inverters
I, $q=1$	$n-2$	$n-2$
I, $q=2$	$\frac{n-4}{2}$	$\frac{n-4}{2}$
I, $q=3$	$\frac{n-6}{3}$	$\frac{n-6}{3}$
II, $q=1$	$n-1$	$n-1$
II, $q=2$	$\frac{3n-6}{2}$	$\left[\frac{7n-18}{4}\right]$
II, $q=3$	$\frac{5n-18}{7}$	$\frac{5n-18}{7}$
III, $q=1$	$n-2$	$n-2$
III, $q=2$	$\frac{n-4}{2}$	$\left[\frac{3n-12}{4}\right]$
III, $q=3$	$\frac{n-6}{3}$	$\frac{2n-12}{3}-2\left(\frac{n}{6}-\left[\frac{n}{6}\right]\right)$
IV, $q=1$	$3n-8$	$3n-8$
IV, $q=2$	$\frac{5n-20}{2}$	$\frac{5n-20}{2}$
IV, $q=3$	$\frac{7n-36}{3}$	$\frac{7n-36}{3}$

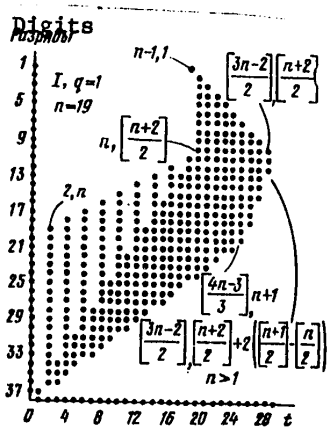


Fig. 4.4.14

Therefore, the study of the decay processes in array simultaneous multipliers, just as the of the processes of establishing the solution, has definite value. The problem of analysis of the processes of decay, apparently, was stated for the first time in work [41].

Decay processes may be examined by a technique based on the following assumptions.

1) All partial products become equal to zero simultaneously at the moment t = 0.

2) From each one-digit binary adder, the correct carry signal (zero) is emitted within the "clock period" after two correct (i.e. zero) signals of the terms are fed to the input, and the sum signal (also zero) is generated with the "clock period" after all three (zero) terms are fed.

These rules are common to "simultaneous" and "two-clock-period" adders. The validity of this statement follows from an examination of the circuits shown in fig. 4.4.2.

Following this technique, timing diagrams were built for all the arrays shown in fig. 4.4.3, and precise formulas for the coordinates of the characteristic points of these diagrams were derived [41]. One of the 12 diagrams is shown in fig. 4.4.14. It corresponds to the specific value of n = 19, but the formulas are valid for any n greater than 1.

These diagrams are consolidated in simplified form in fig. 4.4.15 in a general table, as was done earlier for the processes of establishing the solution. Indicated in the figure are the precise values of the full decay time T_z and the approximated values of the coordinates of the characteristic points (as above, the precise formulas are replaced by approximated only for clarity).

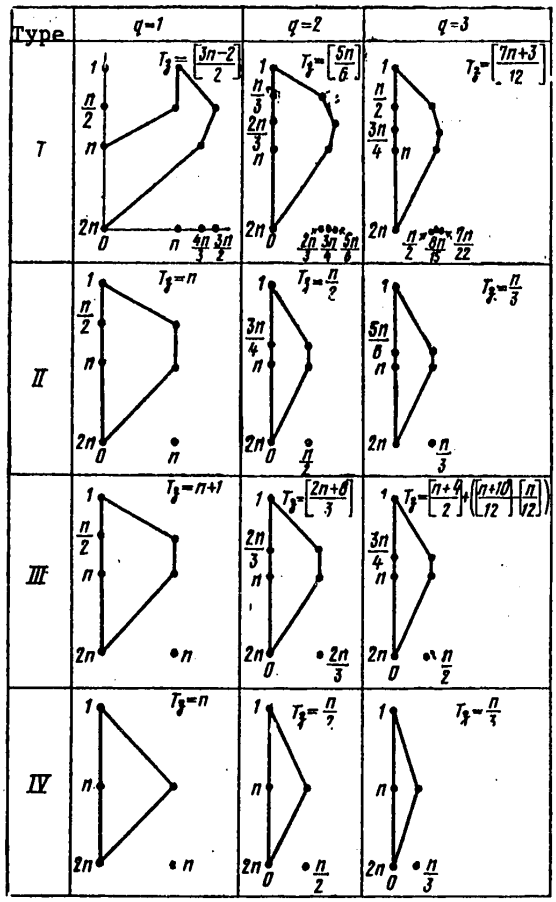


Fig. 4.4.15

It can be seen from an examination of the timing diagrams that decay occurs slower in type I arrays than in the other three types. When q = 1, arrays II–IV have practically identical rate of decay, but when q = 2 or 3, decay occurs somewhat faster in arrays II and IV than in the type III array. It is also evident that the decay process cannot practically be speeded up.

The diagrams of decay, just as the diagrams for establishing the solution, can be used to select a type of homogeneous array and a one-digit adder circuit. They make it possible to select the necessary measures of protection in those cases when presence of decay leads to reduction in speed of the arithmetic unit.

It should be noted that the time of one "clock period," adopted in this chapter as the unit of measurement of time, although it was defined as the maximal time of delay of the corresponding element, in reality can be equated to the value, only somewhat exceeding the average time of operation of this element. This is explained by the fact that in the long chain of serially connected elements, the probability of large deviation of total delay from the value equal to the sum of the average delays is very insignificant. This feature of "single-pass" (simultaneous, noncyclic) arithmetic and other devices in general and of adder arrays in particular provides for, in essence, additional speeding up of their operation and raising of reliability [1, pp 442-443].

4.4.4. Multilayer Simultaneous Array Multipliers

1°. Let us call multilayer that array simultaneous multiplier, built with one-digit binary adders and half-adders, in which there is at least one one-digit adder or half-adder, into the inputs of which are fed the output signals of the sum of two or three other one-digit adders. In other words, in a multilayer array simultaneous multiplier, the one-digit adders pertaining to a given position of a device, are not

stretched out in a chain, as is done, in particular, in the homogeneous structures discussed in sections 4.4.1-4.4.3, but form a certain ramified "tree."

One could suggest quite a few structures of simultaneous multipliers that fit this definition. We shall discuss just some of the most well known structures.

Let us analyze some general properties possessed by the majority of multilayer structures discussed in this book [46].

Let us assume that a simultaneous multiplier consists of a series of serially connected layers: the output signals of an i-th layer are the input signals to an (i + 1)-th layer (i = 1, 2, ...). The n_{i-1} numbers go from the preceding, (i - 1)-th layer to the inputs of each i-th layer. The apparatus of the i-th layer processes (adds) by certain rules common to each layer these numbers and generates n_i other numbers that go to the next (i + 1)-th layer. The sum of the numbers present at the inputs or outputs of any layer equals the same value--the product of the initial cofactors A and C.

We shall see subsequently that in many multilayer simultaneous multipliers known and discussed below, the function $n_i = f(n_{i-1})$ satisfies the following five conditions.

Condition 1. For any positive integer n_{i-1} there is a corresponding unique positive integer n_i, i.e. the function $n_i = f(n_{i-1})$ is single-valued.

Condition 2.

$$n_i < n_{i-1} \text{ when } n_{i-1} > 2 \quad (i = 1, 2, \ldots)$$

(i.e. when more than two numbers enter the inputs of any layer, fewer numbers than those that entered the inputs result at the outputs of this layer).

Condition 3.

$$0 \leqslant f(n_{i-1}+1) - f(n_{i-1}) \quad (n_{i-1} = 1, 2, \ldots)$$

(i.e. the quantity of n_i numbers at the outputs of a layer cannot decline when the quantity of n_{i-1} numbers increases at the inputs of the layer).*

Condition 4.

$$f(n_{i-1}+1) - f(n_{i-1}) \leqslant 1 \quad (n_{i-1} = 1, 2, \ldots).$$

Condition 5. For any number N, no matter how large, there exists such n'_{i-1}, that $n_i > N$ when all $n_{i-1} > n'_{i-1}$. In other words, $\lim_{n_{i-1} \to \infty} n_i = \infty$.

* It is assumed that a change in the quantity of numbers at the inputs of a layer is accompanied by a corresponding rearrangement of the apparatus in the layer.

We shall see later that these conditions are met not only in multilayer simultaneous multipliers built from one-digit adders, but also in simultaneous multipliers built from parallel counters * (see subsections 4°, 5°, 9°, 10°, 11° and 12° of this section), short-bit parallel adders (section 6.3.1) or from special translators of N-bit code into two-bit code (section 6.3.2).

Let us examine several general corollaries stemming from the fulfillment of the above five conditions and thus relevant to all the multilayer simultaneous multipliers just listed (as well as to all the simultaneous multipliers that can still be suggested and developed and in which the conditions listed are also met).

Corollary 1. It follows from conditions 1 and 2 that the quantity of n_1, n_2, ... numbers at the outputs of the sequential layers form a series whose terms descend until the first of them satisfies the inequality

$$n_i \leqslant 2.$$

Let us designate the number of this term by the letter M. Let us assume that the simultaneous multiplier contains M layers.** Thus, the initial $n = n_0$ numbers are sequentially translated ("removed") M times so that there are satisfied the inequalities

$$n_0 > n_1 > n_2 > \ldots > n_{M-1} > n_M;$$
$$n_i > 2 \text{ when } i < M;\ n_M \leqslant 2.$$

Corollary 2. It folows from condition 1 and corollary 1 that for any n, there is a corresponding unique M, i.e. that the function M(n) is single-valued.

Corollary 3. The function M(n) is non-decreasing, i.e. $M(n') \leqslant M(n'')$, if $n'' > n'$. This follows from corollary 1 and condition 3. In fact, when n increases, not one of the values n_1, n_2, ... can decrease and, consequently, M cannot decrease.

Corollary 4. When n increases by one, the quantity of layers either does not change or is increased by one, i.e. $M(n+1) - M(n) \leqslant 1$.
This follows from corollary 1 and conditions 3 and 4.

Corollary 5. The number of layers M can be as large as desired: $\lim_{n \to \infty} M = \infty$. (from corollary 1 and condition 5).

---

* We call a parallel counter (k, m) an assembly that adds k binary digits of the same weight (let us assume the weight equals 1) and generates their sum in the form of an m-bit binary number. Since the maximal value of this number is $2^m - 1$, then m is the smallest integer solution to the inequality $2^m - 1 \geqslant k$.

A particular case of a parallel counter is the counter (3, 2) which is a well known one-digit binary adder.

** We assume that n_0 is greater than 2.

Corollary 6. For any integer non-negative M, there is some maximal value n which we shall designate in the form $n^{(M)}$. In other words, $n^{(M)}$ is the maximal quantity of numbers that can be contracted into two-digit code by using M layers. Corollary 6 follows from corollaries 4 and 5.* Let us note that $n^{(0)}=2$.

Corollary 7. From corollaries 3 and 6, it follows that $n^{(T-1)}<n^{(T)}$ $(T=1, 2, \ldots)$.

Thus, the numbers $n^{(0)}$, $n^{(1)}$, $n^{(2)}$, ... form an ascending sequence.

Corollary 8. It follows from corollaries 3, 6 and 7 that when $n\geqslant 3$ $M(n)=T$, if $n^{(T-1)}<n\leqslant n^{(T)}$ $(T=1, 2, \ldots)$. This means that the quantity of layers of simultaneous multipliers when any $n\geqslant 3$ can be determined, after selecting from the series of numbers $n^{(0)}$, $n^{(1)}$, ... the two adjacent numbers $n^{(T-1)}$ and $n^{(T)}$ that satisfy the given inequality. Then M(n)=T. We will often use this technique later. If there is known the time tau_i of operation of any i-th layer and if $tau_1=tau_2=\ldots=tau_M=$ tau, then the value of M defines the time of operation of the entire device from the moment the $n=n_0$ initial numbers (partial products) are fed to the inputs of the first layer to the result of the two-digit code of the product; this time equals M tau.

Corollary 9. If $n_{i-1}=n^{(T)}$, then $n_i=n^{(T-1)}$. In fact, since in this case n_i numbers are contracted T-1 layers, then owing to corollary 8

$$n^{(T-2)}<n_i\leqslant n^{(T-1)}.$$

If it would turn out that $n_i<n^{(T-1)}$, then when n_{i-1} increases by one, n_i would increase by no more than one (condition 4), and would become equal to n'_i, the inequality

$$n^{(T-2)}<n'_i\leqslant n^{(T-1)}$$

would be met and therefore n'_i numbers would be contracted T-1 layers. But this would mean that $n^{(T)}+1$ numbers were contracted by T layers which is impossible. Therefore, $n_i=n^{(T-1)}$.

Corollary 9, in particular, means that if $n_i=n^{(T-1)}$, then $\max(n_{i-1})=n^{(T)}$. We will use this possibility for computing values of $n^{(T)}$.

---

* Fulfillment of condition 5 is necessary since if it were not met, i.e. if there were such X and such $n_{i-1}=\tilde{n}$, that $n_i=X$ when all $n_{i-1}\geqslant\tilde{n}$, then when all $n_0\geqslant\tilde{n}$ n_i would equal X, and M would equal some constant maximal value $\tilde{M}$ (i.e. corollary 5 would not exist). In the process, $n^{(\tilde{M})}$ would equal infinity, and $n^{(M)}$ when $M>\tilde{M}$ would not exist (i.e. corollary 6 also would not exist).

Using corollary 8, one can write the following system of inequalities:

$$n^{(M-1)} < n_0 \leqslant n^{(M)}, \quad n^{(M-2)} < n_1 \leqslant n^{(M-1)}, \ldots,$$

$$n^{(M-i-1)} < n_i \leqslant n^{(M-i)}, \quad \ldots n^{(0)} < n_{M-1} \leqslant n^{(1)},$$

$$n_M = n^{(0)} = 2.$$

It is evident that if we construct some i-th layer not in accordance with the function $n_i=f(n_{i-1})$, but some other method, but so that in the process the inequality $n^{(M-i-1)}<n_i\leqslant n^{(M-i)}$ is not violated, then the total number of layers M does not change. We will use this possibility later to modify the methods under investigation for constructing multilayer simultaneous multipliers.

2°. The first of the multilayer simultaneous multipliers that we shall discuss is organized the following way [1, p 454].

Let us subdivide a set consisting of n initial terms (partial products) into non-intersecting groups of two numbers each per group. The number of such groups (let us call them full) is equal to [n/2]; in addition, with an odd n, there will be one more group not full that contains one number. In each full group, let us add the two numbers by using the usual parallel binary adder with serial propagation of the carries. Let us not translate the number that has fallen into the group not full. Needed for translation, evidently, are $lambda_1=[n/2]$ parallel adders, the aggregate of which let us call the first layer of the device. Let us consider as outputs of the layer the [n/2] numbers obtained at the outputs of the adders and the number in the unfull group that, thus, passes through the layer without translation. The quantity of nu_1 numbers that end up in the unfull group, evidently, is equal to $n-2[n/2]$. It is evident that nu_1 equals 0 or 1. One can state that in the first layer $n=n_0$ numbers by using $lambda_1$ parallel adders is translated into

$$n_1 = \lambda_1 + \nu_1 = \left[\frac{n_0}{2}\right] + n_0 - 2\left[\frac{n_0}{2}\right] = n_0 - \left[\frac{n_0}{2}\right]$$

numbers. Each i-th layer is constructed similarly, i.e.

$$n_i = \lambda_i + \nu_i = n_{i-1} - \left[\frac{n_{i-1}}{2}\right]. \qquad (4.4.1)$$

Here $lambda_i$ is the number of parallel adders in the i-th layer.

A graph of the function $n_i=f(n_{i-1})$ is shown in fig. 4.4.16a (p 203). The continuous line shows the part of the graph that is later periodically repeated (broken line). It can be seen from (4.4.]) and the graph that with this method of constructing a multilayer simultaneous multiplier, all the conditions listed above (conditions 1-5) are met and therefore all nine corollaries are valid. The structure of such a simultaneous multilayer multiplier is described by the relationships

$$n_i = \lambda_i + \nu_i, \ \lambda_i = \left[\frac{n_{i-1}}{2}\right],$$

$$\nu_i = n_{i-1} - 2\lambda_i, \ n_M = 2, \ i = 1, 2, \ldots, M. \quad (4.4.2)$$

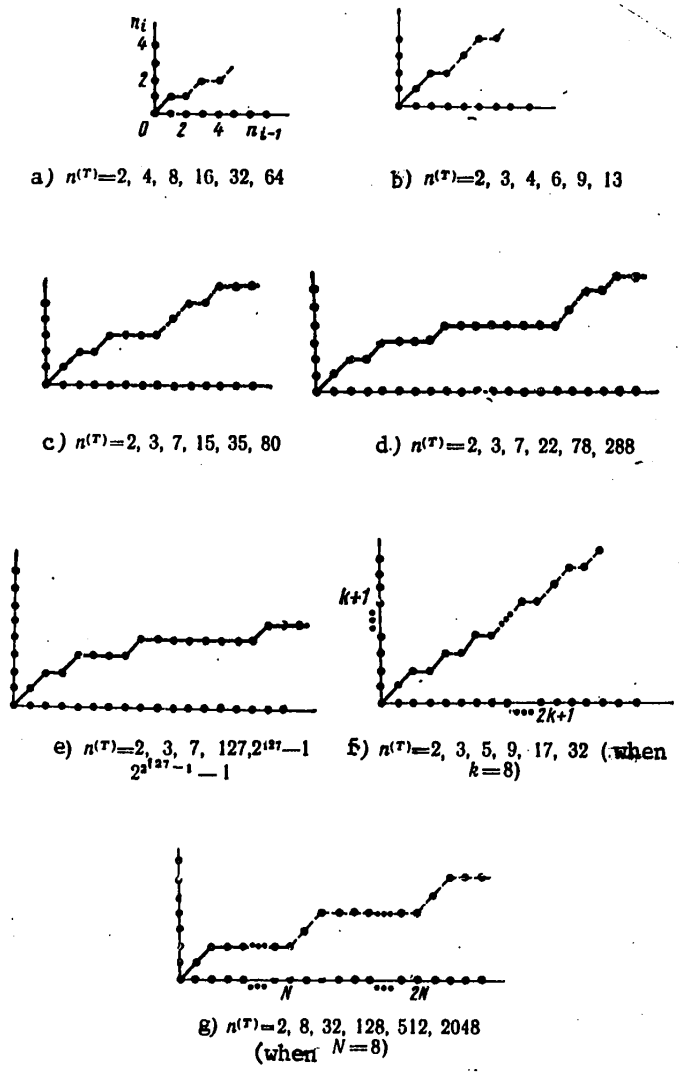


Fig. 4.4.16. Type assembly: a) and b)—counters (3, 2); c)—counter (7,3); d)—counter (15, 4); e)—counter (k, m); f)—k-bit adder; g)—translator N→2. T = 0, 1, 2, 3, 4, 5.

The sum of the two numbers obtained at the outputs of the last, M-th layer, as stated earlier, equals the product of AC.

It can also be seen from (4.4.1) that with the given n_i the maximal value of n_{i-1} is $2n_i$. Therefore, from corollary 9

$$n^{(T)}=2n^{(T-1)} \quad (T=1, 2, \ldots).$$

Since $n^{(0)}=2$, then $n^{(1)}=4$, $n^{(2)}=8$, ..., i.e.

$$n^{(M)}=2^{M+1} \quad (M=0, 1, \ldots). \qquad (4.4.3)$$

The first six numbers of $n^{(T)}$ are given in fig. 4.4.16, a.

Since for any n there is such T, that

$$n^{(T-1)}=2^T<n\leqslant 2^{T+1}=n^{(T)},$$

then from corollary 8, for this n there is an M = T.
In other words,

$$M=]\log_2 n[-1^{*}. \qquad (4.4.4)$$

*]d[designates the smallest integer not less than d.

For example, when n = 48, M = 5 (since $n^{(4)}=32<n\leqslant 64=n^{(5)}$).

The relationships (4.4.3) and (4.4.4) fully define the direct dependency between the number of terms n and the number of layers M, and the inverse dependency between the number of layers M and the maximal number of terms $n^{(M)}$.

Since each parallel adder reduces the quantity of numbers being added by one, then the full number of these adders equals

$$\sum_{i=1}^{M} \lambda_i = n-2.$$

Shown in fig. 4.4.17 is the location of the numbers at the inputs and outputs of the layers of the simultaneous multiplier of the type under consideration for the case n = 12; also shown in the drawing is the procedure for completing addition of the two numbers, after which the final product is obtained. Each dot in the figure represents a binary digit. The initial array A of the numbers is translated by the first layer into the array B; array C is obtained after the second layer, then D and, finally, the result E. Each rectangle in the figure encloses the numbers being entered to the inputs of one parallel adder. The horizontal lines

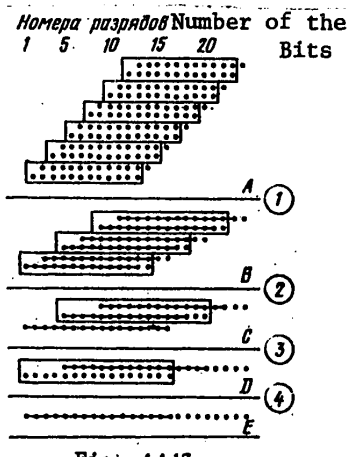


Fig. 4.4.17

connect the output digits of each parallel adder. Let us assume that the last adder that translates array D makes up an additional, (M + 1)-th layer of the device. It can be seen from the figure that there are 11 parallel adders and 146 one-digit adders. It can also be seen that the maximally full operating time of this device is

$$(M+1)\tau_{CM}+(2n-2)\tau_E=]\log_2 n[\tau_{CM}+(2n-2)\tau_E,$$

where tau_{sum} and tau_E are respectively the time for forming the sum digit and the carry digit in a one-digit adder. In fact, the second layer begins operating within $tau_E + tau_{sum}$ after the start of operation of the first layer (let us ignore the time for forming the partial products), the third layer--within 2 tau_E+ tau_{sum}, and the fourth--within 4 tau_E + tau_{sum}. By this time, the carries will actually pass through 1 + 2 + 4 low-order positions of the device (from the 2n-1-th to the 2n-8-th). In the last layer, the carries pass through the positions left before the first position. In addition to the delays associated with generation of the carries, there is also the delay tau_{sum} in each layer.

It can be seen from fig. 4.4.17 that this simultaneous multiplier (and the figure) was constructed actually according to the following rules.

1) The initial array of terms (partial products) is shown in the form of a parallelogram. The rows of digits obtained at the outputs of each layer are shown in the figure one under the other in the same order as the rectangles reflecting the corresponding rows of apparatus (in this case--parallel adders) at the inputs of the layer.

2) The quantity of n_i numbers at the outputs of each i-th layer is defined by the function $n_i=f(n_{i-1})$; however, the quantity of digits at the outputs of one position is n_i only in some middle positions of the layer; in the extreme positions due to the restricted length of the terms, the number of digits at the outputs is less than n_i.

3) One can state that the apparatus (adders) were installed at first as if the terms were infinitely long numbers, and then taking into account the actual width of the terms, all "superfluous" assemblies (one-digit adders) were removed from the circuit, i.e. those parts of the apparatus that had no effect at all on the position of the significant digits at the outputs of the layers.

One can note that these rules clearly define the structure shown in fig. 4.4.17. Let us call this structure "basic." Later we will often encounter various "basic" structures of simultaneous multipliers that differ from each other only by the fact that some other type assemblies will be installed in the layers instead of binary adders.

Let us also note that after changing the formulated rules, other structures can be constructed too (with the same function of $n_i=f(n_{i-1})$). For example, if the first of the three rules is removed, one can obtain a set of structures, one of which is shown in fig. 4.4.18. In the process, the number of layers M does not change.

3°. In accordance with the remark made at the end of subsection 1°, one can modify somewhat the design of the device just discussed, after preserving the same number of layers M=]log n_0[-1. Let us try, for example, to reduce to the maximum the number of parallel adders in the first layer. Since there must be met the inequality

$$n^{(M-2)} < n_1 \leqslant n^{(M-1)}, \qquad (4.4.5)$$

the number of adders lambda$_1$ = [n_0/2] can be reduced to the value

$$\lambda'_1 = n_0 - n^{(M-1)} = n_0 - 2^M = n_0 - 2^{[\log_2 n_0]-1}.$$

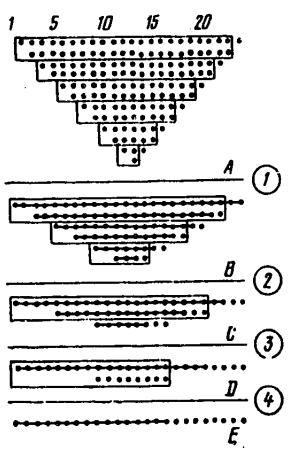


Fig. 4.4.18

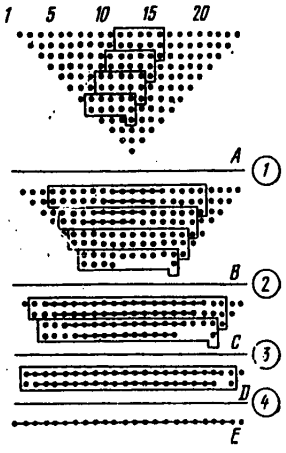


Fig. 4.4.19

These lambda'$_1$ adders add in pairs the 2 lambda'$_1$ numbers, and the remaining n_0-2 lambda'$_1$ numbers pass through the layer without translation. Obtained then at the outputs of the first layer are

$$n_1 = \lambda'_1 + (n_0 - 2\lambda'_1) = n^{(M-1)} = 2^M$$

numbers, which corresponds to the right boundary of the inequality (4.4.5). The remaining layers must be constructed according to the old rules, i.e. lambda$_i$ = [n_{i-1}/2] (i=2, 3, ..., M). It is evident that in the process

$$n_2 = \frac{n_1}{2} = n^{(M-2)} = 2^{M-1}, \quad \lambda_2 = n_2,$$

$$n_3 = \frac{n_2}{2} = n^{(M-3)} = 2^{(M-2)}, \quad \lambda_3 = n_3,$$

. . . . . .

$$n_M = \frac{n_{M-1}}{2} = n^{(0)} = 2, \quad \lambda_M = 2.$$

The number of layers M, as we see, is retained. It is interesting that the number of parallel adders in this case also does not change:

$$(n_0 - 2^M) + 2^{M-1} + 2^{M-2} + \ldots + 2^1 = n_0 - 2.$$

Shown in fig. 4.4.19 is the modified structure for the case n = 12. A number of features of the new structure can be seen from fig. 4.4.19 in addition to that $n_1 = n^{(3)} = 8$ (in contrast to $n_1 = 6$ in fig. 4.4.17), $n_2 = 4$ and $n_3 = 2$. The rules by which the new structure was built can be formulated the following way.

1) The apparatus is installed in a way such that the quantity of digits n^j_i at the outputs of each j-th position of each i-th layer meets the inequality $n^j_i \leqslant n^{(M-i)}$.

2) In each layer are installed only those assemblies (one-digit adders in this case) the removal of any one of which would lead to nonfulfillment of the requirement $n^j_i \leqslant n^{(M-i)}$.

3) The assemblies in the layers are used the "maximal" way (as many inputs of these assemblies as possible are fitted; in fig. 4.4.19, it is shown in particular that even the input of the carry of the low-order position of the parallel adder is used).

Let us call the device built by these rules "economical." It can be seen from fig. 4.4.19 that in this case, the "economical" simultaneous multiplier contains just as many layers (4) as the basic, and just as many parallel adders (11), but the quantity of one-digit adders has declined considerably (132 instead of 146). Moreover, device operating time has also been reduced. It can be seen from the figure that now it is

$$\begin{aligned} &(2n-2)\tau_E \quad \text{when } \tau_E \geqslant \tau_{CM}, \\ &(2n-4)\tau_E + 2\tau_{CM} \quad \text{when } \tau_E < \tau_{CM}. \end{aligned}$$

[cm = sm = sum]

We shall encounter later also the "economical" structures. From corollary 8, the "economical" modification can be built for any simultaneous mu ltiplier meeting conditions 1-5.

It can be noted that the formulated rules for building the "economical" circuit allow different structures, i.e. the simultaneous multiplier structure is not unequivocally defined.

Besides the basic and the "economical," many other modifications could be suggested, but we will not do this.

Given in table 4.4.2 as yet another example are the values of n_i and $lambda_i$ of the basic and "economical" structure of the type of simultaneous multiplier under discussion for the case $n_0=48$.

The main shortcoming of both modifications of the device in question consists in it being inexpedient to make a fast adder of the last, M + 1-th layer, since the carry propagation in it is nevertheless delayed by the adders of the preceding layers. We observed a similar situation in homogeneous array simultaneous multipliers of type I (see fig. 4.4.7).

4°. This shortcoming stems from the fact that within each layer carries are propagated from low-order to high-order positions. Later we shall investigate a series of structures for simultaneous multipliers in which high speed is attained because there are no carry propagation processes within layers. But first let us describe a structure that can be considered one-layer and which in a certain sense is the source for all subsequent modifications. This simultaneous multiplier is a chain of 2n-1 serially connected parallel counters (k, m). In each position of the device there is one parallel counter that adds all the digits of the partial products

relevant to a given position (i.e. up to n digits), and besides that, the carries coming from some counters located in the lower-order positions of the device [21]. Such a simultaneous multiplier for the case n_0=12 is illustrated by figs. 4.4.20 and 4.4.21; shown in the first one are several low-order positions of the device, and in the second--the arrangement of numbers at the inputs and outputs of the counters. In fig. 4.4.21, the outputs of each counter are connected by a slanting line. It can be seen from the figure that the most complex counter--(15, 4)--is in the 13th position. There are 2n-1 counters.

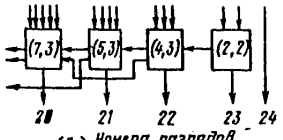


Fig. 4.4.20

Key:
1. Number of the positions

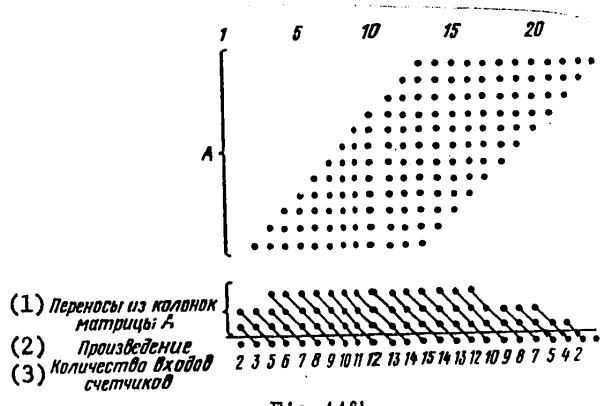


Fig. 4.4.21

Key:
1. Carries from columns of array A
2. Product
3. Number of inputs of counters

This type of device has a simple structure but substantial shortcomings: the parallel counters with a large number of inputs are complex circuits and carry propagation along the chain of counters takes much time.

5°. Now let us move on to multilayer structures in which there is no carry propagation within layers. Let us change the preceding device [21]. Let us formulate the product in two stages--first we obtain the two-row code of the product, i.e. the two numbers whose sum is the product sought, and then we add these two numbers in a fast adder.

In each of the M layers of the device that forms the two-row code of the product, in one position will be no more than one parallel counter (k, m). There will be no connections between counters within a layer. Let us connect the layers serially, just as before. If only one digit enters the input of a given layer in some position, then it will pass through the layer without translation (two digits in one position will also not be translated if in each of the lower-order positions at input of this layer there will also be no more than two digits).

Such a device for n=12 is illustrated in fig. 4.4.22, in which, as in fig. 4.4.21, slanting lines connect the output digits of the individual parallel counters. It

Table 4.4.2.

i	Основное ОУ (1)		"Экономичное" ОУ (2)		i	Основное ОУ (1)		"Экономичное" ОУ (2)	
	n_i	λ_i	n_i	λ_i		n_i	λ_i	n_i	λ_i
0	48		48		3	6	6	8	8
1	24	24	32	16	4	3	3	4	4
2	12	12	16	16	5	2	1	2	2

Key:
1. Basic simultaneous multiplier 2. "Economical" simultaneous multiplier

can be seen from fig. 4.4.22 that the device contains 3 layers in all; array A is translated into array B, then B into C and finally, C into D. It is evident that the maximal number of inputs of one counter in such a device is n.

It follows from the description of the method for constructing the layers that for a gievn n_{i-1} there is an n_i that is the smallest integer that satisfies the inequality

$$2^{n_i}-1 \geqslant n_{i-1}.$$

Hence it follows that

$$n_i = [\log_2 n_{i-1}] + 1. \qquad (4.4.7)$$

A graph of the function $n_i = f(n_{i-1})$ is shown in fig. 4.4.16e (p 203). It can be seen from the description of this dependency that in this case too conditions 1-5 are met. Since $\max(n_{i-1}) = n^{(T)}$ when $n_i = n^{(T-1)}$ (corollary 9), then there is derived from (4.4.7):

$$n^{(T-1)} = \log_2 (n^{(T)} + 1),$$

i.e.

$$n^{(T)} = 2^{n^{(T-1)}} - 1. \qquad (4.4.8)$$

Several of the first values of $n^{(T)}$, computed by using relationship (4.4.8), are given in fig. 4.4.16e.

From corollary 8

$$M = T \text{ when } n^{(T-1)} < n \leqslant n^{(T)}.$$

For example, M = 3 when n = 48, since

$$n^{(2)} = 7 < n \leqslant 127 = n^{(3)}.$$

The sequence of numbers n_i for this example has this form: 48, 6, 3, 2.

It is evident that when this method is used, $n^{(T)}$ increases rapidly with the increase of T. For real values of n, the number of layers does not exceed the value 2-3.

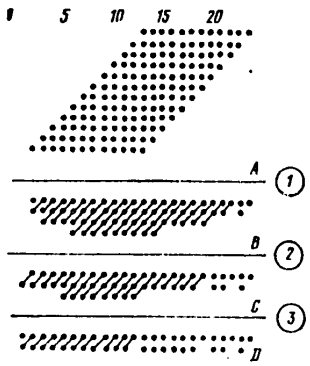


Fig. 4.4.22

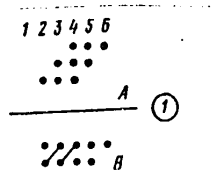


Fig. 4.4.23

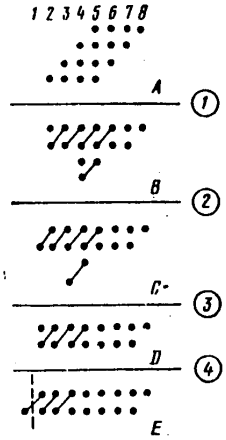


Fig. 4.4.24

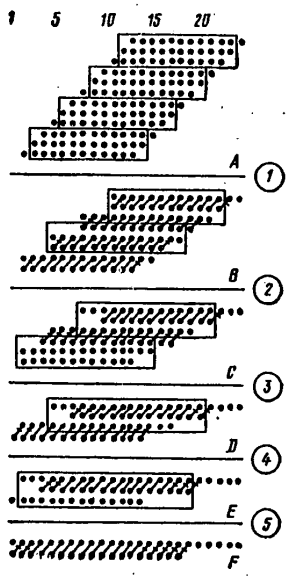


Fig. 4.4.25

It should be noted that prior to the appearance of work [21], this method of adding sets of numbers was suggested in [42] in a more general form for number systems with the arbitrary base r. A table of values of $n^{(T)}$ for r = 2 was given in [42]. This same table in [21] contained a number errors:
for $n^{(3)}$, $n^{(4)}$ and $n^{(5)}$, 15, 32 and 64 were indicated instead of 127, $2^{127}-1$ and $2^{2^{127}-1}-1$.

6°. The method just described provides for a very small number of layers, but parallel counters with up to n inputs are needed to use the method. When n is large, such assemblies are bulky. Therefore, it is advisable to consider methods that allow using parallel counters (k, m) with limited k.

One of the earliest works in this field was [42] by V. M. Khrapchenko; he described a method of fast addition of a large number of numbers with the base r by using identical parallel counters (k, m), each of which adds k base-r digits of equal weight. This method was developed independently of works [19 and 43] and is a generalization of the idea advanced in them. Multilayer array simultaneous multipliers, built with counters (k, m), with different k, were also investigated in works [17 and 21] and in a number of later publications.

Let us begin with the case k=2 and r=2 (subsequently in this section we will be talking about binary addition only). The use of the counters (2, 2), i.e. binary half-adders, when n =3 and n=4 is shown in figs. 4.4.23 and 4.4.24. It is evident that when n=3, the device contains one layer, but when n=4, four layers. Just as in the preceding case (fig. 4.4.22), within each layer of the device that forms the two-row code of the product, there are no connections between counters. The difference is that now in one position of one layer there can be several counters (when n=4, there are two counters each in the 5th position of the 1st layer and in the 4th position of the 2nd layer).

In investogating the structure when n is larger, one can be assured that the number of layers in a device built with counters (2, 2) by this method is approximately 3.7n-10.4.* Such devices, of course, are too slow.

7°. Let us go on to the case of k=3. By now there have been a considerable number of suggestions for methods of building multilayer simultaneous multipliers with counters (3, 2), i.e. with one-digit binary adders. Let us look at one of the earliest proposals [17 and 18].

Shown in fig. 4.4.25, taken from work [21], is the arrangement of numbers at the inputs and outputs of each layer of the suggested device for the case of n=12. The principle for building this simultaneous multiplier is as follows.

Positioned in the first layer of the device are $\text{lambda}_1=[n/3]$ rows of one-digit binary adders. Accordingly, n initial terms are divided into non-intersecting groups: $\text{lambda}_1=[n/3]$ full groups of 3 numbers each and one unfull group containing $\text{nu}_1=n-3[n/3]$ numbers ($\text{nu}_1=0$, 1 or 2). Each of the lambda_1 rows of adders adds the 3 terms of the corresponding full group (each such triad of numbers in the figure is enclosed by a rectangle) and produces two numbers--a series of digit sums and a series of cascaded carries, i.e. the three-row code is translated into two-row. At the outputs of the first layer there is formed the n_1-row code, and

$$n_1=2\lambda_1+\nu_1=2[n_0/3]+n_0-3[n_0/3]=n_0-[n_0/3]$$

(nu_1 terms of the unfull group pass through the layer without translation). In fig. 4.4.25, $\text{lambda}_1=4$, $\text{nu}_1=0$ and $n_1=8$. The lines connecting the output digits of the one-digit adders that add two digits each and therefore can be replaced by half-adders are crossed out by short lines in this and the following figure.

Arranged in similar fashion in the second layer are $\text{lambda}_2=[n_1/3]$ rows of one-digit adders. The second layer replaces the n_1-row code by n_2-row, and

$$n_2=2\lambda_2+\nu_2,$$

where $\nu_2=n_1-3\lambda_2$. In fig. 4.4.25, $\lambda_2=2$, $\nu_2=2$, $n_2=6$.

---

* The error of this empirical formula when n is less than or equal to 80 is no more than ± 1.8. The formula $M=2^{n-2}$ from [21] is incorrect.

Each i-th layer is built by the same method, i.e.

$$n_i=2\lambda_i+\nu_i=2[n_{i-1}/3]+(n_{i-1}-3[n_{i-1}/3])=n_{i-1}-[n_{i-1}/3]. \qquad (4.4.9)$$

The graph of the function $n_i=f(n_{i-1})$ is given in fig. 4.4.16b (p 203). From the graph and (4.4.9) it is evident that conditions 1-5 are met. The structure of this simultaneous multiplier is described by the relationships

$$n_i=2\lambda_i+\nu_i,\ \lambda_i=[n_{i-1}/3],$$

$$\nu_i=n_{i-1}-3\lambda_i,\ n_M=2,\ i=1, 2, \ldots, M.$$

When $n_i=n^{(T-1)}$, the maximal value of $n_{i-1}=n^{(T)}$ is obtained, evidently in the case when the layer contains lambda=$[n_i/2]$ rows of adders, i.e. for each pair of numbers at the outputs of the layer there are 3 numbers at the inputs of the layer. In the process, nu = n_i-2 lambda numbers (nu = 0 or 1) pass through the layer without translation. Therefore

$$n^{(T)}=3\lambda+\nu=3\left[\frac{n^{(T-1)}}{2}\right]+\left(n^{(T-1)}-2\left[\frac{n^{(T-1)}}{2}\right]\right)=n^{(T-1)}+\left[\frac{n^{(T-1)}}{2}\right]. \qquad (4.4.10)$$

Using this relationship, one can calculate the sequence of values of $n^{(T)}$; the first few are given in fig. 4.4.16b.

Just as before, M = T when

$$n^{(T-1)}<n\leqslant n^{(T)}. \qquad (4.4.11)$$

For example, M = 9 when n = 48, since

$$n^{(8)}=42<n\leqslant 63=n^{(9)}.$$

The sequence of numbers of n_i for this example can be determined by using formula (4.4.9): 48, 32, 22, 15, 10, 7, 5, 4, 3, 2.

It is pointed out in work [44] that when this method is used

$$M=\left]\log_{\frac{3}{2}}\frac{n}{2}\right[\qquad (4.4.12)$$

or

$$M=\left]\log_{\frac{3}{2}}\frac{n}{2}\right[+1^{*}). \qquad (4.4.13)$$

Comparing (4.4.12) and (4.4.13) with (4.4.4), we see that the number of layers is increased considerably, but it should be considered that the delay in each layer is now equal only to the time of operation of one one-digit adder.

Concerning the structure of the simultaneous multiplier corresponding to fig. 4.4.25, one can note that it belongs to the class of "basic" structures since it was built by the rules given at the end of subsection 2° of this section.

8°. Let us describe an "economical" modification to the device just discussed [21]. In accordance with the rule given in 3°, first let us put in the first layer not lambda$_1$=$[n_0/3]$ rows of adders, but lambda'$_1$=n_0-$n^{(M-1)}$ rows. Then

$$n_1=2\lambda'_1+(n_0-3\lambda'_1)=n^{(M-1)}.$$

*]d[is the smallest integer not less than d.

In each of the remaining layers, just as in the preceding device, the number of rows of adders will be $\text{lambda}_i=[n_{i-1}/3]$. With that, as noted earlier, $n_2=n^{(M-2)}$, $n_3=n^{(M-3)}$, ..., $n_M=n^{(0)}=2$. Second, each row of adders in each layer will maximally contract from both sides (until further contraction would lead to the derivation of an n_i greater than $n^{(M-i)}$). Fig. 4.4.26 shows the structure of such an "economical" simultaneous multiplier (n=12, as in fig. 4.4.25). It is evident from the figure that now installed in the first layer of the device are just those one-digit adders, the lack of any one of which would cause the quantity of numbers n_1 at the outputs of the layer to be greater than $n^{(M-1)}=n^{(4)}=9$. In the process, the one-digit adders are replaced by half-adders (marked by small lines) where possible. Similar measures are also taken in the remaining layers. In comparing the last two figures, one can see that the quantity of layers in the "economical" device is the same as in the preceding, but the volume of apparatus is less.

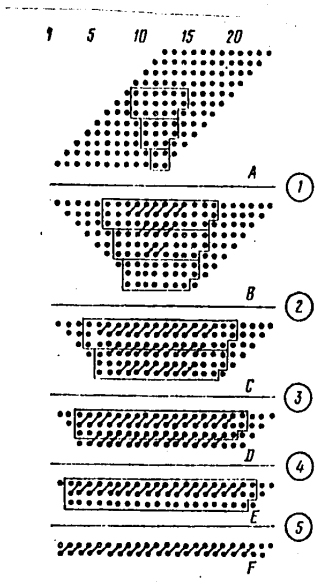


Fig. 4.4.26

It can be shown that the "economical" simultaneous multiplier of this type contains (n-1)(n-2) one-digit adders and half-adders. This value is the minimal for any simultaneous multiplier built of such assemblies and forming the two-row code of the product. In this case (fig. 4.4.26), this qunatity is 11 x 10 = 110, i.e. 26 fewer than in the preceding device.

It should be noted that prior to the appearance of work [21], practically the same method for constructing multilayer simultaneous multipliers with counters (3, 2) was described in work [17]. The main difference is that a "truncated" device containing n basic and several additional positions was discussed in [17].

9°. Let us now assume that counters (7, 3) and (3, 2)* [21] are available to the developer. Each layer will be built by a method which to a certain extent is similar to the preceding one. In each i-th layer, let us put $\text{lambda}_i=[n_{i-1}/7]$ rows of counters (7, 3). One such row translates 7-row code into 3-row. The remaining $n_{i-1}-7\,\text{lambda}_i$ input numbers of the i-th layer either pass through the layer without translation (when $n_{i-1}-7\,\text{lambda}_i$ is greater than or equal to 2) or are translated by yet another row of counters (when $n_{i-1}-7\,\text{lambda}_i$ is less than or equal to 3). It is evident that when $n_{i-1}-7\,\text{lambda}_i = 3$, two-row code is formed at the outputs of this additional row of counters, and three-row code when it is greater than 3.

* The availability of counter (3, 2) in the process does not change the essence of the problem, since it can be replaced by the counter (7, 3); generally any counter (k_1, m_1) can be replaced by counter (k_2, m_2) when k_2 is greater than or equal to k_1 (zeros are fed to the k_2-k_1 extra inputs, but of course the counter (k_1, m_1) is simpler than (k_2, m_2)).

The graph of the function $n_i=f(n_{i-1})$ is given in fig. 4.4.16c (p 203). It follows from the description of this function that conditions 1-5 are met. The structure of the simultaneous multiplier is described by the relationships

$$n_i=3\lambda_i+\nu_i,\ \lambda_i=[n_{i-1}/7],$$
$$\nu_i=\begin{cases}n_{i-1}-7\lambda_i \text{ when } n_{i-1}-7\lambda_i\leqslant 2,\\ 2 \quad \text{when } n_{i-1}-7\lambda_i=3,\\ 3 \quad \text{when } n_{i-1}-7\lambda_i>3,\end{cases} \qquad (4.4.14)$$
$$n_M=2,\ i=1, 2, \ldots, M.$$

In accordance with corollary 9, the maximal value of $n_{i-1}=n^{(T)}$ when $n_i=n^{(T-1)}$ is obtained, evidently, in that case when the layer contains $\lambda'=[n_i/3]$ rows of counters (7, 3) and $\lambda''=[(n_i-3\lambda')/2]$ rows of counters (3, 2) (lambda" = 0 or 1), but $\nu=n_i-3\lambda'-2\lambda''$ numbers pass through the layer without translation (ν = 0 or 1). In other words, for each triad of numbers at the outputs of the layer there are 7 numbers at its inputs; if the quantity of $n_i-3\lambda'$ numbers left is 2, then for them there are another 3 numbers at the inputs of the layer, but if $n_i-3\lambda'=1$, then the corresponding number has passed through the layer without translation. Thus,

$$n^{(T)}=7\lambda'+3\lambda''+\nu,\ \lambda'=[n^{(T-1)}/3],$$
$$\lambda''=[(n^{(T-1)}-3\lambda')/2],\ \nu=n^{(T-1)}-$$
$$-3\lambda'-2\lambda'',\ T=1, 2, \ldots \qquad (4.4.15)$$

The first few numbers of $n^{(T)}$, derived by using these relationships, are given in fig. 4.4.16c.

Just as before, $M=T$ when $n^{(T-1)}<n\leqslant n^{(T)}$.
For example, M=5 when n=48. The sequence of numbers n_i for this example can be derived by using (4.4.14):

48, 21, 9, 5, 3, 2.

In comparing the values of $n^{(T)}$ in figs. 4.4.14c and b (p 203), we see that now when n is greater than 4, there are fewer layers in the new device.

Let us note that one could put in each i-th layer only $\text{lambda}_i=[n_{i-1}/7]$ rows of counters (7, 3), and the remaining $n_i-7[n_{i-1}/7]$ numbers would pass through the layer without translation. For a structure of this type, $n^{(T)}=7[n^{(T-1)}/3]+(n^{(T-1)}-3[n^{(T-1)}/3])$ and therefore the sequence of numbers $n^{(0)}, n^{(1)}, \ldots$ would change.* Condition 3 is not met in the new structure, i.e. n_i can decline as

* It is possible that the author in work [21] had precisely this method of constructing simultaneous multipliers in mind, since in it instead of $n^{(5)}=80$, the value 79 is indicated.

n_{i-1} increases, because of which the quantity of layers M can be greater than in the structure just described, but the volume of apparatus—less. Such structures can be suggested also when other type assemblies are used—counters (15, 4) (see subsection 11° of this section), k-bit adders (section 6.3.1), translators N—2 (section 6.3.2) and others. Such structures in translators N—2 are discussed in work [45].

10°. Similar to the way this was done in subsections 3° and 8° of this section, one can build an "economical" multilayer simultaneous multiplier with the counters (7, 3) and (3, 2). In such a device, after the first layer are obtained $n_1=n^{(M-1)}$ numbers, after the second—$n_2=n^{(M-2)}$ etc. For example, if n=48, then $n_1=35$, $n_2=15$, $n_3=7$, $n_4=3$ and $n_5=2$.

The basic and modified ("economical") simultaneous multipliers for the case n=12 are illustrated in the figs. 4.4.27 and 4.4.28 respectively. It is evident that there is less apparatus in the latter device.

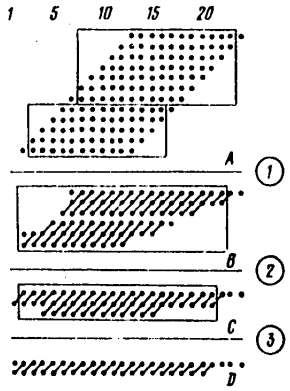


Fig. 4.4.27

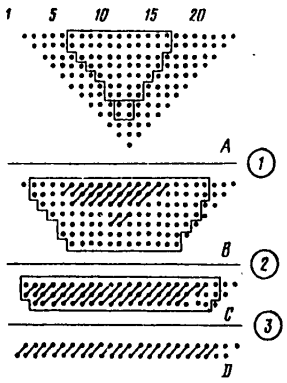


Fig. 4.4.28

11°. Let us now assume that a multilayer simultaneous multiplier is built from the counters (15, 4), (7, 3) and (3, 2). Discussion similar to the preceding leads to the formulas

$$\left.\begin{aligned} &n_i=4\lambda_i+\nu_i,\quad \lambda_i=[n_{i-1}/15],\\ &\nu_i=\begin{cases} n_{i-1}-15\lambda_i & \text{when } n_{i-1}-15\lambda_i\leqslant 2,\\ 2 & \text{when } n_{i-1}-15\lambda_i=3,\\ 3 & \text{when } 3<n_{i-1}-15\lambda_i\leqslant 7,\\ 4 & \text{when } n_{i-1}-15\lambda_i>7, \end{cases}\\ &n_M=2,\ i=1,\ 2,\ \ldots,\ M, \end{aligned}\right\} \quad (4.4.16)$$

$$n^{(T)}=15\lambda'+7\lambda''+3\lambda'''+\nu,\quad \lambda'=\left[\frac{n^{(T-1)}}{4}\right],$$

$$\lambda''=\left[\frac{n^{(T-1)}-4\lambda'}{3}\right],\quad \lambda'''=\left[\frac{n^{(T-1)}-4\lambda'-3\lambda''}{2}\right],$$

$$\nu=n^{(T-1)}-4\lambda'-3\lambda''-2\lambda''',\quad T=1,\ 2,\ \ldots \quad (4.4.17)$$

The graph of the function $n_i=f(n_{i-1})$ and the first few values of $n^{(T)}$ are shown in fig. 4.4.16d (page 203).* Conditions 1-5 are met. It is evident from fig. 4.4.16d that when $16\leqslant n\leqslant 22$, $36\leqslant n\leqslant 78$, $n\geqslant 81$, there are fewer layers in this device than in the preceding one built from counters (7, 3). For example, when n=48, there are 4 layers (since $22=n^{(3)}<48<n^{(4)}=78$).

From (4.4.16) for $n_0=48$ we obtain all remaining values of n_i: $n_1=14$, $n_2=4$, $n_3=3$ and $n_4=2$.

12°. When counters (15, 4) are used, just as in the other cases, one can build "economical" simultaneous multipliers in which $n_i=n^{(M-i)}$ $(i=1, 2, \ldots, M)$.

Another two methods of building multilayer array simultaneous multipliers will be discussed in sections 6.3.1 and 6.3.2 in chapter 6. By the first of these methods, the multiplier is made with large-scale integrated (LSI) circuits, each of which is a k-bit binary parallel adder. The second method calls for using LSI circuits, each of which is a small-bit converter of N-row code into two-row. Despite the difference in the element base, the structures of the simultaneous multipliers in both these cases are related to the structures discussed in this section. Use of the k-bit adders and the N→2 translators is reflected in fig. 4.4.16g (p 203).

Aside from the methods depicted in fig. 4.4.16, there are also others. For example, in work [47], there are discussed multilayer simultaneous multipliers in which the type assembly is an adder that adds r k-bit binary numbers and produces their sum in the form of one d-bit number (such an assembly can be made in the form of ROM containing 2^{rk} numbers of d bits each). It is suggested that such r and k be chosen that d is a multiple of k. A chain (series) of such adders translates the group of r multibit numbers into s numbers, and

$$d=sk, \qquad r=2^{(s-1)k}+2^{(s-2)k}+\ldots+1.$$

The graph $n_i=f(n_{i-1})$ for this simultaneous multiplier is shown in fig. 4.4.29 (the case selected is r=9, k=3, d=6 and s=2). It is evident that condition 3 is not met if r-1 is greater than s (it is met only when r=3, k=1, d=2 and s=2 which corresponds to the method reflected in fig. 4.4.16b (p 203) and which is thus a particular case of the methods discussed in [47].

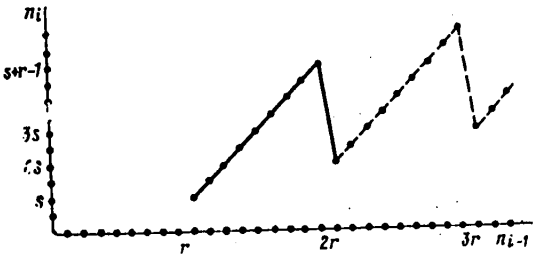


Fig. 4.4.29

* Relationships (4.4.16) and (4.4.17) are given here for the first time. In work [21], 21, 61 and 226 are erroneously indicated for $n^{(3)}$, $n^{(4)}$ and $n^{(5)}$.

5. High-Speed Synchronous Devices for Division

Until recently, there was less development of methods for speeding up division compared to those for addition, subtraction and multiplication. This is apparently due, on the one hand, to the "sequential" nature of the procedure for conventional division (the next digit of the quotient can not be selected and the new remainder cannot be computed until the preceding remainder is derived and analyzed) and, on the other, to division being a relatively infrequent operation. In some digital computers, direct performance of division is "avoided" by using standard subroutines that include multiplication and addition-subtraction. However, division is now included in the list of operations of almost all modern general-purpose machines and the necessity of speeding up division follows from the very existence of the very efficient methods for speeding up multiplication: a non-fast operation for division makes no sense when fast addition-subtraction and multiplication are available since in this case the division subroutine may afford greater speed than the hardware solution. It must be noted that in recent years there have appeared a number of new, powerful, synchronous methods for speeding up division and the development of new methods is continuing. The main known methods are discussed in this chapter.

Synchronous methods for speeding up division are conveniently classified into two groups: speeding up division in each cycle of which one or more digits of the quotient is selected and the new partial remainder * computed, and that which "avoids" division by using multiplication or another procedure.

Methods of fast division, performed by a conventional circuit--analysis of the remainder and divider, determination of the next digit of the quotient and computation of the new remainder--may also be divided into two groups.

The first includes methods using algorithms under which changing the sign of the remainder as a result of the next cycle of division does not lead to the need for returning to the preceding remainder. These methods for speeding it up are called nonrestoring division methods (section 5.4). As these methods evolved, they were enriched with new ideas such as using a redundant set of digits for advance formation of the quotient, deriving the next remainder in the form of two-row code and using arrays of adders.

Methods in the second group return to the preceding remainder after the remainder sign is changed, i.e. they restore it (section 5.1).

Of the methods that perform division by "avoiding" it, we shall discuss iterative methods that use multiplication (section 5.3) and the group of methods suggested by Stefanelly (section 5.2).

---

* From now on, let us call the unshifted partial remainder simply the remainder and designate it by B_i; the dividend and divisor will be designated by the letters A and C respectively.

5.1. Methods of Fast Restoring Division

The main way to speed up restoring division consists in advance formation of several multiples of the divisor and in the simultaneous production, during one division cycle, of the several differences between the (shifted) remainder and each of the several multiples. Analysis of the signs of the differences produced allows deriving in the process several digits of the quotient at once. As an example, we can cite the division method [1] in which in each division cycle there are computed three differences: $4B_i-C$, $4B_i-2C$ and $4B_i-3C$. By the signs of three differences derived, it is easy to determine the quaternanry (i.e. two binary) digits of the quotient. One of the differences or the value of $4B_i$ becomes the new remainder. When, for example, $4B_i-2C\geqslant 0$, $4B_i-3C<0$, then it is evident that the digit 2 (i.e. 10) should be written to the quotient and $B_{i+1}=4B_i-2C$.

It is seen that this group of methods actually performs restoring division.

Some speed-up in conventional restoring division (with definition of one binary digit per cycle) is made possible by an additional adder [2]. In one version of this method, the additional adder repeats the operation of the main adder and "lags behind" it somewhat. If in the main adder there is derived a $2B_i-C$ less than 0, then a zero is written to the quotient, and the additional adder, which at this time contains the value of $2B_i$, becomes the main one; a new shift is made in it and a new cycle begins: $2B_i \times 2 = 2B_{i+1}$, $2B_{i+1}-C$ etc. In the process, the main adder becomes the additional, and in it the remainder is restored: $(2B_i-C) + C = 2B_i$, $2B_i \times 2 = 2B_{i+1}$. In another version of the method, the additional adder also "lags behind" the main one. If in the main one there is produced a $2B_i-C$ less than 0, then 0 is written to the quotient and the number $2B_i$ from the additional adder is added to the main one: $(2B_i-C) + 2B_i = 4B_i-C$, i.e. the remainder $2B_i$ is restored at once, it is doubled and a new cycle is performed: $2B_{i+1}-c$. In the additional adder at this time, the number is doubled:

$$2B_i\times 2=2B_{i+1};$$

if it again turns out that the number in the main adder is negative, the procedure is repeated.

Restoring division can also be speeded up by making a device for division in the form of an iterative array. Such devices were already mentioned in section 4.4.1 (see [22-24, 31-34]) of chapter 4.

For example, the device described in [3] consists of identical elements, each of which (fig. 5.1.1) performs the function of a one-digit binary subtracter (when D=0) or the function of a one-digit switch (when D = 1)*.

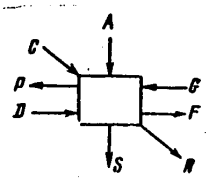


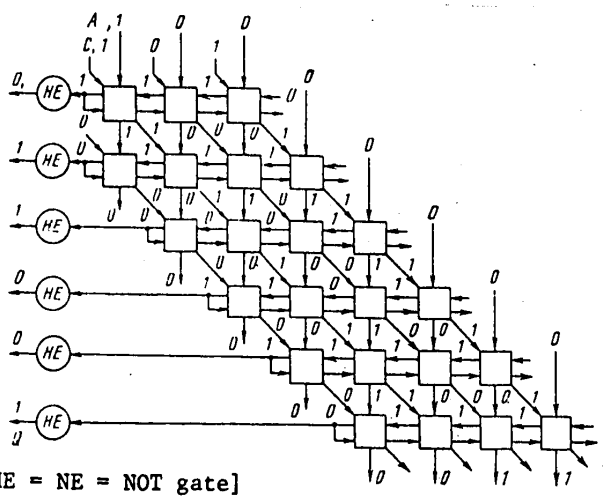
Fig. 5.1.1

*)In fig 5.1.1 $F=D$, $R=C$, $P=\bar{A}C \vee \bar{A}G \vee CG$, $S=\bar{A}D \vee A\bar{C}\bar{G}D \vee \bar{A}CGD \vee \bar{A}\bar{C}GD \vee ACG\bar{D}$.

In the first case (D=0), the result of the subtraction is sent to the output S, in the second (D=1)--the digit of the minuend (digit of the preceding remainder). The entire circuit consists of layers (fig. 5.1.2), in each of which the divisor C is subtracted from the doubled preceding remainder coming from the outputs of the preceding layer. Along the layer in the process, the signals P and G of the borrow are propagated (see figs. 5.1.1 and 5.1.2). The borrow signal in the high-order position of the layer determines the sign of the difference $2B_i-C$ (and the next digit of the quotient) and controls the generation of the signals S in all elements of a given layer. Fig. 5.1.2 "describes" the example of the division of A=0.100 by C=0.101, by which there is derived the quotient Q=0.11001 and the remainder B_6=0.11; the precise relationship between A, C, Q and B_6 in this device takes the form

$$\frac{A}{C} = Q + \frac{B_6}{C2^5}.$$

In this device, division is speeded up not by complicating the algorithm, but by use of high-speed miniature elements connected by short regular bonds.*



[HE = NE = NOT gate]

Fig. 5.1.2

5.2. Stefanelly Methods

Stefanelly methods are based on several algorithms for deriving the inverse value 1/C of the binary divisor C situated in the range between 1/2 and 1; these methods may be expanded to also compute the quotient from division of two numbers.

The quotient $Q \approx 1/C$ is formed in two stages--first the quotient is produced in the form of a binary number with redundant representation of the digits, and then it is translated into simple binary form.

Generally speaking, any positive or negative integers are used as the digits x_i in the redundant binary number

$$X = x_1 2^{-1} + x_2 2^{-2} + \ldots$$

* Strictly speaking, this method is a particular case of restoring division--nonperforming division, by which the subtraction is carried to conclusion only in the case of a positive difference of $2B_i-C$; otherwise, subtraction is interrupted and the remainder is shifted (see section 7.8).

The main procedure for forming the inverse value of the divisor

$$C = c_1 2^{-1} + c_2 2^{-2} + \ldots + c_n 2^{-n}$$

provides for deriving Q in the redundant form

$$Q = a_0 2^0 + a_1 2^{-1} + a_2 2^{-2} + \ldots$$

The product of CQ has to be equal, evidently, to one. Let us assume that it is equal to the binary number 0.111 .... In arranging in matrix form the partial products of the multiplicand C by the individual digits a_i of the multiplier Q, we see in the first column of this matrix the term $c_1 a_0$, in the second--the terms $c_2 a_0$, $c_1 a_1$, in the third--$c_3 a_0$, $c_2 a_1$, $c_1 a_2$ etc. The main idea of the method consists in selecting the redundant digits a_i in such a way that the sum of the terms in one column are exactly equal to one, i.e. to the next digit of the product of CQ. It is easy to show that in the process the values of a_i are defined by the relationships

$$\begin{aligned}
a_0 &= 1,\\
a_1 &= 1 - c_2,\\
a_2 &= 1 - (c_2 a_1 + c_3),\\
a_3 &= 1 - (c_3 a_2 + c_2 a_1 + c_4),\\
&\ldots\ldots\ldots\ldots\ldots\ldots\\
a_l &= 1 - (c_3 a_{l-1} + c_4 a_{l-2} + \ldots + c_l a_1 + c_{l+1}).
\end{aligned}$$

Described in work [5] is an array device that computes the digits a_i by these formulas. This device is very unwieldy due primarily to the large range of possible values for the digits a_i. For example, with a 12-bit divisor

$$C \quad -21 \leqslant a_9 \leqslant 6, \quad -28 \leqslant a_{14} \leqslant 232, \quad -80 \leqslant a_{18} \leqslant 1578.$$

Further in work [5] there is suggested successively three modifications to the method aimed at reducing the range of values for the digits a_i and simplifying the device. In the last modification, the digits a_i are within the range -3 ... +3, which considerably simplifies computing the values of a_i and translating Q into conventional binary form.

5.3. Performing Division by Multiplication

In the appartus of arithmetic units of modern digital computers, there are often realized methods of division that use particular iterative formulas that include multiplication [6-12]. In such arithmetic units, the quotient or inverse value of the divisor are found by successive approximations. The majority of these methods are based on the known iterative formula

$$y_{i+1} = y_i(2 - y_i C), \quad i = 0, 1, 2, \ldots, \qquad (5.3.1)$$

used for the approximate computation of the value of 1/C. Here y_0, y_1, y_2, ... are successive approximations of the value of 1/C. It is known that if the first approximation of y_0 is within the range of $0 < y_0 < 2/C$,

then $y_i \to 1/C$ when $i \to \infty$. To limit the values of y_i (including y_0) at the top, the value of C is limited at the bottom. It is conventionally assumed that $1/2 \leqslant C < 1$.

Let delta_i be the relative error of the value of y_i:

$$y_i = \frac{1}{C}(1 - \delta_i).$$

After substituting this expression in (5.3.1), we get:

$$y_{i+1} = \frac{1}{C}(1 - \delta^2_i),$$

which means doubling the quantity of valid digits after each iteration.

The number of iterations that must be carried out to obtain the required accuracy of the result, evidently, is strongly dependent on the choice of the first approximation. Therefore, much attention is paid to choosing the value of y_0. There are several methods for choosing y_0 [9]. The most precise (up to 8-9 valid binary signs in y_0) is afforded by tabular methods [9, 11, 12].

The moment division ends in the arithmetic unit is defined not by comparing the difference $|y_{i+1} - y_i|$ with some tolerance, as occurs when performing division by a a subroutine, but by counting the number of iterations. For example, if y_0 contains 8 valid digits $(|\delta_0| \leqslant 2^{-8})$, then to obtain, let us say, a 15-bit quotient, one iteration is performed; two iterations are needed for a 30-bit quotient, etc.

There are a large number of different machine algorithms to implement the iterative process based on the formula (5.3.1). To increase the rate of convergence of the iterations, some of these algorithms provide for specific changes in the basic formula (5.3.1) (see for example, [13], pp 535-536). Four of the most well known algorithsm will be briefly discussed here.

The procedures performed by the first of these algorithms literally follow the very writing of the relationship (5.3.1):

$$\begin{aligned}
&y_0 C,\\
&2 - y_0 C,\\
&y_1 = y_0 (2 - y_0 C),\\
&y_1 C,\\
&2 - y_1 C,\\
&y_2 = y_1 (2 - y_1 C),\\
&\cdots\cdots\cdots\\
&y_i C,\\
&2 - y_i C,\\
&y_{i+1} = y_i (2 - y_i C).
\end{aligned}$$

In the process, each cycle of the iterative process consists of three sequentially performed operations: multiplication, taking of the complement and multiplication again.

For convenience of presentation, let us introduce the designations:

$$x_i=y_{i-1}C, \qquad R_i=2-x_i.$$

One (i-th) iteration then can be described by the relationships

$$x_i=y_{i-1}C, \quad R_i=2-x_i, \quad y_i=y_{i-1}R_i. \qquad (5.3.2)$$

When $i\longrightarrow\infty$, the values of x_i and R_i tend to 1, but y_i, as was noted, tends to 1/C.

In the second algorithm, the i-th iteration is described by the relationships [6, 10]

$$x_i=x_{i-1}R_{i-1}, \quad R_i=2-x_i,$$
$$y_i=y_{i-1}R_i \ (i=2, 3, \ldots). \qquad (5.3.3)$$

The first iteration is made somewhat differently, but precisely as with the first algorithm:

$$x_1=y_0C, \quad R_1=2-x_1, \quad y_1=y_0R_1.$$

The values of x_i, R_i and y_i in these two algorithms have completely identical numerical values (if apparatus errors are ignored), which can easily be seen by translating the relationship (5.3.2):

$$x_i=y_{i-1}C=Cy_{i-2}R_{i-1}=x_{i-1}R_{i-1}$$

(we arrive at the relationship used in the formulas (5.3.3)). The second algorithm differs from the first only in that x_i is computed by multiplying x_{i-1} by R_{i-1} instead of using the equivalent formula $x_i=y_{i-1}C$. However, this difference has considerable effect on apparatus implementation since after deriving R_i, the values of y_i and x_{i+1} can be computed simultaneously.

Iteration in the third algorithm [8, 10, 11] is performed by the same relationships (5.3.3) as with the second (for i=2, 3, ...). The difference is in the first iteration:

$$x_1=y_0C, \quad R_1=2-x_1, \quad y_1=y_0AR_1,$$

where A is the dividend. The values of x_i and R_i in the process match the similar values derived with the first two algorithms (again if apparatus errors of multiplication and subtraction are ignored). But y_i in this case tends to the quotient of A/C. The third algorithm was implemented in the arithmetic unit of the IBM 360/91 machine [11].

The fourth algorithm was developed in the work [7]. The next, i-th, iteration in this case consists in computing the values of

$$w_i=w^2_{i-1}, \ r_i=1+w_i, \ \varphi_i=\varphi_{i-1}r_i \ (i=2, 3, \ldots). \quad (5.3.4)$$

The first iteration is described by the relationships

$$w_1=1-y_0C, \quad r_1=1+w_1, \quad \varphi_1=y_0Ar_1.$$

If we ignore the apparatus errors of multiplication and subtraction, then the third and fourth algorithms practically coincide, since

$$x_i=1-w_i, \quad R_i=r_i, \quad y_i=\varphi_i \quad (i=1, 2, \ldots).$$

However, the presence of apparatus errors leads to the computations by (5.3.3) and (5.3.4) not proceeding identically. For example, in the first case, the value of

$R_i=2-x_i=2-x_{i-1}R_{i-1}$ may be less than one, but in the second case

$r_i=1+w_i=1+w^2_{i-1}$ cannot be less than one.

The processes of accumulation of apparatus and methodological errors for the division methods described in this section are rather complex and proceed differently when the different algorithms are used. Apparently, there are now no methods for precise analysis of these processes. Therefore, relatively rough estimates of errors are generally used in practice. These estimates are based on the assumption that the apparatus errors of the different operations (for example, the error of multiplication of $y_{i-1}R_i$ in the first algorithm and the error of the multiplication of y_iC performed after it) are independent random values, which, generally speaking, is incorrect. The error after the i-th iteration, i.e. the difference between y_i and the quotient sought, is usually estimated "at the top" in the assumption that apparatus errors of all intermediate operations performed after the start of division had the most "disadvantageous" signs and absolute values.

Since several multiplications (usually 4-5) have to be performed in every division by any of the methods discussed here, these methods are especially efficient in arithmetic units with fast (for example, array) multipliers.

5.4. Fast Nonrestoring Division

Discussed in this section are various synchronous methods of fast nonrestoring division; the nonrestoring division method described in section 7.8 is a particular case of the group of methods considered here.

5.4.1. General Description of Nonrestoring Division

In general form, the synchronous method of nonrestoring division can be described as follows. The operation consists of repeating one-type cycles, of which only the first and last somewhat differ from the rest. During the next i+1-th cycle of division in a device, the next remainder B_{i+1} is computed:

$$B_{i+1}=B_ir-a_{i+1}C. \qquad (5.4.1)$$

Here i+1 = 1, 2, 3, ... is the number of the next cycle of division; B_i is the preceding remainder; r is the number system base in which the division is performed;*

* The value of r can also be defined as the number system base in which the quotient is formed in advance.

C is the divisor and a_{i+1} is the next digit of the quotient.

From (5.4.1):

$$B_0/C=\sum_{i=1}^{k} a_i r^{-i}+B_k/r^k C. \qquad (5.4.2)$$

Here k designates the full number of cycles of division. Thus, the initial remainder, i.e. the first of the remainders B_i, in which the formula (5.4.1) is applied to compute B_{i+1} is the remainder B_0. This initial remainder in the general case differs from the dividend by some constant positive multiplier h:

$$\frac{A}{C}=\frac{hB_0}{C}=h\sum_{i=1}^{k} a_i r^{-i}+h\frac{B_k}{r^k C}. \qquad (5.4.3)$$

The sum $h\sum_{i=1}^{k} a_i r^{-i}$, derived in the process of performing the operations in the arithmetic unit, is usally taken as the result of the division, and the remainder term forms, thus, the error of the operation.

Each (i+1)-th cycle of division consists in selecting a_{i+1} and computing B_{i+1}. The digits of a_{i+1} are selected in the device by certain rules that include, in the general case, analysis of the values of C and B_i and certain operations with them. The remainder B_{i+1}, as mentioned before, is computed in accordance with the relationship (5.4.1). There is also produced an "attachment" of the next digit of the quotient a_{i+1} to the sum formed in the preceding cycles: $\sum_{i=1}^{l} a_i r^{-i}$,

i.e. the value $\sum_{i=1}^{l} a_i r^{-i}+a_{l+1}r^{-l-1}$. (5.4.4)

is formed.

Let us introduce the designation

$$Q=\sum_{i=1}^{k} a_i r^{-i}. \qquad (5.4.5)$$

The value of Q with precision to the remainder term $B_k/r^k C$ is equal to B_0/C. The value of hQ with precision to $hB_k/r^k C$ is equal to the sought quotient of A/C:

$$A/C=hQ+hB_k/r^k C. \qquad (5.4.6)$$

The digits a_i are selected from some set xi of permissible values of the digits:

$$\xi=\{\xi_1, \xi_2, \ldots, \xi_m\}. \qquad (5.4.7)$$

For determinacy let us assume that

$$\xi_1<\xi_2<\ldots<\xi_m. \qquad (5.4.8)$$

It should be noted that the relationship used to compute B_{i+1} could be written in another form. For example:

$$B_{i+1}=rB_i-a_iC$$

or

$$B_{i+1}=r(B_i-a_{i+1}C),$$

or

$$B_{i+1}=r(B_i-a_iC)$$

etc. In the process, the form of notation of the relationships (5.4.2), (5.4.3) and a number of other formulas changes. However, these translations of formulas denote only a change in the names of the variables and constants and do not change the essence of the method of pressure [as published] and its hardware implementation.

The set of values of xi_j (j = 1, 2, ..., m) and the base r are the main parameters for the method of nonrestoring division. To a considerable extent, they determine the speed and hardware complexity of the specific variant of the method.

As seen from (5.4.5), the number system base, in which the quotient Q is formed, is the number r, and as digits a_i in the r-nary digits of the number Q, the values of xi_1, xi_2, ..., xi_m are used.

Let us assume for generality that this number system differs from the system in which all numbers in a machine are represented (and, in particular, the dividend and divisor). Let us assume that used in the machine is the conventional p-nary system with the base p and the digits 0, 1, 2, ..., p-1.

The derived quotient Q must be translated from the r-nary form (5.4.5) into the p-nary form

$$Q=\Sigma d_i p^{-i}, \qquad (5.4.9)$$

in which the digits d_i are selected from the set

$$d=\{0, 1, \ldots, p-1\}. \qquad (5.4.10)$$

This translation may be done at the end of the operation, however also possible is the translation of each partial quotient

$$\sum_{i=1}^{i} a_i r^{-i}$$

in each i-th cycle (i = 1, 2, ..., k).

One can note that the relationships (5.4.1)-(5.4.3) also occur in synchronous restoring division. The characteristic feature of nonrestoring division is precisely the need of translating the quotient from the form (5.4.5) into the form (5.4.9). In contrast to this, in restoring division, the digits of the quotient are selected at once from the set (5.4.10).

Since keeping track of the signs of the dividend and divisor and the form of representation of negative numbers (sign and magnitude form or two's complement code) in the machine presents no fundamental difficulties for nonrestoring division methods,

for simplicity of presentation, let us further assume in this chapter that A and C are negative numbers.*

The first cycle of division may differ from the latter ones by the presence of a preparatory operation for forming the value of $B_0=A/h$, and by the initial translations of the starting values of the dividend and divisor described in the latter sections of the chapter.

The last cycle of division may in a particular way differ from the preceding ones as a function of the requirements made in a given device on the last remainder hB_k. In the simplest case, the remainder B_k is simply not calculated.

Each remainder B_i must not go beyond a certain range of values. This requires first that the remainder term in (5.4.6) be smaller than the permissible error of division, i.e. that when k is fairly large, this term become fairly small, and second, that each remainder B_i can be formed in some real device. Thus, the inequality

$$B_{min} \leqslant B_i \leqslant B_{max}, \qquad (5.4.11)$$

must be adhered to, where B_{max} and B_{min} are the bounds of the range of permissible values of the remainder. Let us note that this requirement must be met:

$$\left.\begin{aligned} B_{max} &\leqslant \xi_m C/(r-1), \\ B_{min} &\geqslant \xi_1 C/(r-1). \end{aligned}\right\} \qquad (5.4.12)$$

In fact, if, for example, $B_0=\xi_m C/(r-1)+|\delta|$, then even when $a_1=\xi_m$ there is derived $B_1=B_0 r-\xi_m C=\xi_m C/(r-1)+|\delta| r$. In selecting further $a_2=a_3=\ldots=a_k=\xi_m$, we derive $B_k=B_0+|\delta| r^k$, , i.e. the remainders B_i will increase continuously, the remainder term in (5.4.3) will equal

$$h\frac{B_k}{r^k C}=h\frac{B_0}{r^k C}+h\frac{|\delta|}{C}$$

and with any large $|\delta|$, the value of $hB_k/(r^kC)$ will not be able to be disregarded. But if as a_i are selected not xi_m, but other digits of the set, then the remainders will increase even faster. Similarly, the remainders B_{i+1}, B_{i+2}, ... begin to continuously increase in absolute value if

$$B_i < \xi_1 C/(r-1).$$

The ranges of permissible values of the magnitudes of the dividend A, the divisor C and the quotient of A/C depend on the number representation form adopted in the machine and in the general case are prescribed by the inequalities

* In reality, representation of negative dividends and divisors in two's complement code has an effect on the assemblies of the arithmetic unit that perform normalization of the divisor, selection of the next digit of a_{i+1}, translation of the quotient and others.

$$A_{min} \leqslant A \leqslant A_{max}, \qquad (5.4.13)$$

$$C_{min} \leqslant C \leqslant C_{max}, \qquad (5.4.14)$$

$$(A/C)_{min} \leqslant A/C \leqslant (A/C)_{max}, \qquad (5.4.15)$$

where A_{min}, A_{max}, C_{min}, C_{max}, $(A/C)_{min}$ and $(A/C)_{max}$ are CONSTANTS.

5.4.2. Classical Method of Division

Let us call classical the conventional method of nonrestoring division [14]. Division by this method is done in the r-nary number system using the following set of permissible values of the digits of the quotient:

$$\xi = \{-(r-1), -(r-2), \ldots, -2, -1, 1, 2, \ldots, r-1\}. \qquad (5.4.16)$$

The boundaries of the range (5.4.11) of permissible values of the remainders for the classical method are the values

$$B_{max} = \xi_m C/(r-1),$$

$$B_{min} = \xi_l C/(r-1), \qquad (5.4.17)$$

i.e. the values of +C and -C. Thus,

$$-C \leqslant B_i \leqslant C. \qquad (5.4.18)$$

The rules for selecting the next digit a_{i+1} for any of the division methods described in section 5.4 can, as will be shown later, be modified somewhat. In particular, for the method using set (5.4.16), one can suggest different rules. The most well known of them has been formulated in work [14, p 42] in the following form: if $B_i \geqslant 0 (B_i < 0)$, then the divisor C is repeatedly subtracted (added) from $B_i r$ so long as the sign of this difference (sum) is opposite to the sign of the value of B_i.* In the process, the digit a_{i+1} in absolute value is equal to the number of subtractions (additions) made and has a plus (minus) sign.

Let us show that if B_i satisfies the inequality (5.4.18), then the cited rule for selecting the digit a_{i+1} provides for fulfillment of it and for the following remainder B_{i+1}. Let us write the value of rB_i in the form $rB_i = (M+\varepsilon)C$, where

* A more precise formulation would have to be the following: if $B_i \geqslant 0$ $(B_i < 0)$, then the divisor C is repeatedly subtracted (added) from $B_i r$ so long as the sign of this difference (sum) is opposite to the sign of the value of B_i or until the quantity of additions (subtractions) made equals r-1.

M is an integer $(-r \leqslant M \leqslant r)$, and ε satisfies the inequality $0 \leqslant \varepsilon < 1$ (when M=r, it should be assumed that $\varepsilon=0$ as required by (5.4.18)). The dependency of a_i and B_{i+1} on M, realized by the given rule for selection of a_{i+1}, is shown in table 5.4.1.

Table 5.4.1.

M	a_{i+1}	B_{i+1}
r	$r-1$	$rC-(r-1)C=C$
$r-1$	$r-1$	$(r-1+\varepsilon)C-$ $-(r-1)C=\varepsilon C$
$0, 1, 2, \ldots, r-2$	$M+1$	$(M+\varepsilon)C-$ $-(M+1)C=-C(1-\varepsilon)$
$-(r-1), -(r-2), \ldots$ $\ldots, -2, -1$	M	$(M+\varepsilon)C-MC=\varepsilon C$
$-r$	$-(r-1)$	$(-r+\varepsilon)C+(r-1)C=$ $=-C(1-\varepsilon)$

It is evident from the table that the new remainder B_{i+1} is within the same range (5.4.18) as B_i (here and from now on it is assumed that if $\overline{rB_i-\rho C=0}$, where ρC is a multiple of the divisor C, then this difference has a plus sign).

Let us assume that the numbers in a machine are represented in a conventional r-nary system using the set of digits

$$d=\{0, 1, 2, \ldots, r-1\}, \qquad (5.4.19)$$

and let us show that one can relatively simply translate the sum $\sum_{t=1}^{l+1} a_t r^{-t}$ into this number system in each (i+1)-th cycle of division.

Let the number $\sum_{t=1}^{l+1} a_t r^{-t}$, which has in the system with the set (5.4.16) the form

$$0, a_1 a_2 \ldots a_l a_{l+1},$$

after translation into the system with the set (5.4.19) look like this

$$0, d_1 d_2 \ldots d_l d'_{l+1},$$

i.e. be equal to $\sum_{t=1}^{l} d_t r^{-t} + d'_{l+1} r^{-l-1}$. Since B_0 is greater than or equal to 0, then a_1 is equal to or greater than 1 and therefore after the first cycle of division, the number 0, a_1 in the number system that uses the set (5.4.16), and this same number 0, d'_1 in the system that uses the set (5.4.19), are written identically, i.e. $d'_1=a_1$. After the second cycle, the numbers 0, a_1a_2 and 0, $d_1d'_2$ are also written identically, if B_1 is greater than or equal to 0, since in the process, a_2

is greater than 0. But if B_1 is less than 0, then a_2 is greater than 0 and $d_1=d'_1-1$ and $d'_2=r+a_2$. Continuing similar discussions, we arrive at the fact that if B_i is greater than or equal to 0, then a_{i+1} is greater than 0, and the translated quotient after (i+1)-th cycle has the form

$$0,\ d_1d_2\ \dots\ d_{i-1}d_id'_{i+1}=0,\ d_1d_2\ \dots\ d_{i-1}d'_ia_{i+1},$$

i.e. in this case $d_i=d'_i,\ d'_{i+1}=a_{i+1}$. But if B_i is less than 0, then a_{i+1} is less than 0 and the translated quotient is written in the form

$$0,\ d_1d_2\ \dots\ d_{i-1}(d'_i-1)(r+a_{i+1}),$$

i.e. $d_i=d'_i-1,\ d'_{i+1}=r+a_{i+1}$. Thus, Thus, to calculate the sum (5.4.4) with simultaneous translation of the quotient, all that is needed in addition to a shift register is a two-digit r-nary subtraction circuit.

Of special interest is the case of r = 2 which corresponds to the widely spread binary nonrestoring division discussed in section 7.8. A feature of this method is the simplicity of selecting the digit a_{i+1}, calculating the new remainder B_{i+1} and translating the quotient:

$$\begin{aligned} &\text{if} \quad B_i\geqslant 0, \quad \text{then} \quad a_{i+1}=1,\ d_i=d'_i=1,\ d'_{i+1}=a_{i+1}=1,\\ &\text{or if} \quad B_i<0, \quad \text{then} \quad a_{i+1}=-1,\ d_i=d'_i-1=0,\ d'_{i+1}=r+a_{i+1}=1. \end{aligned}$$

If we assume that the (i+1)-th cycle is called the aggregate of the following operations:

a) selection of a_{i+1} by the sign of B_i,

b) selection of d_i and writing of d_i as the final digit of the quotient, and

c) calculation of $B_{i+1}=2B_i-a_{i+1}C$, then it should be stated that k+1 cycles in all are performed in the division, but the first and last of them are incomplete: in the first cycle, only B_1 is calculated (a_1 is necessarily equal to 1 and d_0 is not selected), and in the last cycle, only d_k is selected (since a_{k+1} and B_{k+1} are not needed). After completion of the division, the precise relationships between the dividend, the divisor, the derived quotient and the last remainder B_k have the following form:

$$\begin{aligned} &\text{if} \quad B_k\geqslant 0, \quad \text{then} \quad \frac{A}{C}=h\frac{B_0}{C}=h\sum_{i=1}^{k}d_i2^{-i}+h\frac{B_k}{2^kC},\\ &\text{or if} \quad B_k<0, \quad \text{then} \quad \frac{A}{C}=h\frac{B_0}{C}=h\sum_{i=1}^{k}d_i2^{-i}+h\frac{B_k+C}{2^kC}. \end{aligned}$$

In the division examples given in section 7.8, the constant h was equal to 2.

Speeding up conventional nonrestoring division (r = 2), just as speeding up restoring division (see section 5.1), can be achieved by making a device for division in the form of an iterative array.

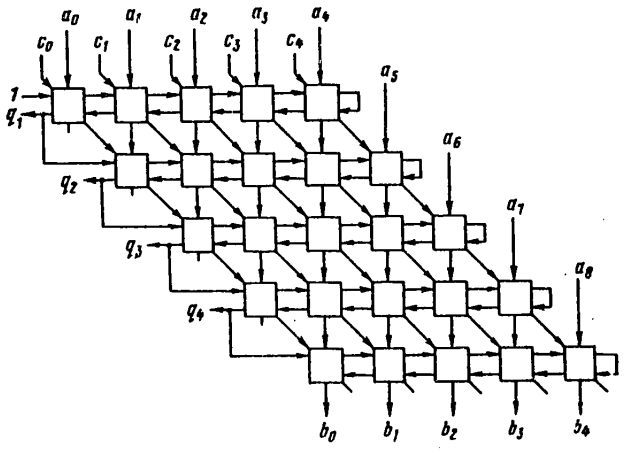


Fig. 5.4.1

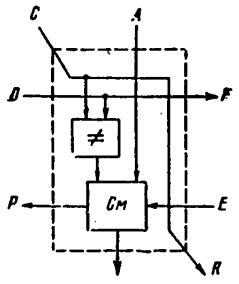


Fig. 5.4.2

[Cm = Sm = Adder]

Shown in fig. 5.4.1 is one of the simplest modifications of such an array, made up of identical elements. Each element (fig. 5.4.2) contains a coincidence-type one-digit binary adder Sm and an EXCLUSIVE OR element that realizes the function $c\bar{q} + \bar{c}q$ and controls the input of the divisor into the adder. The 8-bit dividend $A = a_0, a_1a_2a_3a_4a_5a_6a_7a_8$, the 4-bit divisor $C = c_0, c_1c_2c_3c_4$ and each remainder B_i (in particular, the last remainder $B_5 = b_0, b_1b_2b_3b_4$) are represented in two's complement code. It is assumed that the dividend and divisor are positive normalized fractions (i.e. $a_0 = c_0 = 0$, $a_1 = c_1 = 1$). Since 1/2 is less than or equal to A is less than 1 and 1/2 is less than or equal to C is less than 1, then the quotient $Q = q_1, q_2q_3q_4$ is a positive number lying within the range 1/2 is less than Q is less than 2, i.e. q_1 is an integral part of this number. Each digit of the quotient q_i is the control signal for the next row of the matrix that determines what operation--addition or subtraction--has to be performed in this row. Subtraction, as is evident from figs. 5.4.1 and 5.4.2, is performed by forming the two's complement code of the divisor.

Division is speeded up in this device, just as in the device analyzed earlier (fig. 5.1.2), not by improving the algorithm for the operation, but by using high-speed miniature elements connected by short conductors.

Let us now discuss a modification to the circuit just described that allows achieving an additional increase in the speed of division [4]. The main idea of the modification consists in eliminating the time for carry propagation along each row of the matrix. To this end, first, each remainder B_i in each row

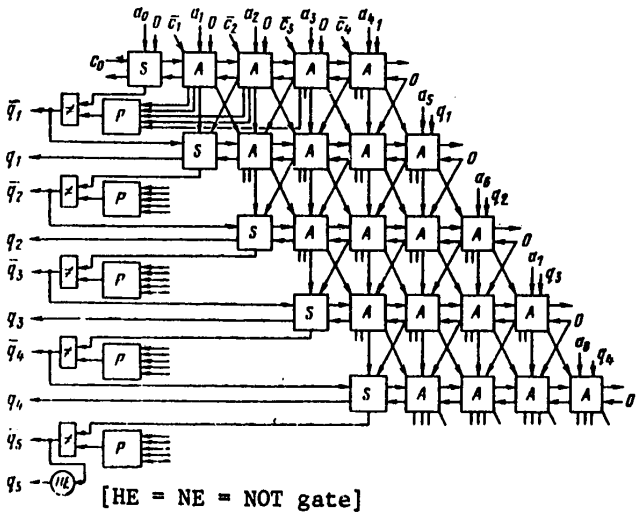


Fig. 5.4.3

will be computed in the form of two numbers: the number S, made up of the step-by-step sums s_j, and the number E, made up of the step-by-step carries e_j (the sum of these two numbers equals the remainder B_i), and second, the carry into the sign bit of the row, the next digit of the quotient and the control signal for the next row (to add or subtract) will be formed by using a carry advance circuit.

Shown in fig. 5.4.3 is a modified iterative array for a 4-bit divisor and an 8-bit dividend. The array uses three types of cells (A, S and P) and minor complementary logic.

The operations performed in the i-th row of the matrix (i = 1, 2, ...) correspond to the operations performed during the next i-th cycle of conventional nonrestoring division: if $q_{i-1} = 1$ (0), then the divisor C is subtracted (added) from the next remainder $2B_{i-1}$. In this case, this procedure is performed the following way:

$$\begin{array}{l} 2S = s_0, s_1 s_2 \ldots s_{n-2} s_{n-1} a_{n+1} \\ 2E = e_0, c_1 c_2 \ldots c_{n-2} 0 q_{i-1} \end{array} \Big\} 2B_{i-1}$$

$$\pm c_0, c_1 c_2 \ldots c_{n-2} c_{n-1} c_n$$

$$\begin{array}{l} S' = s'_0, s'_1 s'_2 \ldots s'_{n-2} s'_{n-1} s'_n \\ E' = e'_0, e'_1 e'_2 \ldots e'_{n-2} e'_{n-1} 0 \end{array} \Big\} B_i .$$

Here n is the number of digits in the divisor.

The divisor is subtracted by adding the two's complement code of the number -C; the two's complement code is derived by inverting all digits c_j and adding one ($q_{i-1} = 1$) to the low-order digit.

The P cells (fig. 5.4.4) contain the logic for advance determination of the carry into the sign (zero) bit of the next remainder. The advance circuit in the i-th row must generate the signal

$$P=D_1+R_1D_2+R_1R_2D_3+\ldots+R_1R_2\ldots R_{n-2}D_{n-1},$$

where $D_j=s'_je'_j$, $R_j=s'_j+e'_j$ are the auxiliary bit functions for origination of the carry and permission for carry propagation. It is obvious that this part of the circuit depends on the length of the operands n; for a large length of n, a multistage carry advance circuit has to be used (shown in fig. 5.4.4 is a cell, with the use of which the advance circuit will be single-stage only when n is less than or equal to 3). The P cell in the i-th row of the matrix generates a carry to the zero bit derived during addition of the numbers

$$\begin{array}{l} s'_1s'_2\ldots s'_{n-1} \\ + \\ e'_1e'_2\ldots e'_{n-1} \end{array}$$

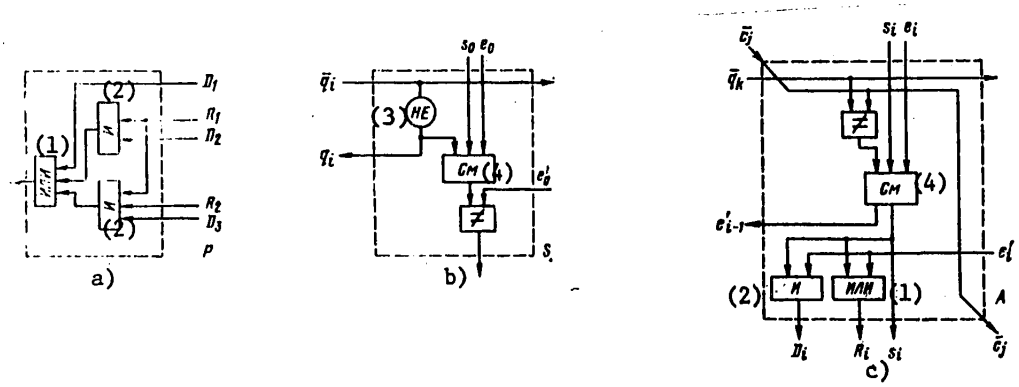


Fig. 5.4.4.

Key:
1. OR
2. AND
3. NOT
4. Adder

The S cell (fig. 5.4.4) of the i-th row sums the digits s_0, e_0 and c_0 (this sum is equal to 0 or 1 as a function of whether the number of one's among the digits s_0, e_0 and c_0 is even or odd); the resulting sum is then added to the carry e'_0 derived during addition with storage of the carries in the adjacent A element (the structure of this element is shown in fig. 5.4.4). Finally, the result is summed with the carry obtained in the P element. The final sum is the sign bit of the remainder B_i, i.e. the value of $\bar{q}_i$. The digit q_i itself is formed by the inverter in the S cell of the next row.

As usual in nonrestoring division, the relationship

$$A/C=Q+B_5/2^kC \ (\text{when} B_5\geqslant 0),$$
$$A/C=Q+(B_5+C)/2^kC \ (\text{when} B_5<0).$$

is observed between the dividend, divisor, quotient and last remainder (in this case, the remainder B_5).

It should be noted that using just one carry advance circuit in each row of a conventional matrix (fig. 5.4.1) would not produce such results; carries would have to be speeded up in all digits of the remainder and a large quantity of additional apparatus would be required for this. It is precisely the combination of forming the remainder in two-row code and obtaining the advance carry into the sign bit that allows achieving considerable speed-up of division with moderate outlays. It can be seen from fig. 5.4.3 and 5.4.4 that total operation time is

$$m(\tau_A+\tau_P+\tau_{\text{EX-OR}}),$$

where m is the number of bits of the quotient; tau_A is the delay time for one A cell; tau_P is the operating time for a P element (here it is assumed that the advance circuit is single-stage); and $\text{tau}_{\text{EX-OR}}$ is the delay time for the EXCLUSIVE OR element generating the digit $\overline{q}_i$.

5.4.3. Graphic-Analytic Method of Division Process Analysis

Before considering the nonrestoring division methods that use sets differing from the classical set (5.4.16), let us introduce into the discussion the graphic representations of the zones xi_j on the plane B_iC [15, 17]. These representations allow us to analyze the connections between the value of r and the structure of the circuit for selecting the digit a_{i+1}.

Let us assume that the bounds of the range (5.4.11) of permissible values of remainders B_i are the values (5.4.17), i.e. that

$$\frac{\xi_1 C}{r-1}\leqslant B_i\leqslant\frac{\xi_m C}{r-1} \ (i=1, 2, \ldots, k). \qquad (5.4.20)$$

This means that with given B_i and C, as a_{i+1} one can select some digit xi_j from the set xi, if

$$\frac{\xi_1 C}{r-1}\leqslant B_i r-\xi_j C\leqslant\frac{\xi_m C}{r-1},$$

i.e. if

$$\frac{\xi_1 C}{r(r-1)}+\frac{\xi_j C}{r}\leqslant B_i\leqslant\frac{\xi_m C}{r(r-1)}+\frac{\xi_j C}{r} \qquad (5.4.21)$$

or

$$\frac{\xi_m C}{r-1}-\frac{(\xi_m-\xi_j)C}{r}-\frac{(\xi_m-\xi_1)C}{r(r-1)}\leqslant B_i\leqslant\frac{\xi_m C}{r-1}-\frac{(\xi_m-\xi_j)C}{r}. \qquad (5.4.22)$$

Corresponding to the inequality (5.4.22) is the fig. 5.4.5 in which on the plane B_iC is shaded the zone of values of the magnitudes of B_i and C with which (5.4.22) is satisfied. Reflected in the figure is a certain special case, for which

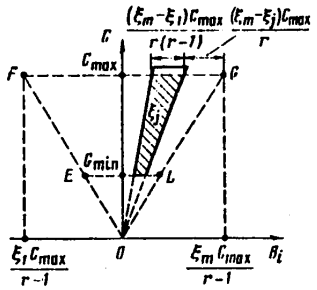


Fig. 5.4.5

$$C_{min}>0,\ \xi_1<0,\ \xi_j<\xi_m,\ \frac{\xi_m C}{r-1}-\frac{(\xi_m-\xi_j)C}{r}-\frac{(\xi_m-\xi_1)C}{r(r-1)}>0.$$

Nevertheless, the following presentation is general in nature.

It is easy to see the correspondence between the fig. 5.4.5 and the terms in the left and right parts of the double inequality (5.4.22);

the term $\frac{(\xi_m-\xi_1)C}{r(r-1)}$ corresponds to the width of the zone of the digit xi_j (it is evident that this width is not a function of the value of xi_j itself), and

the term $\frac{(\xi_m-\xi_j)C}{r}$ corresponds to the horizontal distance between the right edge of the zone and the line GL.

And so, if a point with the coordinates of B_i, C falls in fig. 5.4.5 within the shaded zone that corresponds to some digit xi_j, then this digit can be selected as a_{i+1}. It is evident that for the division process to be realized using some set xi of the digits xi_j, each point within the quadrangle EFGL in fig. 5.4.5 must enter into the zone of at least one of the digits of the set. For this, on the plane B_iC, the zones of the adjacent digits of the set (5.4.7) must overlap or at least be tangent to each other, i.e. the shift between zones of each two adjacent digits xi_j and xi_{j+1} must be less than or equal to the width of the zone. Since the shift between the indicated zones (fig. 5.4.6) is equal to

$\frac{(\xi_{j+1}-\xi_j)C}{r}$, then the requirement $(\xi_{j+1}-\xi_j)C/r\leqslant(\xi_m-\xi_1)C/r(r-1)$ must be met, i.e.

$$\xi_{j+1}-\xi_j\leqslant(\xi_m-\xi_1)/(r-1)\ (j=1, 2,\ldots,m-1). \quad (5.4.23)$$

From (5.4.23), one can derive the requirement of $m\geqslant r$.

In fact,

$$\xi_2-\xi_1<(\xi_m-\xi_1)/(r-1),$$
$$\xi_3-\xi_2<(\xi_m-\xi_1)/(r-1),$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$\xi_m-\xi_{m-1}<(\xi_m-\xi_1)/(r-1).$$

Adding separately the left and right parts of all of these inequalities, we get

$$\xi_m-\xi_1\leqslant(m-1)(\xi_m-\xi_1)/(r-1),$$

hence it follows that

$$m\geqslant r. \qquad (5.4.24)$$

Shown in fig. 5.4.7 are the zones of the digits of the set (5.4.16) used with classical division (values of xi_i are indicated in the upper part of the zones). It is assumed that $C_{min} = 0$. It can be seen from the figure that the zones of the digits overlap in such a way that in the cases of

$$C(r-1)/r<B_i\leqslant C,\ 0<B_i<C/r,$$
$$-C/r<B_i<0,\ -C\leqslant B_i<-C(r-1)/r$$

a certain digit of the set must be selected, and namely the digit (respectively) r-1, 1, -1, -(r-1).

For the cases of

$$B_i=\rho C/r\ (\rho=\pm2, \pm3, \ldots, \pm(r-2)),$$

which correspond to the bounds of the zones, one of three digits can be selected:

$$\pm(\rho-1),\ \pm\rho,\ \pm(\rho+1).$$

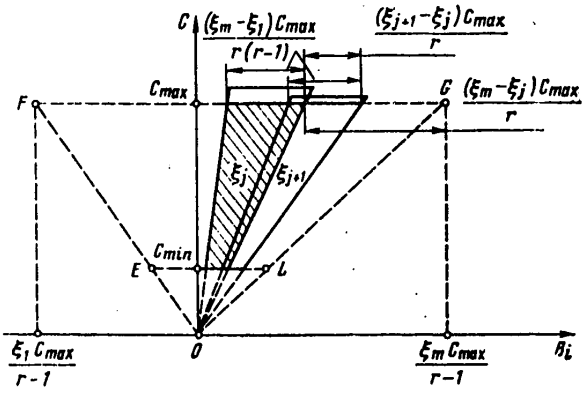


Fig. 5.4.6

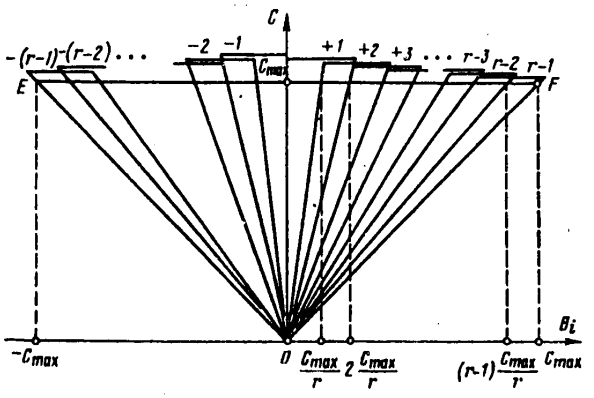


Fig. 5.4.7

In all remaining cases, one of two possible digits can be selected since all corresponding sectors of the plane B_iC are covered by two zones. Just which of these digits is selected is a function of the rule adopted for selecting the digits a_i.

From fig. 5.4.7, we also see that a characteristic feature of set (5.4.16) is that the zones of the two middle digits (+ 1 and -1) do not overlap, but are merely tangent to each other. Therefore, in selecting a_{i+1}, it is necessary to know the sign of the remainder B_i and consequently, the remainder must be computed in the arithmetic unit in the form of one number, i.e. one-row code (the circuit in fig. 5.4.3 is the exception).

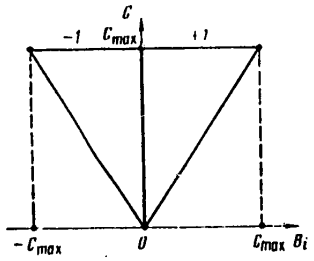


Fig. 5.4.8

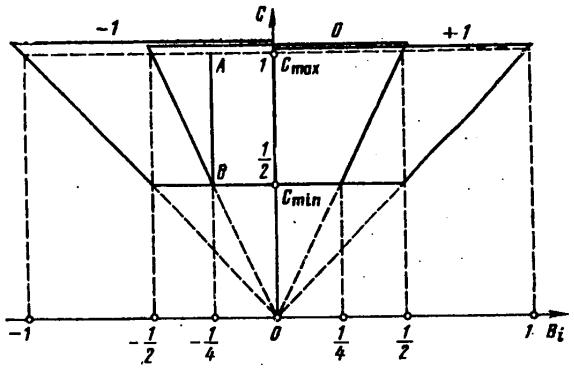


Fig. 5.4.9

Shown in fig. 5.4.8 are the zones for the particular case of the set (5.4.16) that corresponds to conventional nonrestoring binary division (r = 2). In this case, the zones of the digits xi_j do not overlap (fig. 5.4.8).

The presence of overlap of the zones of the digits xi_j on the plane B_iC reflects the redundancy of the set xi which leads to the capability of ambiguous representation of numbers in the number system using the digits xi_j. This capability, as will be shown in the next section, in certain cases allows a substantial increase in the speed of division.

5.4.4. Division Using Symmetrical Set of Integers Including Zero

Until recently, in addition to the classical set (5.4.16), only one form of sets xi:

$$\xi=\{-q, -(q-1), \ldots, -1, 0, 1, \ldots, q-1, q\}, \quad (5.4.25)$$

where q is an integer and satisfies the inequality

$$\frac{r-1}{2} \leqslant q < r-1.$$

was still being used for representation of the digits of the quotient being formed in nonrestoring division methods.

The first of the division methods using sets of this form was the method [18-20] in which

$$r=2, \xi=\{-1, 0, +1\}. \quad (5.4.26)$$

Besides that, a division method [20, 21] was described with the parameters

$$r=4,\ \xi=\{-2,\ -1,\ 0,\ 1,\ 2\}, \qquad (5.4.27)$$

the method [22] with the parameters

$$r=4,\ \xi=\{-3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3\} \qquad (5.4.28)$$

and the method [20] with the parameters

$$r=10,\ \xi=\{-7,\ -6,\ -5,\ -4,\ -3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7\}. \qquad (5.4.29)$$

In general form the set of the form (5.4.25) was described in work [20].

The sets (5.4.25) when q is greater than (r - 1)/2 are redundant. In works [18, 19], there was suggested the method of using this redundancy for set (5.4.26) which consists in beginning to calculate the remainder B_{i+1} by formula (5.4.1) in the form of two-row code in an adder with storage of carries, i.e. in the form of a series of step-by-step sums and a series of step-by-step carries. This sharply reduces the time of one cycle and consequently the time for the whole division.

Shown in fig. 5.4.9 is the arrangement of zones for the set (5.4.26). Redundancy of the set leads to overlapping of the zones of adjacent digits and allows selecting a_{i+1} based on the simple rule of direct analysis of the high-order bits of the number B_i. With a normalized divisor ($1/2 \leqslant C < 1$) and a remainder lying within the range of $-C \leqslant B_i \leqslant C$, for proper selection of the digit a_{i+1}, it is sufficient to know only the four high-order bits of the two-row code of the remainder B_i. If the two indicated four-bit numbers are added, then upon obtaining the four-bit sum B^*_i, one can select the digit a_{i+1} by using a simple table (table 5.4.2).

Table 5.4.2.

B^*_i	a_{i+1}	B^*_i	a_{i+1}
0,XXX 1,11X	1 0	1,0XX 1,X0X	—1 —1

The X's in the table designate the bits, the values of the digits in which can be arbitrary. The validity of the rule reflected in the table is confirmed by fig. 5.4.9 from which it can be seen that since

$$0 < B_i - B^*_i < \frac{1}{4},$$

the selection, for example, of $a_{i+1}=0$ when B^*_i = 1.110 (i.e. -1/4) cannot lead to an incorrect result of division since all remainders

$$-\frac{1}{4}+0 < B_i < -\frac{1}{4}+\frac{1}{4}$$

irrespective of the value of C lie in the zone of the digit $xi_2=0$.

The solid vertical lines in this figure divide the zones of operation of the decoders that select the digit a_{i+1} (in the process it should be assumed that the value of B_i^* is plotted along the axis of the abscissa): left of the line AB is selected $a_{i+1}= -1$, between AB and the axis of the ordinate is selected $a_{i+1}=0$, and to the right of the axis of the ordinate $a_{i+1}=+1$ is selected. This confirms the evident idea that the bounds of the zones of operation of the decoders must pass within the sectors of overlapping of the zones of the corresponding digits xi_j.

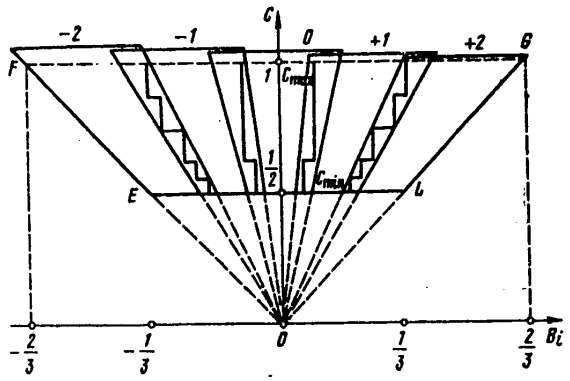


Fig. 5.4.10

Shown in fig. 5.4.10 are the zones of the digits xi_j for another of the sets mentioned--(5.4.27). It is evident from the figure that if a_{i+1} is selected through direct analysis of the remainder and divisor, the decoders of the digits in this case must be substantially more complex. First, it can be seen that in addition to the remainder B_i, the divisor C should also be analyzed since within the sector of overlap of the zones for the digits 1 and 2, a vertical line cannot pass from the line EL to the line FG. Second, it can be seen that a relatively large number of bits of the numbers B_i and C must be analyzed. In fact, let us assume for simplicity that B_i is calculated in the form of one-row code. The decoder of any digit xi_j can be represented as several AND circuits whose outputs are logically summed in an OR circuit. Several bits of the numbers B_i and C are fed to each AND circuit. In the process, for each such AND circuit on the plane B_iC there are one or more rectangles whose sides are parallel to the axes of the coordinates. The rectangles corresponding to all the AND circuits of the decoder of some digit xi_j must fully cover over the plane of B_iC of the sectors, the right and left bounds of which in the process are made up of the "stairs," which must be positioned within the sectors of overlapping of the zone of the digit xi_j with the two adjacent digits of the set xi. Shown in fig. 5.4.10 is one of the possible subdivisions of the plane into zones of operation of the decoders. It is evident that the more "steps" the bounds of its zone contain, the more complex the decoder turns out. Hence it follows that the more complex the decoder, the narrower the sectors of overlapping of the zone of the digit xi_j with the zones xi_{j-1} and xi_{j+1} and the greater the slope of the sectors of overlapping.

For the set (5.4.25) with given r and q, the sectors of overlapping of the zones of the extreme pairs of digits (xi_1 and xi_2, xi_{m-1} and xi_m) have the maximal slopes. The width of the sector of overlapping for the set (5.4.25), as seen from fig. 5.4.6, is equal to

$$\frac{(\xi_m-\xi_1)C}{r(r-1)}-\frac{(\xi_{l+1}-\xi_l)C}{r}=\frac{2q-r+1}{r(r-1)}C.$$

With a given r, the greater q is, the greater is this value. The maximum value (when q = r-1) is C/r. But even with maximal q, the number of steps in the stairs increases rapidly as the values of r become larger. This follows from the diminishment of the width of the sector of overlap equal to C/r; the width of the whole range of remainders when q = r-1 and variation of r remains constant is:

$$\frac{\xi_m C_{\max}}{r-1} - \frac{\xi_1 C_{\max}}{r-1} = C_{\max}.$$

It is easy to show that for other types of sets xi too when the method of selection of a_{i+1} by direct decoding of B_i and C is used, the decoders become more complex (under otherwise equal conditions) as r increases. The quantity of m decoders also, naturally, increases since m is greater than or equal to r.

In work [23], a division method with the parameters $r=2$, $\xi=\{-2, -1, 0, +1, +2\}$ was suggested; in this method, the remainder B_{i+1} is computed in the form of two-row code, however these two rows are not made up by step-by-step sums and carries of a conventional binary adder. The remainder B_{i+1} is formed in the redundant binary number system (r = 2) with permissible values of the digits -1, 0 and +1. The remainder is computed with high speed since there is no carry propagation in doing so. But due to the increase in quantity of permissible digits in the set xi and the complexity of the input circuits of the adder associated with this, this method apparently has no advantages compared to the methods described in [18, 19].

Another method for selecting the digit a_{i+1} was suggested in work [20]. It is based on an analysis of the signs of the approximately computed differences

$$B_i - M_j C \ (j=1, 2, \ldots, m-1),$$

where M_j are some constants. For example, for the set (5.4.27), the differences are computed

$$B_i + \tfrac{3}{8}C,\ B_i + \tfrac{1}{8}C,\ B_i - \tfrac{1}{8}C,\ B_i - \tfrac{3}{8}C. \quad (5.4.30)$$

Given in fig. 5.4.11 on the plane B_iC are the straight lines, the equation of which corresponds to the differences (5.4.30) computed for selecting of the digit of the quotient when the set (5.4.27) is used. The sign of the difference is indicated by the side of the corresponding straight line on which the point with the coordinates B_i, C is located. It is evident that the quantity of constants M_j, actually, must be equal to m-1, each line B_i-M_jC must pass within the sector of overlap of zones of the digits xi_j and xi_{j+1}, and the digits a_{i+1} themselves must in the process be selected by the rule (let us assume that $M_1 < M_2 < \ldots < M_{m-1}$):

$$a_{i+1} = \xi_1, \text{ if } B_i - M_1 C < 0,$$
$$a_{i+1} = \xi_m, \text{ if } B_i - M_{m-1} C \geqslant 0,$$
$$a_{i+1} = \xi_j, \text{ if } \begin{cases} B_i - M_j C < 0, \\ B_i - M_{j-1} C \geqslant 0. \end{cases}$$

It also follows from the figure that the error in computing the difference $B_i - M_jC$ must not exceed the distance between the line $B_i - M_jC = 0$ and the boundary of the sector of overlap of the two zones within which this line must pass.

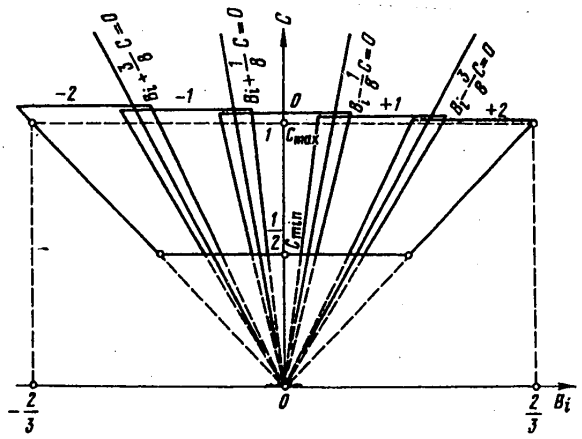


Fig. 5.4.11

Since m is greater than or equal to r, then in general as r increases, the quantity of the differences of $B_i - M_jC$ increases; in the process, the apparatus difficulties and the time needed to select the next r-nary digit of the quotient increase accordingly.

In [24], a method was suggested that allows achieving addition speed-up in nonrestoring division when the set (5.4.26) is used with computation of B_{i+1} in the form of two-row code. It was suggested that a matrix of one-digit adders be used, in each stage of which there is determined the next digit a_{i+1} and the new remainder B_{i+1} computed. The gain in speed in the process can be very great. In fact, when one n-bit adder is used to compute the two-row code of B_{i+1}, the transmission of the preceding remainder from the outputs of the adder to its inputs (with simultaneous shift by one position to the left) with the use of flip-flop registers or delay lines may take no less time than the computation itself of the new remainder. The matrix of adders in such a device may also be used for multiplication.

With small values of r (for example, when r is less than or equal to 4) and two-row representation of the remainder, the matrices of adders can evidently be used also for division with other sets of permissible values of the digits of the quotient.

It follows from relationship (5.4.2) that an increase in the value of r means in essence a proportional increase in the speed of division (assuming the cycle time remains unchanged). For example, one, two, three or four bits of the quotient are determined at once within one cycle when r = 2, 4, 8 or 16 respectively.

The main and essentially sole obstacle to increasing the value of r used in division is the apparatus complexity (or greater inputs of time) for selecting the next digit a_{i+1}. In this and the preceding sections, three main methods of selecting a_{i+1} were described:

1) repeated subtraction (addition) of C from rB_i (for classical division);

2) approximate computation of auxiliary differences of type $B_i - M_jC$; and

3) direct decoding of the values of B_i and C.

Significant increase in time losses or apparatus complexity of the circuit for selecting a_{i+1} as r increase when the first two methods are used are evident, since as r increases, the quantity of digits in sets (5.4.16) and (5.4.25) is increased, and consequently, the number of addition-subtraction procedures increases for both methods. The increase in quantity and complexity of decoders for selecting a_{i+1} with direct analysis of B_i and C has already also been mentioned.

Division methods with large values of r will be covered in the next section of this chapter. In these methods, direct analysis of B_i and C will be performed in selecting a_{i+1}. It should be noted that if the remainder B_i is formed in the form of two-row code, then the "staircase" of the decoder must not approach on the plane B_iC closer to the right boundary of the sector of overlap than the distance corresponding to the maximal difference between the part B_i^* of the remainder being analyzed and the precise value of the remainder B_i. This distance therefore must not exceed the width of the sector of overlap and even must be substantially less than it, since otherwise the quantity of "steps" and decoder complexity associated with it may increase intolerably. With large values of r, decoders of digits become complicated (even with one-row code of the remainder). The presence of two-row code of the remainder owing to the indicated circumstance complicates these decoders even more since it leads to an additional increase of steps in the "staircases." On the other hand, if there are entered in the inputs of the decoders not the digits of two-row code, but the outputs of an additional adder that adds several high-order bits of the two-row code (this is done to simplify the decoders), then the requirement to reduce the maximal difference between B_i^* and B_i leads to the necessity of increasing the length of this adder, which in turn reduces the gain in speed. Therefore, as well as to facilitate the subsequent presentation, let us assume that B_i is computed in the form of one-row code, although the methods to be presented subsequently are also applicable in the case of two-row code of the remainder.

5.4.5. Generalized Method of Nonrestoring Division and Investigation of It

Described here is the synchronous method of division suggested in 1966 [17] whose main feature is the capability of using large values of r while affording high speed in selecting digits of the quotient a_{i+1}. This method has permitted, in particular, development of an arithmetic unit that performs division in the number system with the base of r=16, i.e. that determines within one cycle at once 4 binary positions of the quotient [15].

It was shown in the preceding section that an attempt to seriously increase the value of r leads to considerable complexity of the circuit for selecting the digit a_{i+1}. Let us show that a method of selecting a_{i+1} based on direct decoding of B_i and C can nevertheless permit switching to considerably large values of r if other sets xi are used in the process.

This capability is based on the following idea. Since when the set with equally spaced digits xi_j is used, the decoders of the individual digits xi_j become more complicated as the absolute value of xi_j increases because of the increase in slope of the boundaries of the zone of this digit on the plane B_iC, then we should switch to

those sets in which the overlap of zones of adjacent digits increases as the absolute values of these digits increase. In the process, the increase in slope will to a considerable extent be compensated by the increase in overlap so that the quantity of steps in the "staircases" will not increase rapidly. In other words, we have to switch from sets (5.4.16) and (5.4.25) that contain only equally spaced numbers to sets that do not contain equally spaced numbers. The possibility of this solution is concealed in the inequality (5.4.23) which is the sole limitation imposed on the difference between the adjacent digits of the set xi and which permits with given xi_1 and xi_m including in the set xi any real numbers that satisfy this inequality.

But on the other hand, expanding the overlap sectors leads to an increase in the number of digits in the set xi which increases the quantity of divisor multiples to be generated and complicates the input circuits of the adders that compute B_{i+1}. The optimum (under which the total apparatus of the circuit for selecting a_{i+1}, the circuit for computing B_{i+1} and the circuit for translating the quotient is minimal) depends on the element system used. Therefore, here we can restrict ourselves to just a general consideration of the fact that in realizing this method, the overlaps of the zones of the middle digits of the set xi must be, apparently, less than for the set (5.4.16) since the decoders of these digits are very simple and they can be made somewhat more complicated, having obtained as compensation a reduction in the quantity of digits xi_i, but for the extreme digits of the set, the overlaps must be larger than for the set (5.4.16), although in the process the number of digits in the set xi also increases.

Before moving on to a discussion of the concrete sets xi, let us clear up the limitations imposed on the extreme digits of these sets, i.e. on the values of xi_1 and xi_m.

Let us note that the quotient of B_0/C is found in the range that depends on h and on the range (5.4.15) of the quotient of A/C:

$$\frac{1}{h}\left(\frac{A}{C}\right)_{\min}=\left(\frac{B_0}{C}\right)_{\min}\leqslant\frac{B_0}{C}\leqslant\left(\frac{B_0}{C}\right)_{\max}=\frac{1}{h}\left(\frac{A}{C}\right)_{\max}. \tag{5.4.31}$$

On the other hand, B_0 must satisfy the inequality (5.4.20). After substituting $A/h = B_0$ in (5.4.20), we obtain the requirement

$$\xi_1 C/(r-1)\leqslant A/h\leqslant\xi_m C/(r-1), \tag{5.4.32}$$

which together with (5.4.15) defines the relationship between the constants h, xi_1 and xi_m.

Let us consider first the case when $(A/C)_{\min}=0$ which occurs when the numbers are represented in fixed-point form. It follows from (5.4.32) and (5.4.15) that in the process the values of xi_1 and xi_m must satisfy the requirements

$$\xi_1\leqslant 0, \tag{5.4.33}$$

$$\xi_m\geqslant (A/C)_{\max}(r-1)/h. \tag{5.4.34}$$

Of course, one can assume that xi_1 and xi_m are given (and $\xi_1 \leqslant 0$) and instead of inequality (5.4.34) derive from (5.4.32) and (5.4.15) the inequality that imposes a restriction on the constant h:

$$h \geqslant (A/C)_{max}(r-1)/\xi_m. \qquad (5.4.35)$$

Let us now consider the case when $(A/C)_{min}$ is greater than 0 which corresponds to representation of numbers in floating-point form. It follows from (5.4.32) and (5.4.15) that in the process there can be selected both the value xi_1 is less than or equal to 0 and xi_1 is greater than 0. If xi_1 is less than or equal to 0, then xi_m must in design be selected on the basis of requirement (5.4.34) or, which is the same, h must be selected based on (5.4.35). If xi_1 is greater than 0, then from (5.4.32) and (5.4.15) there must be derived the requirement

$$(A/C)_{max}(r-1)/\xi_m \leqslant h \leqslant \leqslant (A/C)_{min}(r-1)/\xi_1. \qquad (5.4.36)$$

One can assume that (5.4.36) indicates the permissible bounds for h with given xi_1 and xi_m. One can conversely assume that the value of h is given and the parameters xi_1 and xi_m are selected with regard to the requirements

$$\xi_1 \leqslant (A/C)_{min}(r-1)/h, \quad \xi_m \geqslant (A/C)_{max}(r-1)/h. \qquad (5.4.37)$$

It can also be seen from (5.4.36) that when xi_1 is greater than 0, irrespective of the value of h there must be met the requirment

$$\xi_m/\xi_1 \geqslant (A/C)_{max}/(A/C)_{min}. \qquad (5.4.38)$$

Let us illustrate the meaning of inequalities (5.4.37) and (5.4.38) in the example of division of the mantissas of A and C of numbers represented in floating-point form. In this case

$$N \leqslant A \leqslant M, \quad N \leqslant C \leqslant M, \qquad (5.4.39)$$

where M and N are constants, and $N>0$. In the process

$$N/M \leqslant A/C \leqslant M/N, \qquad (5.4.40)$$

i.e. the requirment (5.4.38) in this case has the form

$$\xi_m/\xi_1 \geqslant (M/N)^2.$$

In fig. 5.4.12, the shaded zone PRST shows the permissible values of A and C, and in fig. 5.4.13, zone EFGL shows the permissible values of the quantities C and B_i. When

$$\xi_1 = (N/M)(r-1)/h, \quad \xi_m = (M/N)(r-1)/h,$$

i.e. in the case of the "boundary" meeting of the requirements (5.4.37) and (5.4.38), the square PRST after substitution of $B_0=A/h$ is transformed into the shaded rectangle on the plane B_iC. It is evident from the figures that violation of any of the inequalities (5.4.37) leads to the shaded rectangle in fig. 5.4.13 starting to go beyond the limits of the zone of permissible values.

It should be noted that both approaches taken (with the one, it is assumed that xi_1 and xi_m are given, while with the other, that h is given) differ only methodologically.

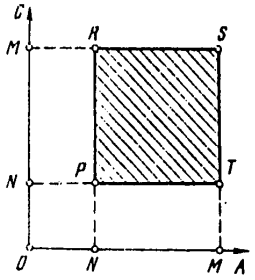


Fig. 5.4.12

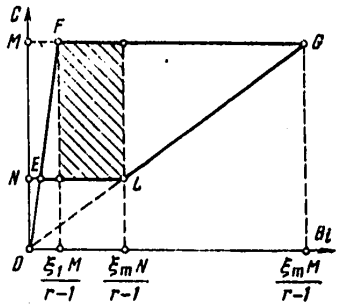


Fig. 5.4.13

In considering the relationship between h and xi_1 and xi_m, it should be noted that the accepted relationships (5.4.1) and (5.4.3) permit ambiguous description of the same method of division. In fact, the process of dividing the dividend A by the divisor C using the set xi of digits xi_j with a certain constant h and remainders B_i lying within the range (5.4.20) described by the relationships (5.4.1) - (5.4.3), (5.4.7), (5.4.8), (5.4.11), (5.4.13) - (5.4.15) and (5.4.20) will as before be described by these relationships if instead of h, xi_j and B_i (including B_0) we will consider the values of h/phi, phi xi_j, phi B_i where phi is any positive constant. Thus, for example, the division method defined by the relationships

$$r=4;\ \xi=\left\{-2,\ -1\frac{1}{8},\ 0,\ 1\frac{1}{8},\ 2\right\};$$

$$0\leqslant A\leqslant 1;\ \ \frac{1}{2}\leqslant C\leqslant 1;\ \ -\frac{2}{3}C\leqslant B_i\leqslant\frac{2}{3}C;\ \ h=4,$$

and the method

$$r=4;\ \xi=\left\{-4,\ -2\frac{1}{4},\ 0,\ 2\frac{1}{4},\ 4\right\};$$

$$0\leqslant A\leqslant 1;\ \ \frac{1}{2}\leqslant C\leqslant 1;\ \ -\frac{4}{3}C\leqslant B_i\leqslant\frac{4}{3}C;\ \ h=2$$

are the same method implementable by the same apparatus.

To achieve uniqueness of description, we shall from now on notate the set xi, as is essentially done for the sets (5.4.16) and (5.4.25), in such a way (i.e. select that constant phi) that, first, all values of xi_j are integers (this is possible since we are dealing with rational numbers of xi_j) and second, that the greatest common divisor of the numbers of xi_j is equal to one. Consequently, the example just cited should be written as follows:

$$r=4;\ \xi=(-16,\ -9,\ 0,\ 9,\ 16);$$

$$0\leqslant A\leqslant 1;\ \ \frac{1}{2}\leqslant C\leqslant 1;\ \ -\frac{16}{3}C\leqslant B_i\leqslant\frac{16}{3}C;\ \ h=\frac{1}{2}. \tag{5.4.41}$$

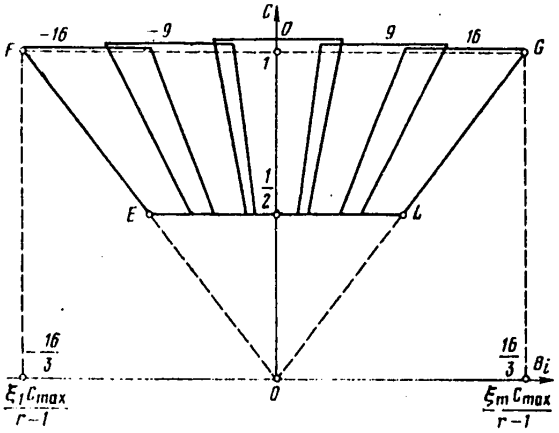


Fig. 5.4.14

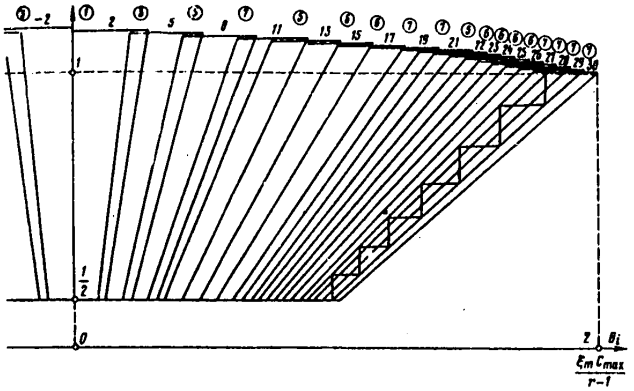


Fig.5.4.15

Let us move now to a consideration of some sets of xi with a variable "step" between the digits of xi_j. The set (5.4.41) is the first of these examples. The arrangement of the zones of digits on the plane B_1C for this set is shown in fig. 5.4.14. It can be seen from the figure that thanks to the variable "step" between the digits of xi_j, the width of the overlap sector between zones 9 and 16 is wider than between the zones 0 and 9. After comparing figs. 5.4.14 and 5.4.10, we see that the overlap of zones 0 and 9 is less than 0 and 1, but in return the overlap of the extreme zones 9 and 16 is larger than 1 and 2.

Another example of a symmetrical set with unequal positioning of the digits is shown in fig. 5.4.15 for the case r = 16:

$$\xi=\{-30, -29, -28, -27, -26, -25, -24, -23,$$
$$-22, -21, -19, -17, -15, -13, -11, -8, -5,$$
$$-2, 2, 5, 8, 11, 13, 15, 17, 19, 21, 22,$$
$$23, 24, 25, 26, 27, 28, 29, 30\}. \qquad (5.4.42)$$

To simplify the figure, the zones of the negative digits are not shown fully.

The number of steps in the ideal "staircases" separating the zones of adjacent digits of the set are indicated in the little circles in this and some subsequent figures. By convention a staircase is said to be ideal when all angles are tangent to the bounds of the overlap sector, i.e. a "staircase" that has the minimal number of steps. These values permit evaluating the complexity of the corresponding decoders that select the digits of the quotient.*

The maximally permitted number of steps in the "staircase" is a function of the element system, the speed required and other circumstances. Let us assume, for instance, that there should be no more than 5 steps in ideal "staircases."

Despite the considerable overlapping of the zones of the extreme digits of the set (5.4.42), the number of steps in the ideal "staircases" is still large--up to 7 steps. It can be seen from fig. 5.4.15 that a further increase in overlap of the zones cannot yield a substantial decrease in the number of steps. In fact, no matter how small, for example, the difference $xi_m - xi_{m-1}$, the "staircase" between the zones of these two digits will still contain at least 6 steps. But even 6 steps, as stipulated above, is too many. Let us examine a method of countering this difficulty.

A more precise formulation of the rule for depicting the "staircase" is desirable for further presentation, i.e. for the boundary between the zones of operation of the decoders of two numbers of the set xi on the plane B_iC. Refinement concerns the points belonging directly to the very sections of the lines forming the "staircase." The rule is explained by fig. 5.4.16. The arrows indicate which zone--right or left--the points situated on the sections of the "staircases" are considered to fall in. Points of the plane that are located outside the "staircase" itself on the right or left of it fall, naturally, in the right or left zones. As indicated earlier, the "staircase" that separates the zones of operation of the decoders of the two adjacent digits xi_j and xi_{j+1} of the set xi must be located within

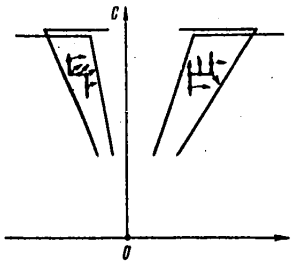


Fig. 5.4.16

* It is noted that after work [17] appeared, the suggestions on using graphic plots of the zones of the digits of the quotient on plane B_iC and on estimating the apparatus outlays for the circuit to select the digits of the quotient by the number of steps in the "staircases" (admittedly, only for sets (5.4.25)) were also made in works [22, 25 and 16].

the sector of overlap of the zones of these two digits, can touch the bounds of the sector of overlap, but must not go beyond them.

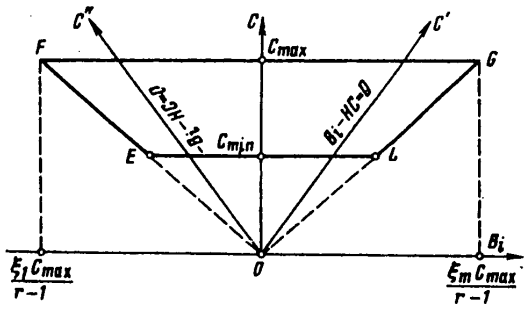


Fig. 5.4.17

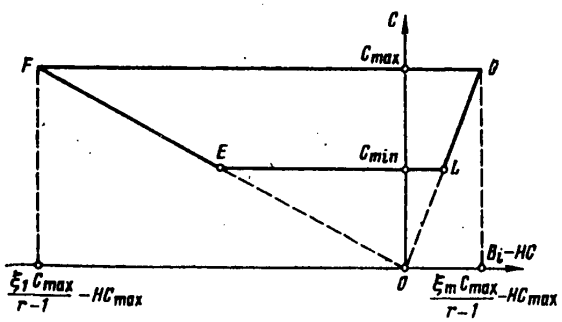


Fig. 5.4.18

From now on, we will follow this same rule in depicting the zones of digits in other coordinates.

Let us now show how the decoders that select the digits of a_i can be simplified. Let on the plane B_iC be shown (fig. 5.4.17) the zone EFGL of permitted remainders and divisors. The bounds of the zones of the individual digits of xi_j are straight lines whose equations are written in this form

where

$$B_i=KC,$$

$$K=\frac{\xi_m}{r(r-1)}+\frac{\xi_l}{r}$$

for the right boundary of the zone of ξ_l and

$$K=\frac{\xi_1}{r(r-1)}+\frac{\xi_l}{r}$$

for the left zone.

Let us show this entire system of lines in a new system of coordinates, in which on the axes are plotted the values of B_i-HC and C (fig. 5.4.18), where H is a positive constant. On the plane B_iC (see fig. 5.4.17), corresponding to the value of H is the line OC' with the equation

$$B_i-HC=0.$$

The quadrangle EFGL "moves" from plane B_iC to plane B_i-HC, C in such a way that the sections FG and EL keep their length. The same occurs from the zone of any digit of xi_j: neither its width (with a given C) nor the width of the overlap sectors changes. But the slopes of the boundaries of all the zones change. In particular, the slopes of those zones which in fig. 5.4.17 were located on the right of OC' or on the left of OC', but closer to OC' than to OC, change in such a way that the new "staircases," drawn for these zones in the new system of coordinates, have fewer

steps than in the old system (it is assumed that each of the segments of the lines that make up the "staircase" is parallel to one of the axes of the coordinates). One can similarly draw in fig. 5.4.17 on the left of OC another line OC" with the equation

$$-B_i - HC = 0,$$

so that in the third system of coordinates obtained, $-B_i - HC$, C, the left zones of the digits of xi_j occupy a favorable position.

As an example, the zones of the extreme digits of the set (5.4.42) shown earlier in fig. 5.4.15 are shown in fig. 5.4.19 in the system of coordinates $B_i - \frac{2}{3}C$, C.

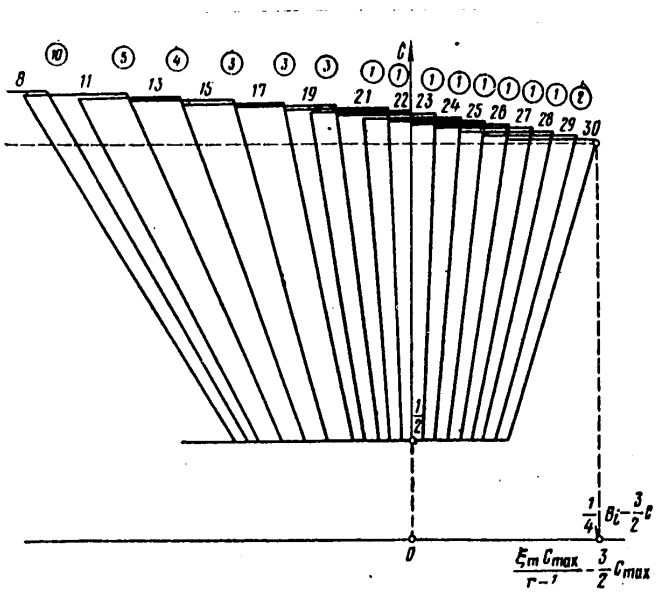


Fig. 5.4.19

It is seen from the figure that the number of steps has been sharply reduced in the ideal "staircases" for the digits of 13, 15, 17, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30.

The zones of the negative digits $-13, -15, -17, -19, -21, -22, -23, -24, -25, -26, -27, -28, -29, -30$ occupy precisely the same position on the plane $-B_i - \frac{3}{2}C$, C.

It follows from what has been said that if in a device for division that uses set (5.4.42), except the remainder B_1, the differences $B_i - \frac{3}{2}C$, $-B_i - \frac{3}{2}C$ are computed and the decoders built to select digits 13, 15, 17, ..., 30 by decoding the high-order positions of the values of $B_i - \frac{3}{2}C$, C, the digits -13, -15, -17, ..., -30 by decoding the values of $-B_i - \frac{3}{2}C$, C and the digits -8, -5, -2, 2, 5 and 8 by conventional decoding of B_i and C, then the decoders of the extreme digits of the set will be significantly simplified.*

Since there are now rather few steps in the "staircases," it is advisable to reduce the number of digits in the set xi, which in a number of cases can yield a substantial gain in apparatus.

* Decoders like the others cannot be built for the digits +11 and -11; these digits can be selected by the following simple rule: $a_{i+1} = +11\ (-11)$, if $B_i \geq 0$ ($B_i < 0$) and no other digit of the set xi is selected in the process.

The set in question, (5.4.42), can be replaced, for example, by the set

$$\xi=\{-30, -27, -24, -21, -18, -16, -14, -12, -11, -9, -7, -5, -2, 2, 5, 7, 9, 11, 12, 14, 16, 18, 21, 24, 27, 30\}, \quad (5.4.43)$$

the arrangement of zones of which on the planes

$$B_iC;\ B_i-\frac{3}{2}C,\ C;\ -B_i-\frac{3}{2}C,\ C$$

is shown in figs. 5.4.20 and 5.4.21; in fig. 5.4.21, two identical figures are imposed on each other: one for the zones of positive digits in coordinates

$$B_i-\frac{3}{2}C,\ C,$$

and the other for the zones of the negative digits in coordinates

$$-B_i-\frac{3}{2}C,\ C.$$

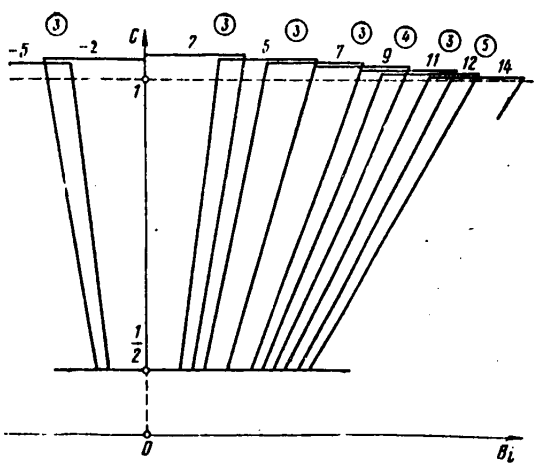


Fig. 5.4.20

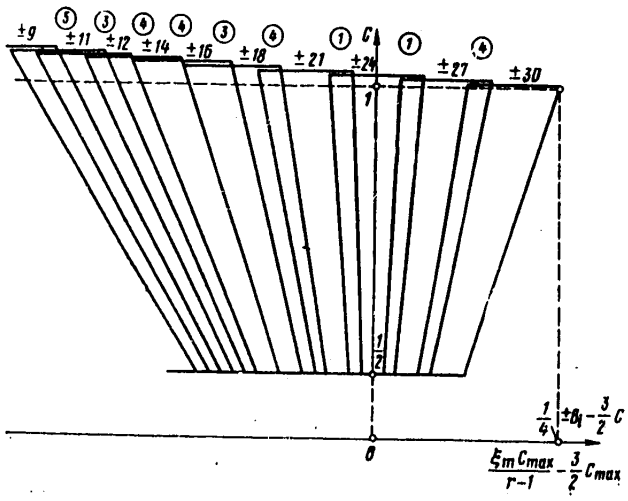


Fig. 5.4.21

The digits -11, -9, -7, -5, -2, 2, 5, 7, 9 and 11 can be selected by direct decoding of the values of B_i and C; the digits 14, 16, 18, 21, 24, 27 and 30 by analysis of

$$B_i-\frac{3}{2}C \quad \text{and C}$$

and the digits -30, -27, -24, -21, -18, -16, and -14 by analysis of

$$-B_i-\frac{3}{2}C \quad \text{and C}$$

It is seen from the figures that in the process, all ideal "staircases" contain no more than four steps. The digits -12 and +12 can be selected by a rule similar to that given earlier for the digits +11 and -11 of the set (5.4.42).

Let us consider the question of the quantity of binary positions in the supplementary adders that compute the difference of the type $\pm B_i$-HC. There is no need for these differences to be computed precisely. In these adders, it is sufficient to process the values represented by several high-order positions of the numbers B_i, HC. However, in the process, the rules for designing the decoders of the digits x_{i_j}

and the rules for drawing the "staircases" within the sectors of overlap of the zones of the adjacent digits of the set must be changed in a suitable way.

Let 2^{l} и 2^{-l} be the weights of the high-order and low-order significant binary positions of the adder that computes the difference of the type $\pm B_i$-HC. Let us assume that the negative numbers -HC enter the inputs of the adder in inverse code and that there is no ring carry circuit in this adder. Let us also stipulate that the positive and negative numbers at the output of the adder are represented in sign-and-magnitude code and complement code (complement with respect to 2^{t+1}). The approximate number $\pm B_i$-HC, obtained at outputs of the adder will be designated in the form $(\pm B_i\text{-HC})^*$ in contrast to the precise number $\pm B_i$-HC.

The values of B_i and HC can be represented in the following form:

$$B_i = K_1 2^{-l} + \beta, \quad HC = K_2 2^{-l} + \gamma,$$

where K_1 and K_2 are integers, K_2 is greater than 0, and beta and gamma satisfy the inequalities

$$0 \leqslant \beta < 2^{l}, \quad 0 \leqslant \gamma < 2^{l}.$$

First let us consider the computation of the difference of B_i-HC. Since this difference is used to select a_{i+1} only when B_i is greater than 0, it can be assumed that the number B_i is positive and represented in sign-and-magnitude code. Let us also assume that the number B_i is in some register B. The high-order digits of the number B_i go from the register B into the adder of B_i-HC also in sign-and-magnitude code. The precise difference of B_i-HC is $B_i - HC = (K_1 - K_2)2^{-l} + (\beta - \gamma)$.

In the adder in the process are added the numbers $K_1 2^{-l}$ and $(2^{l+1} - K_2 2^{-l} - 2^{-l})$ **.

If K_1 is greater than K_2, at the outputs of the adder is obtained the negative number $(B_i - HC)^* = (K_1 - K_2)2^{-l} - 2^{-l}$.

If K_1 is less than or equal to K_2, then obtained at the adder outputs is the number

$$2^{l+1} - (K_2 - K_1)2^{-l} - 2^{-l},$$

which corresponds to representation in the complement code of the negative number

$$(B_i - HC)^* = (K_1 - K_2)2^{-l} - 2^{-l}.$$

In both cases

$$0 < B_i - HC - (B_i - HC)^* = 2^{-l} + (\beta - \gamma) < 2 \cdot 2^{-l}. \qquad (5.4.44)$$

** Since the maximal value of B_i is $\xi_m C_{max}(r-1)$,

the requirement $2^{l+1} > \xi_m C_{max}/(r-1) \geqslant 2^{l}$ must be satisfied.

Consequently, the number B_i-HC lies on the numeric axis to the right of the number $(B_i\text{-HC})^*$ at a distance less than the value of two one's of the low-order position of the adder of B_i-HC.

Let us now go to the adder of $-B_i$-HC. Since this adder is used when B_i is less than 0, let us assume that K_1 is less than 0. The precise difference of $-B_i$-HC is

$$-B_i-HC=(-K_1-K_2)2^{-l}-(\beta+\gamma).$$

Let us also assume that the number B_i is represented in register B in complement code:

$$2^{l+1}-|B_i|=2^{l+1}+K_1 2^{-l}+\beta$$

and that the high-order bits of the register B are inverted prior to transmission to the adder of $-B_i$-HC.

The numbers $(2^{l+1}-2^{l+1}-K_1 2^{-l}-2^{-l})=(-K_1 2^{-l}-2^{-l})$ and $(2^{l+1}-K_2 2^{-l}-2^{-l})$ ** are added in the adder.

If $-K_1 \geqslant K_2+2$, then at the adder outputs there is obtained the negative number $(-B_i-HC)^*=(-K_1-K_2)2^{-l}-2\cdot 2^{-l}$.

If $K_1 \leqslant K_2+1$, the addition result is $2^{l+1}-(K_1+K_2)2^{-l}-2\cdot 2^{-l}$, which corresponds to the representation in complement code of the negative number

$$(-B_i-HC)^*=(-K_1-K_2)2^{-l}-2\cdot 2^{-l}.$$

In both cases

$$0<B_i-HC-(B_i-HC)^*=2\cdot 2^{-l}-(\beta+\gamma)\leqslant 2\cdot 2^{-l},$$

i.e. again the number $-B_i$-HC lies on the numeric axis to the right of the number $(-B_i\text{-HC})^*$ at a distance that does not exceed the value of two one's of the low-order position of the adder of $-B_i$-HC.

Stemming from what has been said and from fig. 5.4.16 is the following rule for constructing "staircases" within the sectors of overlap of the zones of the digits xi_j on the plane $|B_i|$ - HC, C: "Staircases" must not go beyond the left boundary of the overlap zone and must not approach closer to the right boundary than the distance equal to the value of one one of the low-order position of the adder of $\pm B_i$-HC.

** Since the maximum value of $|B_i|$ when B_i is less than 0 is $|\xi_1|C_{max}/(r-1)$, then $2^{l+1}>|\xi_1|C_{max}/(r-1)\geqslant 2^l$.

When necessary (for example, with small overlaps of the zones of the digits of xi_j) one can compute several differences of the type $\pm B_i$-HC with various constants H; each of these differences can be used to construct decoders of a certain group of digits of xi_j.

One fine point should be noted in the interpretation of the graphic depiction of the zones of the digits and the zones of operation of the decoders in the coordinates $|B_i|$ -HC, C: the digit zone boundaries are represented on the assumption that the precise difference of $|B_i|$ -HC is plotted on the axis of the abscissas, and the decoder operation zone boundaries are constructed on the assumption that the value derived at the adder of $|B_i|$ -HC outputs, i.e. the value of $(|B_i| - HC)^*$, is indicated on the axis of the abscissas.

Usually the assemblies of a device for division are substantially simplified with a symmetrical set of xi (i.e. if $xi_j = -xi_{m-j+1}$; $j = 1, 2, \ldots, m/2$ with even m; $j = 1, 2, \ldots, (m-1)/2$ with odd m). In particular, one common decoder as shown in fig. 5.4.22 can be built for the pair of symmetrical digits of xi_j and xi_{m-j+1}.

The signals +B and -B are the output signals of the sign bit of register B, i.e. +B=1 when B_i is greater than or equal to 0, and -B=1 when B_i is less than 0. If the set xi is symmetrical, then the assembly for computing B_{i+1} and the assembly for translating the quotient are also simplified.

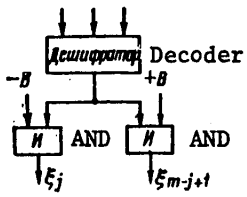


Fig.5.4.22

The use, described in this section, of approximate differences of the type of $\pm B_i$-HC differs essentially from the use of similar differences described in work [20] (see section 5.4.4). There the approximate differences were computed only to determine the signs of these differences and the quantity of differences to be computed increased rapidly as m increased. In our case, the difference of $(\pm B_i\text{-HC})^*$ is needed to construct decoders; therefore, not only its sign is used, but also its magnitude. The quantity of differences is not large: even for m = 20-30, it is sufficient to compute two differences of the type of $\pm B_i$-HC.

Let us discuss briefly the problem of detecting overflow of the quotient that occurs during division of fixed-point numbers.

Up to now in this chapter, we have assumed that the dividend A and divisor C are within the ranges (5.4.13) and (5.4.14), with C_{min} greater than 0. However, during division of fixed-point numbers, the true dividend A^* and divisor C^* are located in the range

$$0 \leqslant A^* \leqslant R,\ 0 \leqslant C^* \leqslant R \qquad (5.4.45)$$

and therefore before beginning the division, A^* and C^* must be translated into A and C. In the process, the ratio of these two values must be preserved.

An overflow signal must be generated if the quotient exceeds the value of R. This may occur both when $C^*=0$ and when $C^* \neq 0$. The first situation causes no apparatus difficulties. Let us consider in more detail the case of $C^* \neq 0$.

The simplest implementation of the procedure for translating A^* and C^* into A and C is the normalization of the divisor C^* and the same shift to the left of the dividend A^*:

$$A=p^fA^*, \qquad C=p^fC^*. \tag{5.4.46}$$

Here p is the base of the number system used for number representation in the machine; f is the number of zeros before the high-order significant digit in the n-position number C^*. It is evident that if $C^* \neq 0$, then

$$0 \leqslant f \leqslant n-1. \tag{5.4.47}$$

Let us assume that A^* and C^* are translated into A and C by using normalization.

An overflow can be detected by various methods. For example, the following method can be used. Since C is less than or equal to R, let us compare A and R^2. If

$$A>R^2, \tag{5.4.48}$$

then let us assume that an overflow occurs. But if A is less than or equal to R^2, this does not mean that there is no overflow.

Let us require that fulfillment of the inequality

$$B_0 \leqslant \xi_m C_{\min}/(r-1). \tag{5.4.49}$$

be automatically provided for when A is less than or equal to R^2. Since in fixed-point division, $(A/C)_{min}=0$, then xi_1 is less than or equal to 0 and therefore fulfillment of the inequality (5.4.49) means that the point with the coordinates B_0, C falls on the plane B_iC in the zone of permissible values. Since we now assume that A is less than or equal to R^2, then, after substituting $B_0=A/h=R^2/h$ in (5.4.49), we obtain the requirement

$$h \geqslant R^2(r-1)/C_{\min}\xi_m. \tag{5.4.50}$$

Shaded in fig. 5.4.23 is the rectangle on the plane AC, in which the point with coordinates A, C may be found, and the rectangle on the plane B_iC, in which the point B_0,C may fall when

$$h=R^2(r-1)/C_{\min}\xi_m.$$

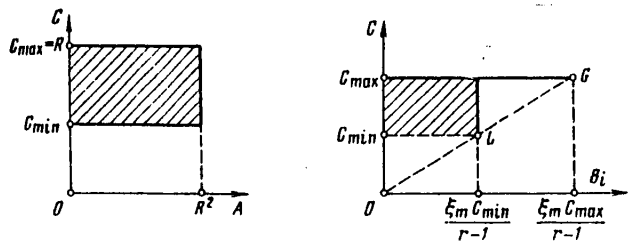


Fig. 5.4.23

Thus, the constant h should be selected with regard to requirement (5.4.50). In the process, division will be correctly performed for all k cycles (i.e. all B_i will be found in the range of permissible values), after which will remain only checking the inequality

$$hQ > R, \qquad (5.4.51)$$

the fulfillment of which should be evaluated as an indication of overflow.

An example of application of this method of overflow detection will be described in section 5.4.6.

5.4.6. Example of Realization of Generalized Method of Nonrestoring Division

Let us briefly consider the structure of a device for division built on the basis of results presented in this chapter.

The division method implemented in the indicated arithmetic unit and the unit itself have the following parameters [15, 17]: p = 2, n = 28, point is fixed in front of the high-order position, r = 16, h = 1, k = 7,

$$\xi = \{-30, -29, -28, -27, -26, -24, -22, -19, -16, -13, -10, -8, -5, -2, 2, 5, 8, 10, 13, 16, 19, 22, 24, 26, 27, 28, 29, 30\}. \qquad (5.4.52)$$

It is evident that $R = 1 - 2^{-28}$ and therefore the initial dividend A^* and divisor C^* are located in the ranges

$$0 \leqslant A^* \leqslant 1-2^{-28}, \quad 0 \leqslant C^* \leqslant 1-2^{-28}$$

(the case of $C^*=0$ is not considered here).

After normalization of the numbers A^* and C^*, the values of A and C are located in the ranges

$$0 \leqslant A \leqslant 2^{+27}-2^{-1}, \quad 2^{-1} \leqslant C \leqslant 1-2^{-28},$$

i.e. $$C_{min} = 2^{-1}, \quad C_{max} = 1-2^{-28}.$$

If A is greater than or equal to 1, i.e. if A is greater than $R^2 + 2^{-28}$, then an overflow signal is generated. It is evident that in the process, A^*/C^* is greater than 1. But if

$$0 \leqslant A \leqslant 1-2^{-28}, \quad 2^{-1} \leqslant C \leqslant 1-2^{-28},$$

then division proceeds correctly for all 7 cycles since the selected value of h provides for fulfillment of the inequality (5.4.49).

At the end of the division operation, a check is made of the inequality (5.4.51) which in this case has the form

$$hQ \geqslant 1.$$

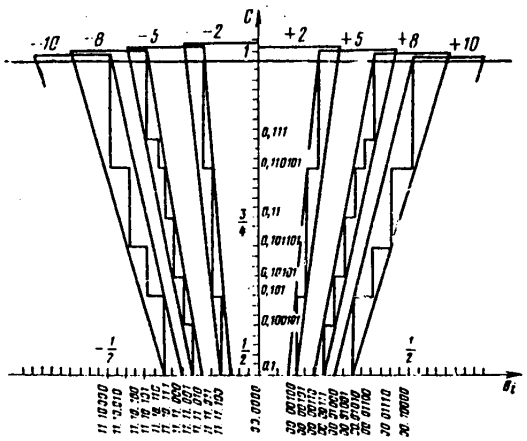


Fig. 5.4.24

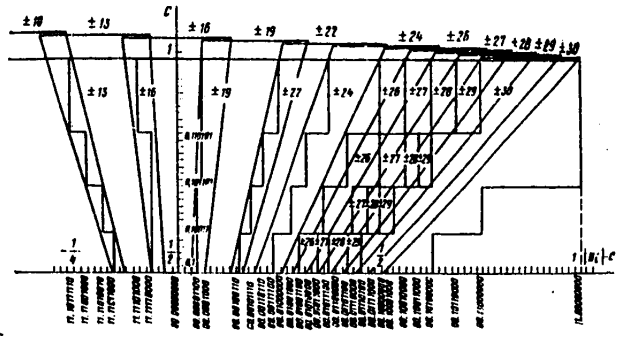


Fig. 5.4.25

If hQ is greater than or equal to 1, then an overflow indication is also generated.

The range (5.4.20) of permissible values of remainders B_i has the form

$$-2C \leqslant B_i \leqslant 2C.$$

The middle digits of the set xi (digits -8, -5, -2, 2, 5 and 8) are selected by using decoders that analyze the high-order positions of the values of B_i and C. The arrangement of the zones of these digits and the zones of operation of the corresponding decoders on the plane B_iC is shown in fig. 5.4.24.

Two supplementary adders that compute differences of the type of $\pm B_i - HC$, namely the differences of

$$B_i - C \quad \text{and} \quad -B_i - C$$

are used in the unit to select the extreme digits of the set xi (the digits -30, -29, -28, -27, -26, -24, -22, -19, -16, -13, 13, 16, 19, 22, 24, 26, 27, 28, 29 and 30).

The arrangement of the zones of the extreme digits of the set (5.4.52) and the zones of operation of the corresponding decoders on the plane $|B_i| - C$, C is shown in fig. 5.4.25. The decoders of the extreme digits are constructed in accordance with the fig. 5.4.22, i.e. combined in pairs.

The double supplementary adder of $|B_i| - C$ contains 9 positions; the weights of the high-order and low-order positions of this adder equal 1 and 2^{-8} respectively.

Therefore, the "staircases" of the decoders in fig. 5.4.25 must not approach (and do not approach) closer to the right boundary of the sector of overlap of the zones than by 2^{-8}.

The digit -10 (+10) is selected automatically when B_i is less than 0 (B_i is greater than or equal to 0) and when no other digit of set xi is selected in the process. The decoders of the digits xi_i, just as all the other assemblies in the arithmetic unit are built with the potential system of elements, which contains a NOT gate and an F shaper (amplifier), implemented by a transistor current switch circuit, emitter followers and diode AND and OR logic circuits. A circuit for selecting the digit xi_{12}=-8 is shown as an example in fig. 5.4.26. One can trace the correspondence of this circuit to the zone shown in fig. 5.4.24. Positions of the numbers are numbered from left to right: 0 (weight 1), 1 (weight 2^{-1}), 2 (weight 2^{-2}) etc. The signal -B is the output of the sign register B and equals 1 when B_i is less than 0. The auxiliary signals D_1, D_2 and D_3 are generated by the individual circuits in such a way that

D_1 = 1 when 1/2 is less than or equal to C is less than 37/64,

D_2 = 1 when 45/64 is less than or equal to C is less than 53/64, and

D_3 = 1 when 53/64 is less than or equal to C is less than 1.

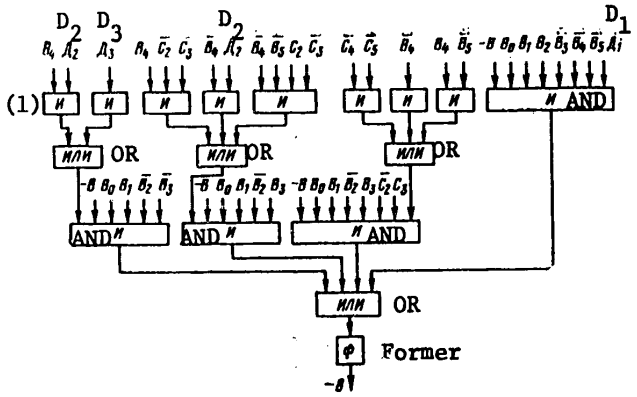


Fig. 5.4.26

Key:

1. [all nine blocks on this level represent] AND circuits

Quotient digit decoders contain from 22 to 80 diodes each. The decoders of +2 and -2 (22 diodes each) and of ±24 (24 diodes) have the simplest circuits; decoders of +5 and -5 (80 diodes each) have the most complex circuits.

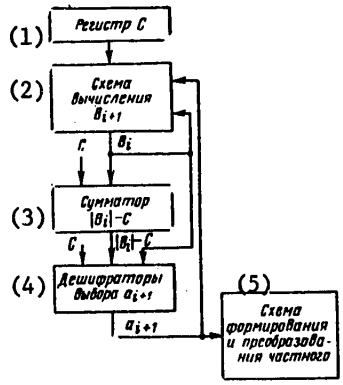


Fig. 5.4.27

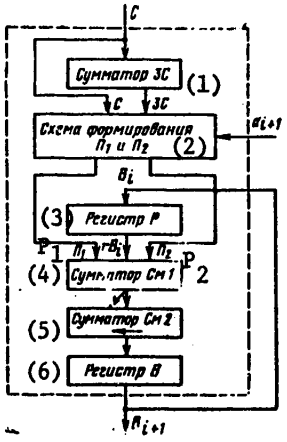


Fig. 5.4.28

Key:

1. Register C
2. Circuit to compute B_{i+1}
3. Adder of $]B_i[- C$
4. Decoders to select a_{i+1}
5. Circuit to form and translate quotient

Key:

1. Adder 3C
2. Circuit to form P_1 and P_2
3. Register P
4. Adder Sm1
5. Adder Sm2
6. Register B

Table 5.4.3.

a_{i+1}	$П_1$ P_1	$П_2$ P_2	a_{i+1}	$П_1$ P_1	$П_2$ P_2
±2	0	∓2C	±22	∓24C	±2C
±5	∓8C	±3C	±24	∓24C	0
±8	∓8C	0	±26	∓24C	∓2C
±10	∓8C	∓2C	±27	∓24C	∓3C
±13	∓16C	±3C	±28	∓24C	∓4C
±16	∓16C	0	±29	∓32C	±3C
±19	∓16C	∓3C	±30	∓32C	±2C

A block diagram of a unit for division is shown in fig. 5.4.27; the circuit that computes the next remainder B_{i+1} that enters the unit is shown in fig. 5.4.28.

The values of the multiples P_1 and P_2 of the divisor C, selected from the set

$$P_1 = (\pm 32C, \pm 24C, \pm 16C, \pm 8C, 0)$$

$$P_2 = (\pm 4C, \pm 3C, \pm 2C, 0)$$

by the rules defined by table 5.4.3 provide for fulfillment of the relationship

$$P_1 + P_2 = -a_{i+1}C.$$

As seen from the table, each of the multiples P_1 or P_2 is formed from the number C shifted by the needed number of positions, or the number 3C generated by the auxiliary adder 3C.

Look-ahead carries have been organized in adder Sm2.

The circuit that translates the quotient Q into binary form is essentially a 7-bit binary adder. The values of xi_j of the set (5.4.52) are 5-bit binary numbers. A direct exhaustive search of all possible cases shows that in addition of two binary numbers

$$\sum_{t=1}^{i} a_t 16^{-t} + a_{t+1}16^{-t-1}$$

the carry run does not exceed three bits, and the borrow run, seven bits. A 3-bit carry is obtained, for example, if a_i = 19 or 27, and a_{i+1} is greater than or equal to 16:

$$\begin{array}{l} \times\ldots\times\times011 \longleftarrow \sum_{t=1}^{i} a_t 16^{-t} \\ + \\ \qquad\qquad 1\times\times\times\times \longleftarrow a_{t+1}16^{-t-1} \\ \hline \times\ldots\times\times100\times\times\times\times \longleftarrow \sum_{t=1}^{i} a_t 16^{-t} \end{array}$$

A 7-bit borrow run occurs if a_i = 5, 13 or 29, a_{i+1} = -16, and a_{i+2} is less than 0:

$$\begin{array}{l} \times\ldots\times101\longleftarrow \sum_{t=1}^{i} a_t 16^{-t} \\ + \\ \qquad -10000\longleftarrow a_{i+1}16^{-i-1} \\ \hline \times\ldots\times1000000 \\ + \\ \qquad\quad -\times\times\times\times\times\longleftarrow a_{i+2}16^{-i-2} \\ \hline \times\ldots\times011111\times\times\times\times\times\longleftarrow \sum_{t=1}^{i+2} a_t 16^{-t} \end{array}$$

The 7-bit adder in the circuit that translates the quotient is built with the use of parallel-parallel logic, i.e. there are no carry or borrow signals in the circuit, and the time to form all seven bits of the sum is the delay time of one active element.

The successful combination of multiplication and division operations in the arithmetic unit developed should be noted. Multiplication is performed in 5 cycles; during each cycle, the product of the multiplicand by 6 successive digits of the multiplier A is added to the running sum of the partial products. In the process,

the same adders Sm1, Sm2 and 3C (fig. 5.4.28), the same multiples P_1 and P_2 of the multiplicand C (in the multiplication operation, another two values of the multiple P_2 equal to $\pm C$, are used), and the same registers B and P are used. The apparatus, installed in an arithmetic unit just for the division operation (circuit to translate the quotient, adders of $\pm B_i - C$ and decoders to select the digit a_{i+1}) and not used in other operations (multiplication, addition, subtraction, shifts and logic operations) makes up only 10 percent of the total unit size. This indicates that under certain conditions, the described division methods can be successfully combined with the well-known methods for performing the other operations without requiring substantial additional outlays for equipment.

It should be noted that at present there are apparently no other machines, in the arithmetic units of which, synchronous methods of nonrestoring division would afford obtaining four or at least three binary digits of the quotient at once in one cycle.

Obtaining more than four bits of the quotient in a cycle can be effected by units that perform division by using iterations (cycles), during each of which, multiplication operations are performed (see section 5.3). In such units, the number of quotient bits obtained can, for example, be doubled after each cycle. However, use of these methods is warranted only in arithmetic units that perform multiplication at very high speed. Conversely, the generalized method of synchronous nonrestoring division permits considerably raising the speed of division in arithmetic units that use relatively inexpensive (in the sense of apparatus outlays) methods for speeding up multiplication and which contain therefore a relatively small amount of hardware.

BIBLIOGRAPHY

For Chapter 1

1. Glushkov, V. M.; Kapitonova, Yu. V. and Letichevskiy, A. A., "Theory of Data Structures and Synchronous Parallel Computations," KIBERNETIKA, No 6, 1976, pp 2-15.
2. Glivenko, Ye. V., "A Principle of Parallel Computations," PROGRAMMIROVANIYE, No 1, 1977, pp 3-9.
3. Kartsev, M. A., "Problems of Structure of Multiprocessor Computing Systems," VOPROSY RADIOELEKTRONIKI. SER. EVT, No 5-6, 1970, pp 3-19.
4. (Enslow), F. G., ed., "Multiprocessor Systems and Parallel Computations," translated from English, Moscow, Mir, 1976, 383 pages.
5. (Baytser), B., "Computer Systems Architecture," translated from English, Moscow, Mir, 1974, Vol 1, 498 pages; Vol 2, 566 pages.
6. Baer, J. L. and Estrin, G. A., "A Priory Scheduling of Bi-logic Graph Models of Computations," INTERNET, 1969, pp 119-127.
7. Baer, J. L. and Estrin, G., "Bounds for Maximum Parallelism in a Bilogic Graph Model of Computation," IEEE TRANS. COMPUT., Vol C-18, No 11, 1969, pp 1012-1014.
8. Martin, D. and Estrin, G., "Models of Computations and Systems Evaluation of Vertex Probabilities in Graph Models of Computations," J. ACM, Vol 14, No 2, 1967, pp 281-300.

9. Golovkin, B. A., "Structure of Probabilistic Model and Analysis of Parallel Computational Processes," IZV. AN SSSR. TEKHNICHESKAYA KIBERNETIKA, No 3, 1973, pp 86-96.

10. Golovkin, B. A., "A Class of Dynamic Multidimensional Models," VOPROSY RADIOELEKTRONIKI. SER. EVT, No 7, 1973, pp 25-30.

11. Yevreinov, E. V. and Kosarev, Yu. G., "Odnorodnyye universal'nyye vychislitel'nyye sistemy vysokoy proizvoditel'nosti" [Homogeneous General-Purpose High-Throughput Computer Systems], Novosibirsk, Nauka, 1966, 308 pages.

12. Pospelov, D. A., "Vvedeniye v teoriyu vychislitel'nykh sistem" [Introduction to Theory of Computer Systems], Moscow, Sovetskoye radio, 1972, 280 pages.

13. Golubev-Novozhilov, Yu. G., "Mnogomashinnyye kompleksy vychislitel'nykh sistem" [Multimachine Complexes of Computer Systems], Moscow, Sovetskoye radio, 1967, 424 pages.

14. Kartsev, M. A., "Paralleling of Iterative Computational Algorithms," VOPROSY RADIOELEKTRONIKI. SER. EVT, No 9, 1971, pp 36-39.

For Chapter 2

1. Kartsev, M. A., "Arkhitektura tsifrovykh vychislitel'nykh mashin" [Digital Computer Architecture], Moscow, Nauka, 1978, 296 pages.

2. [Same as Chap 1, Ref 4]

3. Crane, B. A.; Gilmartin, H. J.; Huttenhoff, J. H.; Rux, R. T. and Shively, R. R., "PEPE Computer Architecture," in "COMPCON-72, 6th Annual IEEE Comput. Soc. Int. Conf., San Francisco, 1972, pp 57-60.

4. Evensen, A. J. and Troy, J. L., "Introduction to the Architecture of a 288-Element PEPE," in "Proc. Sagamore Computer Conf. on Parallel Processing," Springer-Verlag, NY, 1973, pp 162-169.

5. Slotnick, D. L.; Borck, W. C. and McReynolds, R. C., "The Solomon Computer," in "1962 Fall Joint Computer Conf. AFIPS Proc., Vol 22, Washington, 1962, pp 97-108.

6. Barnes, G. H.; Brawn, R. M.; Kato, M.; Kuck, J.; Slotnick, D. L. and Stokes, R. A., "The ILLIAC IV Computer," IEEE TRANS. COMPUT., Vol C-17, No 8, 1968, pp 746-757.

7. Bouknight, W. J.; Dennenberg, S. A.; McIntyre, D. E.; Sameh, A. H. and Slotnick, D. L., "The ILLIAC IV System," PROC. OF IEEE, Vol 60, No 4, 1972, pp 369-388.

8. VOPROSY RADIOELEKTRONIKI. SER. EVT, No 9, 1980.

9. Martin, D. F. and Estrin, G., "Experiments on Models of Computation and Systems," IEEE TRANS. COMPUT., Vol EC-16, No 1, 1967, pp 59-69.

10. Noor, A. K. and Fulton, R. E., "Impact of CDC STAR-100 Computer on Finite Element Systems," J. STRUCT. DIV. PROC. AMER. SOC. CIV. ENG., Vol 101, No 4, 1975, pp 731-750.

11. Chap 1, Ref 3

12. Abel, N. E.; Budnik, P. P.; Kuck, D. J.; Muraoka, Y.; Northoote, R. S. and Wilhelmson, R. B., "TRANQUIL: A Language for an Array Processing Computer," in "Spring Joint Computer Conference," 1969, pp 57-73.

13. Lamport, L., "The Parallel Execution of DO Loops," COMMUNICATIONS OF THE ACM, Vol 17, No 2, 1974, pp 83-93.

14. Lawrie, D. H.; Layman, T:; Baer, D. and Randal, J. M., "Glypnir: A Programming Language for Illiac-IV," CACM, Vol 18, No 3, 1975, pp 157-164.

15. Kartsev, M. A., "Arifmetika tsifrovykh mashin" [Digital Computer Arithmetic], Moscow, Nauka, 1969, 576 pages.

16. Kartsev, M. A., "M-10 Computer," DAN, Vol 245, No 2, 1979, pp 309-312.

For Chapter 3

1. Weinberger, A. and Smith, J. L., "A One-Microsecond Adder Using One-Megacycle Circuitry," IRE TRANS. ON ELECTRON. COMPUT., Vol EC-5, No 2, 1956, pp 65-73.

2. Rowe, W. D., "A New Approach to High-Speed Logic, in "Proc. of the West Joint Computer Conf.," San Francisco-NY, 1959, pp 277-283.

3. RCA Solid State Division, "Arithmetic Arrays Using Standard COS/MOS Building Blocks," 1973, pp 373-378.

4. Zhuk, A. I. and Zhuk, V. I., "Circulation of Signals through Carry Circuit in Adders of Inverse Codes," VOPROSY RADIOELEKTRONIKI. SER. EVT, No 4, 1968, pp 24-33.

5. Potemkin, I. S., "Summatory" [Adders], Ministry of Higher and Secondary Specialized Education of the USSR, Moscow Order of Lenin Power Engineering Institute, Moscow, 1975, 66 pages.

6. Wilkes, M. V., "Automatic Digital Computers," London, Methuen and Co., 1958, 305 pages.

7. Morgan, L. P. and Jarvis, D. B., "Transistor Logic Using Current Switching and Routing techniques and Its Application to a Fast-Carry Propagation Adder," IEE PROC., Vol 106, No 29, Pt. B, 1959, pp 467-468.

8. Lehman, M. and Burla, N., "Skip Techniques for High-Speed Carry Propagation in Binary Arithmetic Units," IRE TRANS. ON ELECTRON. COMPUT., Vol EC-10, No 4, 1961, pp 691-699.

9. Heijn, H. I. and Selman, . . C., "The Philips Computer Pascal," IRE TRANS. ON ELECTRON. COMPUT., Vol EC-10, No 2, 1961, pp 175-183.

10. Majersky, S., "On Determination of Optimal Distributions of Carry Skips in Adders," IEEE TRANS. COMPUT., Vol 16, No 1, 1967, pp 45-58.

11. Nadler, M., "A High-Speed Electronic Arithmetic Unit for Automatic Computing Machines, " ACTA TECHNICA, Vol 1, No 6, 1956, pp 464-478; Vol 2, No 1, 1957, pp 101-102.

12. Pat. No 3230354 (USA), NKI 235-164.

13. Sclansky, J., "Conditional-Sum Addition Logic," IRE TRANS. ON ELECTRON. COMPUT., Vol EC-9, No 2, 1960, pp 226-231.

14. Bedrij, O. J., "Carry-Select Adder," IRE TRANS. ON ELECTRON. COMPUT., Vol EC-11, No 3, 1962, pp 340-346.

15. Belyakov, M. I. and Brik, V. A., "Definition of Optimal Parameters of a Fast Adder," VOPROSY RADIOELEKTRONIKI. SER. EVT, No 9, 1970, pp 62-66.

For Chapter 4

1. [same as 2-15].
2. Khetagurov, Ya. A., "Circuit for Reducing Multiplication Time," in "Vychislitel'naya tekhnika" [Computer Technology], Moscow, Atomizdat, 1960, pp 48-53.
3. Kartsev, M. A., "Arifmeticheskiye ustroystva elektronnykh tsifrovykh mashin" [Arithmetic Units for Electronic Digital Computers], Moscow, Fizmatgiz, 1958, 158 pages.
4. Brik, V. A. and Lushpin, L. I., "Structure of LSI Simultaneous Multipliers," VOPROSY RADIOELEKTRONIKI. SER. EVT, No 11, 1974, pp 41-52.
5. Dean, K. J., "Versatile Multiplier Arrays," ELECTRONIC LETTERS, Vol 4, No 16, 1968, pp 333-334.
6. Hoffman, J. C.; Lacaze, B. and Csillag, P., "Multiplier parralele a circuits logiques iteratifs," ELECTRONIC LETTERS, Vol 4, No 9, 1968, p 178.
7. Bandyopadhyay, S.; Basu, S. and Choudhury, A. K., "An Iterative Array for Multiplication of Signed Binary Numbers," IEEE TRANS. COMPUT., Vol 21, No 8, 1972, pp 921-922.
8. Khetagurov, Ya. A.; Popov, Yu. A. and Lyubentsov, V. M., "An Array for Multiplication," in "Voprosy vychislitel'noy matematiki i vychislitel'noy tekhniki" [Problems of Computational Mathematics and Computer Technology], Moscow, Mashgiz, 1963, pp 157-165.
9. Brik, V. A., "On the Speed of Pyramids of Adders in Multipliers for Digital Computers," VOPROSY RADIOELEKTRONIKI. SER. EVT, No 5-6, 1970, pp 79-87.
10. Brik, V. A.; Borisov, Yu. M. and Tanetov, G. I., "Pyramid of Adders for the Arithmetic Unit of a Digital Computer," in "Elementy i ustroystva upravlyayushchikh mashin" [Elements and Devices for Control Computers], Moscow, 1966, pp 159-166.
11. Majithia, J. C. and Kitai, R., "An Iterative Array for Multiplication of Signed Binary Numbers," IEEE TRANS. COMPUT., Vol 20, No 2, 1971, pp 214-216.
12. Deegan, I. D., "Cellular Multiplier for Signed Binary Numbers," ELECTRONIC LETTERS, Vol 7, No 151, 1971, pp 436-437.
13. Kingsbury, N. G., "High-Speed Binary Multiplier," ELECTRONIC LETTERS, Vol 7, No 10, 1971, pp 277-278.
14. Springer, J. and Alfke, P., "Parallel Multiplier Gets Boost from IC Iterative Logic," ELECTRONICS, Vol 43, No 21, 1970, pp 89-93.
15. Habibi, A. and Wintz, P. A., "Fast Multipliers," IEEE TRANS. COMPUT., Vol 19, No 2, 1970, pp 153-157.
16. Stylianos, P., "A 40-ns 17-bit by 17-bit Array Multiplier," IEEE TRANS. COMPUT., Vol 20, No 4, 1971, pp 442-447.
17. Khrapchenko, V. M., "Methods of Speeding Up Arithmetic Operations Based on Translation of Multirow Code," VOPROSY RADIOELEKTRONIKI. SER. VII EVT, No 8, 1965, pp 121-144.
18. Wallace, C. S., "A Suggestion for a Fast Multiplier," IEEE TRANS. COMPUT., Vol EC-13, No 1, 1964, pp 14-17.

19. Patent 126668 (USSR), published in Bulletin of Inventions, No 5, 1960.

20. Jacobsohn, D., "A Suggestion for a Fast Multiplier," IEEE TRANS. COMPUT., Vol EC-13, No 6, 1964, pp 754-755.

21. Dadda, L., "Some Schemes for Parallel Multipliers," ALTA FREQUENZA, Vol 34, No 5, 1965, pp 349-356.

22. Kartsev, M. A., "A Device for Fast Multiplication and Division," VOPROSY RADIOELEKTRONIKI. SER. XII OBSHCHETEKHNICHESKAYA, No 12, 1959, pp 43-61.

23. Gex, A., "Multiplier-Divider Cellular Array," ELECTRONIC LETTERS, Vol 7, No 15, 1971, pp 442-444.

24. Deegan, I., "Concise Cellular Array for Multiplication and Division," ELECTRONIC LETTERS, Vol 7, No 23, 1971, pp 702-704.

25. Hemel, A., "Making Small ROM's Do Math Quickly, Cheaply and Easily," ELECTRONICS, Vol 43, No 10, 1970, pp 104-111.

26. Johnson, N., "Improved Binary Multiplication System," ELECTRONIC LETTERS, Vol 9, No 1, 1973, pp 6-7.

27. Patent 3670956 (USA), MKI G06f7/52.

28. Lamdan, T. and Aspinal, D., "Some Aspects of the Design of a Simultaneous Multiplier," in "Proc. IFIP Congr., New York City, 1965," Vol 2, Washington, D.C., Spartan Books; London, Macmillan and Co., 1966, pp 440-441.

29. Moroz, I. G., "Some Questions of Simultaneous Multiplication," in "Vychislitel'naya matematika i tekhnika" [Computer Mathematics and Technology], Kiev, AN UkSSR, 1962, pp 85-94.

30. Hennie, F. C., "Iterative Arrays of Logical Circuits," New York, MIT Press; London, Wiley, 1961, 242 pages.

31. Cappa, M. and Hamacher, V.C., "An Augmented Iterative Array for High-Speed Binary Division," IEEE TRANS. COMPUT., Vol 22, No 2, 1973, pp 172-175.

32. Guild, H. H., "Some Cellular Logic Arrays for Non-Restoring Binary Division," THE RADIO AND ELECTRONIC ENGINEER, Vol 39, No 6, 1970, pp 345-348.

33. Majithia, J. C., "Nonrestoring Binary Division Using a Cellular Array," ELECTRONIC LETTERS, Vol 6, No 10, 1970, pp 303-304.

34. Dean, K. J., "Binary Division Using a Data-Dependent Iterative Array," ELECTRONIC LETTERS, Vol 4, No 14, 1968, pp 283-284.

35. Dean, K. J., "Cellular Logic Array for Extracting Square Roots," ELECTRONIC LETTERS, Vol 4, No 15, 1968, pp 314-315.

36. Devries, R. C. and Chao, M. H., "Fully Iterative Array for Extracting Square Roots," ELECTRONIC LETTERS, Vol 6, No 8, 1970, pp 255-256.

37. Dean, K. J., "Logical Circuits for Use in Iterative Arrays," ELECTRONIC LETTERS, Vol 4, No 5, 1968, pp 81-82.

38. Dean, K.J., "Cellular Logical Array for Obtaining the Square of a Binary Number," ELECTRONIC LETTERS, Vol 5, No 16, 1969, pp 370-371.

39. Anderson, S. F.; Earle, J.G.; Goldschmidt, R.E. and Powers, D. M., "The IBM System/360, Model 91. Floating-Point Execution Unit," IBM J. RES. AND DEVELOPMENT, Vol 11, No 1, 1967, pp 34-53.

40. (Buchholz), V., ed., "Design of High-Speed Systems. Complex (Stretch)," translated from English, edited by A. I. Kitov, Moscow, Mir, 1965, 348 pages.

41. Brik, V. A. and Lengnik, T. M., "Analysis of 'Decay' Processes in Array Multipliers," VOPROSY RADIOELEKTRONIKI. SER. EVT, No 9, 1971, pp 65-69.

42. Khrapchenko, V. M., "A Method for Translating Multirow Code to One-Row," DOKLADY AN SSSR, Vol 148, No 2, 1963, pp 296-299.

43. Ofman, Yu., "Algorithmic Complexity of Discrete Functions," DOKLADY AN SSSR, Vol 145, No 1, 1962, pp 48-51.

44. Borisov, Yu. M., "Apparatus Methods of Second Order for Speeding Up Multiplication," in "Vychislitel'naya tekhnika, algoritmy i sistemy upravleniya" [Computer Technology, Algorithms and Control Systems], Transactions of Conference of INEUM [Institute of Electronic Control Machines], July 1966, Moscow, INEUM, 1967, pp 5-19.

45. Brik, V. A. and Lushpin, L. I., "Translation of Multirow Code into Two-Row by Using One-Type Assemblies," VOPROSY RADIOELECTRONIKI. SER. EVT, No 7, 1973, pp 94-116.

46. Brik, V. A., "Some Common Properties of Multilayer Arrays," VOPROSY RADIOELEKTRONIKI. SER. EVT, No 6, 1978, pp 61-69.

47. Stenzel, W. J.; Kubitz, W. J. and Garcia, G. H., "A Compact High-Speed Parallel Multiplication Scheme," IEEE TRANS. COMPUT., Vol 26, No 10, 1977, pp 948-957.

For Chapter 5

1. Patent 3293418 (USA) NKI 235-156.

2. Akushskiy, I. Ya., "Mnogoregistrovyye skhemy vypolneniya arifmeticheskikh operatsiy: Voprosy teorii matematicheskikh mashin. Sbornik pervyy" [Multiregister Circuits for Performing Arithmetic Operations: Problems of Theory of of Mathematical Machines. First Collection], ed. by Yu. Ya. Bazilevskiy, Moscow, Fizmatgiz, 1958, pp 192-218.

3. [same as 4-34].

4. [same as 4-31].

5. Stefanelly, R. A., "A Suggestion for a High-Speed Parallel Binary Divider," IEEE TRANS. COMPUT., Vol C-21, No 1, 1972, pp 42-55.

6. [same as 4-18].

7. [same as 4-17].

8. Flynn, M. J., "Very High-Speed Computing System," PROC. OF THE IEEE, Vol 54, No 12, 1966, pp 1901-1909.

9. Ahmad, M., "Iterative Schemes for High-Speed Division," COMPUT. J., Vol 15, No 4, 1972, pp 333-336.

10. Shaham, Z. and Riesel, Z., "A Note on Division Algorithms Based on Multiplication," IEEE TRANS. COMPUT., Vol C-21, No 5, 1972, pp 513-514.

11. [same as 4-39].

12. Brik, V. A.; Gavrilin, V. A.; Zhuk, V. I.; Zlatnikov, V. M.; Kislinskiy, V. A.; Lengnik, T. M.; Lushpin, L. I. and Petrova, G. N., "Multiprocessor Arithmetic Unit," VOPROSY RADIOELECTRONIKI. SER. EVT, No 5, 1972, pp 56-67.

13. Kartsev, M. A., "Arifmetika tsifrovykh mashin" [Digital Computer Arithmetic], Moscow, Nauka, 1959, 576 pages.

14. (Berks, A.; Goldstein, G. and Neyman, G.), "Preliminary Consideration of Logic Design of an Electronic Computing Device," KIBERNETICHESKIY SBORNIK, No 9, Moscow, Mir, No 9, 1964, pp 7-67.

15. Brik, V. A.; Gavrilin, V. A.; Zhuk, V. I.; Zakharov, V. G.; Lushpin, L. I. and Petrova, G. N., "High-Speed Arithmetic Unit," VOPROSY RADIOELEKTRONIKI. SER. EVT, No 5-6, 1970, pp 97-108.

16. Atkins, D. E., "The Analysis and Design of a Class of Quotient Digit Selectors," 5th Annual IEEE Int. Comput. Soc. Conf., Boston, Mass., 1971; Conf. dig., New York, 1971, pp 201-202.

17. Brik, V. A., "A Method of Nonrestoring Division," Digital Computer Technology and programming, ed. by A. I. Kitov, Moscow, Sov. radio, No 5, 1969, pp 72-84.

18. Nadler, M., "A High-Speed Electronic Arithmetic Unit for Automatic Computing Machines, ACTA TECHNICA, Vol 1, No 6, 1956, pp 464-478; Vol 2, No 1, 1957, pp 101-102.

19. Kartsev, M. A., "A Method of Binary Division," in "Materialy nauchno-tekhn. konferentsii 'Novyye razrabotki v oblasti vychislitel'noy matematiki i vychislitel'noy tekhniki'" [Materials from the Scientific and Technical Conference on "New Developments in Computational Mathematics and Computer Technology"], Kiev, VTs AN UkSSR, 1960, pp 409-418.

20. Robertson, J. E., "A New Class of Digital Division Methods," IRE TRANS. ON ELECTRON. COMPUT., Vol EC-7, No 3, 1958, pp 218-222.

21. Brailovskiy, V. L. and Glukhov, Yu. N., "Methods of Nonrestoring Division," VOPROSY RADIOELEKTRONIKI. SER. XII, OBSHCHETEKHNICHESKAYA, No 13, 1961, pp 40-59.

22. Patent 3621218 (USA).

23. Soceneantu, A. and Toma, C. I., "Cellular Logic Array for Redundant Binary Division," PROC. IEE, Vol 119, No 10, 1972, pp 1452-1456.

24. [same as 4-22].

25. Atkins, D. E., "Higher-Radix Division Using Estimates of the Divisor and Partial Remainders," IEEE TRANS. COMPUT., Vol C-17, No 10, 1968, pp 925-934.

For Chapter 6

1. Gramolin, V. V., "Enhancing the Regularity of Computer Units with the Use of Special Arithmetics," Transactions of the Moscow Institute of Rail Transport Engineers, No 395, "Problems of Theory of Computing Systems," 1971, pp 73-83.

2. Karasik, V.M., "Some Problems of Design of LSI Digital Computers," VOPROSY RADIOELEKTRONIKI. SER. EVT, No 1, 1971, pp 3-13.

3. Zeydenberg, V. K.; Matveyenko, N. A. and Tarovatova, Ye. V., "Survey of Foreign Computer Technology as of 1971," Moscow, Institute of Precision Mechanics and Computer Technology, 1971, 316 pages.

4. Cloyd, E. M., "MOS/LSI Throughout," in "IEEE Int. Convent. Dig.," New York, NY, 1971," New York, NY, 1971, pp 74-75.

5. NEW SCIENTIST, 3/V, No 844, 1973.

6. Al'tman, "Functional Approach to Design of LSI Circuits," ELEKTRONIKA, No 17, 1975, p 58.

7. Tarovatova, Ye. V.; Golovnya, Ye. V. and Matveyenko, N. A., "Foreign Computer Technology in 1975," ed. by V. K. Zeydenberg, Moscow, Institute of Precision Mechanics and Computer Technology imeni S. A. Lebedev, USSR Academy of Sciences, 1976, 275 pages.

8. Prangishvili, I. V.; Abramova, N. A.; Babicheva, Ye. V. and Ignatushchenko, V. V., "Mikroelektronika i odnorodnyye struktury dlya postroyeniya logicheskikh i vychislitel'nykh ustroystv" [Microelectronics and Homogeneous Structures for Building Arithmetic and Logic Units], Moscow, Nauka, 1967, 228 pages.

9. Dukhnich, Ye. I. and Makarevich, O. B., "LSI Circuits and Software for Digital Integrating Structures," in "Vychislitel'nyye sistemy i sredy" [Computer Systems and Environments], Taranrog, 1972, pp 433-435.

10. Zeydenberg, V. K.; Matveyenko, N. A. and Tarovatova, Ye. V., "Obzor zarubezhnoy vychislitel'noy tekhniki po sostoyaniyu na 1972" [Survey of Foreign Computer technology as of 1972], Moscow, Institute of Precision Mechanics and Computer Technology, 1972, 355 pages.

11. Arluckle, W. L. and Mattson, R. C., "Macromodularity: A Design Concept to End Computer Generation Gaps," COMPUTER DESIGN, Vol 9, No 8, 1970, pp 69-79.

12. Russo, R. L., "On the Tradeoff between Logic Performance and Circuit-to-Pin Ratio for LSI," IEEE TRANS. COMPUT., Vol 21, No 2, 1972, pp 147-153.

13. [Same as 3-3].

14. Belkov, M. S.; Bratal'skiy, Ye. A. and Lushpin, L. I., "Method of Building a Superparallel Adder," VOPROSY RADIOELEKTRONIKI. SER. EVT, No 7, 1975, pp 65-71.

15. [Same as 4-13].

16. [Same as 4-14].

17. [Same as 4-15].

18. [Same as 4-16].

19. Brik, V. A. and Lushpin, L. I., "Method of Translating Multirow Code by Using Binary Adders," VOPROSY RADIOELEKTRONIKI. SER. EVT, No 6, 1977, pp 119-124.

20. [Same as 4-45].

21. [Same as 4-25].

22. [Same as 4-26].

23. [Same as 4-27].

24. "The TTL Data Book Supplement to CC-401 for Design Engineers," Deutschland, Texas Instruments, 1973.

25. Belkov, M. S.; Bratal'skiy, Ye. A.; Brik, V. A.; Lushpin, L. I. and Pakhunov, V. N., "Development of LSI System for Building Digital Computer Assemblies," VOPROSY RADIOELEKTRONIKI. SER. EVT, No 7, 1975, pp 72-80.

26. Patent 3866030 (USA). MKI G06f7/39.

For Chapter 7

1. McKeeman, W. M., "Representation Error for Real Numbers in Binary Computer Arithmetic," IEEE TRANS. COMPUT., Vol 16, No 5, 1967, pp 682-683.
2. Hamming, R. W., "Numerical Methods for Scientists and Engineers," New York, McGraw-Hill Book Co., 1962, 411 pages.
3. Potemkin, I. S., "Postroyenie funktsional'nykh uzlov na potentsial'nykh sistemakh elementov" [Building Functional Assemblies with Potential Systems of Elements], Moscow, MEI [Moscow Power Engineering Institute], 1974, 126 pages.
4. [Same as 2-15].
5. Brik, V. A., "Precision of Performance of Floating-Point Addition-Subtraction Operations," VOPROSY RADIOELEKTRONIKI. SER. EVT, No 5, 1972, pp 74-81.
6. Khrapchenko, V. M., "Estimating Error of Binary Multiplication," in "Problems of Cybernetics," ed. by A. A. Lyapunov, Moscow, Fizmatgiz, No 10, 1963, pp 165-177.

Subject Index

Table of Contents

8545
CSO: 8144/1812

- END -