MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

UKRAINIAN STATE UNIVERSITY OF SCIENCE AND TECHNOLOGIES

Department «Electronic computing machines»

In the author's edition

O. Yehorov, V. Dziuba, P. Ivin

COMPUTER ARCHITECTURE

Methodical recommendations for performing practical works

Electronic analogue
of the printed edition

Dnipro
2023

УДК 004.2
У 39

Authors:
O. Yehorov, V. Dziuba, P. Ivin

Recommended by the MCF «CTS» (protocol No. 604 dated 01.05.2023).

Registered SMD USUST (No. 604 from 01.05.2023)

Yehorov O.
Computer architecture [Text]: methodical recommendations for performing practical works / authors: O. Yehorov, V. Dziuba, P. Ivin; Ukrainian state university of science and technologies. – Dnipro : USUST, 2023. – 37 p.

Methodical recommendations are intended for students of the 3rd year of the specialty 123 "Computer engineering" to prepare for practical work in the discipline "Computer architecture".

Illustrations 23. Tables 7. Bibliography: 9 titles.

CONTENT

INTRODUCTION

The purpose of the practical work is to acquire practical skills in the use of acquired knowledge in the development of structures, algorithms and microprograms for the operation of the main devices of the arithmetic logic unit (ALU) of the central processor, as well as to consolidate the main theoretical provisions of the course. As a result of practical work, students should get a distinct notion of the interaction of the main components and blocks of the ALU processor in the process of performing arithmetic operations and learn to use the apparatus, methods and tools of computer design. This goal is best met by students' independent implementation of the development of the structure, algorithmic description, schemes.

The implementation of the above practical works helps:

– improving the understanding of the processes of performing arithmetic operations of ALU;

– obtaining practical skills in the development of devices, algorithms and microprograms for the operation of ALU;

– formation of the ability to document and present the results of the development of ALU elements through the compilation of report documentation and the presentation of practical works.

Program learning results, the achievement of which the publication contributes to:

– to know and understand the scientific provisions underlying the functioning of computer tools, systems and networks (ПРН1);

– to be able to apply knowledge to identify, formulate and solve technical problems of the specialty, using methods that are most suitable for achieving the set goals (ПРН6);

– to be able to apply knowledge of technical characteristics, design features, purpose and rules of operation of software and technical means of computer systems and networks to solve technical problems of the specialty (ПРН9).

PRACTICAL WORK № 1

GETTING TO KNOW THE LANGUAGE OF DESCRIPTION OF LOGIC SCHEMES JOLS-M. EXECUTION OF OPERATIONS IN THE JOLS-M LANGUAGE

Objective:

1. Get acquainted with the syntax of the JOLS-M language.
2. To study the peculiarities of representing variables and performing arithmetic and logical operations on numbers.
3. To acquire the skills of forming the structure of the device and writing a microprogram in the JOLS-M language.

1. Theoretical information

Several integrated development environments have been developed for creating microprograms in the JOLS-M. JOLS-M allows you to perform:

- description of the structure of the elements of the simulated device;
- input of a microprogram that simulates the operation of the device;
- execution of a microprogram;
- step-by-step microprogram debugging;
- print the structure of elements, the listing of microprogram and simulation results.

A microprogram on JOLS-M is a finite number of lines with one microinstruction in each line. The core of the JOLS-M includes the following microinstructions:

– INPUT
– OPERATION
– PRINT
– GOTO
– IF
– END
– Comment

The listed microinstructions allow using of labels.

Microinstruction INPUT

The **INPUT** microinstruction is intended for changing the contents of any register or memory element during the execution of the microprogram. At the same time, the current contents of the specified register or memory element in binary code are displayed in a special input window. The execution of the microprogram is interrupted for inputting a new value or changing the old one. After entering the value, the execution of the microprogram continues

```
{ mmm } INPUT [operand1],
```

where **{ mmm }** – label (1...999);

operand1 – any of the following variants:
– pXX – a register with number XX (0..39);
– пXXX – a memory cell with an address XXX (0..254);
– p(pXX) – a register with the address stored in the register XX;
– п(пXXX) – a memory cell with an address stored in an address cell XXX;
– p(пXXX) – a register whose address is stored in a memory cell;
– п(pXX) – a memory cell whose address is stored in a register.

Microinstruction OPERATION

The **OPERATION** microinstruction is designed to perform various operations on registers, memory elements and constants, such as: addition, subtraction, logical AND, logical OR, assignment, modulo addition 2, cyclic shift, logical shift, addition with cyclic carry, logical NOT.

With the help of a special sign (~), the inverse value of this operand will participate in the operation, but the operand itself will not change. A register or memory element, a separate bit or a group of bits, constants in the binary, decimal or hexadecimal numbering system, as well as registers or memory elements that are addressed by an indirect address can participate in operations.

```
{ mmm } OPERATION [operand1] [function] [operand2]{~},
```

where **{ mmm }** – label (1..999);
operand1 – any of the following variants:
– pXX;
– pXX(NN);
– pXX(NN:nn);
– пXXX;
– пXXX(NN);
– пXX(NN:nn);
– p(pXX);
– p(пXXX);
– п(пXXX);
– п(pXX),
where XX – register number (0..39);
XXX – memory element number (0..254);
NN – high bit number (inclusive);
nn – low bit number (inclusive);
operand2 – can take the same values as operand1 and any of the following:
– dddd – decimal constant ($-2^{31}..2^{31}$);
– $hhh – a hexadecimal constant (0..FFFFFFFFh);
– #bbb – a binary constant (0.. 1111 1111 1111 1111 1111 1111 1111 1111);
function – any of the following operations on operands:
– = assignment;
– + addition;

– - subtraction;
– / logical addition (OR);
– & logical multiplication (AND);
– @ modulo addition 2;
– >> logical shift to the right;
– << logical shift to the left;
– >] cyclic shift to the right;
– [< cyclic shift to the left;
– ++ addition with cyclic transfer;
– ~ sign of inversion of the second operand.

Microinstruction PRINT

The **PRINT** microinstruction enables the output of the values of registers, memory cells, as well as text messages during the execution of the microprogram.

а) **{mmm} PRINT [operand1]..[operand7],**

where **{mmm}** – label (1..999);
operand1 .. operand7 – any of the following variants:
– pXX;
– пXXX;
– p(pXX);
– p(пXXX);
– п(пXXX);
– п(pXX),
where XX – register numbers (0..39);
XXX – memory element number (0..254);

б) **{mmm} PRINT «any text»**

where **{mmm} –** label (1..999).

Microinstruction GOTO

The **GOTO** microinstruction is intended for the unconditional transfer of control of the microinstruction with a label. At the same time, the microprogram continues to be executed from the line with the specified label. The label must be in the range of values 1..999. The value of a label, register or indirectly addressed memory element can be specified directly in the **GOTO** microinstruction.

{mmm} GOTO [expression],

where **{mmm}** – label (1..999);
expression – any of the following variants:
– ddd – transition label (1..999);

– pXX;
– пXXX;
– p(pXX);
– p(пXXX);
– п(пXXX);
– п(pXX),
where XX – register number (0..39);
XXX – memory element number (0..254).

Microinstruction IF

The **IF** microinstruction performs relational operations on operands such as =, <>, <=, >= . If the relation is true, then the command specified after **IF** microinstruction in the same line is executed next, otherwise the next line will be executed.

{mmm}IF[operand1][relation][operand2][microinstruction],

where **{mmm}** – (1..999);
operand1 – any of the following variants:
– pXX;
– pXX(NN);
– pXX(NN:nn);
– пXXX;
– пXXX(NN);
– пXX(NN:nn);
– p(pXX);
– p(пXXX);
– п(пXXX);
– п(pXX),
where XX – register number (0..39);
XXX – memory element number (0..254);
NN – number of the most significant bit (inclusive);
nn – number of the least significant bit (inclusive)
(ATTENTION !!! NN => nn);
operand2 – can take the same values as **operand1**, as well as any of the following:
– dddd – decimal constant (-1*2^31 .. 2*2^31);
– $hhh – hexadecimal constant (0.. FFFFFFFFh);
– #bbb – binary constant (0 .. 1111 1111 1111 1111 1111 1111 1111 1111);
relation – any of the following relationship microinstructions:
– = is equal to;
– < less;
– > more;
– <= less than or equal to;

– >= greater than or equal to;
– < > unequal;

microinstruction – any of the following:
– OPERATION;
– GOTO;
– INPUT;
– PRINT;
– END;
– Comment.

ATTENTION!!! The **IF** microinstruction is not allowed to be used as a statement.

Microinstruction END

The **END** microinstruction stops the execution of the microprogram.

```
{mmm} END,
```

where **{mmm}** – (1..999).

Comment

The comment microinstruction (single quote `) is intended to indicate comments in the firmware text, which dramatically increases the visibility of the firmware. Ignored during firmware execution.

2. Problem definition (Goal)

Develop a microprogram to demonstrate the execution of microinstructions and operations.

3. The order of work

1. Download JOLS-M.
2. Describe the arbitrary structure of the device using registers and memory cells.
3. Compile a microprogram that includes:
   - entering information into several registers and memory cells;
   - transfer of information from the register to the memory cell;
   - transfer of information from the memory cell to the register;
   - execution of logical operations "AND" and "OR", addition by modulo 2 over the contents of two registers, the contents of a register and a memory cell;
   - logical and cyclic shift to the left and right of the content of registers by 1 and 4 digits;
   - a logical shift to the left and to the right, in which the number of shifts is indicated in the register;

– logical shift to the left and right by 1 and 4 digits of the content of part of the register digits (to perform a shift of approximately half of the register digits);
– transfer information from one register to another with a skew of 2 digits to the left and 3 digits to the right;

4. Perform simulation on a computer.

The result of each operation, including the output numbers, should be printed.

The output data for the performance of each operation should be selected in such a way that the characteristic features of the performance of this operation are visible.

4. The content of the report

The report should contain brief theoretical information, printed firmware and simulation results.

5. Control questions and tasks

1. Is there an option in the JOLS-M language to perform actions on part of the contents of a register?
2. How is information equalized when it is transferred from a register with a smaller digit to a register with a larger digit: by lower or higher digits?
3. How to transfer information from a register to a register without performing a shift micro-operation (skewed transfer)?
4. How to swap the contents of two registers without using an intermediate one?
5. What are the main operations used in JOLS-M?
6. What is the difference between cyclic and logical shift?
7. What microinstructions are used in the JOLS-M language?

PRACTICAL WORK № 2

STUDY OF ADDITION METHODS FOR THE NUMBERS WITH FIXED POIN

Objective:

1. Get acquainted with the rules and methods of adding numbers with a fixed point.
2. Get the skills to use reverse and additional codes.

1. Theoretical information

1.1. Return and additional codes. To perform an addition operation with different signs, one complement code (1cc) or two complement code (2cc) are used. The leading digit of the number is used as a sign. The number 0 in the sign digit indicates the positive sign of the number ("+"), and the number 1 - the negative ("-").

If it is necessary to represent a negative number when using the 1cc, then "1" is written in the sign digit, and all significant digits are inverted.

If the number needs to be represented in 2cc, then, in addition to the inversion of significant digits, "1" must be added to the significant digits. Figure 2.1 shows the number «-5» in 1cc and 2cc.

$DC: -5_{10}=1.0101_2$ $1cc: -5_{10}=1.1010_2$ $2cc: -5_{10}=1.1011_2$

Figure 2.1. Representation of the number -5 in direct code, 1cc and 2cc

In 1cc, if after performing the operation of adding numbers, the bit grid overflows (there is a transfer from the higher digit), then one should be added to the result (addition with a cyclic transfer). An example of the addition operation using the reverse code is shown in Figure 2.2.

$DC: -5_{10}=1.0101_2$ $-4_{10}=1.0100_2$ $(-5)+(-4)=?$
$1cc: -5_{10}=1.1010_2$ $-410=1.1011_2$

```
      1.1010
      1.1011
  1← 1.0101
        ____1
  1cc: 1.0110
  DC: 1.1001
```

Figure 2.2. The process of performing an addition operation using a return code

1.2. Modified 1cc (M1cc) and modified 2cc (M2cc) codes are used to determine the fact of overflow. Another additional bit is used for this. If the number is negative, the additional and sign digits are equal to 1 ("11"), if the number is positive - 0 ("00"). At the same time, if after performing the addition operation, the additional and sign digits are not equal to each other ("10" or "01"), then an overflow has occurred.

Code conversion and all other actions are performed similarly to ordinary 1cc and 2cc. For example, let's add numbers «5» and «13» using M2cc and a 4-bit grid. The progress of the addition operation is shown in Figure 2.3.

```
00.0101  (5)
00.1101  (13)
01.0010  (overflow!)
```

Figure 2.3. An example of an addition operation with overflow

Figure 2.4 shows the structure of a device for adding two n-bit numbers with a fixed point.

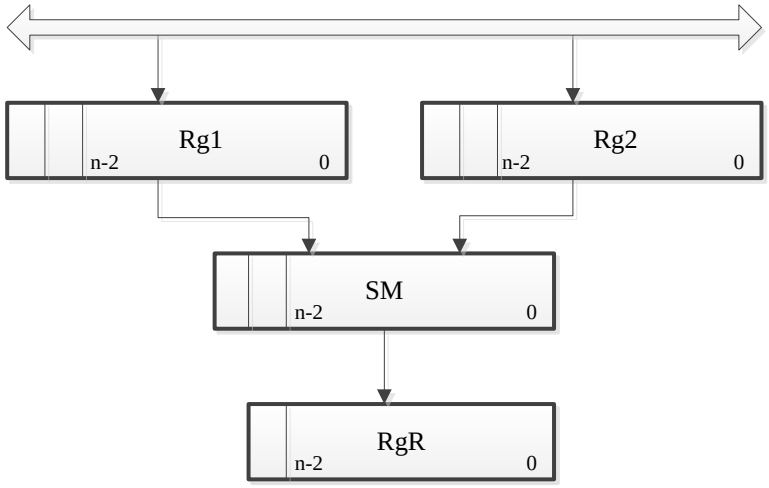

Figure 2.4. The structure of the device for adding two n-bit numbers with a fixed point

Registers Rg1 and Rg2 contain two output operands, the most significant bit of which is a sign. Addition of numbers takes place on the adder SM. It should be taken into account that the adder is accumulative. The result of the operation is entered in the RgR result register.

Figure 2.5 shows the general algorithm for adding two n-bit numbers with a fixed point.

2. Problem definition (Goal)

Develop the structure, algorithm and microprogram of the device to perform the operation of adding two n-bit numbers with a fixed point according to the option (Table 2.1).

3. The order of work (according to the variant)

1. Make the structure of the addition device.
2. Develop an addition algorithm.

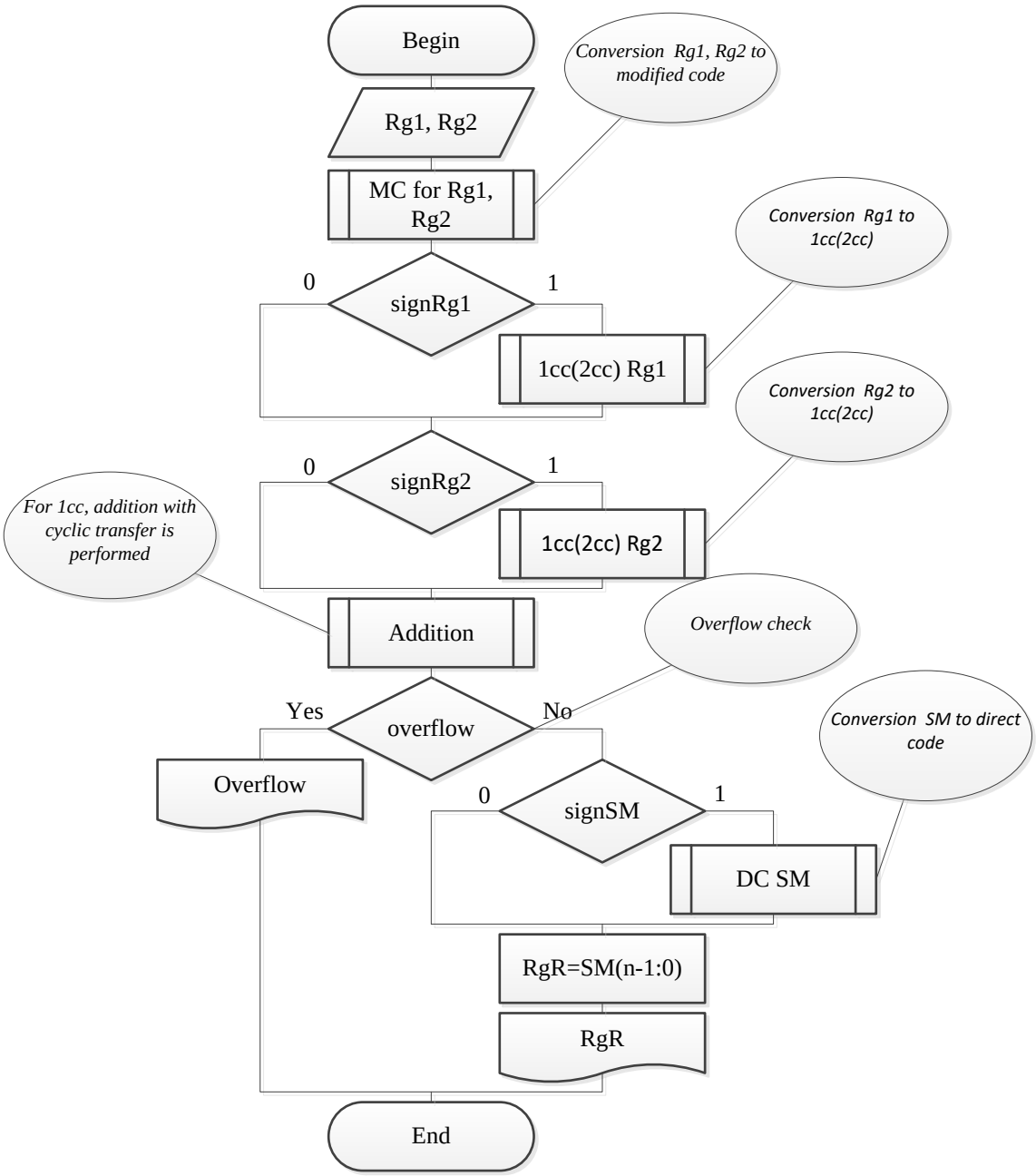

Figure 2.5. A general algorithm for adding two n-bit fixed-point numbers

3. Download JOLS-M.

2. Describe the structure of the addition device.

3. Compile a microprogram in which to foresee the execution of the operation of adding numbers with different signs, with the simulation of the application of the reverse or additional modified code.

Table 2.1 – Task options for performing the addition operation

Variant number	Bit depth	Code	Variant number	Bit depth	Code
1	4	1cc	14	10	2cc
2	4	2cc	15	11	1cc
3	5	1cc	16	11	2cc
4	5	2cc	17	12	1cc
5	6	1cc	18	12	2cc
6	6	2cc	19	13	1cc
7	7	1cc	20	13	2cc
8	7	2cc	21	14	1cc
9	8	1cc	22	14	2cc
10	8	2cc	23	15	1cc
11	9	1cc	24	15	2cc
12	9	2cc	25	16	1cc
13	10	1cc	26	16	2cc

ATTENTION!!!

Turn the operands in such a way that you take one additional, one visible result, that re-ordering.

The result of skin examination, including the number of days to be followed, should be shown to others.

The specific data for the specific skin operation should be chosen in such a way that it was possible to see the characteristic features of this operation.

4. The content of the report

The report should contain brief theoretical information, developed structure, algorithm (block diagram) and firmware for performing the operation of adding two operands, as well as simulation results for numbers with different signs.

5. Control questions and tasks

1. How are reverse and additional codes formed?
2. What is the difference between adding to the 1cc and adding to the 2cc?
3. What are reverse and additional codes used for?
4. What is bit grid overflow? How is overflow determined?
5. Why does a zero value of the overflow bit of a negative number indicate an overflow?
6. How is overflow determined if there is no overflow bit in the adder?

PRACTICAL WORK № 3

STUDY OF MULTIPLICATION METHODS FOR THE NUMBERS WITH FIXED POIN

Objective:

1. Get acquainted with the existing methods of multiplying numbers with a fixed point.
2. To simulate the operation of multiplication methods on a computer according to the variant of the task.

1. Theoretical information

A partial product is a product multiplied by one or more digits of the factor. When analyzing one digit of the multiplier, the value of the partial product can be equal to zero (when the digit of the multiplier is "0") or the value of the product (when the digit of the multiplier is "1").

There are several ways to multiply binary numbers.

The beginning of multiplication from the lower digits of the multiplier and the shift of the sum of partial products (Figure 3.1, method №1). In each cycle of multiplication, the lowest digit of the multiplier is analyzed. If it is equal to 1, then the value of the multiplicand is added to the content of the sum of partial products (SPP).

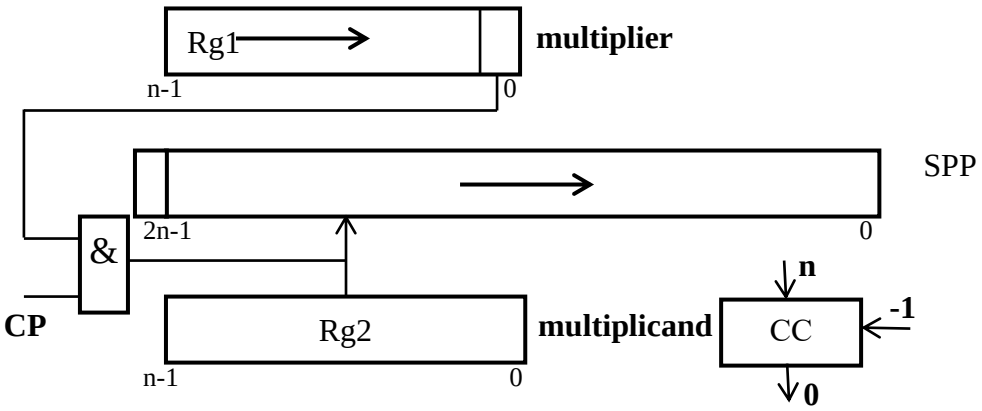

Figure 3.1. The structure of the multiplication device from the lower digits of the multiplier and the shift of the sum of partial products

Moreover, the product is added to the larger part of the sum of partial products. Next, the sum of the partial products and the multiplier are shifted to the right by one digit.

ATTENTION!!! *An additional digit is required for the adder.*

Figure 3.2 shows an example of performing a multiplication operation in this way.

Multiplier: $5_{10}=0101_2$
Multiplicand: $13_{10}=1101_2$

```
SPP                  Multiplier
0 00000000            0101 R1
0 1101
0 11010000 RП1
0 01101000            0010 R1
0 0000
0 01101000 R1
0 00110100            0001 R1
0 1101
1 00000100 R1
0 10000010            0000 R1
0 0000
0 10000010 R1
0 01000001
```

Figure 3.2. An example of multiplication from the lower digits of the multiplier and the shift of the sum of partial products

The beginning of multiplication from the lower digits of the multiplier and the shift of the multiplicand (Figure 3.3, method №2).

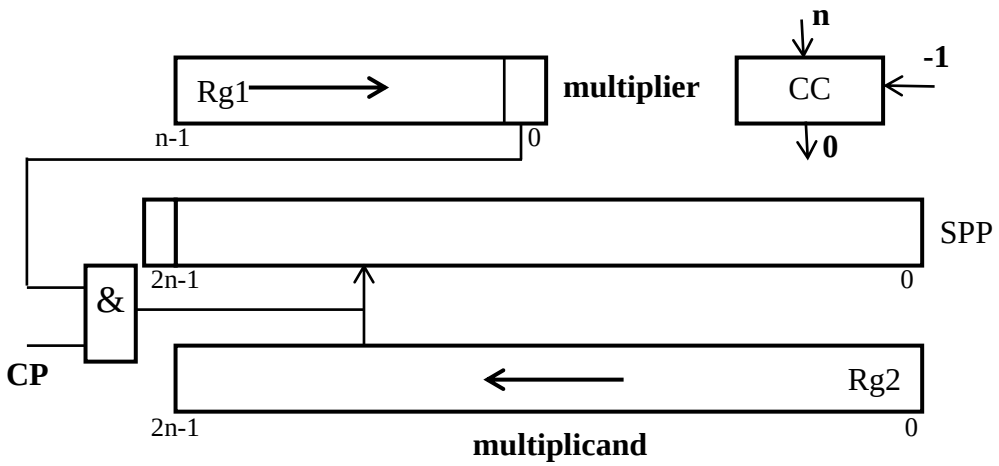

Figure 3.3. The structure of the multiplication device from the lower digits of the multiplier and the shift of the multiplicand

Before the beginning, the product is placed in a double digit register starting from the lowest digits. In each cycle of multiplication, the lowest digit of the

multiplier is analyzed. If it is equal to 1, then the value of the multiplicand is added to the content of the sum of partial products. Next, the multiplicand t is shifted to the left by one digit, and the multiplier is shifted to the right by 1 digit.

Figure 3.4 shows an example of performing a multiplication operation in this way.

Multiplier: $9_{10}=1001_2$
Multiplicand: $11_{10}=1011_2$

```
SPP                Multiplier      Multiplicand
0 00000000          1001 R1
0 00001011                          00001011 L1
0 00001011          0100 R1
0 00000000                          00010110 L1
0 00001011          0010 R1
0 00000000                          00101100 L1
0 00001011          0001 R1
0 01011000                          01011000 L1
0 01100011
```

Figure 3.4. An example of multiplication from the lower digits of the multiplier and the shift of the multiplicand

The beginning of multiplication from the higher digits of the multiplier and the shift of the sum of partial products (Figure 3.5, method №3). In each cycle of multiplication, the higher digit of the multiplier is analyzed. If it is equal to 1, then the value of the multiplicand is added to the content of the sum of partial products. Moreover, the multiplicand is added to the younger part of the sum of partial products. Next, the sum of the partial products and the multiplier are shifted to the left by one digit.

Figure 3.6 shows an example of performing a multiplication operation in this way.

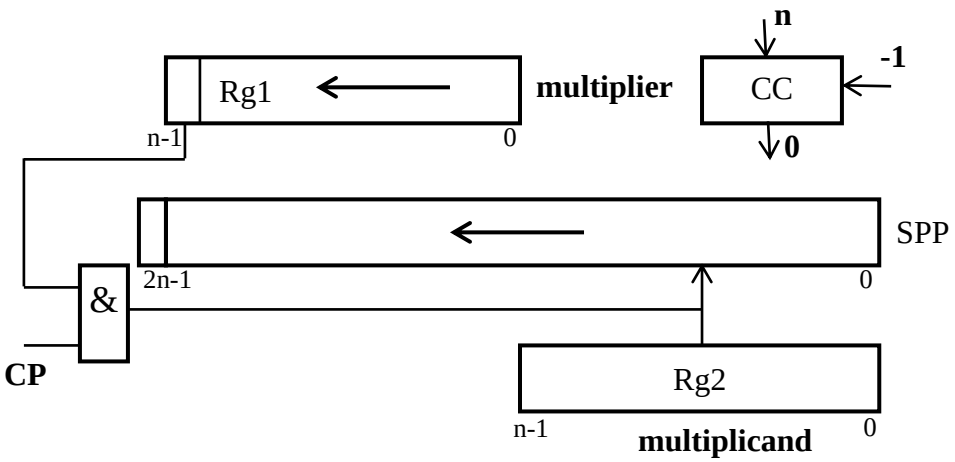

Figure 3.5. The structure of the multiplication device from the higher digits of the multiplier and the shift of the sum of partial products

ATTENTION!!! *In this method, the last shift is not performed.*

Multiplier: $5_{10}=0101_2$
Multiplicand: $13_{10}=1101_2$

```
SPP                Multiplier
0 00000000           0101 L1
0     0000
0 00000000 L1
0 00000000           1010 L1
0     1101
0 00001101 L1
0 00011010           0100 L1
0     0000
1 00011010 L1
0 00110100           1000 L1
0     1101
0 01000001
```

Figure 3.6. An example of multiplication from the higher digits of the multiplier and the shift of the sum of partial products

The beginning of multiplication from the higher digits of the multiplier and the shift of the multiplied (Figure 3.7, method №4) Before the beginning, the product is in a register with double digits starting from the higher digits. Each cycle begins with a shift multiplicand by 1 digit to the right. The higher digit of the multiplier is analyzed. If it is equal to 1, then the value of the multiplicand is added to the content of the sum of partial products. Next, the multiplier is shifted by 1 digit to the left.

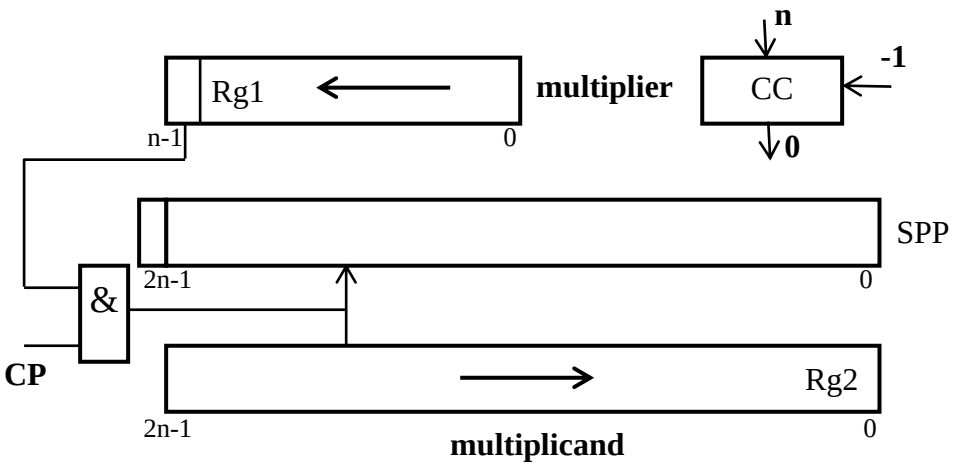

Figure 3.7. The structure of the multiplication device from the higher digits of the multiplier and the shift of the product

Figure 3.8 shows an example of performing a multiplication operation in this way.

ATTENTION!!! *In this method, the shift is performed first, and then the addition. An additional digit is required for the adder.*

Multiplier: $9_{10}=1001_2$
Multiplicand: $11_{10}=1011_2$

СЧД	Multiplier	**Multiplicand**
0 00000000	1001 L1	
0 01011000		R1 01011000
0 01011000	0010 L1	
0 00000000		R1 00101100
0 01011000	0100 L1	
0 00000000		R1 00010110
0 01011000	1000 L1	
0 00001011		R1 00001011
0 01100011		

Figure 3.8. An example of multiplication from the higher digits of the multiplier and the shift of the multiplicand

The number of cycles for each method corresponds to the bit size of the operands.

The time of the multiplication operation can be estimated by the formula:

$$T_{множ} = n\times(t_{дод}+t_{зс}), \quad (3.1)$$

where *n* – number of digits;

$t_{дод}$ – time of addition;

$t_{зс}$ – shift execution time.

The general multiplication algorithm is shown in Figure 3.9. Registers Rg1 and Rg2 contain two output operands. The sum of partial products is formed on the SPP adder. The CC counter sets the number of iterations of addition operations and depends on the bit size of the operands. The result of the operation is formed in the SPP.

2. Problem definition (Goal)

Develop the structure, algorithm and microprogram of the device to perform the operation of multiplying two n-bit numbers with a fixed point according to the option (Table 3.1). The sign of the operands can be ignored.

3. The order of work (according to the variant)

1. Make the structure of the multiplication device.
2. Develop a multiplication algorithm.
3. Download JOLS-M.

4. Describe the structure of the multiplication device.

5. Compile microprograms for implementing the given method of multiplication.

6. Simulate the operation of the multiplication device on a computer by printing the contents of the multiplier register and the register of sums of partial products after each multiplication cycle.

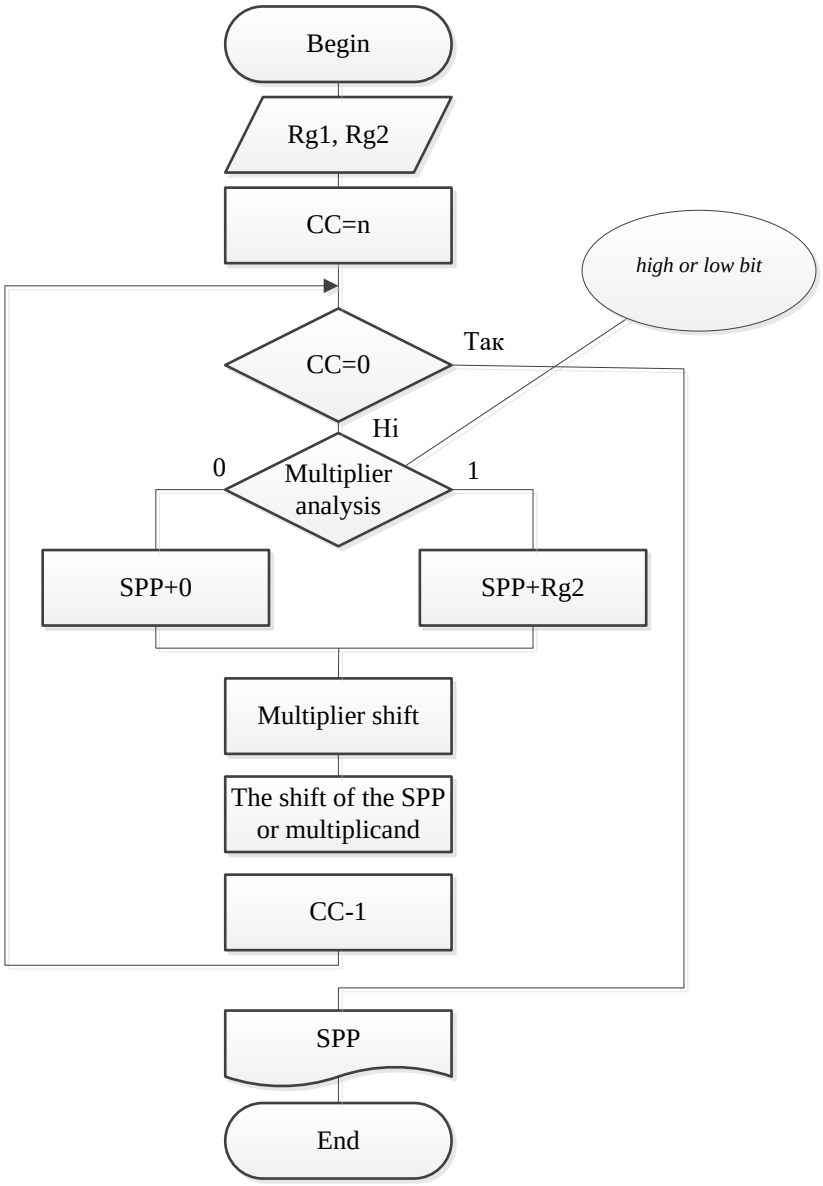

Figure 3.9. A general algorithm for multiplying two n-bit numbers with a fixed point

4. The content of the report

The report should contain brief theoretical information, developed structure, algorithm (block diagram) and microprogram for performing the operation of multiplication of two operands, as well as simulation results.

Table 3.1 –Variants of tasks for performing the multiplication operation

Variant number	Multipliers bit depth	Method of multiplication	Variant number	Multipliers bit depth	Method of multiplication
1	4	1	14	7	2
2	4	2	15	7	3
3	4	3	16	7	4
4	4	4	17	8	1
5	5	1	18	8	2
6	5	2	19	8	3
7	5	3	20	8	4
8	5	4	21	9	1
9	6	1	22	9	2
10	6	2	23	9	3
11	6	3	24	9	4
12	6	4	25	10	1
13	7	1	26	10	2

Note: The methods of multiplication are marked as follows: 1 – the beginning of multiplication from the lower orders of the multiplier and the shift of the sum of partial products; 2 – the beginning of multiplication from the lower orders of the multiplier and the shift of the multiplicand; 3 – the beginning of multiplication from the higher digits of the multiplier and the shift of the sum of partial products; 4 - the beginning of multiplication from the higher digits of the multiplier and the shift of the multiplicand.

5. Control questions and tasks

1. What is a partial product?
2. How do machine methods of multiplication differ from each other?
3. Why can the next partial product in the binary number system be equal to?
4. What is the digit value of the product with n-digit coefficients?
5. What is the purpose of rounding the product?
6. What microoperations are performed in each cycle of multiplication?
7. What is the multiplication time?
8. What logical methods of multiplication do you know? Name the time to perform the multiplication using these methods.

PRACTICAL WORK № 4

STUDY OF ACCELERATED LOGICAL MULTIPLICATION METHODS FOR THE NUMBERS WITH FIXED POIN (MULTIPLICATION WITH ANALYSIS OF SEVERAL DIGITS)

Objective:

1. Get acquainted with the existing methods of accelerated multiplication of numbers with a fixed point.
2. To simulate the operation of accelerated multiplication methods on a computer according to the variant of the task.

1. Theoretical information

The structural diagram of the device for accelerated multiplication with the analysis of several digits is similar to the device for multiplication with bit-by-bit analysis of the multiplier. But there are several features of this operation. In each cycle, ***k*** digits of the multiplier are analyzed, the partial product is determined accordingly, which is added to the sum of the partial products. The shift of the multiplier and the sum of the partial products (or the multiplied - depending on the method) is performed by ***k*** digits.

Tables 4.1 and 4.2 show the values of the partial products at different values of the digits of the multiplier for the method with the analysis of 2 digits. Table 4.1 shows the values for the method of multiplication starting from the lowest digits, in table 4.2 - from the highest digits (note: A is multiplicand).

Table 4.1. Values of partial products for the method of multiplication starting from the lowest digits

Current pair of digits	**A sign of correction in the current measure**	**Partial product**	**A sign of correction in the next measure**
00	0	0	0
01		A	0
10		2A	0
11		-A	1
00	1	A	0
01		2A	0
10		-A	1
11		0	1

Table 4.2. Values of partial products for the method of multiplication starting from higher digits

Current pair of digits	**Next digit**	**Partial product**
00	0	0
01		A
10		-2A
11		-A
00	1	A
01		2A
10		-A
11		0

Start of multiplication with lower digits. The correction sign bit can be generated automatically. For this, an additional digit (the oldest) should be used in the register of the sum of partial products. To multiply the sum of partial products with a shift, in addition to this digit, 2 additional ones should be used (to organize

the shift, etc.). It should be noted that if a correction digit equal to one is received in the last cycle of multiplication, then one additional cycle of multiplication must be performed.

Start of multiplication with higher digits. Multiplication begins with the analysis of the fictitious senior pair of "00" digits. Further, all actions are performed according to the table. 4.2. When multiplying with a shift of the product in the first cycle, the product is not shifted. The multiplication ends with an addition operation.

The number of clocks for the multiplication operation is chosen according to the digit ratio of the coefficients and is equal to n/2 digits (rounded up).

The time of performing the operation of accelerated multiplication with the analysis of two digits can be estimated by the formula:

$$T_{множ} = n/2 \times (t_{дод} + t_{зс}), \quad (4.1)$$

where *n* – number of digits;

$t_{дод}$ – time of addition;

$t_{зс}$ – shift execution time.

2. Problem definition (Goal)

Develop the structure, algorithm and microprogram of the device to perform the operation of multiplying two n-digit numbers with a fixed point with the analysis of two digits according to the option (Table 4.3). The sign of the operands can be ignored.

Table 4.3. Variants of tasks for performing the multiplication operation with the analysis of two digits

Variant number	Multipliers bit depth	Method of multiplication	Variant number	Multipliers bit depth	Method of multiplication
1	4	4	14	10	3
2	4	3	15	10	2
3	4	2	16	10	1
4	4	1	17	12	4
5	6	4	18	12	3
6	6	3	19	12	2
7	6	2	20	12	1
8	6	1	21	14	4
9	8	4	22	14	3
10	8	3	23	14	2
11	8	2	24	14	1
12	8	1	25	16	2
13	10	4	26	16	1

3. The order of work (according to the variant)

1. Compile the structure of the multiplication device with the analysis of two digits.

2. Develop a multiplication algorithm with the analysis of two digits.

3. Download JOLS-M.

4. Describe the structure of the multiplication device with the analysis of two digits.

5. Compile microprograms for implementing the given method of multiplication.

6. Simulate the operation of the multiplication device, performing a printout of the successive digits of the multiplier and the contents of the register of the sum of partial products in each cycle of multiplication.

4. The content of the report

The report should contain brief theoretical information, the developed structure, algorithm (block diagram) and microprogram for performing the operation of multiplication of two operands with analysis of two digits, as well as simulation results.

5. Control questions and tasks

1. Why can the partial product be equal when multiplying by two digits of the factor and starting the multiplication from the lower digits of the factor? (from senior ranks)? When multiplying by three digits?

2. What determines the need to correct the next group of digits of the multiplier when starting multiplication from the lowest digits of the multiplier? (from older grades)?

3. Why, when multiplying by a group of digits of the multiplier, starting from the youngest digits, and shifting the sum of partial products, additional overflow digits are needed in the adder?

4. What is the multiplication time when multiplying with the analysis of two digits of the multiplier? Three digits of the multiplier?

PRACTICAL WORK № 5

STUDY OF DIVISION METHODS FOR THE NUMBERS WITH FIXED POINT

Objective:

1. Get acquainted with the existing ways of division numbers with a fixed point.
2. Model the operation of division methods on a computer according to the task variant.

1. Theoretical information

The general idea of the division operation is to successively subtract the divisor from the divident. At the same time, the next digit of the quotient (result) is determined by the inversion of the sign digit of the remainder. Depending on the

method of division, the remainder can be restored. After that, the divisor is shifted relative to the divisor (to the left by 1 digit), or the divisor is shifted relative to the divisor (to the right by 1 digit).

Before starting the division, a trial cycle is performed to check the possibility of further execution of the operation. If at this stage we receive a positive remainder, then further execution is impossible, as an overflow occurs (as a result of dividing a larger number by a smaller one, we get a whole part that has nowhere to store).

Completion of the division operation takes place according to the clock counter.

Two types of devices are used to perform the operation of dividing numbers:

– a device with a shift of residues to the left (Figure 5.1);
– a device with a shift of residues to the right (Figure 5.2).

On both devices, it is possible to perform division in two ways: with remainder recovery and without current.

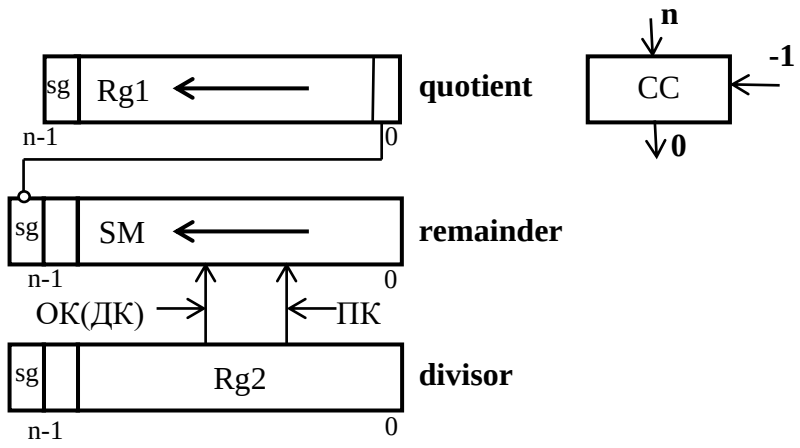

Figure 5.1. The structure of the division device with a shift of the remainder

Division with restitution. If a negative remainder is obtained after subtraction from the divisor, then it should be restored by adding the divisor to the remainder.

The execution time of division with remainder recovery can be estimated using the following formula:

$$T_{дidiv} = n \times (1.5 \times t_{add} + t_{sh}), \quad (5.1)$$

where *n* – number of digits;

t_{add} – time of addition;

t_{sh} – shift execution time.

Division without restitution. If a negative remainder is obtained after subtraction from the divisor, then in the next cycle the value of the divisor will be added to the remainder, if the remainder is positive, the divisor will be subtracted from the remainder in the next cycle.

The execution time of division with remainder recovery can be estimated using the following formula:

$$T_{діл} = n \times (t_{add} + t_{sh}), \quad (5.2)$$

where *n* – number of digits;

t_{add} – time of addition;

t_{sh} – shift execution time.

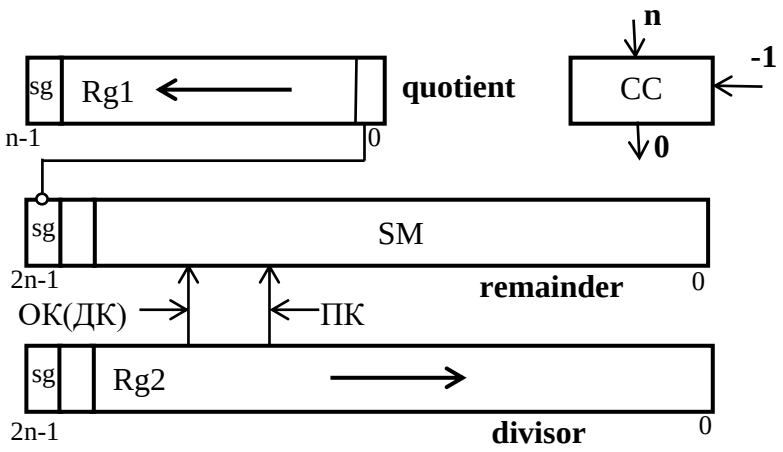

Figure 5.2. The structure of the division device with a divisor shift

Figures 5.3, 5.4, 5.5 and 5.6 show examples of performing the division operation in different ways (B – divider).

The general algorithm for division numbers with remainder restoration is shown in Figure 5.7. Register Rg2 contains the divisor, SM contains the divided. A balance occurs on the SM adder. The CC counter sets the number of iterations of operations to form the final value of the share. The result of the operation is formed in Rg1.

2. Problem definition (Goal)

Develop the structure, algorithm and microprogram of the device to perform the operation of division two n-bit numbers with a fixed point according to the option (Table 5.1). The sign of the operands can be ignored.

3. The order of work (according to the variant)

1. Make the structure of the division device.
2. Develop a division algorithm.
3. Download JOLS-M.
4. Describe the structure of the division device.
5. Compile microprograms to implement the given method of division.
6. Simulate the operation of a division device on a computer by printing the contents of the quotient register and the adder with remainders after each division cycle.

Dividend: $9_{10}=1001_2$
Divisor: $13_{10}=1101_2$ 0011_{2cc}

Remainder		Quotient
0 0 1001		
1 1 0011	-B	
1 1 1100		0.
0 0 1101	+B	
0 0 1001	L1	
0 1 0010		
1 1 0011	-B	
0 0 0101	L1	0.1
0 0 1010		
1 1 0011	-B	
1 1 1101		0.10
0 0 1101	+B	
0 0 1010	L1	
0 1 0100		
1 1 0011	-B	
0 0 0111	L1	0.101
0 0 1110		
1 1 0011	-B	
0 0 0001		0.1011

Figure 5.3. An example of division with a shift of the remainder and with the restoration of the remainder

Dividend: $9_{10}=1001_2$
Divisor: $13_{10}=1101_2$ 0011_{2cc}

Remainder		Quotient
0 0 1001		
1 1 0011	-B	
1 1 1100	L1	0.
1 1 1000		
0 0 1101	+B	
0 0 0101	L1	0.1
0 0 1010		
1 1 0011	-B	
1 1 1101	L1	0.10
1 1 1010		
0 0 1101	+B	
0 0 0111	L1	0.101
0 0 1110		
1 1 0011	-B	
0 0 0001		0.1011

Figure 5.4. An example of division with a shift of the remainder and without restoration of the remainder

Dividend: $7_{10}=0111_2$
Divisor: $12_{10}=1100_2$ 0100_{2cc}

Remainder		Quotient	Divisor
0 0 01110000			
1 1 01000000	-B		01000000_{2cc} R1
1 1 10110000		0.	
0 0 01100000	+B		01100000 R1
0 0 00010000		0.1	
1 1 11010000	-B		11010000_{2cc} R1
1 1 11100000		0.10	
0 0 00011000	+B		00011000 R1
1 1 11111000		0.100	
0 0 00001100	+B		00001100
0 0 00000100		0.1001	

Figure 5.5. An example of division with a shift of the divisor and with the restoration of the remainder

Dividend: $7_{10}=0111_2$
Divisor: $12_{10}=1100_2$ 0100_{2cc}

Remainder		Quotient	Divisor
0 0 01110000			
1 1 01000000	-B		01000000_{2cc} R1
1 1 10110000		0.	
0 0 11000000	+B		
0 0 01110000			
1 1 10100000	-B		10100000_{2cc} R1
0 0 00010000		0.1	
1 1 11010000	-B		11010000_{2cc} R1
1 1 11100000		0.10	
0 0 00110000	+B		
0 0 00010000			
1 1 11101000	-B		11101000_{2cc} R1
1 1 11111000		0.100	
0 0 00011000	+B		
0 0 00010000			
1 1 11110100	-B		11101000_{2cc}
0 0 00000100		0.1001	

Figure 5.6. Example of division with divisor shift and no recovery the remainder

4. The content of the report

The report should contain brief theoretical information, the developed structure, algorithm (block diagram) and microprogram for performing the operation of division of two operands, as well as simulation results.

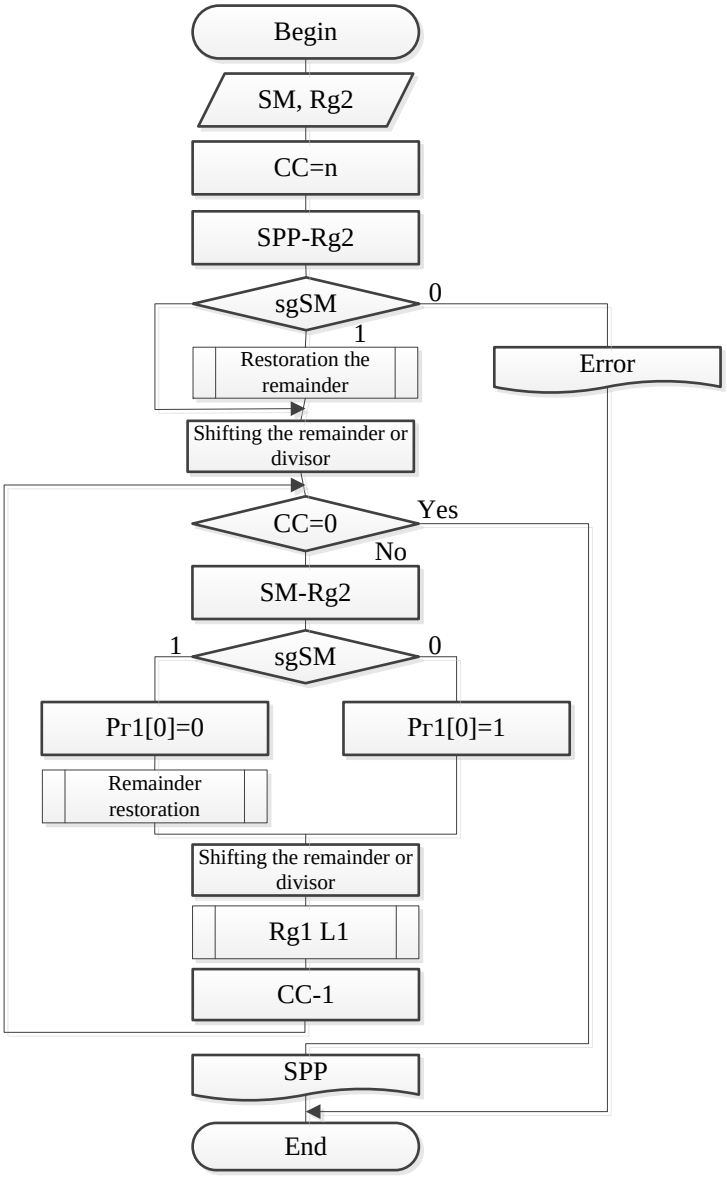

Figure 5.7. A general algorithm for dividing two n-bit numbers with remainder recovery

Table 5.1. Variants for performing the division operation

Variant	Division method	Recovery of the remainder	bit depth	Variant	Division method	Recovery of the remainder	bit depth
1	Remainder shift	yes	6	4	Remainder shift	no	9
2	Remainder shift	no	6	5	Divider shift	yes	9
3	Divider shift	yes	6	6	Divider shift	no	9
7	Divider shift	no	6	17	Divider shift	no	5
8	Remainder shift	yes	7	18	Divider shift	yes	5
9	Remainder shift	no	7	19	Remainder shift	no	5
10	Divider shift	yes	7	20	Remainder shift	yes	5
11	Divider shift	no	7	21	Divider shift	no	10
12	Remainder shift	yes	8	22	Divider shift	yes	10

Table 5.1 (continued)

13	Remainder shift	no	8	23	Remainder shift	no	10
14	Divider shift	yes	8	24	Remainder shift	yes	10
15	Divider shift	no	8	25	Divider shift	no	11
16	Remainder shift	yes	9	26	Remainder shift	yes	11

4. The content of the report

The report should contain the developed algorithm (block diagram), printed microprogram and simulation results, structural diagram of the division operation device.

5. Control questions and tasks

1. What machine methods of division do you know?
2. How is overflow determined when dividing fractional numbers?
3. How is each subsequent digit of the fraction determined?
4. In what cases is the remainder restored during division with remainder restoration?
5. Prove the possibility of performing division without restoring the remainder.
6. How is the sign of the particle determined?
7. How is rounding performed during division?
8. Why is the time for a division operation with restorative and without restorative equal?
9. In what situation during division with remainder restoration will the division time be maximum (minimum)?

PRACTICAL WORK № 6

STUDY OF ADDITION METHODS FOR THE NUMBERS WITH FLOAT POINT

Objective:

1. Get acquainted with ways of representing floating point numbers.
2. Learn the procedure for adding floating-point numbers.
3. Simulate the operation of adding floating-point numbers on a computer according to the task option.

1. Theoretical information

Numbers with a floating point are presented in the form of mantissa and exponent: $A=Ma\cdot 2^{Pa}$, $B= Mв\cdot 2^{Pв}$. At the same time, the number of digits of the mantissa is equal to *m*, of exponent *n*.

For binary numbers, the addition operation is performed in the following sequence.

1. Alignment of exponents. The exponent of the second number is subtracted from the exponent of the first number. The mantissa of a number with a lower

exponent is shifted to the right by the difference of exponents. Which of the exponents is smaller is determined by the sign of the difference of exponents. At this stage, the result of the entire operation can be determined without performing the addition itself, when the difference in exponents exceeds the number of digits of the mantissa. In this case, the result will be a number with a higher exponent.

2. Adding mantises. Mantises are added in the usual way, taking into account the code in which the terms are presented. When adding, overflow is possible.

3. Normalization of the result. Overflow may occur after adding mantises. It can be eliminated by shifting the mantissa to the right by one digit and increasing the exponent of the result by 1. At the same time, if an overflow occurs when the exponent is increased, it will be a true overflow. Also, after adding mantises, the result may need to be normalized. Normalization is performed by shifting the mantissa to the left and, accordingly, reducing the exponent by the amount of the shift. At the same time, if there is a negative overflow of exponent, then a situation of "disappearance of exponent" occurs.

Figure 6.1 shows the structure of a device for adding two floating-point numbers. Adders SMM and SME contain the values of the mantissa and the exponent of the first operand, registers RgM and RgE of the second. The trigger T is used to store the sign of the difference in the exponent of the operands. Rg1 are used to temporarily store the exponent of the first operand. The result of the addition operation is stored in the adders SMM and SME.

Figure 6.2 shows the general algorithm for adding two floating-point numbers.

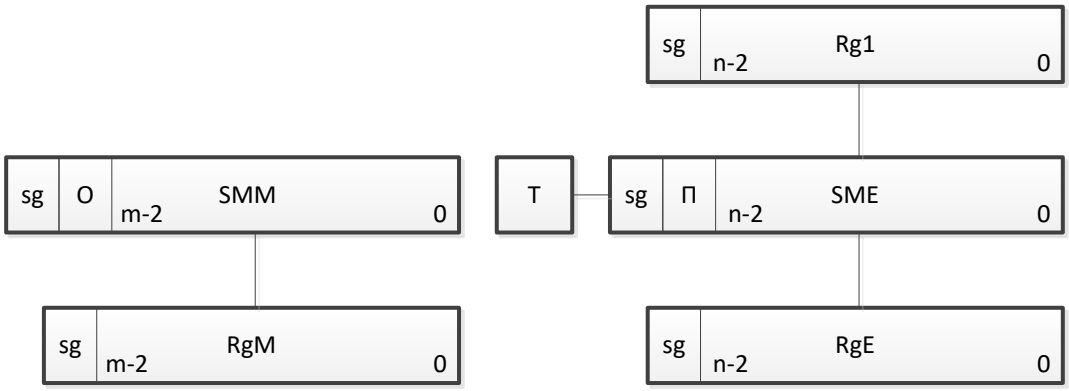

Figure 6.1. Device structure for adding two floating-point numbers

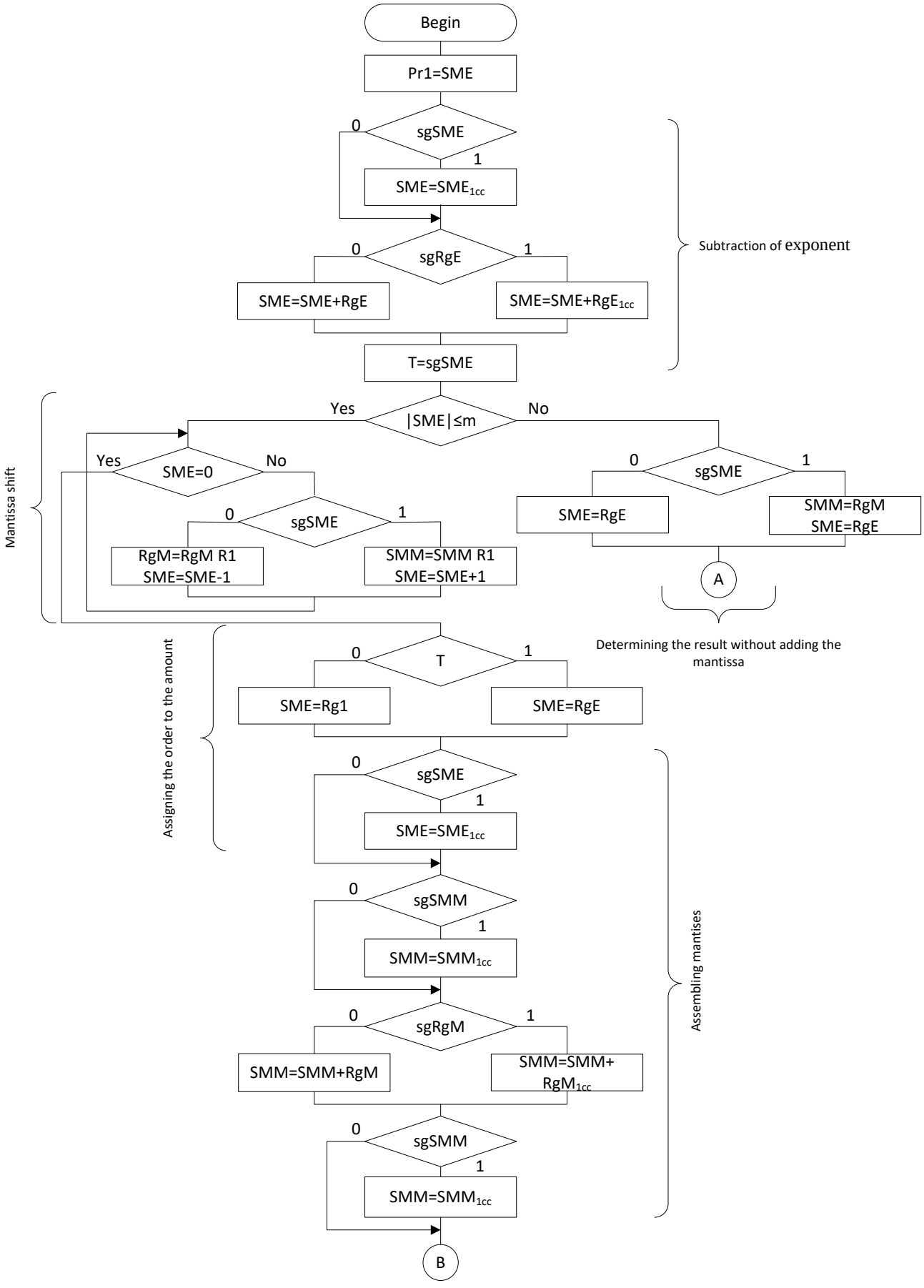

Figure 6.2. General algorithm for adding two floating-point numbers

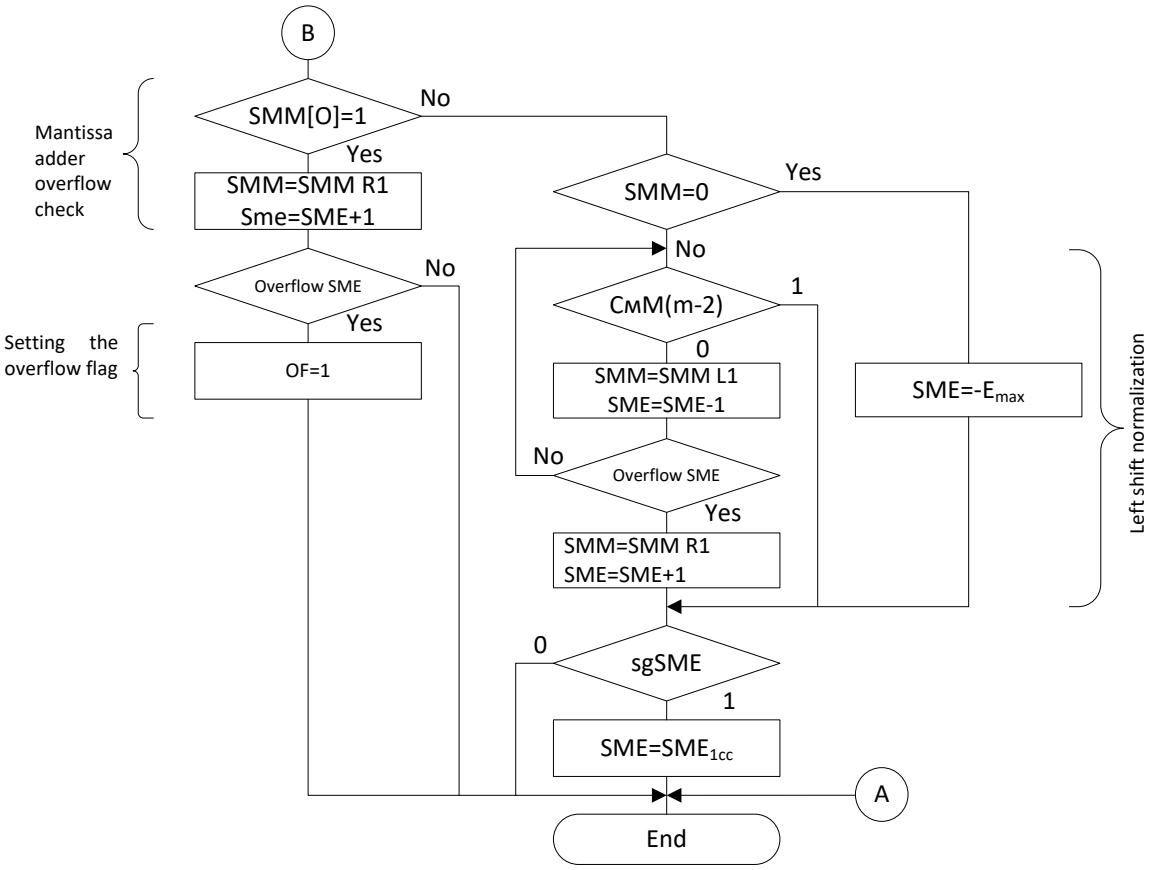

Figure 6.2. General algorithm for adding two floating-point numbers (continued)

2. Problem definition (Goal)

Develop the structure, algorithm and microprogram of the device to perform the operation of adding two floating-point numbers according to the option (Table 6.1).

3. The order of work (according to the variant)

1. Compile the structure of the device for adding two floating-point numbers.
2. Develop an algorithm for adding two floating-point numbers.
3. Download JOLS-M.
4. Describe the structure of the device for adding two floating-point numbers.
5. Compile microprograms for implementing the specified method of adding two floating-point numbers.
6. Perform simulation of ALP work on four examples:

1) $A>0$, $B>0$, $P_A>0$, $P_B<0$; 2) $A>0$, $B<0$, $|A|>|B|$, $P_A>0$, $P_B>0$;

3) $A>0$, $B<0$, $|A|<|B|$, $P_A<0$, $P_B>0$; 4) $A<0$, $B<0$, $P_A>0$, $P_B>0$.

The numbers are chosen arbitrarily, but in each example $P_A \neq P_B$. In the firmware, provide for the printing of the mantissa and the exponent after each cycle of exponent alignment and normalization.

Table 6.1. Task options for adding floating-point numbers

Variant number	bit depth		Code that is applied
	mantissa	exponent	
1	10	5	1cc
2	11	6	2cc
3	12	7	1cc
4	13	8	2cc
5	14	5	1cc
6	15	6	2cc
7	16	7	1cc
8	17	8	2cc
9	18	5	1cc
10	19	6	2cc
11	20	7	1cc
12	10	8	2cc
13	11	5	1cc
14	12	6	2cc
15	13	7	1cc
16	14	8	2cc
17	15	5	1cc
18	16	6	2cc
19	17	7	1cc
20	18	8	2cc
21	19	5	1cc
22	20	6	2cc
23	15	7	1cc
24	16	5	2cc
25	17	6	1cc

Note: 1cc – one complement code; 2cc – two complement code.

4. The content of the report

The report should contain brief theoretical information, the developed structure, algorithm (block diagram) and microprogram for performing the operation of adding floating-point numbers, as well as simulation results

5. Control questions and tasks

1. What is the main purpose of using floating point representation?
2. What characterizes the exponent bit and the mantissa bit?
3. Name the main stages of adding floating-point numbers.
4. How is the exponent compared?
5. The exponent of which term is the exponent of the result?
6. In what situations can the result be obtained without performing addition (that is, the result is equal to one of the terms)?
7. In what situation can a true overflow occur? What is a sign of overflow?

8. What actions are performed if an overflow occurred when adding mantises?

9. Why does the result of the operation, as a rule, normalize?

10. When can loss of exponent occur?

11. What are the advantages of representing floating-point numbers with shifted exponent (characteristic)?

12. How does a result with a zero mantissa appear?

LIST OF REFERENCES

1. Харріс Д.М. Цифрова схемотехніка та архітектура комп'ютера [Текст] / Д.М.Харріс. – Морган Кауфман, 2013. – 1662 с.
2. Матвієнко М.П. Архітектура комп’ютерів: Навчальний посібник [Текст] / Матвієнко М.П., Розен В.П., Закладний О.М. – К.: Ліра-К, 2019. – 264 с.
3. Карачка А. Ф., Дудко О. І. Архітектура комп’ютерів: Навч. посіб. [Текст] / За ред. А. О. Саченка. – Тернопіль: Економічна думка, 2009. – 180 с.
4. Тарарака В.Д. Архітектура комп'ютерних систем: навчальний посібник [Текст] / В.Д.Тарарака – Житомир : ЖДТУ, 2018. – 383 с.
5. Антоненко О. В., Бардус І. О. Архітектура комп’ютера та конфігурування комп’ютерних систем (на основі фундаменталізованого підходу) : навч. посіб. [Текст] / О. В. Антоненко, І. О. Бардус − Бердянськ : БДПУ, 2018. – 292с.
6. Голотенко О.С. Архітектура комп’ютерних систем: конспект лекцій для студентів усіх форм навчання з курсу «Архітектура комп’ютерних систем» [Текст] / О.С. Голотенко– Тернопіль : Вид-во ТНТУ ім. Івана Пулюя, 2016. – 120 с.
7. Linda Null, Julia Lobur. Essentials of Computer Organization and Architecture [Текст] / Jones & Bartlett Learning, 2018. – 744 с.
8. Paul D. Crutcher, Neeraj Kumar Singh, Peter Tiegs. Essential Computer Science [Текст] / Apress, 2021. – 290 с.
9. William Stallings. Computer Organization And Architecture [Текст] / PEARSON, 2015. – 864 с.
10. СДН «Лідер». Архітектура комп’ютерів. Процесори (https://lider.diit.edu.ua/course/view.php?id=1792).

Educational and methodical edition

Yehorov Oleh
Dziuba Volodymyr
Ivin Pavlo

COMPUTER ARCHITECTURE

Methodical recommendations for practical works

In the author's edition
Computer layout O. Yehorov

Формат 60x84 1/16. Ум. друк. арк. 2,15. Обл.-вид. арк. 2,17.
Зам. № 153.

Ukrainian state university of science and technology Publishing house

Certificate of the subject of publishing activity DK № 7709 dated 14.12.2022
Address of the publishing house and department of operational polygraphy:
St. Lazaryana, 2; Dnipro, 49010