

"ELEKTRONIKA MK 90"
M I C R O C A L C U L A T O R

M O D E R N I Z E D
O P E R A T I N G M A N U A L

1 9 8 8

Contents

page

1. General instructions	4
2. Technical data	4
3. Delivery set	6
4. Safety requirements	6
5. Preparing for use	6
6. Operating procedure	9
6.1. Keyboard and LCD operating modes	9
6.1.1. Boot mode	9
6.1.2. Special keys	11
6.1.3. Keyboard operation in upper and lower case mode	12
6.1.4. Keyboard operation in Russian/Latin layout modes	14
6.1.5. Keyboard operation in functional mode	15
6.1.6. LCD operating mode	16
6.2. Calculation functions	17
6.2.1. Constants and calculation accuracy	17
6.2.2. Arithmetic functions of the microcalculator's BASIC	17
6.2.3. Variables	19
6.3. Microcalculator's BASIC commands	20
6.3.1. Immediate mode commands	20
6.3.1.1. AUTO command	20
6.3.1.2. DELETE command	21
6.3.1.3. EDIT command	21
6.3.1.4. HELP command	22
6.3.1.5. LIST command	22
6.3.1.6. MEM command	23
6.3.1.7. INIT command	23
6.3.1.8. SAVE command	24
6.3.1.9. NAME/AS command	24
6.3.1.10. LOAD command	25
6.3.1.11. KILL command	25
6.3.1.12. DEV command	25
6.3.1.13. FILES command	26
6.3.1.14. RUN command	27
6.3.2. Program mode commands	27
6.3.2.1. DEF FN operator	27
6.3.2.2. DIM operator	30
6.3.2.3. END operator	33
6.3.2.4. FOR-TO-STEP/NEXT operator	33
6.3.2.5. GOSUB, RETURN operators	34
6.3.2.6. GOTO operator	35
6.3.2.7. IF/THEN operator	37
6.3.2.8. INPUT operator	39
6.3.2.9. LET operator	39
6.3.2.10. PRINT operator	40

	page
6.3.2.11. READ, DATA, RESTORE operators	42
6.3.2.12. REM (REMARK) operator	45
6.3.2.13. RANDOMIZE operator	46
6.3.2.14. CLS operator	47
6.3.2.15. DRAW operator	47
6.3.2.16. LOCATE operator	53
6.3.2.17. STOP operator	54
6.3.2.18. DIS operator	54
6.3.2.19. WAIT operator	54
6.3.2.20. PLAY operator	55
6.3.3. Built-in functions	56
6.3.3.1. SIN, COS functions	56
6.3.3.2. ATN function	56
6.3.3.3. LOG function	57
6.3.3.4. EXP function	59
6.3.3.5. ABS function	59
6.3.3.6. INT function	60
6.3.3.7. INC function	61
6.3.3.8. SGN function	61
6.3.3.9. SQR function	61
6.3.3.10. RND function	62
6.3.3.11. PI function	62
7. Storage rules	63
Appendix 1. Keyboard diagram for working in functional mode	64
Appendix 2. Microcalculator's BASIC commands	65
Appendix 3. Error messages	76
Appendix 4. Examples of programs	79
Appendix 5. Character code tables	88
Appendix 6. Recommendations for working with SMP	91

Memo for foreigners

МК 90's BASIC prints messages in Russian. In the listings, these messages aren't translated since this is what you will see on the real screen. So there are translations of these messages:

БЕЙСИК = BASIC

Готов = Ready

ОСТ В СТРОКЕ = STOP AT LINE

вр / нр = upper / lower case

лат / рус = Latin / Russian layout

фк = Function key (pressed)

су = Control key (pressed)

ОШ = ОШИБКА = ERROR

СТР = СТРОКА = LINE

Справочник SM* = SM* disk directory

Программа = Program

Свободно = Free

СМП = Сменный Модуль Памяти =
Removable Memory Cartridge (SMP / RMC)

Распределение памяти (в байтах) =
Memory allocation (in bytes)

1. General instructions

- 1.1. Before using the "Elektronika MK90" microcalculator, carefully read this user manual. Compliance with its requirements will ensure reliable operation of the product for a long time.
- 1.2. To ensure long and reliable operation of the microcalculator, avoid using or storing the product in places subject to sudden temperature changes, high humidity, do not allow the microcalculator case to heat up from direct sunlight, protect it from mechanical damage.
- 1.3. Do not remove the plug from the diagnostic connector of the microcalculator in order to protect the product from static electricity.
- 1.4. Use only a soft and clean cloth to wipe the microcalculator case.
- 1.5. Do not expose the microcalculator to mechanical shock, since the liquid crystal display (LCD) screen is made of glass.
- 1.6. Do not apply great force to the LCD contrast regulator to avoid breakage.
- 1.7. Do not store the removable memory cartridge (SMP) in the calculator without the batteries installed. The SMP should be stored in the case intended for its storage and included in the delivery set.
- 1.8. Do not touch the SMP connector contacts in order to protect the SMP from static electricity.
- 1.9. After transporting the calculator in winter conditions, keep it at room temperature for two hours before using it.

2. Technical data

- 2.1. Microcalculator "Elektronika MK90" is designed to perform scientific, engineering, statistical calculations; for operational processing, storage and display of various text, symbolic and graphic information.
- 2.2. The microcalculator provides the user with the following options:
 - Operation as an information device using a personal set of programs generated by the user and stored in the SMP;
 - Input, editing, debugging and execution of programs in the BASIC programming language;
 - Obtaining reference information on the purpose and format of more than 50 commands and operators of the BASIC language;
 - Plotting various graphs, tables, diagrams and schemes on the LCD screen;
 - Output of alphanumeric information to the LCD screen in various scales and directions;
 - Text editing;
 - Sound signal control.

- 2.3. The microcalculator provides work with the BASIC programming language.
- 2.4. The number system used for input and output of information is decimal.
- 2.5. The number of digits of the mantissa of a number is seven.
- 2.6. The number of digits of the order of the number is three.
- 2.7. Range of numbers presented $\pm 10^{-980}$ to $\pm 10^{979}-1$.
- 2.8. Number of memory registers – 286.
- 2.9. The RAM memory capacity at the user's disposal is 11824 bytes.
- 2.10. The microcalculator provides operation with two SMP.
- 2.11. The capacity of the volatile RAM memory of each SMP is 10 KB, including 8 KB (16 blocks) at the user's disposal.
- 2.12. The SMP ensures storage of information when disconnected from the microcalculator for 1.5 years from the date of installation of the power supply element in the SMP and in the event of:
 - Ambient air temperature $25 \pm 10^{\circ}\text{C}$;
 - Relative air humidity no more than 80%;
 - Atmospheric pressure 630–800 mm Hg.
- 2.13. Information is entered:
 - 1) from the keyboard
 - 2) from two SMP.
- 2.14. Information is output:
 - 1) on a graphic LCD (120 columns by 64 lines, for symbolic representation of information – 8 lines by 20 characters);
 - 2) into two SMP.
- 2.15. The microcalculator is powered by four NKGC-0,45 IIC PBC3.579.000 TU batteries. It is allowed to use power sources of the A-316 "KVANT" or "PRIMA" type instead of NKGC-0,45 batteries. The microcalculator can be powered by an external power supply with an output voltage of 4.5–6.0 V and a load current of at least 0.150 A.
- 2.16. Fully charged batteries of the NKGC-0,45 type provide:
 - 1) in the user program execution mode, at least 5–6 hours of continuous operation of the microcalculator;
 - 2) in text input and editing mode, at least 15–20 hours of operation.Current consumption in the "off" state – no more than 0.7 mA
- 2.17. The microcalculator is designed to operate at temperatures from +1 to 50°C, relative air humidity up to 80% at 25°C.
- 2.18. The overall dimensions of the microcalculator are no more than:
 - length 255 mm;
 - width 100 mm;
 - height 33,5 mm;
- 2.19. Weight no more than 0.55 KG.
- 2.20. The maximum electrical power consumed by the microcalculator is no more than 0.55 W.
- 2.21. Content of precious materials:
 - gold – 0,6252 g;
 - silver – 0,6846 g;
 - platinum – 0,07510 g;
 - palladium – 0,3873 g;
 - ruthenium – 0,0002967 g.
- 2.22. Total mass of non-ferrous metals and their alloys:
 - aluminum and aluminum alloys – 3,3468 g;
 - copper and copper-based alloys – 21,06 g.

3. Delivery set

3.1. The microcalculator comes with:

– Microcalculator "Elektronika MK90" bK0.310.095 TU	1 pc.
– Cover 8.634.022	1 pc.
– Case 4.166.005	1 pc.
– Memory cartridge 3.065.003	2 pcs.
– Casing 4.161.016	2 pcs.
– Box 8.865.405	2 pcs.
– Accumulator NKGC-0,45 II C PBC3.579.000 TU	4 pcs.
– Charger "Elektronika ZU-1" in packaging with operating instructions 12.MO.081.159 TU	1 pc.
– User manual 3.055.003 RE	1 pc.
– Packaging box 4.180.267	1 pc.

4. Safety requirements

- 4.1. It is prohibited to open the microcalculator or perform repair or adjustment work when the power supply is connected.
- 4.2. Upon completion of work, and also in the event of malfunctions, turn off the microcalculator and disconnect the power supply first from the 220 V AC network, and then from the microcalculator (when operating from the power supply).

5. Preparing for use

- 5.1. The microcalculator is supplied with a set of uncharged batteries NKGC-0,45.
Charge the batteries using the charger "Elektronika ZU-1" 12.MO.081.159 TU, included in the delivery set.
- 5.2. Install the power supply elements into the microcalculator, for which:
 - 1) remove cover 1 (see Fig. 5.1) of the battery compartment (lightly press on the cover's knurling and slide it in the direction of the arrow indicated on the cover);
 - 2) install the batteries, observing the polarity indicated in the battery compartment;
 - 3) replace the battery cover.

Rear panel of the microcomputer

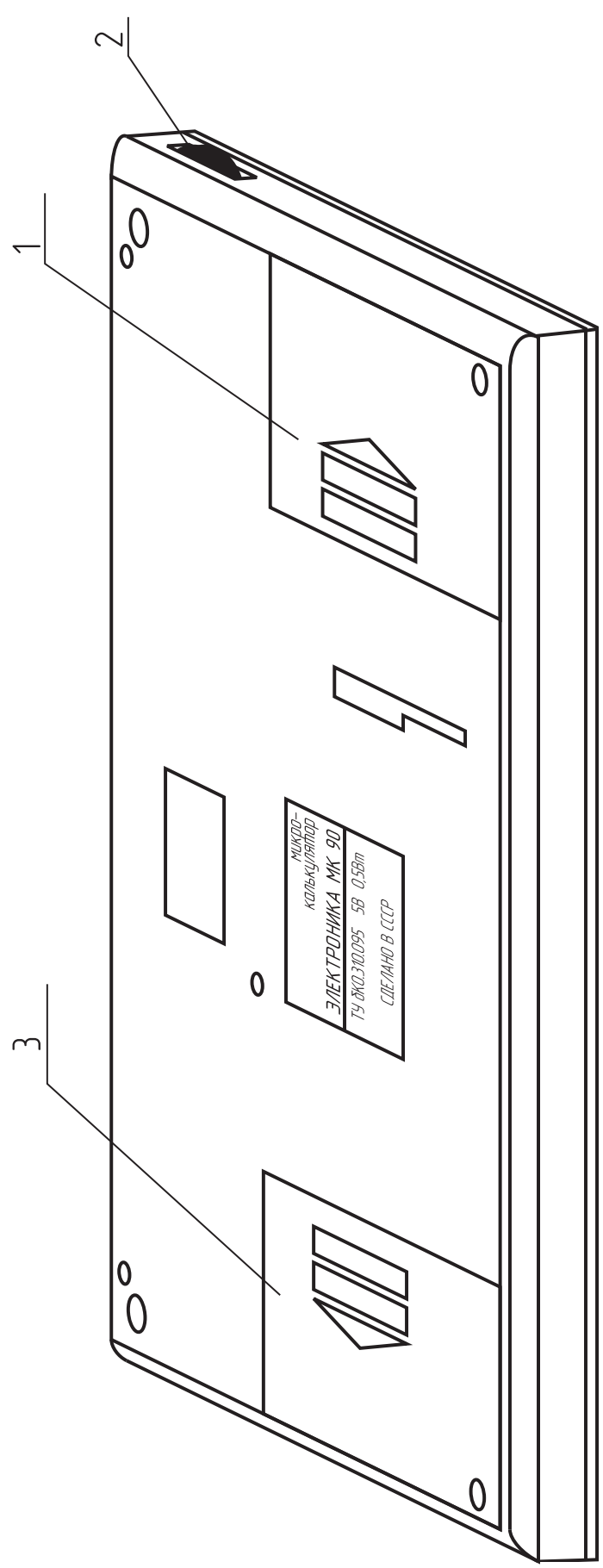


Fig. 5.1

- 1 — battery compartment cover,
- 2 — contrast control,
- 3 — SMP compartment cover.

Front panel of the microcomputer

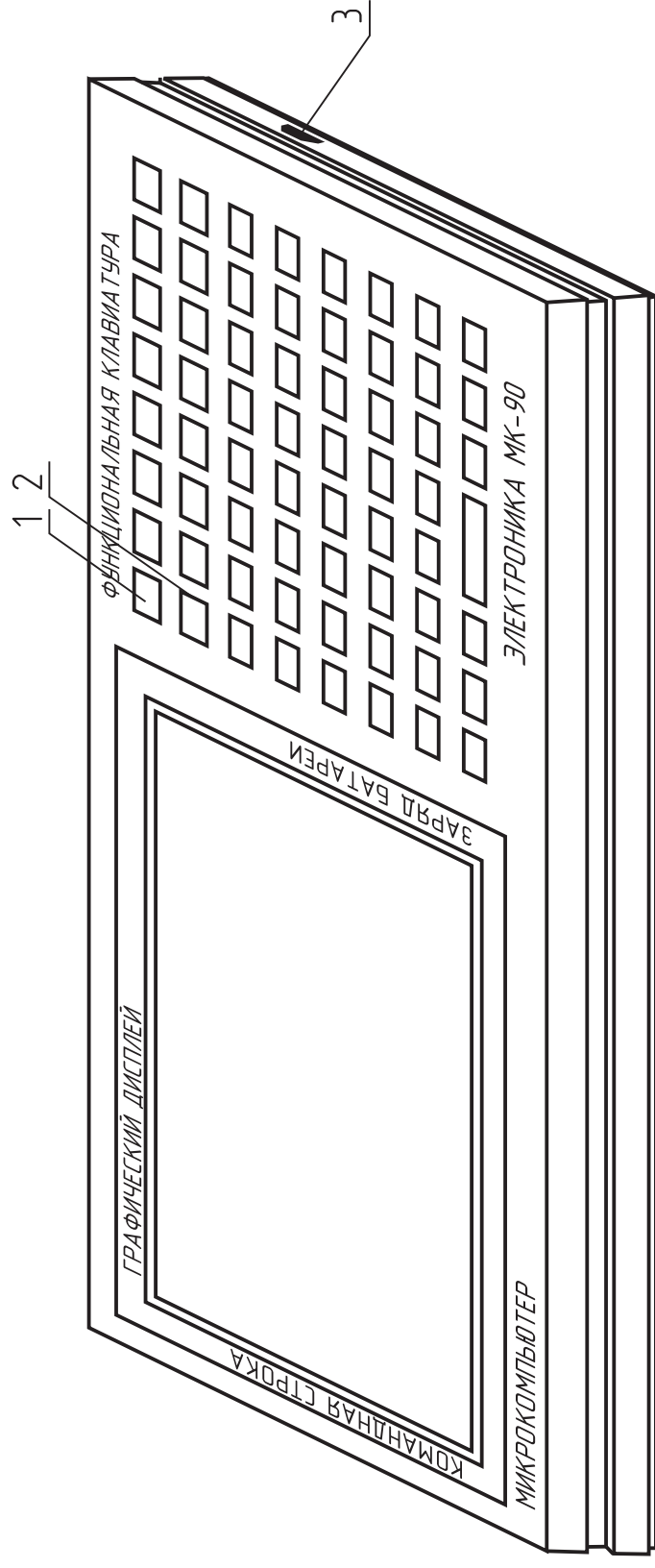
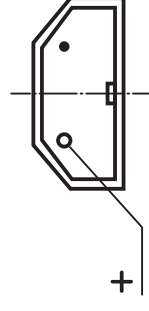


Fig. 5.2

- 1 — power off key,
- 2 — power on key,
- 3 — external power supply connection point.



- 5.3. Press the power-off key 1 and power-on key 2 in sequence (see Fig. 5.2). The LCD screen should display the boot dialog information (see section 6.1.1.), indicating that the microcalculator is ready for operation.
- 5.4. Using contrast control 2 (see Fig. 5.1), set the optimal contrast of the LCD screen.
- 5.5. The microcalculator has connection point 3 for an external power supply with the technical data specified in paragraph 2.15.
- 5.6. To connect the SMP to the microcalculator, you need to:
- 1) Remove cover 3, which closes access to the SMP compartment (press lightly on the cover's knurling and slide it in the direction of the arrow indicated on the cover).
 - 2) Insert the SMP into the slot marked with the number 0 or 1;
 - 3) Carefully move the SMP housing until the SMP plug is fully inserted into the microcalculator socket;
 - 4) Replace the SMP compartment cover.
- 5.7. It is allowed to connect (disconnect) the SMP to (from) the microcalculator only when it is switched off.
- 5.8. The microcalculator provides information about the discharge of the microcalculator's batteries, which is indicated by the display of two additional dots in the upper left and lower right corners of the LCD.
- 5.9. If tests or programs fail to run or run incorrectly when the calculator is powered from an AC mains via an external power source, it is recommended to switch to an autonomous power source.

If after performing item 5.3 the microcalculator does not turn on, it is recommended that after pressing the "On" key, remove one power element (battery) or, if there are none, remove the power source connector from the socket and install it back after 5–10 seconds, then repeat the operation according to item 5.3.

In this case, the SMP should be removed from the socket before turning on the microcalculator and installed after powering up the microcalculator, checking its power and then pressing the "Off" key.

6. Operating procedure


6.1. Keyboard and LCD operating modes

6.1.1. Boot mode

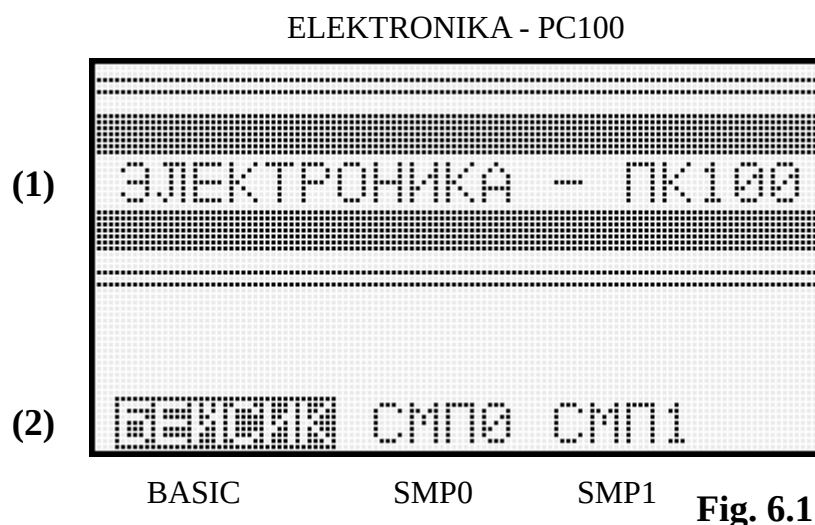
Function keys used when booting the calculator software.

 — move frame to the right;

 — move frame to the left;

 — enter (pressing this key will boot from the selected device).

When the calculator is powered on, the following boot dialog information is displayed on the screen:




The information in line (2) of the screen indicates possible boot devices.

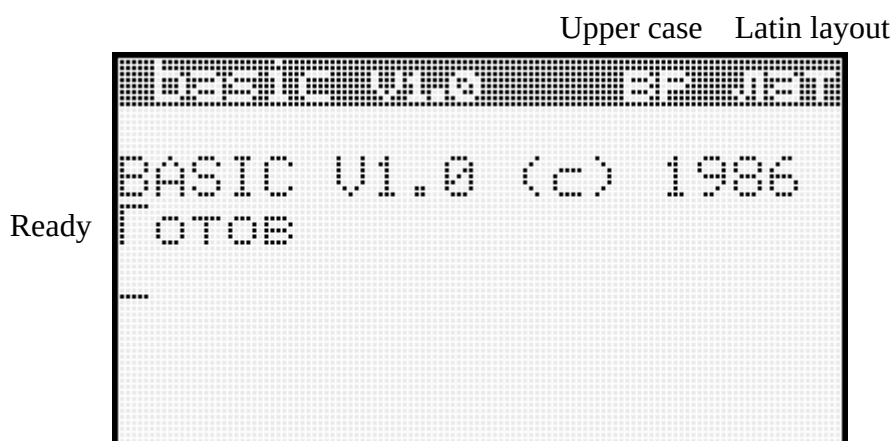
Using the selection frame, moving it to the right or left using the keys described above, select the device from which the software will be loaded.

The boot mode from SMP0 (SMP1) is provided for further expansion of the microcalculator software.

Initially, the frame is placed on the text 


and pressing the key  launches the BASIC interpreter from the internal permanent memory.


After starting the BASIC interpreter, the following information appears on the screen:







6.1.2. Editing keys and special keys



The keys described below perform special functions and are independent of other keys being pressed.


 — enter a space in all keyboard modes.
(space)


 — the last entered character or symbol is deleted and
(back the cursor is moved to the left position.
space)


 — performs input of programs and commands.
(carriage
return)


   — when these keys are entered in sequence, the program execution stops, except
for the commands PLAY, LOAD, SAVE, NAME, FILES, KILL, INIT.

  — when you press these keys in sequence, the sound for pressing keys is
enabled/disabled.

 — used to switch the keyboard to functional mode.

By pressing the key  and any other key in sequence, a BASIC operator (function) is
entered, which can also be entered symbol by symbol.


There is a key  to set the keyboard to upper or lower case mode. When the calculator is
turned on, the upper case mode is always set.


To set the keyboard to Latin or Russian register mode, there is a special key . When the
calculator is turned on, the Latin case mode is always set.


The keyboard operating modes are displayed in the upper indication line of the LCD when
working with the BASIC interpreter.







6.1.3. Keyboard operation in upper and lower case mode


When entering from the keyboard in upper (lower) case mode, the character indicated on the key (above the key) is entered for the next group of keys that do not depend on the Russian/Latin case mode.


* — multiplication sign;
 — colon;


+ — addition sign;
 — semicolon;

? — question mark;
 — slash;



 ...  ...  — special characters;
 ...  ...  — numbers;



= — equal sign;
 — minus sign;



< — less than sign;
 — comma;



> — greater than sign;
 — dot;



A group of upper (lower) case keys, the characters entered from which depend on the set Russian/Latin case mode:



А Я
 ...  Uppercase (lowercase) letters of the Russian alphabet when the Russian case mode is set, or uppercase (lowercase) letters of the Latin alphabet when the Latin case mode is set.



Ч
 " Ч " — uppercase (lowercase) in Russian case mode
 " ч " — in latin case mode for upper case

Ш
 " Ш " — uppercase (lowercase) in Russian case mode
 " ш " — in latin case mode for upper case

Щ
 " Щ " — uppercase (lowercase) in Russian case mode
 " щ " — in latin case mode for upper case

Ъ
 " Ъ " — in Russian lowercase mode
 " ъ " — in latin case mode for upper case


Э
 " Э " — uppercase (lowercase) in Russian case mode
 " э " — in latin case mode for upper case


Ю
 " Ю " — uppercase (lowercase) in Russian case mode
 " ю " — in latin case mode for upper case

ч
 " ч " — in latin case mode for lower case

ш
 " ш " — in latin case mode for lower case


щ
 " щ " — in latin case mode for lower case

Ъ
 " ■ " — symbol "backspace" - in Latin lowercase and Russian uppercase mode

Э
 " | " — in latin case mode for lower case

Ю
 " ‘ " — in latin case mode for lower case

6.1.4. Keyboard operation in Russian/Latin register modes

The Russian/Latin register mode is set by pressing a special key  and affects only the group of character keys that depend on the setting of the Russian-Latin register mode.

When the Russian register mode is set, the symbol indicated on the panel above the key is entered. When the upper register mode is set, this symbol is a capital Russian letter, and when the lower register mode is set, it is a lowercase Russian letter.



When the Latin case mode is set, the character indicated directly on the key is entered. When the upper case mode is set, this character is a capital Latin letter, and when the lower case mode is set, it is a lowercase Latin letter.

Entering Russian text should begin with switching the keyboard to the Russian case mode and end with switching the keyboard to the Latin case mode. This input sequence must be strictly followed. If it is violated, errors are possible. Below are examples of correct and incorrect input of Russian text.


Right	Wrong
PRINT "Р/Л Русский текст Р/Л"	PRINT Р/Л "Русский текст" Р/Л PRINT "Р/Л Русский текст" Р/Л
REM Р/Л Комментарий Р/Л	REM Р/Л Комментарий

Throughout the text, angle brackets (<>) indicate the mandatory presence of information, and square brackets ([]) indicate its possibility or optionality.

6.1.5. Keyboard operation in functional mode


To make working with the keyboard easier, a function key  has been introduced, i.e., entering a directive or function can be done by pressing only two keys in sequence:  and any other key from the table of directives and keys. Directives and functions displayed on the screen have the following form:

<space> directive <space>
или
function (

If the key  was pressed accidentally, the mode is cancelled by pressing the key again.

Directives and keys correspondence table.

Directives		Functions
AUTO — A	REM — X	ABS — 3
CLS — C	RESTORE —]	ATN — 9
DATA — D	RETURN — [COS — 6
DEF FN — ,	RUN — R	EXP — 2
DIS — ↑	SAVE — T	INT — 5
DIM — .	STEP — S	LOG — 7
DEV — ←	STOP — U	RND — 0
DELETE — →	THEN — Z	SGN — 8
DRAW — B	WAIT — W	SIN — 1
EDIT — V	PLAY — ↵	SQR — 4
END — E	PRINT — P	
FILES — \	RANDOMIZE — _	
FOR — F	READ — Y	
GOSUB — ↓		
GOTO — G		
INIT — J		
INPUT — I		
HELP — H		
KILL — K		
LET — L		
LIST — M		
LOAD — @		
LOCATE — Q		
NAME — O		
NEXT — N		

The keyboard diagram for operation in functional mode is given in **Appendix 1**. The labels under some keys (LLIST, LPRINT, LFILES) are reserved for expanding functions. When pressing key  after entering the commands LLIST, LPRINT, LFILES, the calculator goes into "hang" mode, which is exited by turning the calculator on again.

6.1.6. LCD operating mode

Information can be displayed on the LCD screen in symbol display mode and in graphic mode.

6.1.6.1. Character display mode

The LCD screen contains 8 lines with 20 characters in each line. After a line is filled with 20 characters, the twenty-first character automatically moves to the next line, and after all the lines are filled, the entire image shifts up one line, erasing the top line and clearing the last line.

6.1.6.2. Graphic mode

The LCD screen can display points with specified coordinates. The location of the points is specified by the DRAW operator (see section 6.3.2.15), which allows drawing straight lines and circles, erasing straight lines consisting of continuously located points. Graphic coordinates consist of 7680 points: 120 points in the X direction and 64 points in the Y direction (see Fig. 6.3).

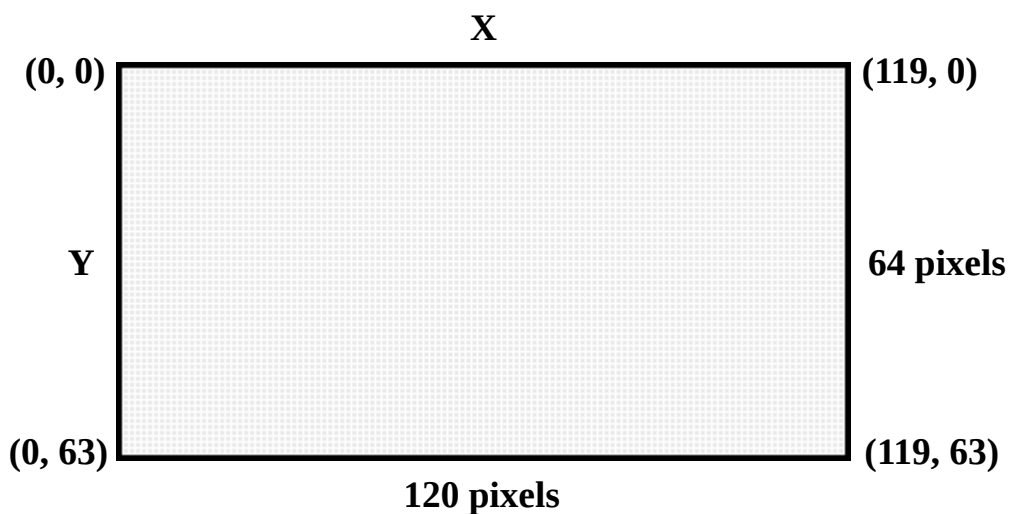


Fig. 6.3

6.2. Calculation functions

6.2.1. Constants and calculation accuracy

In the BASIC language, integer and real constants are defined. Real constants are written in base and exponential forms.

For example, the number 2340,0 in BASIC can be written as the integer 2340 or the real 2340.0 (base form), 23.4E2 or 2.34E3 (exponential form).

To use numbers that have scientific notation, the following rules apply:

— The order may be unsigned if it is positive (1234.56E3).

A negative order must be signed (1234.56E-3);

— The order must be in the range from minus 980 to +980;

The calculation results are displayed on the LCD screen as integers or decimal numbers if nine positions are sufficient to represent the result, i.e. seven digits, a sign (a space in the case of a positive result) and a decimal point. Otherwise, the result is displayed on the LCD screen in exponential notation and occupies a maximum of fifteen positions, i.e. a sign (a space if the result is positive), a decimal point, seven significant digits of the mantissa, the E symbol, an order sign (minus or space) and an order with up to four digits.

Below are some user-entered values with their BASIC-printed equivalents.

Entered value	The value printed by BASIC
.01	.01
9.90000E-3	.0099
999999	999999
1.000000E6	1000000
2.718281828459045	2.718282
0.000000000036218	.36218E-9

BASIC is accurate to at least seven digits, with an accuracy of one in the seventh place. Therefore, the number 4.0000000225 is equivalent to 4.0.

6.2.2. Arithmetic functions of the BASIC microcalculator

- ⌈ raising to a power
- +, - addition, subtraction
- *, / multiplication, division

Limitation of the computational domain:

- 1) Division by 0 results in an error;
- 2) On overflow, when the result does not fit into the calculation range, an error occurs.

Allowed powers:

$$X^Y$$

Y – any real number, error when $X \leq 0$

Examples:

1.	$0.5 \uparrow 0$	1
2.	$-0.5 \uparrow 0$	-1
3.	$(-0.5) \uparrow 0$	ERROR
4.	$0.5 \uparrow 2$	0.25
5.	$0.5 \uparrow (-2)$	4
6.	$(-0.5) \uparrow (-2)$	ERROR
7.	$0.5 \uparrow 0.5$	0.7071068
8.	$(-0.5) \uparrow 0.5$	ERROR
9.	$2 \uparrow (-0.5)$	0.7071068
10.	$(2) \uparrow -0.5$	ERROR

Priority of calculation sequence:

- 1) Elements in curly braces.
- 2) Functions (built-in functions (**see section 6.3.3.**) and user functions (**see section 6.3.2.1.**))
- 3) Exponentiation (\uparrow)
- 4) Multiplication ($*$) and division ($/$)
- 5) Addition ($+$), subtraction ($-$)
- 6) Comparison operators \leq , \geq , $<$, $>$, $><$, $=$ etc.

Comparison operators

Comparison operators can only be used with the IF statement (see section 6.3.2.7.).

=	equal to
<>, ><	not equal
<	less than
>	greater than
=>, >=	equal to or greater than
=<, <=	equal to or less than

Example: $A+B \neq 0$. The result of adding A and B is not zero.

6.2.3. Variables

A variable is a value whose value can change during the execution of a program. A variable is designated by a single letter or a single letter (capital Latin) followed by a digit.


For example:

I, B3, C8

Just like constants, variables can be integer or real.

6.3. Microcalculator BASIC commands

6.3.1. Immediate mode commands

Some BASIC commands can be entered on a line without a number. In this case, the command is executed after pressing the  key at the end of the line. The result of the execution is stored for later use, but the command is lost. This mode is called immediate.

The immediate mode is necessary when editing and debugging programs written in program mode.





In the given command formats, square brackets [] indicate the possibility or optionality of the presence of information.

A brief description of the BASIC commands of the microcalculator is given in **Appendix 2**. Error messages when entering commands are listed in **Appendix 3**.

6.3.1.1. **AUTO** command

Function: Sets BASIC to automatic line numbering mode during program entry.

Syntax: `AUTO [<line number>][, [<stepping>]]`

Executing this command causes line numbers to be generated each time the  key is pressed at the end of a program line. The numbering starts with <line number>, and subsequent line numbers differ by the amount specified by <stepping>. If <line number> or <stepping> is not specified, the default value is 10. Auto-numbering can be stopped by pressing the keys    .

Example:

```
AUTO 100,50
```

Generate line numbers with a step of 50 and a starting number of 100 (100, 150, 200, ...).

The minimum line number and minimum step are 1. The maximum line number is 8191. The line size must not exceed 80 characters (4 lines of the LCD screen).

6.3.1.2. **DELETE** command

Function: Erases the line with the specified number

Syntax: `DELETE <line number N1>[, <line number N2>]`

N1 – the number of the first line to be deleted

N2 – last line number

The DELETE command must have an argument.

If only one line number is given in the DELETE command, the corresponding line is deleted. If two numbers are given, separated by a comma, the corresponding two lines and all the lines between them are deleted.

Example:

```
DELETE 54
```

Line 54 is erased

```
DELETE 56, 876
```

Lines 56 through 876 inclusive are erased.

If in the DELETE command the starting line number N1 is greater than the ending line number N2, BASIC responds "Готов" (Ready), but no lines are deleted.

```
DELETE 1, 8191
```

All program lines are erased.

6.3.1.3. **EDIT** command

Function: Puts BASIC into the mode of editing the program located in the command buffer.

Syntax: `EDIT <line number>`

The EDIT command is used in immediate mode to switch BASIC to line editing mode.

Editing Russian text is not allowed.

The following keys are used for editing.


 — move the cursor to the left position;

 — move the cursor to the right position;

 — switching the editor to insert-replace mode;



 — shift line left to cursor position;

  — placing a cursor at the beginning of the edited line;

  — move to the previous word in the line;

  — delete characters from a line, starting from the cursor position to the end of the line;

  — move to the next word in the line;


  — placing a cursor after the last character of a line;

 — write the edited line to the command buffer and switch to command mode.

6.3.1.4. **HELP** command

Function: Displays information on the LCD screen about the composition of BASIC language commands, their purpose, syntax and binding to the functional keyboard (in Russian)

Syntax: HELP *
 HELP <topic>

HELP information is printed line by line at the press of a key  .

If you need information about a specific command or function, type HELP <topic>.

Information on all commands and functions can be obtained by typing HELP *.

Examples:

```
HELP *  
HELP DRAW/A  
HELP HELP
```


6.3.1.5. **LIST** command

Function: Prints program lines to the screen

Syntax: LIST [<line number N1>[,<line number N2>]]

To get the program text on the screen, you need to specify the LIST command. After printing, BASIC will go to "Готов" (Ready).

To print a single line, specify the line number in the LIST command. To print a series of lines, specify the numbers of the starting and ending lines in the LIST command; in this case, all lines between them will be printed in addition to the lines specified.

If there are more than 7 program lines between the specified numbers, then 7 lines are displayed on the screen at once, and then one line at a time after pressing the key  .

Note: When calling the LIST command on a non-existent line number(s), the closest existing line(s) to it (them) is displayed on the screen.

Example:

```
LIST 37
```

Line 37 is printed

```
LIST 37,126
```

Lines 37 through 126 inclusive are printed.

```
LIST
```

The entire program is printed, starting from the line with the lower number.

6.3.1.6. **MEM** command

Function: Shows the current memory allocation.

Syntax: MEM

Example:

MEM

Prints memory allocation in the following form:

Memory allocation
(in bytes)

Program:

Free:



This command provides information about the size of the typed program text, while it is necessary to take into account that the amount of memory occupied by a variable or array element (**see section 6.3.2.2.**) is 6 bytes.

6.3.1.7. **INIT** command

Function: Erasing all files from SMP and formatting it

Syntax: INIT ["<device name>:"]

If <device name> is not specified, then the working SMP is erased and formatted (**see section 6.3.1.12.**). All information from the SMP disappears.

Examples:

```
INIT "SM0 : "  
INIT
```

Note: the INIT command, like all subsequent commands intended for working with the SMP, must be used only when the SMP is connected.

6.3.1.8. **SAVE** command

Function: Saves files on SMP

Syntax: `SAVE "<file specification>"`

This command saves a BASIC program to the SMP. To do this, specify the device name, file name and file type. If the device name is omitted, it is considered a working SMP (see section 6.3.1.12.). If the file type is missing, it writes the type "BAS".

Example 1:

```
SAVE "ADDR"
```

(the ADDR.BAS file is written to the working SMP)

Example 2:

```
SAVE "SM1:PROG.ABC"
```

(file PROG.ABC is written to SMP1)

The file name must be no more than 6 characters long, and the file type must be no more than 3 characters long.

6.3.1.9. **NAME/AS** command

Function: Changes file names on SMP

Syntax: `NAME "<old file specification>" AS "<new file specification>"`

Both <old file specification> and <new file specification> are specified by the device name, file name and file type. The device name may be omitted if it is a working SMP.

The file name specified in <old file specification> must be the name of an existing file. If the new file name already exists, the file with that name will be deleted from the SMP, and its name will be assigned to the file with the old file specification. This command only changes the name of the specified file, but does not rewrite it to another area of the SMP.

Example:

```
FILES "SM1:"  
PROG1 .BAS      2      4  
PROG3 .BAS      5      6  
ГOTOB
```

```
NAME "SM1:PROG1.BAS" AS "SM1:PROG2.BAS"  
ГOTOB
```

```
FILES "SM1:"  
PROG2 .BAS      2      4  
PROG3 .BAS      5      6  
ГOTOB
```

Note: the typed or corrected program must be saved on the SMP before starting execution with the RUN command. This ensures more compact and reliable storage of the program.

6.3.1.10. **LOAD** command

Function: Loads the program recorded in the SMP into memory

Syntax: `LOAD "<file specification>" [,R]`

The file specification must specify the device name, file name, and file type. If the device name is omitted, the working SMP is assumed; if the file type is not specified, ".BAS" is assumed.

If the LOAD command is executed without specifying "R", then after loading the program BASIC goes into command mode. However, if "R" is specified, then as soon as the program loading is completed, its execution begins immediately.

It is important to remember that as a result of the LOAD command, all variables are cleared, and any program that was previously in the command buffer is destroyed.

Example:

```
LOAD "SM1:PROG1.BAS"
```

6.3.1.11. **KILL** command

Function: Deletes files from SMP

Syntax: `KILL "<file specification>"`

The KILL command can be used to delete files of any type from the SMP. To delete a file, the full file specification must be specified if the file to be deleted is located on a device other than the working one. Otherwise, it is enough to specify the name and type of the file.

Example:

```
KILL "SM1:GRAPH.BAS"
```

The GRAPH.BAS file is deleted from the SM1 device.

6.3.1.12. **DEV** command

Function: Sets <device name> as the working SMP

Syntax: `DEV ["<device name>:"]`

If <device name> is not specified, then SMP0 is selected as the working one. Using the DEV command allows you to not specify <device name> in <file specification> in all commands that work with SMP.

Example 1:

```
DEV "SM1:"
```

Sets SMP1 as the working one.

Example 2:

```
DEV
```

Sets SMP0 as the working one.

6.3.1.13. **FILES** command


Function: Prints file names on <device name>

Syntax: `FILES ["<device name>:"]`

If <device name> is not specified, this command prints a list of all files located on the SMP selected as the working one.

If <device name> is present, then the FILES command prints a list of all files on the specified SMP.

In addition to the file names, the second and third columns of the list print, respectively: the file length in blocks of 512 bytes and the starting address of this file on the SMP.

The list of files is divided and displayed on the screen in pages after pressing the  key.

Example 1:

```
FILES "SM1 : "
```

Prints a list of all files located on SMP1.

Example 2:

```
FILES
```

Prints a list of all files located on the working SMP.

When typing the command line, there may be errors - missing quotation marks, colons, or an incorrectly specified symbolic name of the device, as well as more than six characters in the file name.

In all these cases, the same error message is displayed on the command line.

Notes:

1. Before working with the SMP (commands: INIT, LOAD, SAVE, KILL, FILES), make sure that the required SMP are inserted into the required slots of the microcalculator.
2. Change the SMP only when the microcalculator is switched off, otherwise the microcalculator may freeze or the SMP commands may not be executed correctly.




6.3.1.14. **RUN** command

Function: Runs the program

Syntax: RUN

The RUN command starts the program in program mode.

When the RUN command is used to rerun the program, all variable values are erased from memory. If the RND(X) function is included in the program, its initial value is restored.

You can stop the program by pressing the    keys in sequence.

6.3.2. Program mode commands

Most often, BASIC programs are written in program mode. Where each line begins with a number indicating the sequence of execution. In program mode, commands are called operators. In this mode, the execution of operators is delayed until a special command is given (for example, RUN or GOTO).

Examples of programs are given in **Appendix 4**.

A program loaded from the SMP or typed by the user on the keyboard is deleted when the calculator is turned off, or if another program is loaded from the SMP. When entering a line with an existing number, the latter is written in place of the line with this number. Several operators can be written in one line, separated by a colon and executed sequentially.

6.3.2.1. **DEF FN** operator

Function: Defining user functions

Syntax: DEF FN<letter> (<variable>)=<expression>

In some programs, there is a need to calculate the same mathematical expression, often with different data. In BASIC, it is possible to define functions or expressions and call them in the same way as mathematical functions, i.e. sine, cosine, square root, etc.

A user-defined function is identified by a three-letter name and an argument in parentheses. The first two letters of the function name are FN; the third letter can be any letter of the Latin alphabet. Up to 26 functions can be defined in this way.

Such a function must be defined before it can be used, and is defined using the DEF function definition statement:

```
DEF FNA (параметр) = выражение,
```

Where A is a letter of the Latin alphabet.

For example, using the operator

```
10 DEF FNX (S)=S-2+4
```

the operator

```
20 LET R=FNX (4)
```

will be calculated as

```
LET R=4-2+4,
```

and operator

```
20 LET R=FNX (2)
```

will be calculated as

```
LET R=2-2+4.
```

The parameters specified in a user-defined function are formal parameters. The parameters specified when calling the function are actual parameters.

In our example, S is a formal parameter, 4 and 2 are actual parameters.

The argument in the coin parentheses can be any valid expression; the value of the expression is substituted for the function argument.

In the following example, the FNX function on line 10 will square the number substituted for the function argument. On line 30, the expression 2+A yields 4; 4 is then squared and the value of the FNZ(X) function is printed.

Example:

```
10 DEF FNZ (X)=X^2
```

```
20 LET A=2
```

```
30 PRINT FNZ (2+A)
```

```
ГОТОВ
```

```
RUN
```

```
16
```

```
ОСТ В СТРОКЕ 30
```

```
ГОТОВ
```

Another example:

```
10 DEF FNA (Z) =Z+A+B
20 DATA 2,4
30 READ A,B
40 LET F=FNA (9)+1
50 PRINT F
ГОТОВ

RUN
16
OCT B CTPOKE 50
ГОТОВ
```

The function can be used recursively, as shown in the following example. The expression on line 30 is evaluated as $(2+(2*4))$ before it is squared.

Example:

```
10 DEF FNA (X) =X^2
20 LET A=2
30 PRINT FNA (2+A*FNA (2) )
ГОТОВ
RUN

100
OCT B CTPOKE 30
ГОТОВ
```

If the same function is defined multiple times, the first definition is used and subsequent definitions are ignored.

Example:

```
10 DEF FNX (X) =X^2
20 DEF FNX (X) =X+X
30 PRINT FNX (6)
40 END
ГОТОВ

RUN
36
OCT B CTPOKE 40
ГОТОВ
```

A formal parameter of a function may not be used in a function expression. In the following example, substituting an actual parameter for a formal parameter has no effect on the value of the function.

Example:

```
10 DEF FNA (X) =4+2
20 LET R=FNA (10)+1
30 PRINT R
ГОТОВ

RUN
7
ОСТ В СТПОКЕ 30
ГОТОВ
```

6.3.2.2. DIM operator

Function: Declares an array

Syntax: DIM <array id1> (<index1>[,<index2>])
[,<array id2> (<index1>[,<index2>])...]

DIM reserves memory space for a data array. The maximum value of the index list is 255. The maximum number of all program arrays is no more than 1960. Any variable can be an array identifier.

The maximum value of the dimension is 2, i.e. it is allowed to work with one-dimensional and two-dimensional arrays. The operator must contain the maximum possible indexes:

```
DIM A1 (4,4) , B (10)
```

Since indexes in BASIC start at 0, the above statement will reserve space for array A1, which can hold 25 elements (5*5), and array B, which can hold 11 elements. It is not allowed to specify the dimension of a previously used variable.

For example:

```
10 LET A3=C-C+B/3
.
.
.
40 DIM A3 (2,5)
```

Will generate an error message.

```
ОШ 6 СТП 40
```

Usage.

Let's enter the following program:

```
10 DIM A(19)
20 FOR I=0 TO 19
30 LET A(I)=0
40 NEXT I
60 FOR I=0 TO 4
70 INPUT A(I)
85 NEXT I
```

Run it:

RUN 

?5
?6
?7
?8
?9

Input A(I)

OCT B CTPOKE 85
ГОТОВ

Stopping the program and switching to command mode

Let's look at the result of executing the program:

```
PRINT A
5
PRINT A(1)
6
PRINT A(2)
7
PRINT A(3)
8
PRINT A(4)
9
PRINT A(5)
0
PRINT A(6)
0
PRINT A(29)
```

ОШ 6 CTP 40
ГОТОВ

After entering the five constants, the program execution stops. The PRINT statement is then used in immediate mode to examine the initial elements of the array. Attempting to print the contents of an array cell that is outside the limits set by the DIM statement produces an error message.

Below is a continuation of the program discussed, demonstrating automatic output.

```
10 DIM A(19)
20 FOR I=0 TO 19
30 LET A(I)=0
40 NEXT I
60 FOR I=0 TO 4
70 INPUT A(I)
90 NEXT I
95 REM - LINES 100-130 PRINT THE LIST SEQUENTIALLY
100 FOR I=0 TO 6
110 PRINT "A("I")", A(I)
120 NEXT I
130 STOP
135 REM - LINES 140-160 PRINT PART OF THE LIST
136 REM - IN REVERSE ORDER
140 FOR I=6 TO 0 STEP -1
145 PRINT "A("I")", A(I)
160 NEXT I
ГОТОВ
```

```
RUN
? 1
? 2
? 3
? 4
? 5
A( 0 )    1
A( 1 )    2
A( 2 )    3
A( 3 )    4
A( 4 )    5
A( 5 )    0
A( 6 )    0
```

```
OCT B CТPOKE    130
ГОТОВ
```

```
GOTO 140
A( 6 )    0
A( 5 )    0
A( 4 )    5
A( 3 )    4
A( 2 )    3
A( 1 )    2
A( 0 )    1
```

```
OCT B CТPOKE    160
ГОТОВ
```


6.3.2.3. **END** operator

Function: The last operator of the program

Syntax: END

The END statement should be included in programs intended for repeated use. It should be placed at the end of the program, i.e., on the line that is the logical end of the program.

6.3.2.4. **FOR-TO-STEP/NEXT** operator

Function: FOR-TO-STEP used to indicate the starting point of a program cycle

Syntax: FOR <variable>=E1 TO E2 [STEP E3],
 where E1, E2, E3 – expressions

Function: NEXT used to indicate the end point of a program loop

Syntax: NEXT <variable>

All statements between FOR and its corresponding NEXT statement are executed cyclically according to the conditions specified in the FOR statement. The variable following the FOR statement is considered a control variable. The same variable must be present in the NEXT statement that defines the end of the loop.

The minimum and maximum values represented by the expressions to the left and right of the TO keyword in the FOR statement are the scope of the control variable. The expression following the STEP keyword specifies how much the control variable will change after each pass through the loop. If E3 is not specified, the loop step is 1.

Example:

```
10 LET A=5
20 FOR B=1 TO 5
30 PRINT A+B
40 NEXT B
ГОТОВ

RUN
6
7
8
9
10

OCT B ЦТРОКЕ     40
ГОТОВ
```

In line 20, the variable "B" is assigned the values 1, 2, 3, 4, 5. Since it cannot have all of these values at once, a loop is created, starting with the FOR statement in line 20 and ending with the NEXT statement in line 40. The statements inside the loop are executed five times, with "B" being given a new value each time, in increments of one, and the value of A+B is printed. The NEXT statement causes the loop to jump to line 20 until "B" reaches its final value of 5.

Each FOR statement within a program must be followed by a NEXT statement; NEXT cannot be used without a preceding FOR statement.

6.3.2.5. GOSUB, RETURN operators

Function: Allows you to organize a transition to a subroutine

Syntax: GOSUB N
 where N – line number to which control is transferred

Function: The last statement of a subroutine. Transfers control to the statement following the corresponding GOSUB statement.

Syntax: RETURN

Often, a program needs to write a group of statements (a subroutine) only once, and then repeatedly access the subroutine from different points in the program. The GOSUB statement is used to jump to the first statement in the program. The subroutine ends with the RETURN statement.

The RETURN statement must be used with GOSUB. However, a single RETURN statement is sufficient for the case where multiple GOSUBs are used to branch to a single program.

For example*:

```
5 PRINT " X"; " X^2"; " X^3"; " X^4"
10 FOR A=1 TO 3
15 LET X=A
20 GOSUB 50
25 NEXT A
28 FOR A=1 TO 3
30 LET X=A^2
40 GOSUB 50
45 NEXT A
48 STOP
50 FOR J=1 TO 4
51 PRINT X^J;
52 NEXT J
53 PRINT
54 RETURN
55 END
ГOTO B
```

RUN

X	X^2	X^3	X^4
1	1	1	1
2	4	8	16
3	9	27	81
1	1	1	1
4	16	64	256
9	81	729	6561

```
OCT B CTPOKE 48
ГOTO B
```

The subroutine (lines 50–54) is entered by the GOSUB statements on lines 20 and 40, and exited by the RETURN statement on line 54.

6.3.2.6. GOTO operator

Function: Transfers control to line N and continues program execution from that line.

Syntax: GOTO N

The GOTO operator makes an unconditional jump to the line with the specified number. The program execution continues sequentially, starting with the operator to which the jump was made.

***Editor's note:** the authors of the original manual chose a very bad example. BASIC in MK90 does not allow for an adequate table construction, causing discrepancies in columns visible in the example. In addition, the value of X^5 does not fit into the line. The original program contains many errors, which is why it had to be significantly edited.

Example:

```
10 DATA 1, 2, 3, 4, 5      (see sect. 6.3.2.11. – desc. of DATA and READ operators)
```

```
15 READ X
```

```
20 PRINT X+1
```

```
40 GOTO 15
```

```
ГОТОВ
```

```
RUN
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
ОШ 20 CTP 15
```

```
ГОТОВ
```

Statements 15, 20, and 40 represent the loop in the example above. These three statements are continuously executed with a new value for X each time the line is executed until the last value of the DATA statement constant is used.

The program execution ends with an error message indicating that all constant values have been used.

In the following example, the GOTO statement jumps to the INPUT statement so that the program loops.

```
10 INPUT A                  (see section 6.3.2.8. – description of the INPUT operator)
```

```
20 PRINT A
```

```
30 LET A=A^A
```

```
40 PRINT A
```

```
50 PRINT
```

```
60 GOTO 10
```

```
ГОТОВ
```

```
RUN
```

```
?2
```

```
2
```

```
4
```

```
?3
```

```
3
```

```
27
```

```
?4
```

```
4
```

```
256
```

```
?5
```

```
5
```

```
3125
```

```
?^P
```



```
ГОТОВ
```

When writing multiple statements on a line, separated by the ":" character, the GOTO statement must be the last statement on the line.

6.3.2.7. IF/THEN operator

Function: Provides a conditional jump to the program

Syntax: IF <expression> THEN <operator>
 THEN <line number>
 GOTO <line number>

The word THEN may be followed by any statement, including a second IF statement. When THEN is followed by a line number, the IF-THEN statement acts like an IF-GOTO.

Note: The letter "T" must not be used as a variable name in an IF statement.

When BASIC encounters an IF statement, it evaluates the expressions and compares them according to the requirements of the logical operation, as shown in Table 6.1.

Table 6.1.

Logical operation of the BASIC language	Example	Meaning
=	A = B	A equals B
<	A < B	A is less than B
<=	A <= B	A is less than or equal to B
>	A > B	A is greater than B
>=	A >= B	A is greater than or equal to B
<>	A <> B	A is not equal to B

If the condition is true, the rest of the statement starting with THEN or GOTO is executed. If the condition is false, the next statement after the IF statement is executed.

Example:

IF GOTO <line number> is used

```
10 FOR A=1 TO 10
20 LET X=INT(A^2)
25 IF X>20 GOTO 40
30 PRINT X
35 NEXT A
40 PRINT "Value of A=" A
55 STOP
ГOTOB
```

```
RUN
1
4
9
16
Value of A= 5
```

```
OCT B CTPOKE 55
ГOTOB
```

If we replace line 25 with `X>9 GOTO 40`, the loop will end when `A=4`.

```
RUN
1
4
9
16
Value of A= 4
```

```
OCT B CTPOKE 55
ГOTOB
```

The following example uses IF-THEN <operator> on lines 40, 50:

```
10 REM - ACHIEVEMENT ASSESSMENT PROGRAM
25 INPUT A,B,C,D
30 LET X=(A+B+C+D)/4
40 IF X>=70 THEN PRINT X; "Satsf. rating"
50 IF X<70 THEN PRINT "Undrach. in class"
55 GOTO 25
ГOTOB
```

```
RUN
?55,67,78,89
72.25 Satsf. rating
```

```
?55,44,32,21
Undrach. in class
```

```
?~P   
ГOTOB
```


The IF statement can appear anywhere on a line with multiple statements, except when THEN is followed by a GOSUB or GOTO statement. In this case, the IF statement must be the last statement on the line. The IF-THEN-GOSUB statement provides a branch to a subroutine and must be the last statement on the line.

6.3.2.8. INPUT operator

Function: Performs keyboard input during program execution.

Syntax: INPUT [V1,V2, . . . ,VN]

When executing an INPUT statement, BASIC prints a question mark and waits for the user to enter from one to N numeric values.

When executing an INPUT statement, numeric values must be separated by commas and end with a . Only one question mark is printed for each INPUT statement.




If insufficient data is entered, BASIC prints:

```
ОИИ 121 CTP NNN
```

If too much data is entered, BASIC prints:

```
ОИИ 122 CTP NNN,
```




Where NNN – line number of the INPUT statement.

In both cases, BASIC prints the second question mark and waits for the input to be repeated. The input line must be retyped. Exit to command mode is performed by pressing    in response to the last parameter.

The INPUT statement can be written anywhere in the program, and can also be included in a line with several statements.

Example:

```
10 INPUT A,B,C
20 GOTO 10
ГОТОВ
```

```
RUN
?1,2,3
?4,5,6
?7,8,9
?¬P   
ГОТОВ
```

6.3.2.9. LET operator

Function: Assigns the value of an expression to a variable

Syntax: LET <variable>=<expression>

The following example illustrates the use of the LET operator:

```
10 LET A=1
20 LET B=2
30 LET C=A+1
40 LET X=A+B+C+1
```

After executing these statements, the value of X will be 6. The LET statement can be written anywhere in the program and on a line with several statements.

6.3.2.10. **PRINT** operator

Function: Provides data output to the LCD

Syntax: PRINT [function] [<list>]

A list can contain an expression and/or a text string. List items are separated by "," or ";", which specify the print format.

The PRINT statement without a list is used to output an empty string.

PRINT can be used for calculations: the expression contained in the list is evaluated and its value is printed. The resulting value is not saved for later use.

```
10 LET A=1
15 LET B=5
20 PRINT A+B
ГОТОВ

RUN
6

OCT B CTPOKE 20
ГОТОВ
```

The values of expressions separated in the PRINT statement by the "," character are printed four positions to the right of the previous one. If a more compact arrangement of the value is required during printing, the ";" character is used. In this case, each value is printed two positions to the right of the previous one, provided that this line is not yet finished. Below is an example of specifying print formats.

```
10 DATA 1,2,3
20 READ A,B,C
30 PRINT A;B;C;
40 PRINT A;B;C
50 PRINT A,B,C
60 PRINT A,;B,C
ГОТОВ

RUN
1 2 3 1 2 3
1 2 3
1 2 3

OCT B CTPOKE 60
ГОТОВ
```

To print messages, comments, or any string of characters, the PRINT statement has the functions listed in Table 6.2.

Table 6.2.

S	Changing the size of symbols and characters
Q	Setting the orientation of symbols and signs
Y	Setting the print direction
Z	Setting the distance between characters and between lines
N	Output of information in the negative
P	Output of information in the positive

The list of functions is located after the PRINT operator and is enclosed in angle brackets <>, the functions are separated by the symbol ";". The text to be printed is enclosed in quotation marks.

The PRINT operator functions have the following formats:

1) Function: S

Syntax: S<symbol size in X, symbol size in Y>

The symbol size parameter means how many times the size of the symbol exceeds the standard sizes (5*7). By default, the parameter is "1".

For example, the symbol "M" can be printed using sizes from 1 to 3 as follows:

```
10 CLS
20 DRAW O60,32
30 FOR I=1 TO 3
40 PRINT <SI,I>"M";
50 NEXT I
```

2) Function: Q

Syntax: Q<orientation type>

The orientation type parameter takes values 0, 1, 2, 3. Here is an example of printing the letter "F" in different orientations:

```
10 FOR I=0 TO 3
20 PRINT <QI>"E";:PRINT I
30 NEXT I
```

3) Function: Y

Syntax: Y<print direction>

The print direction parameter takes values 0, 1, 2, 3.

0 – print characters from left to right

1 – print characters top to bottom

2 – print characters from right to left

3 – print characters from bottom to top

It is recommended to use the Y function with LOCATE, since characters that extend beyond the screen are not displayed.

Example:

```
PRINT <Y1>"ABW"
```

4) Function: Z

Syntax: Z<character spacing, line spacing>

Output information with a specified distance between characters and lines. Valid until the end of the PRINT statement.

5) Function: N

Syntax: N

Outputs information in negative. Valid until the <P> function or until the end of the PRINT statement.

6) Function: P

Syntax: P

Outputs information in positive. Valid until the <N> function or until the end of the PRINT statement.

6.3.2.11. READ, DATA, RESTORE operators

Function: Variables from V1 to VN inclusive are assigned the values of the corresponding constants of the DATA statement.

Syntax: READ V1, V2, . . . , VN

Function: Determining the values of constants N1,...,NN for the READ statement

Syntax: DATA N1, . . . , NN

Function: Sets the pointer to the beginning of the DATA constant string.

Syntax: RESTORE

The LET statement was used to assign values to single variables in the previous examples; more LET statements were used for more variables.

However, in programs with many variables, READ and DATA should be used to define them.

The DATA statement introduces a numeric constant or group of constants into the program. The READ statement sequentially associates variable names with the value of constants, which are specified by DATA statements.

READ and DATA statements must accompany each other in user programs, but they do not have to be paired. If one or more READ statements contain nine variables, then one or more DATA statements must contain at least nine constants (an exception is the use of the RESTORE statement).

In the following example, all the constants are specified by a single DATA statement. They are used by READ statements on different lines.

```
10 DATA 1, 5, 3, 7, 9
20 READ A, B
30 LET X=A+B
40 PRINT A, B, X
50 READ U, Q, R
60 LET X=X+U+Q+R
70 PRINT X, U, Q, R
GOTOB

RUN
  1      5      6
 25      3      7      9

OCT B CTPOKE 70
GOTOB
```

When executing a program, BASIC ignores the DATA statement until it encounters a READ statement. It then searches for the DATA statement and finds it on line 10, the first line of the program.

BASIC takes the constant values sequentially and binds them to the variables in the READ statement, which are also selected sequentially: variable A is assigned the value 1, B is assigned the value 5. Having set the pointer to the next data element 3, it returns to line 30, i.e. to the next unexecuted statement.

Line 50 contains another READ statement that contains three new variables. Now BASIC accesses the pointer to obtain the next unused constant: variables U, Q, R are assigned the value of the constants 3, 7, 9.

There are several error messages associated with the DATA and READ statements:

- 19 – invalid variable in READ statement list.
- 20 – the data in the READ statement list is exhausted.
- 21 – invalid format of DATA statement.
- 123 – non-existent variable.

Regarding code 123, it should be remembered that every program variable must be defined by either a READ statement, a LET statement, or a FOR statement before it can be used in an expression or in a PRINT statement list.

The RESTORE statement enables the reuse of constants across DATA statements, starting with the lowest-line-numbered DATA statements in the program.

Example:

```
40 DATA 1, 2
50 READ A, B
60 PRINT A, B
70 RESTORE
80 READ C, D
90 PRINT C+D, C, D
GOTOB

RUN
1      2
3      1      2
```

If you did not have a RESTORE statement, you would get an error message on line 80 indicating that there were no constants for the READ statement.

Example:

```
10 DATA 1, 2
20 READ A, B
30 PRINT A, B
40 RESTORE
50 DATA 3, 4
60 READ C, D
70 PRINT C+D, C, D
GOTOB

RUN
1      2
3      1      2
```

In this example, the RESTORE statement causes the second READ statement on line 60 to receive the value of the constants in the first DATA statement on line 10, rather than the second statement on line 50.

Example:

```
10 DATA 1, 2
20 READ A, B
30 PRINT A, B
40 DATA 3, 4
50 READ C, D
60 PRINT C+D, C, D
GOTOB

RUN
1      2
7      3      4
```

The DATA statement cannot be included in a multiple-statement line; it must be the only statement on a numbered line.

The READ statement can be written anywhere in a multiple-statement line.

The RESTORE statement can be used anywhere in a program; it can be included in a multiple-statement line.

The RESTORE statement has no effect in programs without DATA and READ statements.

6.3.2.12. REM (REMARK) operator

Function: Allows you to print comments to the program

Syntax: REM - COMMENT
 REMARK - COMMENT

Comments can consist of any characters.

REM statements have no effect on program execution, but they do take up memory.

When used in a line with multiple statements, REM must be written last, since BASIC ignores everything that comes after REM on a line.

The REM operator can be replaced by an opening quote <'> (B/H key and 7).

6.3.2.13. **RANDOMIZE** operator

Function: Allows you to get a series of random numbers in a program using the RND function

Syntax: RANDOMIZE

When the RANDOMIZE statement is executed, the RND function selects a seed value to generate a random number.

Example:

```
10 REM - RANDOM NUMBERS USING RANDOMIZE
15 RANDOMIZE
25 PRINT "Random numbers:"
30 FOR I=1 TO 4
40 PRINT RND (0);
50 NEXT I
60 END
ГОТОВ
```

```
RUN
Random numbers:
.5157776 .5863953
.8763733 .980682
```

```
OCT B CTPOKE 60
ГОТОВ
```

```
RUN
Random numbers:
.837677 .9383435
.2109681 .6407166
```

```
OCT B CTPOKE 60
ГОТОВ
```

```
RUN
Random numbers:
.2499695 .7420959
.2028503 .5382385
```

```
OCT B CTPOKE 60
ГОТОВ
```

When we remove the RANDOMIZE operator we get the following:

```
RUN
Random numbers:
.1002502 .9648132
.8866272 .6364441
```

```
OCT B CTPOKE 60
ГОТОВ
```

```
RUN
Random numbers:
.1002502 .9648132
.8866272 .6364441
```

```
OCT B CTPOKE 60
ГОТОВ
```

```
RUN
Random numbers:
.1002502 .9648132
.8866272 .6364441
```

```
OCT B CTPOKE 60
ГОТОВ
```

6.3.2.14. CLS operator

Function: Clears the LCD screen and moves the cursor to the upper left position

Syntax: CLS

Used to clear the LCD screen as needed.

After the operator is executed, the cursor moves to the position (0, 0), the cursor image is absent.

6.3.2.15. DRAW operator

The DRAW operator is used to draw or erase a point (line) on the screen.

1) **DRAW/O** operator

Function: Sets the current screen point

Syntax: DRAW O<X coordinate>,<Y coordinate>

The coordinates of the point X, Y are limited by the screen dimensions: $0 \leq X \leq 119$, $0 \leq Y \leq 63$.

2) **DRAW/H** operator

Function: Draws a point at a given screen position

Syntax: DRAW H<X coordinate>,<Y coordinate>

The coordinates of the point X, Y are limited by the screen dimensions: $0 \leq X \leq 119$, $0 \leq Y \leq 63$.

3) **DRAW/D** operator

Function: Draws line segments between a sequence of given points

Syntax: DRAW D<initial coordinate X>,
 <initial coordinate Y>,
 <X coordinate>,<Y coordinate>

Used to display on the screen a point, straight line, broken line, specified by absolute coordinates X, Y.

The coordinates of the point X, Y are limited by the screen dimensions: $0 \leq X \leq 119$, $0 \leq Y \leq 63$.

If adjacent pairs of X, Y coordinates are the same, the command draws a point at that position.

Example:

```
5  CLS
10 DRAW D50,30,50,30
20 DRAW D70,5,70,5
25  DIS
```


4) **DRAW/E** operator

Function: Erases a point (line) in a given position

Syntax: `DRAW E<initial coordinate X>,
 <initial coordinate Y>,
 <X coordinate>,<Y coordinate>`

Used to erase a point or line on the screen, specified by absolute coordinates X, Y.

The coordinates of the point X, Y are limited by the screen dimensions: $0 \leq X \leq 119$, $0 \leq Y \leq 63$.

If adjacent pairs of X, Y coordinates are the same, then the command erases the point at that position.

Example:

```
5 CLS
10 FOR I=1 TO 3
20 DRAW H10,I
25 REM - DELAY
30 FOR J=1 TO 200: NEXT J
40 DRAW E10,I,10,I
50 NEXT I
```

5) **DRAW/I** operator

Function: Draws a straight line whose coordinates are specified relatively

Syntax: `DRAW I<increment by X>,<increment by Y>`

Used to draw line segments whose endpoint coordinates are specified as increments relative to the current screen point.

Example:

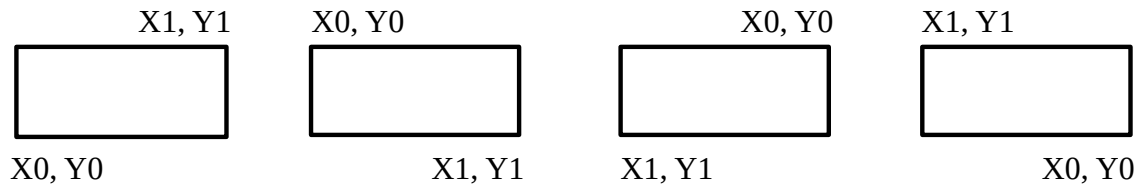
```
5 CLS
10 DRAW O20,32
20 DRAW I0,0,20,10,-5,-30,-10,0
30 REM - DELAY
40 FOR J=1 TO 200: NEXT J
50 REM - SET INDICATION LINE, CURSOR
60 DIS
```

6) **DRAW/A** operator

Function: Draws a rectangle specified by the coordinates of its diagonal

Syntax: `DRAW A<initial coordinate X>,
 <initial coordinate Y>,
 <diagonal X coordinate>,
 <diagonal Y coordinate>`

Used to draw a rectangle whose coordinates are specified as follows:



The coordinates of the point X, Y are limited by the screen dimensions: $0 \leq X \leq 119$, $0 \leq Y \leq 63$.

Example:

```
5 CLS
10 DRAW O60,32
20 FOR I=30 TO 5 STEP -5
30 DRAW A60-I,32+I,60+I,32-I
40 NEXT I
```

The result of execution is shown in Fig. 6.4.

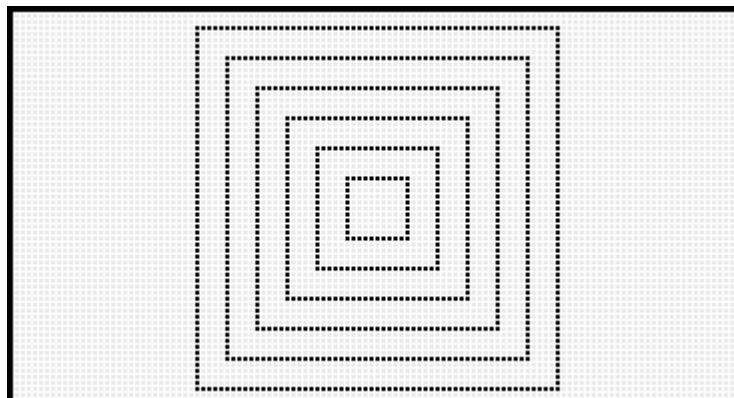


Fig. 6.4

7) **DRAW/X** operator

Function: Draws coordinate axes

Syntax: `DRAW X<direction of coordinate axis>,
<axis division size>,<number of divisions>`

Used to draw coordinate axes for line charts and histograms.

The coordinate axis direction parameter takes values of 0, 1, 2 or 3.

0 – the Y axis is drawn from the origin upwards (+Y)

1 – the X axis is drawn from the origin to the right (+X)

2 – the Y axis is drawn from the origin downwards (-Y)

3 – the X axis is drawn from the origin to the left (-X)

The following parameters are positive integers and specify the distance between scale marks in points and the number of scale marks, respectively.

Example:

```
5 CLS
10 DRAW O60,32
20 FOR I=0 TO 3
30 DRAW XI,5,5
40 NEXT I
```

A graphical example is shown in Fig. 6.5.

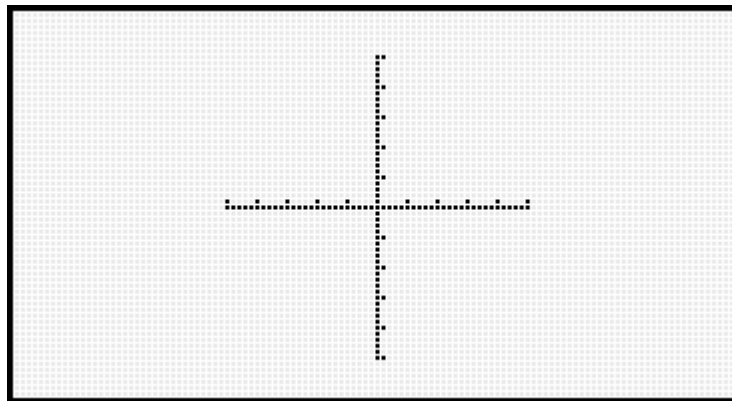


Fig. 6.5

8) **DRAW/G** operator

Function: Displaying vertical or horizontal stripes of a given width

Syntax: `DRAW G<hatch direction>,<X-axis extent>,
<Y-axis extent>[,<distance between lines>]`

Used to hatch a rectangular area in a given direction, with the current point serving as the starting point.

The hatch direction parameter takes values 0, 1, 2

- 0 – without hatch
- 1 – horizontal stripes
- 2 – vertical stripes

The <distance between lines> parameter can be omitted.

Example:

```
5 CLS
10 DRAW OO,0
20 FOR I=0 TO 2
30 DRAW AI*30,20,I*30+20,0
35 LOCATE 30,I
40 DRAW GI,20,20,2
50 NEXT I
```

9) **DRAW/C** operator

Function: Draws a circle given the coordinates of the center and the radius

Syntax: DRAW C<X coordinate of the center>,
<Y coordinate of the center>,<radius>

Used to draw a circle of a given radius around a center specified by absolute coordinates (X, Y). The coordinates of the point X, Y are limited by the screen dimensions: $0 \leq X \leq 119$, $0 \leq Y \leq 63$ and <X> is not equal to <Y>*.

Example:

```
10 CLS
20 FOR I=1 TO 3
30 DRAW C60,32,I*10
40 NEXT I
```

10) **DRAW/M** operator

Function: Displays the mask specified by a hexadecimal number.

Syntax: DRAW M<mask>

Used to fill rectangular areas with a given <mask>. Numbers from the given mask are selected in pairs and formed in screen dots horizontally. Areas are filled from top to bottom from the current cursor position or the position specified by the LOCATE command. Allows you to output irregular figures with an arbitrary configuration.

***Editor's note:** this is due to an error in the program for drawing the circle. When X=Y, the second coordinate quarter of the circle is not drawn.

Example:

```
10 CLS
20 FOR I=1 TO 120
30 DRAW MF0F0F0F00F0F0F0F
40 NEXT I
```

Drawing points is done as follows:

Mask	Binary equivalent	
F0F0	11110000	11110000
AAAA	10101010	10101010

Drawn dots

If the current place of the cursor or position is equal to the point with coordinates (X, Y), then when the DRAW/M operator is running, the Y coordinate is automatically changed for each pair of hexadecimal numbers, i.e. $Y=Y+1$. When $Y=64$, $Y=0$ is automatically assumed, and $X=X+8$. For $X>119$ and $Y>63$, masks are not displayed.

6.3.2.16. LOCATE operator

Function: Sets the cursor position

Syntax: LOCATE <X>, <Y>

Used to change the cursor position. The coordinates of the point X, Y are limited by the screen dimensions: $0 \leq X \leq 119$, $0 \leq Y \leq 63$.

Example:

```
1) 5 CLS
    10 LET X=1
    20 LET X=X+1
    40 PRINT "X="; X
    50 GOTO 20
```

```
2) 5 CLS
    10 LET X=1
    20 LET X=X+1
    30 LOCATE 20, 20
    40 PRINT "X="; X
    50 GOTO 20
```

In case 1), printing of the result starts from the current cursor position.

In case 2), due to the addition of line 30, the printing of the result starts from position (20, 20).

6.3.2.17. **STOP** operator

Function: Stopping program execution. BASIC prints:

```
OCT B CTPOKE      XXX          (STOP AT LINE   XXX)
GOTOB              (Ready)
```

Syntax: STOP

The STOP operator is used for the logical end of a program and can be placed anywhere in the program.

A program intended for temporary use may have no STOP statement; its execution usually ends at the line with the highest number.

The STOP operator helps in debugging user programs, it can be placed at critical points of the program, the program execution stops at these points. When stopping, the PRINT operator can be used in immediate mode to examine the value of specific variables.

If program execution is correct at this point, it can be continued by a GOTO statement in immediate mode, which then calls the statement following the STOP statement, or the STOP statement can be deleted, in which case program execution is resumed by the RUN command.

6.3.2.18. **DIS** operator

Function: Restores the indication line

Syntax: DIS

The DIS operator is used to restore the indication line after using graphical commands that place information in the space allocated for the indication line.

6.3.2.19. **WAIT** operator

Function: Pauses the execution of the program

Syntax: WAIT <A>

The argument size is from 1 to 32536. The minimum delay is at A=1, the maximum is at A=32536. The maximum delay is 0.1 ms.

6.3.2.20. **PLAY** operator

Function: Generates a tone of a given frequency and duration

Syntax: `PLAY <tone>,<duration>`

The <tone> parameter specifies a tone number from 0 to 40.

The <duration> parameter determines the duration of the generated note and is specified by a number from 1 to 32767. The duration of a whole note is determined experimentally.

Example: program for playing the melody "Moscow Nights"

```

10 FOR J=0 TO 2
20 READ K
30 FOR I=1 TO K
40 READ A,B,C
50 PLAY A,B*15
60 WAIT C
70 NEXT I
80 RESTORE
90 NEXT J
100 END
110 DATA 54: 'NUMBER OF NOTES
120 DATA 20,2,1,23,2,1,27,2,1,23,2,1,25,4,1,23,2,1
130 DATA 22,2,1,27,4,1,25,4,1,20,8,1,23,2,1,27,2,1
140 DATA 30,2,1,30,2,1,32,4,1,30,2,1,28,2,1,27,8,1
150 DATA 29,4,1,31,4,1,34,2,1,32,2,1,27,6,1,22,4,1
160 DATA 20,2,1,27,3,1,25,1,1,28,6,1,30,2,1,28,2,1
170 DATA 27,4,1,25,2,1,23,2,1,27,4,1,25,4,1,32,8,1
180 DATA 29,4,1,31,4,1,34,2,1,32,2,1,27,6,1,22,4,1
190 DATA 20,2,1,27,3,1,25,1,1,28,8,1,30,2,1,28,2,1
200 DATA 27,4,1,25,2,1,23,2,1,27,4,1,25,4,1,20,12,1

```

Table of approximate frequencies of each note (Hz):

Note	Freq.	Note	Freq.	Note	Freq.	Note	Freq.	Note	Freq.	Note	Freq.
0	700	7	1050	14	1570	21	2350	28	3515	35	5280
1	740	8	1110	15	1665	22	2495	29	3730	36	5535
2	785	9	1175	16	1760	23	2640	30	3970	37	5925
3	830	10	1245	17	1865	24	2790	31	4190	38	6275
4	880	11	1320	18	1975	25	2960	32	4430	39	6640
5	930	12	1400	19	2095	26	3140	33	4690	40	7050
6	990	13	1480	20	2215	27	3320	34	4970		

The formula for calculating the approximate duration of each note:

$$t = \frac{\text{duration} \times 16}{\text{note frequency}} \text{ [sec]}$$

6.3.3. Built-in functions

BASIC contains ten functions for performing computational operations. These functions are identified by a three-letter name followed by an argument in parentheses. Functions can be used anywhere in a program as expressions or elements of an expression, wherever expressions are valid.

6.3.3.1. SIN(X), COS(X) functions




The argument is an angle expressed in radians. The argument value is limited to ± 51470 . If the angle is expressed in degrees, you need to convert degrees to radians using the following formula:

$$\text{radians} = \text{degrees} \times \frac{\pi}{180}$$

In the example*, 3.14159265 is used as PI. In line 40, this value is used to convert to radians.

```
10 REM - CONVERT ANGLE (X) TO RADIANS
11 REM - CALCULATE SIN AND COS
30 INPUT X
40 LET Y=X*PI/180
60 PRINT "Degrees:" X
70 PRINT "Radians:" Y
80 PRINT "Sine:" SIN(Y)
90 PRINT "Cosine:" COS(Y)
92 PRINT
95 GOTO 30
ГОТОВ
```

```
RUN
?0          ?10          ?45
Degrees: 0   Degrees: 10   Degrees: 45
Radians: 0   Radians: .1745329 Radians: .7853982
Sine: 0     Sine: .1736482   Sine: .7071068
Cosine: 1   Cosine: .9848078 Cosine: .7071068
```

```
?90          ?7P            
Degrees: 90
Radians: 1.570796   ГОТОВ
Sine: 1
Cosine: .1462918E-8
```

6.3.3.2. ATN(X) function




The arctangent function calculates the value of an angle in radians in the range from $+\frac{\pi}{2}$ to $-\frac{\pi}{2}$.

In the program below*, the input number is given in degrees, and line 40 converts it to radians. Line 50 uses the angle value in radians to obtain the tangent using the equation:

$$\tan(X) = \frac{\sin(X)}{\cos(X)}$$

Line 70 specifies the tangent (Z) value as an argument to the ATN function to produce a value that is printed with the heading ATAN(Z). Line 70 converts the arctangent function value in radians back to degrees and places it on the fifth line of the printout to verify the value specified on the first line.

```
20 PRINT "Set angle in deg."  
30 INPUT X  
40 LET Y=X*PI/180  
50 LET Z=SIN(Y)/COS(Y)  
60 PRINT "Degrees:" X  
70 PRINT "Radians:" Y  
75 PRINT "TAN(X):" Z  
80 PRINT "ATAN(Z):" ATN(Z)  
85 PRINT "ATAN degrees:" ATN(Z)*180/PI  
86 PRINT  
90 GOTO 30  
ГОТОВ
```

```
RUN  
Set angle in deg.  
?0  
Degrees: 0  
Radians: 0  
TAN(X): 0  
ATAN(Z): 0  
ATAN degrees: 0  
  
?45  
Degrees: 45  
Radians: .7853982  
TAN(X): 1  
ATAN(Z): .7853982  
ATAN degrees: 45  
  
?90  
Degrees: 90  
Radians: 1.570796  
TAN(X): .6835653E 9  
ATAN(Z): 1.570796  
ATAN degrees: 90  
  
?¬P     
ГОТОВ
```

6.3.3.3. LOG(X) function

The LOG function returns the natural logarithm of its given argument.

***Editor's note:** programs marked with * have been re-adapted from table display to string display for ease of viewing on the device LCD.

Example:

```
10 INPUT X
20 PRINT LOG(X)
25 PRINT
30 GOTO 10
ГОТОВ
```

```
RUN
?54.59815
4
```

```
?22026.47
10
```

```
?100
4.60517
```

```
? .720049E11
25
```

```
?¬P   
```

ГОТОВ

Natural logarithms can easily be converted to logarithms with any other base using the following formula:

$$\log_A(N) = \frac{\log(N)}{\log(A)}$$

Where A is the desired base. The following program illustrates the conversion to common logarithms.

Example*:

```
1 REM - TRANSLATION OF NATURAL LOGARITHM TO DECIMAL
15 INPUT X
17 PRINT "Value:" X
20 PRINT "LN:" LOG(X)
40 PRINT "LOG:" LOG(X)/LOG(10)
45 PRINT
50 GOTD 15
60 END
ГОТОВ
```

RUN		
?4	?250	?60
Value: 4	Value: 250	Value: 60
LN: 1.386294	LN: 5.521461	LN: 4.094345
LOG: .60206	LOG: 2.39794	LOG: 1.778151

```
?¬P   
```

ГОТОВ

6.3.3.4. **EXP(X)** function

The exponential function raises the number "E" to the power X. EXP is the inverse function of the LOG function, i.e.

$$\log(\exp(X)) = X$$

Example:

```
10 INPUT X
20 PRINT EXP (X)
25 PRINT
30 GOTO 10
ГОТОВ
```

RUN

?4

54.59815

?9.421006

12344.99

?25

.790049E 11

?¬P



ГОТОВ

6.3.3.5. **ABS(X)** function

The ABS absolute function calculates the absolute value of any argument.

Example:

```
10 INPUT X
20 LET X=ABS (X)
30 PRINT X
35 PRINT
40 GOTO 10
ГОТОВ
```

RUN

?-35.7

35.7

?2

2

?105555567

.1055556E 9

?-44.555566668899

44.55557

?¬P



ГОТОВ

6.3.3.6. INT(X) function

The integer function calculates the value of the largest integer that does not exceed the value of the argument.

For example:

```
PRINT INT(34.67)
34
```

```
PRINT INT(-5.1)
-6
```

The function can be used to round numbers to the nearest integer using `INT (X+ .5)`.

For example:

```
PRINT INT(34.67+.5)
34
```

```
PRINT INT(-5.1+.5)
-5
```

INT(X) can be used to round to any decimal place by using the following expression as an argument:

$$\frac{\text{int}(X \times 10^D + 0.5)}{10^D}$$

Where D is an integer specified by the user.

Example:

```
20 PRINT "Rounding:"
25 INPUT A
40 PRINT "Num. of dec. digits:"
45 INPUT D
60 LET B=INT (A*10^-D+.5)/10^-D
70 PRINT "A after rnd. =" B
75 PRINT
80 GOTO 20
90 END
ГОТОВ
```

```
RUN
Rounding:
?55.65842
Num. of dec. digits:
?2
A after rnd. = 55.66
```

```
Rounding:
?78.375
Num. of dec. digits:
?-2
A after rnd. = 100
```

?-P



ГОТОВ

6.3.3.7. INC function

Enters the decimal code of the key currently pressed. A zero value as the function result means that either no key is pressed or one of the keys is pressed:



Example:

```
10 LET A=INC
20 IF A=0 THEN GOTO 10
30 PRINT A
40 GOTO 10
```

Tables of character codes are given in **Appendix 5**.

6.3.3.8. SGN(X) function

The sign function SGN(X) gives the value:

1, if $X > 0$
0, if $X = 0$
-1, if $X < 0$

Example:

```
PRINT SGN (3.42)
1
PRINT SGN (-42)
-1
PRINT SGN (23-23)
0
```

6.3.3.9. SQR(X) function

The square root function extracts the square root of any positive value of its argument (X).

Example:

```
10 INPUT X
20 LET X=SQR(X)
30 PRINT X
40 GOTO 10
ГОТОВ
```

RUN

?16

4

?1000

31.62278

?123456789

11111.11

?25E2

50

?↯P

ГОТОВ



6.3.3.10. RND(X) function

The random number function generates a random number or set of random numbers between 0 and 1. The argument (X) is unused and can be any number.

Example:

```
10 PRINT "Random numbers"
30 FOR I=1 TO 15
40 PRINT RND(0)
50 NEXT I
60 END
ГОТОВ

RUN
Random numbers
.1092502 .9648132 .8866272 .6364441 .8390198 .3061218 .285553
.9582214 .1793518 .4521179 .9854126E-1 .5221863 .2462463
.7778015 .450592
OCT B CTPOKE 60
ГОТОВ
```

It is possible to generate random numbers in a given interval. If the interval (A, B) is needed, the expression is used:

$$(B - A) \times rnd(0) + A$$

The following program generates a set of random numbers in the interval (4, 6).

Example:

```
10 REM - SEQUENCE OF RANDOM NUMBERS FROM (4, 6)
20 FOR B=1 TO 15
30 LET A=(6-4)*RND(0)+4
40 PRINT A
50 NEXT B
60 END
ГОТОВ

RUN
4.2005 5.929626 5.773254 5.272888 5.67804 4.612244 4.571106
5.916443 4.358704 4.904236 4.197083 5.044373 4.492493
5.555603 4.901184

OCT B CTPOKE 60
ГОТОВ
```

6.3.3.11. PI function

The function determines the number with an accuracy of up to seven digits: 3.141696.

Used in calculating trigonometric functions. For information on using the PI function, see the section on using trigonometric functions.

Example:

```
LET A=SIN(PI)
```

7. Storage rules

7.1. The microcalculator must be stored in a dry, heated room with no acidic, alkaline or other aggressive impurities in the air at a temperature of +1 to +50°C and at a relative humidity of no more than 85%.

Keyboard diagram for working in functional mode



Microcalculator commands

1. Immediate mode commands

HELP

Function: Displays information on the LCD screen about the composition of BASIC language commands, their purpose, syntax and binding to the functional keyboard (in Russian)

Syntax: HELP *
 HELP <topic>

Examples: HELP DRAW/A
 HELP HELP

DELETE

Function: Removes lines from the program

Syntax: DELETE [starting line number][,end line number]

Examples: DELETE 50
 DELETE 70,120
 DELETE 1,8191

LIST

Function: Prints the program text

Syntax: LIST [starting line number][,end line number]

Examples: LIST
 LIST 20
 LIST 30,50

RUN

Function: Executes the program

Syntax: RUN

Example: RUN

SAVE

Function: Saves programs on the SMP

Syntax: SAVE "[SM*:]file name[.type]"

Examples: SAVE "SM0:TEXT.BAS"
SAVE "SM1:BASIC"

LOAD

Function: Loads a program from the SMP into memory

Syntax: LOAD "[SM*:]name of the input file[.type]"

Examples: LOAD "SM0:TEXT.BAS"
LOAD "SM1:BASIC"

INIT

Function: Initializes the SMP

Syntax: INIT ["SM*:"]

Examples: INIT "SM1:"
INIT

KILL

Function: Removes the specified file from the SMP

Syntax: KILL "[SM*:]file name.type"

Example: KILL "SM0:TEXT.BAS"

NAME (AS)

Function: Renames the file on SMP

Syntax: NAME "<old name>" AS "<new name>"

Example: NAME "SM1:OLD.BAS" AS "NEW.BAS"

FILES

Function: Prints the SMP disk directory listing

Syntax: FILES ["SM*:"]

Examples: FILES
FILES "SM1:"

DEV

Function: Sets the device name as the working SMP

Syntax: DEV ["SM* : "]

Examples: DEV
DEV "SM1 : "

AUTO

Function: Automatically numbers lines (exit mode by CY/P)

Syntax: AUTO [[initial number][,stepping]]

Examples: AUTO
AUTO ,20
AUTO 5,10

MEM

Function: Shows memory allocation

Syntax: MEM

Example: MEM

EDIT

Function: Edits program lines

Syntax: EDIT <line number>

Example: EDIT 150

EDIT subfunctions


CY/A — move cursor to beginning of line


CY/B — move cursor to beginning of previous word


CY/E — erase character from cursor to end of line


CY/F — move cursor to beginning of next word


CY/X — move cursor to end of line

 — insert mode

 — shift line to cursor position

 — move cursor one position to the right

 — move cursor one position to the left

 — exit edit mode

2. Program mode commands

DIM

Function: Reserves memory for array formation

Syntax: DIM array name (numerical value [, numerical value])

Examples: DIM A(10)
 DIM B(5,2)

DEF (FN)

Function: Defines functions and expressions as a single function

Syntax: DEF FN<letter> (parameter)=<expression>

Examples: DEF FNX(S)=S²+4
 DEF FNZ(X)=X²
 DEF FNA(X)=4+2

END

Function: The last operator of the program

Syntax: END

Example: END

FOR-TO-STEP/NEXT

Function: The FOR and NEXT statements are used to indicate the start and end points of a program.

All statements between the FOR statement and its corresponding NEXT statement will be executed cyclically according to the conditions specified in the FOR statement.

Syntax: FOR <variable name>=<numeric expression>
 TO <numeric expression> [STEP <numeric expression>]
 NEXT <variable name>

Examples: FOR I=0 TO 9
 NEXT I

 FOR A=1 TO 9 STEP 2
 PRINT A
 NEXT A

GOSUB/RETURN

Function: Enters and returns from a subroutine

Syntax: GOSUB <line number>
RETURN

Examples: GOSUB 200
RETURN

GOTO

Function: Performs an unconditional jump to the line with the specified number

Syntax: GOTO <line number>

Example: GOTO 210

IF (THEN)

Function: Provides a conditional jump in the program

Syntax: IF <conditional expression> THEN <operator>
THEN <line number>
GOTO <line number>

Examples: IF A>B THEN 10
IF A>4 GOTO 50
IF x>=70 THEN GOSUB 100

INPUT

Function: Enters data from the keyboard during program execution.

Syntax: INPUT <variable name>[, <variable name>, ...]

Examples: INPUT X
INPUT A, B, C

LET

Function: Assigns a value to a variable

Syntax: LET <variable name> = <numeric expression>

Examples: LET A=2
LET X=A+B
LET X=Y*2

PRINT

Function: Outputs (prints) data to the screen

Syntax: PRINT
 PRINT <expression>[,<expression>,...]
 PRINT "character string"["character string",...]

Examples: PRINT A,B
 PRINT A+B
 PRINT "Result"

DATA

Function: Enters into the program numeric constants to which the READ statement sequentially associates variable names

Syntax: DATA <number>[,<number>,...]

Example: DATA 5,6,7,8

READ

Function: Reads the data that was defined in DATA and sequentially binds it to its variables

Syntax: READ <variable name>[,<variable name>,...]

Example: READ S,T

RESTORE

Function: Reuses the constants of the DATA statement that has the lowest line number in the program.

Syntax: RESTORE

Example: RESTORE

REM (REMARK)

Function: Defines comments in the user program

Syntax: REM - comment
 REMARK - comment

Example: REM - CALCULATION OF FACTORIAL X

WAIT

Function: Pauses the execution of the program

Syntax: WAIT A
 Argument A specifies the amount of delay.

Example: WAIT 3530

RANDOMIZE

Function: Allows you to get a series of random numbers in a program using the RND function

Syntax: RANDOMIZE

Example: RANDOMIZE

STOP

Function: Causes the program to terminate execution.

Syntax: STOP

Example: STOP

CLS

Function: Clears the LCD screen and moves the cursor to the far left position

Syntax: CLS

Example: CLS

DIS

Function: Restores the indication line

Syntax: DIS

Example: DIS

DRAW/O

Function: Determines the current point on the screen

Syntax: DRAW O<X coordinate>,<Y coordinate>

Examples: DRAW OX,Y
DRAW O60,32

DRAW/H

Function: Lights a dot at a given position

Syntax: DRAW H<X coordinate>,<Y coordinate>

Example: DRAW HX,Y

DRAW/D

Function: Displays line segments between a sequence of specified points

Syntax: DRAW D<initial coordinate X>,<initial coordinate Y>,
 <X coordinate>,<Y coordinate>

Example: DRAW D34,34,67,50

DRAW/E

Function: Erases a point (line) at a given position

Syntax: DRAW E<initial coordinate X>,<initial coordinate Y>,
 <X coordinate>,<Y coordinate>

Example: DRAW E34,34,67,50

DRAW/I

Function: Draws a straight line whose coordinates are specified relatively

Syntax: DRAW I<increment by X>,<increment by Y>[,...]

Examples: DRAW I20,10,-5,-30
 DRAW I-10,0

DRAW/A

Function: Displays a rectangle specified by the coordinates of its diagonal

Syntax: DRAW A<initial coordinate X>,<initial coordinate Y>,
 <diagonal X coordinate>,<diagonal Y coordinate>

Examples: DRAW A0,0,30,60
 DRAW A-I,I,I,-I

DRAW/C

Function: Displays a circle based on the specified coordinates of the center and radius

Syntax: DRAW C<X coordinate of the center>,
 <Y coordinate of the center>,<radius>

Example: DRAW C60,32,10

DRAW/X

Function: Draws coordinate axes

Syntax: DRAW X<direction of coordinate axis>,
 <axis division size>,<number of divisions>

Example: DRAW XI,5,5

DRAW/G

Function: Drawing vertical or horizontal stripes of a given width

Syntax: DRAW G<hatch direction>,<X-axis extent>,
 <Y-axis extent>[,<distance between lines>]

Example: DRAW GI,20,20,2

DRAW/M

Function: Displaying a mask specified by a hexadecimal number

Syntax: DRAW M<mask>

Example: DRAW MF0F0F0F0F0F0F0F0F

LOCATE

Function: Determines the position of the cursor

Syntax: LOCATE <X coordinate>,<Y coordinate>

Example: LOCATE 60,32

PLAY

Function: Makes sound signals

Syntax: PLAY <tone>,<duration>

Example: FOR I=1 TO 40
 PLAY 1,20
 NEXT I

3. Computational functions

SIN

Function: Trigonometric function of sine – SIN(X)

Syntax: SIN(numeric expression)

Example: SIN (A/B)

COS

Function: Trigonometric function of cosine – COS(X)

Syntax: COS(numeric expression)

Example: COS (A*10)

ATN

Function: Inverse trigonometric function of TAN(X)

Syntax: ATN(numeric expression)

Example: ATN (A/100)

SQR

Function: Square root

Syntax: SQR(numeric expression)

Example: SQR (30)

EXP

Function: Exponential function

Syntax: EXP(numeric expression)

Example: EXP (1)

LOG

Function: Natural logarithm

Syntax: LOG(numeric expression)

Example: LOG (2.71828)

ABS

Function: Absolute value

Syntax: `ABS(numeric expression)`

Example: `ABS(-10.5)`

INT

Function: Integer function. Calculates the value of the largest number that does not exceed the argument value

Syntax: `INT(numeric expression)`

Example: `INT(3.14)`

SGN

Function: Sign of a numerical expression

Syntax: `SGN(numeric expression)`

Example: `SGN(-1)`

RND

Function: Generates a random number or set of random numbers in the range [0, 1]

Syntax: `RND(any number)`

Example: `RND(0)`

INC

Function: Gives the code of the pressed key

Syntax: `INC`

Example: `10 LET A=INC: IF A=0 GOTO 10`
`20 PRINT A`

PI

Function: Determines the number pi to seven decimal places

Syntax: `PI`

Example: `LET A=SIN(PI)`

1. Messages format

Error messages have the following format:

Where XXX – error code, YYY – line number where the error occurred.
If YYY = 0, this means that the error occurred in command mode, i.e. in the last line entered.

Error codes 0 through 64 indicate fatal errors; program execution stops after the error message is printed.
Fatal errors are listed in Table 1.

Error codes 89 through 127 indicate non-fatal errors; program execution continues after the error message is printed. Non-fatal errors are listed in Table 2.

Fatal errors

Error code	Error meaning
0	Out of user allocated memory
1	Unrecognized operator
2	Invalid GOTO or GOSUB statement
3	Invalid character delimiting a statement (usually in the case of a malformed statement that causes the statement's actions to end prematurely)
4	RETURN statement without a corresponding GOSUB statement
5	The index is incorrect
6	The index is not between 0 and 255 or exceeds the maximum set by the program
7	Parentheses mismatch in operator

Continuation of Table 1

Error code	Error meaning
8	Invalid LET operator
9	Invalid comparison sign in IF statement
10	Invalid IF statement
11	Invalid PRINT statement
12	The input string is too long (exceeds 80 characters)
13	Invalid dimension in DIM statement
14	There is not enough space in memory for the array
15	The DEF statement is not formed correctly
16	Invalid row number or dimension value
17	DIM statement for a previously declared or used element
18	Invalid variable in INPUT statement list
19	Invalid variable in READ statement list
20	The data in the READ statement list is exhausted
21	Invalid format of DATA statement
22	Invalid FOR statement
23	FOR is not followed by a corresponding NEXT statement
24	NEXT operator without FOR operator
25	Mismatched quotes in operator
27	Incorrectly formed expression (omitted order of number in E format)
29	Invalid DRAW operator function
30	Parameter not specified
31	Invalid note in PLAY statement
59	No place in SMP
60	The file to be renamed was not found.
61	Invalid format of NAME operator
62	File not found
63	Syntax error in file name

Table 2

Non-fatal errors

Error code	Error meaning
120	Invalid characters when entering
121	Not enough data entered for INPUT operator
122	Too much data entered for INPUT statement
123	Non-existent variable
124	Number too large to fix (probably index combination exceeds range)
125	Overflow or borrow on division/multiplication
126	Square root of a negative number
127	Logarithm of a negative number or zero; Overflow when calculating EXP

Examples of programs

1. Drawing curves

1.1. Drawing a circle

Using the DRAW/C operator, a circle can be drawn. It is enough to determine the coordinates of its center and radius.

The program, with the help of which a circle with a radius of 25 and a center in coordinates (50, 32) is drawn:

```
10 DRAW C50,32,25
```

Now let's draw concentric circles with radii of 10, 20 and 30 with the center at coordinates (50, 32).

```
5 CLS
10 FOR R=1 TO 3
20 DRAW C50,32,R*10
30 NEXT R
```

1.2. Drawing a sine wave

The sine curve drawing program uses the drawing of coordinate axes, the application of scale divisions, and the drawing of the sine curve using the DRAW/X operator.

The scale values can be arranged in the vertical direction using the PRINT operator.

```
10 CLS
20 FOR I=0 TO 2 STEP 2
30 DRAW O20,32
40 DRAW XI,5,4: NEXT I
50 DRAW O20,32
60 DRAW X1,10,8
70 GOSUB 200
80 GOSUB 400
90 GOSUB 600
100 END
200 LET A=28
205 FOR I=90 TO 360 STEP 90
210 DRAW OA,25
220 PRINT <Q3; Y3> I;
225 LET A=A+25
226 IF I=90 THEN LET A=A-6
230 NEXT I
240 RETURN
400 FOR I=-1 TO 1 STEP 1
```

```
410 DRAW O2,30-I*20
420 PRINT I
430 NEXT I
440 RETURN
600 FOR I=0 TO 360 STEP 4.5
610 LET X=20+I*(2/9)
620 LET Y=32+20*SIN(PI/180*I)
630 DRAW HX,Y
640 NEXT I
650 RETURN
```

2. Linear graphs drawing

Since the drawing of the graph requires specifying coordinates, they must be defined first. The origin of coordinates can be placed at the position (X, Y) using the operator:

```
DRAW OX,Y
```

The drawing of coordinate axes is performed using the following operators:

```
DRAW X0,5,10
DRAW X1,5,10
DRAW X2,5,10
DRAW X3,5,10
```

The numbers following X indicate the direction of the axes.

- 0: draws the Y axis from the origin upwards (+Y)
- 1: draws the X axis from the origin to the right (+X)
- 2: draws the Y axis from the origin downwards (-Y)
- 3: draws the X axis from the origin to the left (-X)

The next numerical value (5) determines the distance between scale marks (in dots), the number 10 determines the number of scale marks.

These operators are used as an example in the following program

```
5 CLS
10 FOR I=0 TO 3
20 DRAW O60,32
30 DRAW XI,5,6
40 NEXT I
```

When this program is executed, the coordinate axes are drawn with the origin at point (60, 32), shown in Fig. 1.

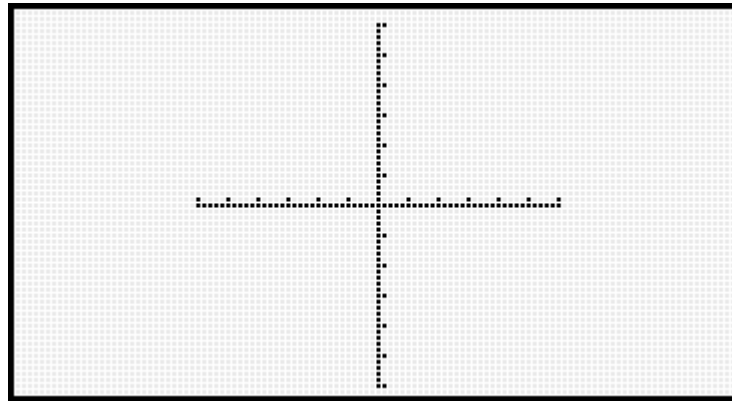


Fig. 1

Linear graphs are often used to study processes that occur over time. Let's look at a program that plots a line graph of average monthly temperature.

Let's assume that the average temperature for a month in a certain city changed in accordance with Table 1. This data is entered into the program using the DATA operator. Then, the program plots a graph of the average temperature for a month. The data is read using the READ operator.

Table 1

Month	Temperature	Month	Temperature	Month	Temperature
January	11,5	May	19,8	September	22,3
February	9,8	June	23,4	October	18,5
March	13,7	July	26,6	November	15,9
April	18,3	August	28,2	December	14,7

```
10 CLS
20 DRAW O30,50
30 DRAW X9,10,4
35 DRAW O30,50
40 DRAW X1,5,12
50 FOR I=1 TO 3
60 LOCATE 15,I*10
70 PRINT 40-I*10
80 NEXT I
90 FOR I=1 TO 12
100 LET X=30+(I-1)*5
110 READ A
120 LET Y=INT(50-A)
130 IF I=1 THEN 150
140 DRAW DP,Q,X,Y
150 LET P=X: LET Q=Y
160 NEXT I
170 DATA 11.5,9.8,13.7,18.3,19.8,23.4
175 DATA 26.6,28.2,22.3,18.5,15.9,14.7
```

3. Histograms drawing

3.1. Drawing histograms using symbols

The principle of constructing a histogram is that columns are drawn on the display screen, the length of which is proportional to the value of N. Since the display screen only fits 20 digits, then in order to display linear values exceeding the number 20, it is necessary to use scaling.

If you run the following program and assign N values from 1 to 20, a line of N characters will appear on the screen. This column will be continuous.

```
10 CLS
20 INPUT N
25 LOCATE 0,32
30 FOR I=1 TO N
40 PRINT "-";
50 NEXT I
60 END
```

3.2. Drawing histograms using graphs

The above program can be modified to use dots instead of symbols. The following program is an example.

With this program, 120 dots can be displayed horizontally, with each dot being six times shorter than the symbol.

```
10 CLS
20 INPUT N
30 FOR I=0 TO N-1
40 DRAW HI,10
50 NEXT I
60 END
```

The histograms obtained with this simple program look like straight lines. You can write a program that will display columns instead of horizontal lines.

```
10 CLS
20 INPUT N
30 DRAW A0,32,N-1,26
40 END
```

The columns obtained with this simple program glow monotonously. It is possible to write a program in which the column will look like a certain pattern. Such patterns can be implemented by making changes and additions to the above program.

1) Horizontal shading of the column:

```
35 DRAW O0,26
36 DRAW G1,N-1,6,2
```

2) Vertical shading of the column:

```
35 DRAW O0,26
36 DRAW G2,N-1,6,2
```

3.3. Drawing Pie Charts

To draw a circle, use the DRAW/C command. However, to draw a pie chart, you must draw lines that define sectors corresponding to certain values.

Below is an example of a program that allows you to divide a circle into 5 parts.

```
10 CLS
20 DRAW C50,32,30
30 LET S=0: LET T=0
40 FOR I=1 TO 5: READ A: LET T=T+A: NEXT I
50 RESTORE: LET A=0
60 FOR I=1 TO 5: LET A1=A
70 READ A: LET S=S+A
80 LET X=50+INT(30*COS(2*PI*S/T))
90 LET Y=32-INT(30*SIN(2*PI*S/T))
100 DRAW D50,32,X,Y
110 LET A2=(A1+A)/2: GOSUB 200
115 NEXT I
120 END
130 DATA 100,200,300,400,500
200 LET A3=COS(2*PI*S/(T+A2))
205 LET X=84
206 IF A3<0 THEN LET X=5
210 LET Y=25-INT(22*SIN(2*PI*S/(T+A2)))
220 DRAW OX,Y
230 PRINT I*100
240 RETURN
```

3.4. Two examples of drawing histograms

Histograms are often used to visualize the relationship of some data.

However, it is necessary that, along with comparative clarity, the histogram bars correctly reflect the quantitatively displayed value. Therefore, it is important to choose the right scale here so that the histogram fits well on the screen.

There are many ways to draw graphs depending on their application. Basically, such histograms are used in which one bar expresses the quantitative value of one value (for example, when illustrating the number of different products). There are histograms in which one bar illustrates both the total number and some details (for example, the ratio of the total number and a specific product).

Let's write programs with the help of which, using the following example, such histograms of two types are drawn.

Example data: the number of cars of two types A and B, manufactured at one plant, is given in Table 2.

Table 2

	1980	1981	1982
Cars A	48200	57200	67200
Cars B	39200	31100	27500

A program for drawing a histogram in which a single column displays the quantitative value of only one quantity.

The histogram column displaying the production of type A cars is light, the column displaying the production of type B cars is dark.

```
10 CLS
20 FOR I=0 TO 2
30 LOCATE 12,10+I*16
40 PRINT 80+I
50 FOR J=1 TO 2
60 READ A
70 LET Y=I*16+(J-1)*8+10
80 DRAW A30,Y+4,30+INT(A/1000),Y
90 IF J=1 THEN GOTO 120
100 DRAW O30,Y
110 DRAW G1,INT(A/1000),4
120 NEXT J: NEXT I
130 DATA 48200,39260,57200,31100,67200,27509
140 END
```

A program for drawing a histogram in which the total quantity and details are displayed in one column at the same time.

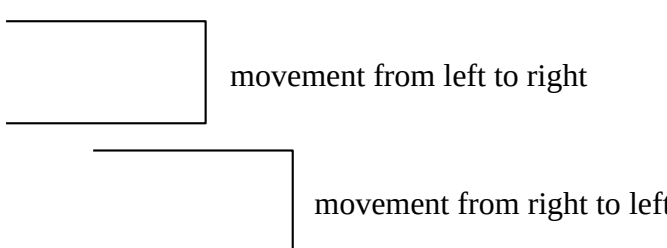
```
10 CLS
20 FOR I=0 TO 2
30 LOCATE 12,10+I*16
40 PRINT 80+I
50 READ A,B
60 LET Y=I*16+10
70 DRAW A30,Y+8,30+INT(A/1000),Y
75 DRAW O30,Y
80 DRAW A30,Y+8,30+INT(B/1000),Y
85 DRAW O30,Y
90 DRAW G1,INT(B/1000),8
100 NEXT I
110 DATA 48200,39200,57200,31100,67200,27500
120 END
```

4. Display of moving objects

4.1. Creating moving objects

If you execute the following program, the '*' mark will move from left to right and right to left.

```
10 CLS
20 FOR I=0 TO 14
30 LOCATE I*6,8
40 PRINT <S2,2>"*"
50 NEXT I
60 FOR I=14 TO 0 STEP -1
70 LOCATE I*6,8
80 PRINT <S2,2>"* "
90 NEXT I
100 GOTO 20
```



The diagram consists of two horizontal rectangular boxes. The top box is positioned to the right of the first loop (lines 20-50) and is labeled "movement from left to right". The bottom box is positioned to the right of the second loop (lines 60-90) and is labeled "movement from right to left".

As shown in the program, when the control variable I in line 20 changes from 0 to 17, the statement in line 30 changes the cursor position from (0, 8) to (114, 8). The mark '*' is displayed on the screen by drawing "**", while the mark '*', which was displayed before, is erased by a space.

When this procedure is repeated, it appears to move from left to right. The statements in lines 60-90 similarly move the mark from right to left.

The vertical position is theoretically performed in a similar manner. However, simultaneous display and erasure cannot occur here, and the program takes the following form:

```
10 CLS
20 FOR I=0 TO 7
30 LOCATE 12,I*8: PRINT "*";
40 IF I=0 THEN 60
50 LOCATE 12,(I-1)*8: PRINT " ";
60 NEXT I
70 FOR I=7 TO 0 STEP -1
80 LOCATE 12,I*8: PRINT "*"
90 IF I=7 THEN 110
100 LOCATE 12,(I+1)*8: PRINT " ";
110 NEXT I
120 GOTO 20
```

4.2. Changing the speed of movement

The previous program moves the '*' label quite quickly. The speed of its movement is controlled by the FOR/NEXT operator.

Let's execute the program obtained by adding the following line to the previous program:

```
45 FOR J=1 TO 50: NEXT J
```

When using this operator, the speed of movement from left to right becomes slightly slower. The speed is increased by decreasing the final value of the FOR/NEXT operator, and is decreased by increasing this value. Speed control is performed by adding this operator to the necessary part of the program.

4.3. Moving a point in a straight line

To move a point in a straight line, you need to run the following program:

10 CLS		
20 FOR I=0 TO 119		
30 DRAW HI,32		
40 IF I=0 THEN 60		
50 DRAW EI-1,32,I-1,32		
60 NEXT I		
70 FOR I=119 TO 0 STEP -1		
80 DRAW HI,32		
90 IF I=119 THEN 110		
100 DRAW EI+1,32,I+1,32		
110 NEXT I		
120 CLS		
130 FOR I=0 TO 63		
140 DRAW H60,I		
150 IF I=0 THEN 170		
160 DRAW E60,I-1,60,I-1		
170 NEXT I		
190 FOR I=63 TO 0 STEP -1		
200 DRAW H60,I		
210 IF I=63 THEN 230		
220 DRAW E60,I+1,60,I+1		
230 NEXT I		
240 GOTO 10		

movement from left to right

movement from right to left

movement from top to bottom

movement from bottom to top

4.4. Moving a point along a curved path

The following program moves a point along a cosine trajectory back and forth.

```
10 CLS
20 LET P=0: LET Q=0
30 FOR I=0 TO 360 STEP 4.5
40 LET X=20+I*(2/9)
50 LET Y=32+20*COS(PI/180*I)
60 DRAW HX,Y
70 DRAW EP,Q,P,Q
80 LET P=X: LET Q=Y
90 NEXT I
100 FOR I=360 TO 0 STEP -4.5
110 LET X=20+I*(2/9)
120 LET Y=32+20*COS(PI/180*I)
130 DRAW HX,Y
140 DRAW EP,Q,P,Q
150 LET P=X: LET Q=Y
160 NEXT I
170 GOTO 30
```

Character code tables

Table 1

Special symbols

Dec. code	Symbol	Dec. code	Symbol	Dec. code	Symbol	Dec. code	Symbol
0		16		32	<i>space</i>	48	0
1		17		33	!	49	1
2		18		34	"	50	2
3		19		35	#	51	3
4		20		36	\$	52	4
5		21		37	%	53	5
6		22		38	&	54	6
7		23		39	'	55	7
8		24		40	(56	8
9		25		41)	57	9
10	<i>ΠC (LF)</i>	26		42	*	58	:
11		27		43	+	59	;
12		28		44	,	60	<
13	<i>BK (CR)</i>	29		45	-	61	=
14	<i>PYC (RUS)</i>	30		46	.	62	>
15	<i>ЛAT (LAT)</i>	31		47	/	63	?

Table 2

Latin layout symbols

Dec. code	Symbol	Dec. code	Symbol	Dec. code	Symbol	Dec. code	Symbol
64	@	80	P	96	‘	112	p
65	A	81	Q	97	a	113	q
66	B	82	R	98	b	114	r
67	C	83	S	99	c	115	s
68	D	84	T	100	d	116	t
69	E	85	U	101	e	117	u
70	F	86	V	102	f	118	v
71	G	87	W	103	g	119	w
72	H	88	X	104	h	120	x
73	I	89	Y	105	i	121	y
74	J	90	Z	106	j	122	z
75	K	91	[107	k	123	{
76	L	92	\	108	l	124	
77	M	93]	109	m	125	}
78	N	94	¬	110	n	126	–
79	O	95	–	111	o	127	3E (BS)

Table 3

Russian language symbols

Dec. code	Symbol	Dec. code	Symbol	Dec. code	Symbol	Dec. code	Symbol
64	ю	80	п	96	Ю	112	П
65	а	81	я	97	А	113	Я
66	б	82	р	98	Б	114	Р
67	ц	83	с	99	Ц	115	С
68	д	84	т	100	Д	116	Т
69	е	85	у	101	Е	117	У
70	ф	86	ж	102	Ф	118	Ж
71	г	87	в	103	Г	119	В
72	х	88	ь	104	Х	120	Ь
73	и	89	ы	105	И	121	Ы
74	й	90	з	106	Й	122	З
75	к	91	ш	107	К	123	Ш
76	л	92	э	108	Л	124	Э
77	м	93	щ	109	М	125	Щ
78	н	94	ч	110	Н	126	Ч
79	о	95	ъ	111	О	127	ЗБ (BS)

Recommendations for working with SMP

ATTENTION!!! To ensure the reliability of the information stored on the SMP, before writing the user program from the microcomputer RAM to the SMP after debugging it (debugging involves at least one execution of the program using the RUN command), it is necessary to adhere to the following rules:

1. Please note that any program must end with the END statement, which has the following format:
`<line number> <space> END.`
2. Enter the STOP operator as the first line at the beginning of the program.
 Run the program with the RUN command. After its execution, delete the entered line with the STOP operator and, using the SAVE command, write the program to the SMP.
3. Use the FILES command to show the listing of files located on the SMP, determining the size occupied by the program (file) in blocks.
4. Use the KILL command to delete the written program on the SMP.
5. Change the format of the last line of the program to the following:
`<line number> END,`
 by removing the space between the line number and the END statement.
6. Perform actions similar to p. 3, writing a new program in the SMP.
7. Use the FILES command to determine the amount of memory occupied by the new program in blocks.
8. If the resulting program takes up the same amount of memory in the SMP as the previous one, then this version of the program is left in the SMP for subsequent use as a working one.
9. If the resulting program takes up less memory in the SMP than the previous one, then this program must be deleted from the SMP, and the last line of the program must be restored in the same way as the previous one, preserving the space between the line number and the END operator. This version of the program should be written in the SMP and used as a working one.
10. After deleting any file from the SMP, it is necessary to ensure that there is only one free area on the SMP, located after the last file. If at least one free area is located between two files, it must be "pushed out" behind the last file. To do this, it is necessary to recopy each file, starting with the first file located after the first free area, using the following sequence of commands:
 LOAD, KILL, SAVE.

Example:

Справочник SM0		
A.BAS	4	4
<свободно>	3	7
B.BAS	5	12
<свободно>	4	16

← first free area

← second free area

To "push out" the first free area behind the last file (B.BAS), you need to execute the following commands in sequence:

LOAD "B" KILL "B.BAS" SAVE "B"		
after "pushing out" the free area		
Справочник SM0		
A.BAS	4	4
B.BAS	5	9
<свободно>	7	16

← free area of the SMP