MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY
"KHARKIV POLYTECHNICAL INSTITUTE"

Department of Computer Engineering and Programming

O.A. Povoroznyuk, A. I. Povoroznyuk

Laboratory workshop on the course
COMPUTER ARCHITECTURE

Approved by the editorial and publishing
department university,
protocol No 1. dated 15.02.2024

Kharkiv – 2024

УДК 004.3(075)
ББК 32.973-02я7
Р 42

Reviewer : V. V. Usyk, Candidate of technical sciences, professor of the department "Multimedia and Internet technologies and systems" NTU "KhPI"

O.A. Povoroznyuk
P42 Laboratory workshop on the course “Computer Architecture” for full-time and part-time students of speciality 123 "Computer Engineering” / O.A. Povoroznyuk, A. I. Povoroznyuk. – Kharkiv: NTU “KhPI”, 2024 – 82 c.

The laboratory workshop includes methodological instructions for 9 two-hour laboratory works, which include work on determining the PC configuration, controlling the timer and sound, the keyboard, drives on magnetic disks, and controlling the video system in text and graphic modes. Each laboratory work contains an individual task, control questions and the item "Peculiarities of programming" in one of the programming languages Turbo-C or Turbo-Pascal, taking into account the depth of knowledge of the students of a specific programming language.

Intended for students of all undergraduate majors in Computer Engineering 123, it can also be useful for both beginners and experienced programmers in creating effective software.

Fig. 39 Tab. 32. Lit. 8 names.

УДК 004.3(075)
ББК 32.973-02я7

CONTENT

INTRODUCTION

Laboratory workshop for practical work in the course "Computer Architecture" contains descriptions of 9 two-hour laboratory works.

The purpose of the laboratory work is to consolidate practical skills in managing PC modules at a low level by programming device adapters at the port level. BIOS interrupts and API functions are used only in cases where control at the port level requires complex control algorithms with specified time delays.

Since modern operating systems limit, and in many cases prohibit, access to PC resources, to ensure the required level of access, a virtual machine with Windows 98 is created, which provides the necessary actions.

Each student receives an individual assignment in accordance with the number in the journal and prepares a report of laboratory work.

Completing laboratory work involves preliminary study of the relevant section of the course and methodological instructions for this work.

To be allowed to perform laboratory work, the student must write the text of the program in accordance with the individual task.

The text of the program must be written on one of the programming languages Turbo-C or Turbo-Pascal, taking into account the depth of students' knowledge of a particular language.

When creating programs, student must use the recommendations in the paragraph "Programming Features" for this laboratory work.

Students who have not passed more than two laboratory works are not allowed to participate in laboratory work.

Laboratory work 1

Topic: Introduction to virtual machine technologies.

Purpose of work: Obtaining practical skills in installing and configuring virtual machines.

1.1. Work description.

All laboratory work is based on obtaining information or programming the hardware that is part of the PC, directly through the input / output ports, interrupts or sections of RAM. Modern operating systems (OS) do not provide such an opportunity, therefore, to perform laboratory work, you should use those OSs that allow it or use the virtualization programs of the corresponding OS. For example, you can use the software products DOSBox, VirtualBox, VMWare Workstation, etc. Next, we will describe how to configure VMWare Workstation for laboratory work.

After installing VMWare Workstation (version 8), select the "File / New virtual machine ..." menu item or press the Ctrl + N combination. On the first page of the New Virtual Machine Wizard, select the "Typical" item. On the second page, you must specify the path to the installation disk of the guest OS (in this case, Windows 98 will be used) or to the disk image file with the OS (Fig. 1.1).

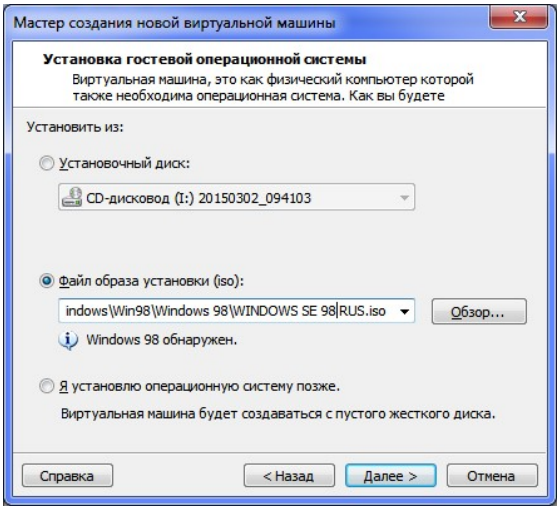

Figure 1.1 - Selecting the installation disk of the guest OS

On the next window of the wizard, you need to select a name for the guest OS and a directory for placing it on the disk (Fig. 1.2).

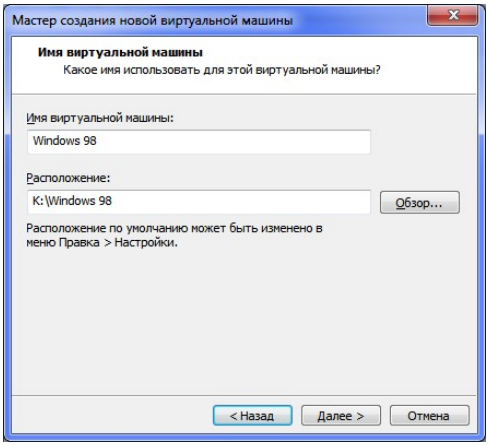

Figure 1.2 - Selecting the directory where the virtual machine files are located

After that, determine the size of the disk for the guest OS and the option of storing it (or one or more files) (Fig. 1.3).

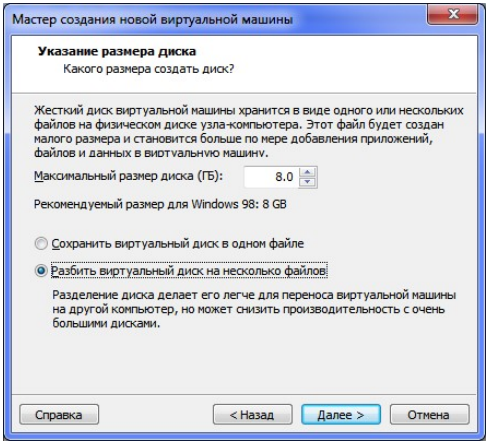

Figure 1.3 - Selecting disk parameters for the guest OS

After that, the process of creating a virtual machine is completed (Fig. 1.4).

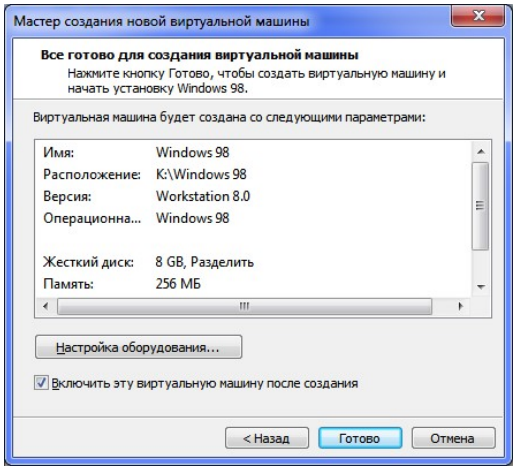

Figure 1.4 - Finishing the creation of the virtual machine

You can also configure hardware (by clicking on the "Hardware setup ..." Fig. 1.4) where you can specify standard parameters (Fig. 1.5), such as: RAM size, processor parameters, CD-drive type, network adapter, USB controller, sound card and display or install additional equipment (by clicking on the "Add ..." button in Fig. 1.5). After that, by clicking on the "Finish" button (Fig. 1.4), you need to install the guest OS.

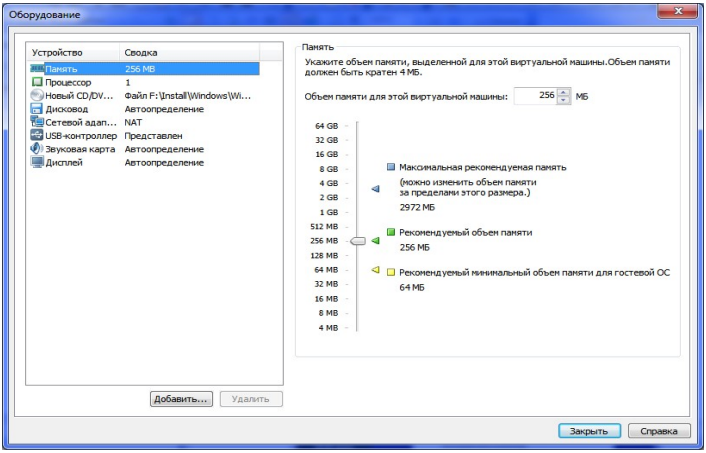

Figure 1.5 - Configuring the hardware of the guest OS

Further, if the "Enable this virtual machine after creation" option was selected (Fig. 1.4), the OS installation process will begin. This process is no different from the process of installing an OS on a real PC. Consider installing Windows 98.

The installation process begins with the selection of a boot device. In this case, you must select option 2: Boot from CD-ROM (Fig. 1.6).

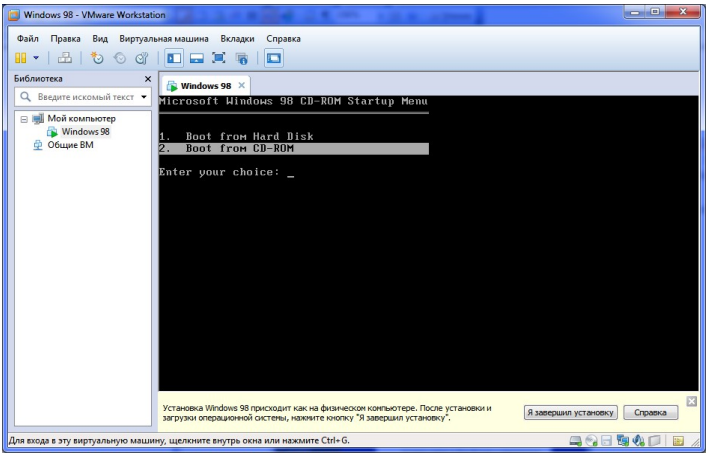

Figure 1.6 - Choosing a boot option

Next, you need to select 1 item: Start Windows 98 Setup from CD-ROM (Fig. 1.7).

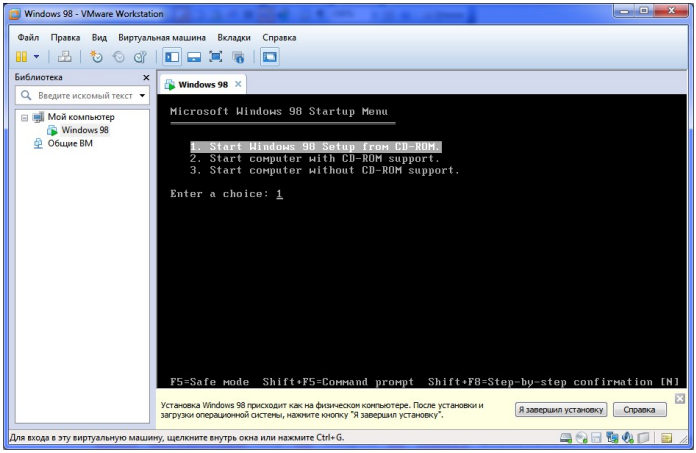

Figure 1.7 - Choosing the option to download and install the OS

At the next stage, you must agree with the installation by pressing the ENTER key (Fig. 1.8).

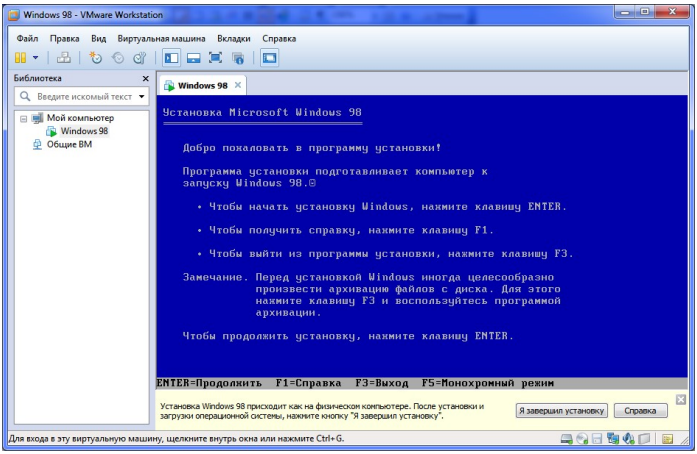

Figure 1.8 – OS installation start window

Next, the OS installer will offer to prepare free space on the disks. It is necessary to confirm this action by pressing the ENTER key (Fig. 1.9).

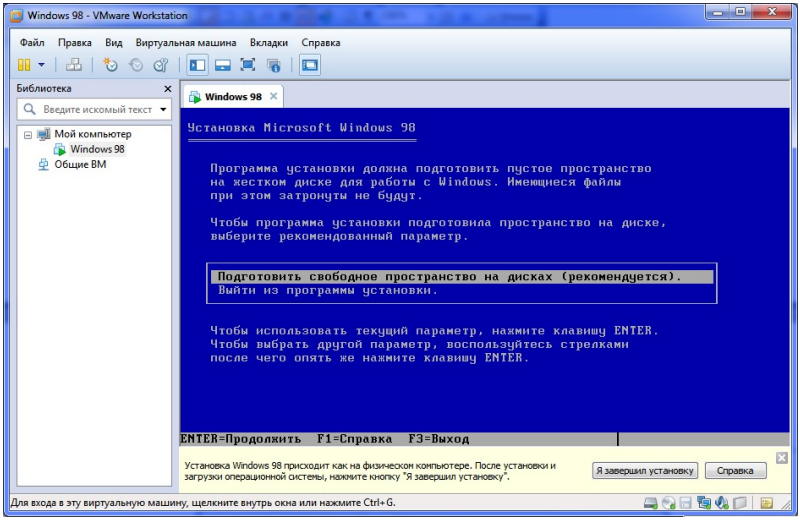

Figure 1.9 – Preparing free space on disks

Then you need to enable support for large disks (Figure 1.10).

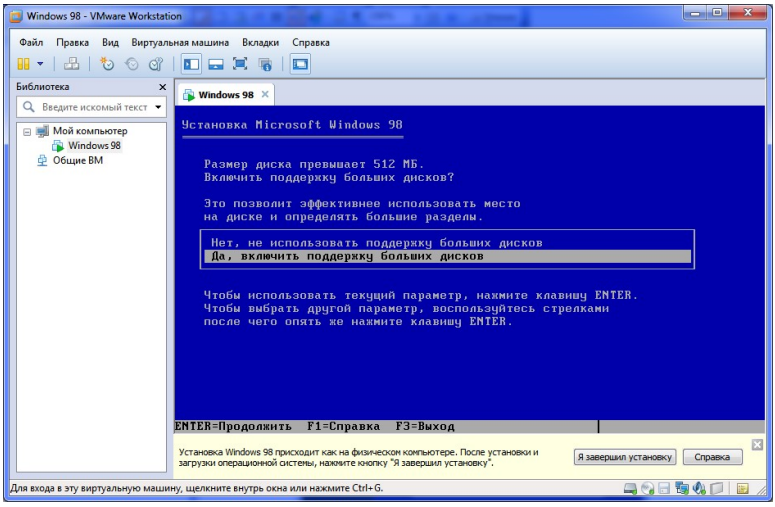

Figure 1.10 – Enabling large disk support

The OS Installation Wizard will then prompt you to reboot the OS. It is necessary to confirm this action by pressing the ENTER key (Fig. 1.11).

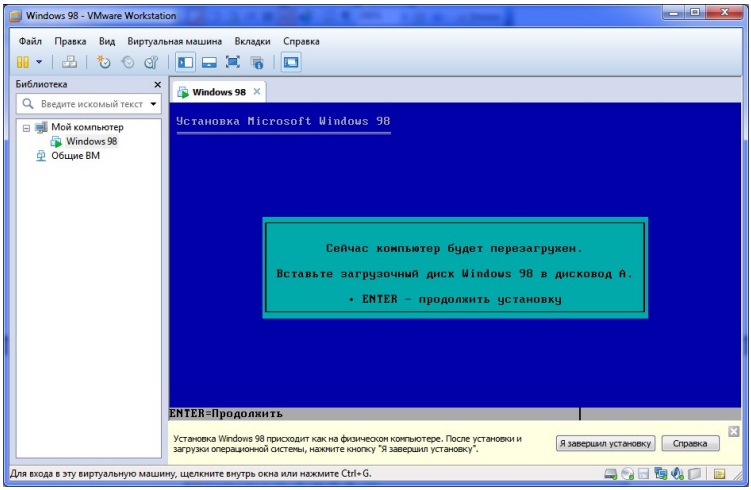

Figure 1.11 – Completing the preparation of the installation

After the reboot, you must select the same options as in fig. 1.6 and fig. 1.7. Then the wizard will format the disk (Fig. 1.12).

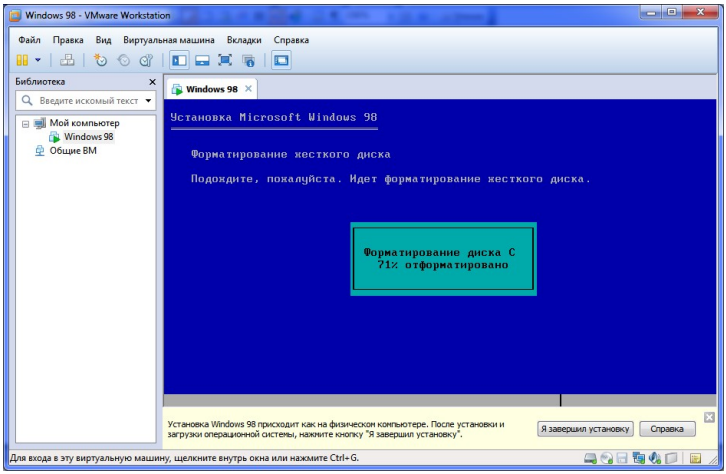

Figure 1.12 – Formatting the disk

Then you must agree with the start of the installation process by pressing the ENTER key (Fig. 1.13).

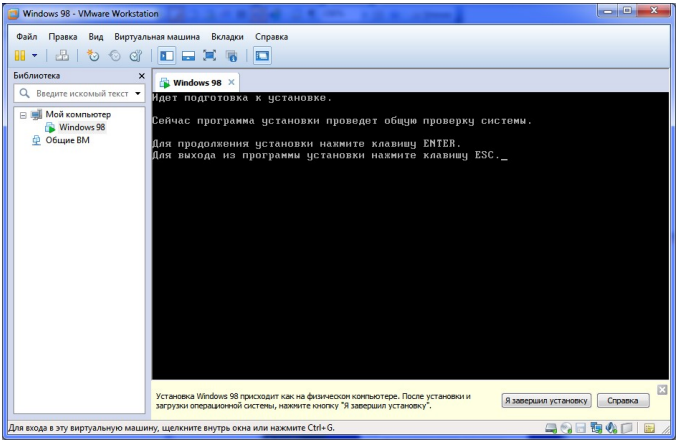

Figure 1.13 – Consent to the installation process

Next, the wizard will check the disk and proceed to the OS installation process (Fig. 1.14).

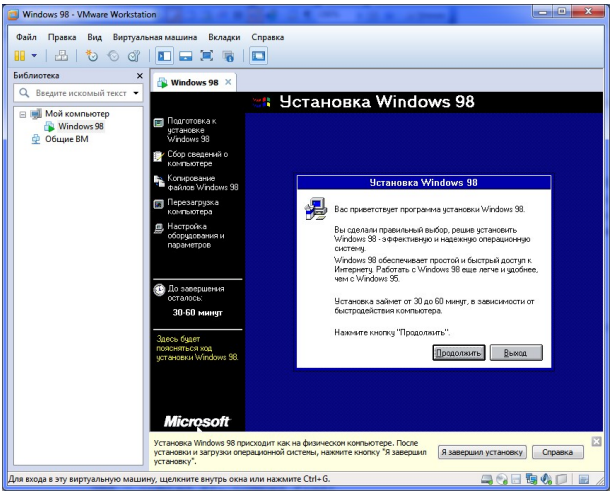

Figure 1.14 – Installing the guest OS

In this case, you need to select the OS installation folder (Fig. 1.15).

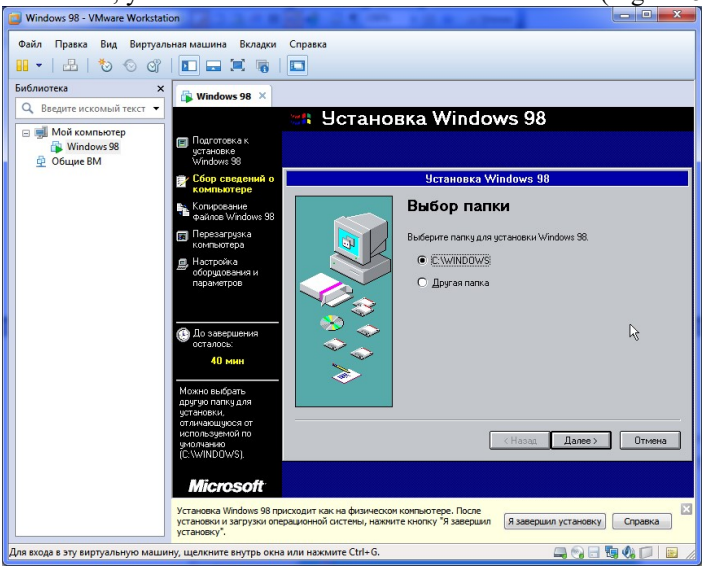

Figure 1.15 – Selecting the OS installation folder

Then you need to select the OS installation option (Fig. 1.16) and agree with the installation of the main components (Fig. 1.17).

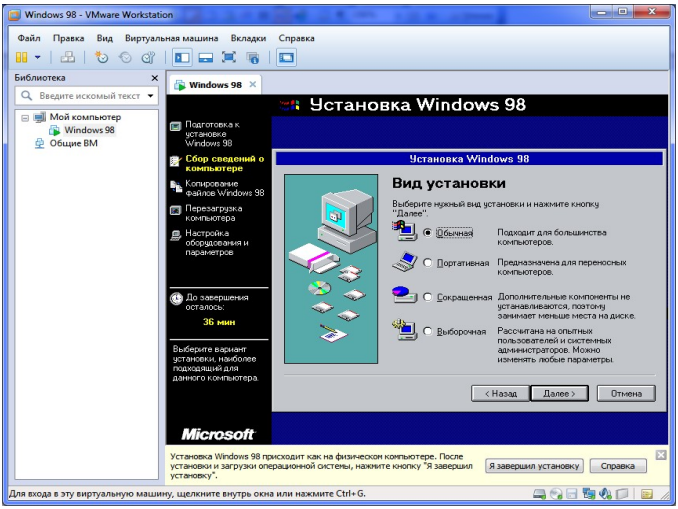

Figure 1.16 – Choosing the OS installation folder

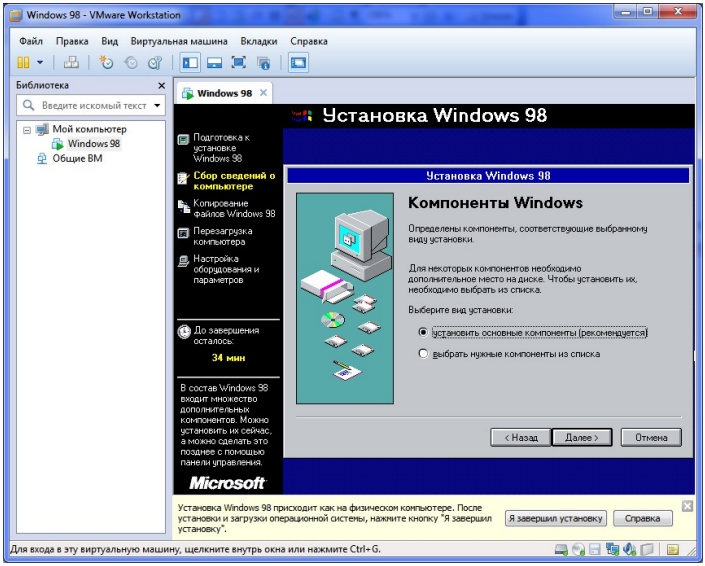

Figure 1.17 – Choosing to install the main OS components

Then you need to specify the name and description of this working machine (Fig. 1.18).

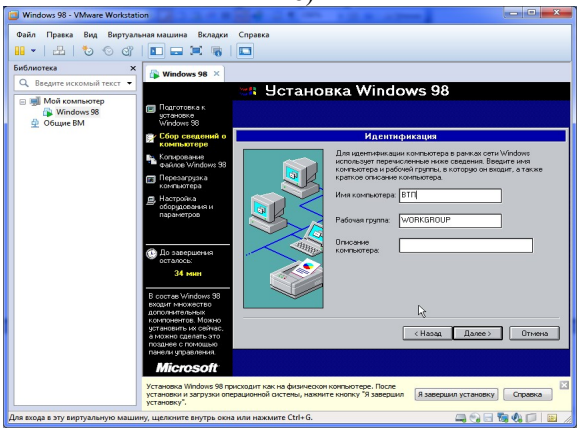

Figure 1.18 – Setting the name and description of the guest OS computer

Next, you need to select a location and agree with the start of the installation process, while the wizard proceeds to the process of copying OS files (Fig. 1.19).

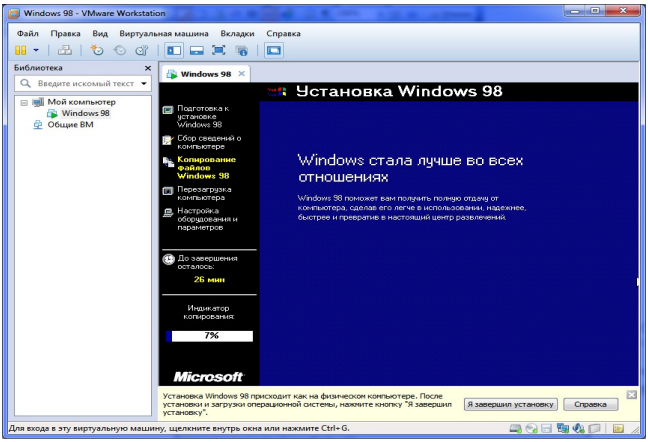

Figure 1.19 – The process of copying files from a guest OS to a virtual machine

After copying the OS files to disk, the guest OS will reboot. Then you need to fill in the information about the user (switching between keyboard layouts is done by pressing the LeftAlt + LeftShift combination), agree to the license agreement and

enter the license key and agree to continue the installation by clicking on the "Finish" button (Fig. 1.20).

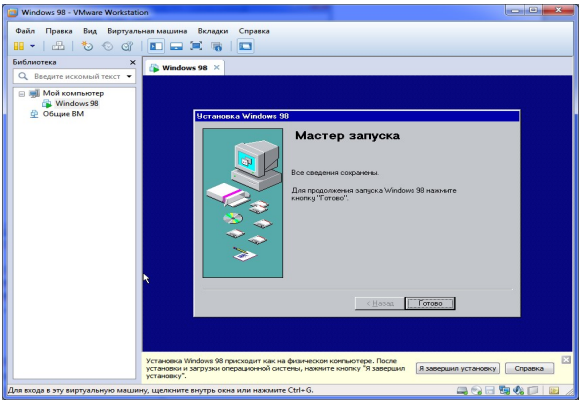

Figure 1.20 – Continuing the installation process

Then the wizard will configure the hardware and after 2 reboots, the OS will be installed. After installing the OS, its launch from VMWare Workstation is carried out by selecting it from the library and pressing the "Enable / Resume" button ( ) or the combination "Ctrl + B" (Fig. 1.21).

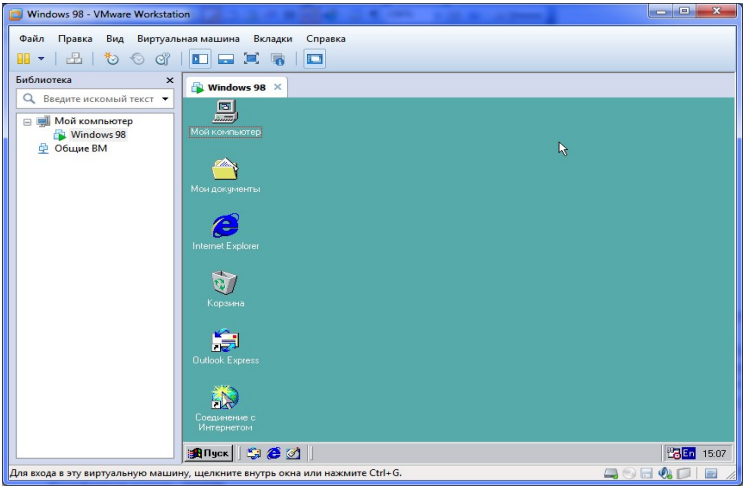

Figure 1.21 – VMWare Workstation window running Windows 98

After installing the guest OS, you need to install the VMWare Tools package for this OS. This package installs the necessary additional drivers (in particular, for Windows 98, the display and mouse drivers are installed), and also provides the ability to copy files from the guest OS to the host OS and vice versa with simple drag-and-drop operations. To install VMWare Tools, select the "Virtual Machine \ Install VMWare Tools" menu item while the guest OS is running.

After installing VMWare Tools, a general view of Windows 98 is shown in Fig. 22. The system is ready for laboratory work on the course "Computer architecture".

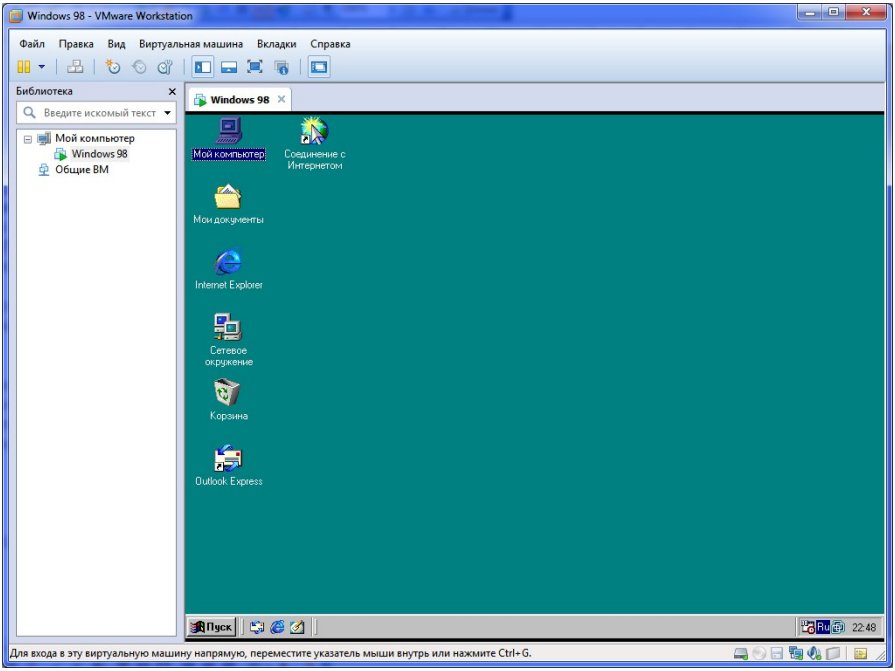

Figure 1.22 – General view of Windows 98

The directory with the installed guest OS (i.e. the OS itself) can be used on another computer with VMWare Workstation installed. To do this, you can simply copy the entire entire directory of the guest OS to another computer and select it in VMWare Workstation from the "File \ Open ..." menu or by pressing Ctrl + O.

After installing the guest OS, you need to add programming language compilers to it to perform the labs. This action can be performed by simply dragging and dropping files from the host OS to the guest OS.

1.2. Content of the report
1.2.1. Purpose of work.
1.2.2. Guest OS installation progress
1.2.3. Conclusions.

Laboratory work 2

Topic: Personal computer configuration

Purpose of work: Obtaining practical skills in determining the configuration and basic characteristics of a PC and its modules.

2.1. Topics for preliminary work

Composition, purpose and characteristics of the main PC modules. PC configuration.

2.2. Work description

During solving some problems, it is necessary to know the type of PC, the type of microprocessor, the composition of the external devices of the machine and their technical characteristics (for example, an attempt to access a program to a non-existent device may lead to the "freezing" of the operating system). This information is contained in specific locations in RAM, ROM and CMOS memory and is listed below.

2.2.1. Determination of the type of PC.

The BIOS ROM contains a byte at the address F000: FFFEh that allows you to identify the type of PC:

FFh - original IBM PC;
FEh - XT, Portable PC;
FDh - PCjr;
FCh - AT;
FBh - XT with 640 K memory on the motherboard;
FAh - PS / 2 model 25 or 30;
F9h - Convertible PC;
F8h - PS / 2 models 55SX, 70.80;
9Ah - Comrad XT, Compaq Plus;
30h - Sperry PC;
2Dh - Compaq PC, Compaq Deskpro.

2.2.2. Determination of the type of microprocessor.

The algorithm for determining the type of microprocessor is based on the differences in the flags registers (RgF) of the microprocessors (MP) 8086, 80286

and 80386 and consists of the following:

0 is written to the flag register. If bits 12-15 of the RgF are set to 1, this is MP 8086. If not, then the F000h code is written to the flag register. If after that bits 12-15 of the RgF remain at 0, then this is MP 80286, otherwise - 80386 and higher.

For MPs higher than 80386, there is a command for determining the type and parameters of the MP: CPUID. The code of this command in machine code is 0Fh 0A2h. You can get information about the manufacturer of the MP by setting the value in the EAX = 0 register. Then CPUID returns the processor manufacturer's identifier (Vendor ID) in the EBX, EDX and ECX registers in the form of 12 ASCII characters. To get the full name of the MP, you must sequentially call the CPUID command with the parameters in EAX = 80000002h, 80000003h and 80000004h. For each request in the registers EAX: EBX: ECX: EDX, a fragment of a 48-character string of the full processor name will be returned

2.2.3. Determining the BIOS creation date.

The BIOS creation date occupies 8 bytes in the BIOS ROM starting from the address F000: FFF5h and is stored in the ASCII format as mm / dd / yy, where

mm - month number; dd - day; yy - year.

2.2.4. Determining the configuration of the IBM PC AT.

2.2.4.1. Data stored in CMOS memory.

CMOS memory is organized on the basis of the MC146818 chip from Motorola and has 64 8-bit cells (registers). Table 2.1 shows the contents of those memory cells in which data about the configuration and state of the machine are stored.

The CMOS (write and read) is addressed as follows: first, the cell address (register number) is entered into port 70h. Then, depending on the operation being performed, data is either written to port 71h, or data is read from port 71h.

Table 2.1 - Contents of cells

Cell address	Content
0Eh	Power on diagnostic status byte
10h	Type of used floppy disk drive
14h	Hardware configuration
15h–16h	Main memory size
30h–31h	Extended memory size

2.2.4.1.1. Diagnostic status byte.

The Diagnostic Status Byte contains the results of the diagnostics performed when the machine is powered on. The status byte format is shown in table. 2.2.

Table 2.2 - Contents of the diagnostic byte

Bits	Values
0-1	Not used, equal to 0
2	0 - incorrect setting of the real time clock; 1 - clock is set correctly.
3	1 - hard disk malfunction, it is impossible to load the operating system from the hard disk; 0 - hard disk is OK.
4	1 - the actual size of the RAM does not match the one specified in the CMOS memory; 0 - the size of the RAM is correct.
5	1 - error in the system configuration; 0 - the configuration is correct.
6	1 - error in the checksum of the CMOS memory; 0 - CMOS checksum is correct.
7	1 - discharge of the battery supplying the CMOS memory and the real time clock; 0 - the battery is functional and charged.

2.2.4.1.2. The type of floppy disks used.

The junior and senior tetrads of this byte describe the second and first floppy disk drives, respectively:

0000 - no drive is installed;
0001 - 360K floppy drive;
0010 - 1.2M floppy drive;
0011 - 720K floppy drive;
0100 - 1.44M floppy drive.

2.2.4.1.3. Hardware configuration.

Table 2.3 - Configuration Byte

Bits	Values
0	1 - the system is equipped with floppy disk drives; 0 - floppy disk drives are not used
1	1 - the arithmetic coprocessor is installed; 0 - coprocessor is not installed..
2-3	Not used, equal to 0
4-5	Video controller type and mode: Bits: 5 4 0 0 - not used or EGA; 0 1 - CGA, EGA, VGA in 40x25 mode; 1 0 - CGA, EGA, VGA in 80x25 mode; 1 1 - monochrome controller.
6-7	The number of used floppy disk drives.

2.2.4.1.4. The size of main memory.

Sell 15h contains the low byte, and sell 16h contains the high byte of the main memory size in kilobytes.

2.2.4.1.5. The size of extended memory.

Sells 17h and 18h contain respectively the low and high bytes of the extended memory size (located above 1M) in kilobytes.

2.2.4.1.6. Data from RTC.

The data in the RTC registers is stored in binary decimal format (BCD), i.e. every 4 bits contain one decimal place. Therefore, to output information from the RTC, it is necessary to convert from BCD to binary and vice versa. Table 2.4 shows the contents of the RTC registers.

During initialization, the BIOS asks the status of the jumpers and analyzes the contents of the CMOS memory. After analysis, the BIOS writes a configuration word to its memory area at 0040: 0010h. The purpose of the individual bits of this word is shown in Table. 2.5.

Table 2.4 - RTC data

Cell address	Content
0h	Current time, seconds
1h	Alarm seconds.
2h	Current time, minutes.
3h	Alarm minutes.
4h	Current time, hours.
5h	Alarm hours.
6h	Current day of the week.
7h	Current day of month.
8h	Current month.
9h	Current year.

Table 2.5 - The content of the configuration word

Bits	Values
0	1 - the system is equipped with floppy disk drives; 0 - floppy disk drives are not used
1	1 - the arithmetic coprocessor is installed; 0 - coprocessor is not installed.
2-3	The size of main memory installed on the system board: Bits: 3 2 0 1 - 16K; 1 0 - 32K; 1 1 - 64K and more.
4-5	Video controller type and mode: Bits: 5 4 0 0 - not used or EGA; 0 1 - CGA, EGA, VGA in 40x25 mode; 1 0 - CGA, EGA, VGA in 80x25 mode; 1 1 - monochrome controller.
6-7	The number of used floppy disk drives.
8	1 - DMA controller is used; 0 - DMA controller is not used.
9-11	The number of serial ports installed.
12	1 - a game adapter (joystick) is used; 0 - no game adapter is used.
13	1 - Serial printer installed (PCjr only).
14-15	The number of printers installed.

2.3. Work order

2.3.1. In accordance with the individual task, using the reference information on the technical characteristics of the PC, given in section 2.2, determine the address or area of ROM, RAM or CMOS memory containing the required information.

2.3.2. Write a program for determining the required characteristics of a PC.

2.4. Programming features

2.4.1. In Pascal

The predefined arrays Mem and MemW are used to access the RAM and ROM cells. For example, to read a word from RAM location 0040: 0010h, use the expression wo: = MemW [$ 0040: $ 0010], where wo is a word type variable; to read a byte from ROM at address F000: FFF5h, use the expression b: = Mem [$ f000: $ fff5] (b is a byte variable).

To access the PC ports, the predefined arrays Port and PortW are used. For example, to write the value 10h to port 70h, use the expression Port [$ 70]: = $ 10; to read from port 71h the expression data: = Port [$ 71] is used, where data is a byte variable.

To highlight a certain bit in a byte, a bit mask is used, which contains one in the checked bit and zeros in the rest. The bitmask is ANDed with the byte to give a value of 0 if the specified bit contains 0 and a value of 1 otherwise. For example, to determine the value of the 0th bit of the variable b (of type byte), you need to write:

```
if(b and $01)=0 then writeln('0-th bit of byte b contains 0')
    else writeln(' '0-th bit of byte b contains 1');
```

2.4.2. In C.

2.4.2.1. To access the RAM and ROM cells, far pointers are used, which are declared in the program as follows:

char far * uk; - to work with bytes

int far * uk; –To work with double-byte words (variables of int type).

In this case, the low byte follows first, then the high byte.

For example, to read a word from RAM at 0040: 0010h, use the expression:

uk = (int far *) 0x00400010;

wo = * uk;

where: wo is a variable of type int.

To read a byte from ROM at address F000: FFF5h, the following expression is

used:

uk = (char far *) 0xF000FFF5;
b = *uk;
where: b is a variable of type char.

To call machine instructions, you must use the built-in assembler. So the program code that determines the manufacturer of the MP will look like:

```
char CPUVendor[12];
 asm {
 mov eax,0h;
 db 0fh,0A2h
 mov dword ptr CPUVendor,ebx
 mov dword ptr CPUVendor+4,edx
 mov dword ptr CPUVendor+8,ecx
 }
```

To access the ports of the PC, the functions of reading and writing the port are used, which are stored in the <dos.h> library. The library is connected by the directive

#include <dos.h>
after which, for example, to write the value 10h to port 70h, use the function:
outportb (0x70,0x10).
To read from port 71h, use the function:
data = intportb (0x71);
where data is a variable of type char.

To highlight certain bits in a byte, a bit mask is used, which is added to the byte according to the bitwise conjunction scheme & (not to be confused with the logical conjunction &&), as a result of which a true value (1) is obtained if the given bit contains 1 and a false value (0) in otherwise. For example, to determine the value of the 0 bit of the variable b (of type char), you need to write:

```
if(b&0x1) printf("0-th bit b contains 1\n");
    else printf("0-th bit b contains 0\n").
```

2.5. Individual tasks

1. Determine the type of IBM; number of floppy disk drives; check if the battery is good.
2. Determine the day the BIOS was created; the size of main memory and the number of connected printers.

3. Determine the month the BIOS was created and the number of connected serial ports; check if coprocessor is installed.

4. Determine the type of 1st floppy disk drive and the year the BIOS was created; check if the PC has a direct memory access controller.

5. Determine the type of video controller, its mode and the date of BIOS creation; check the serviceability of the hard disk.

6. Determine the type of IBM; determine if this PC has an EGA video controller; check if the real time clock is set correctly.

7. Determine the size of extended memory; determine the day the BIOS was created; check if coprocessor is installed.

8. Determine the type of the 2nd floppy disk drive (if any); the size of main memory on the motherboard and the year the BIOS was created.

9. Determine the number of connected printers and the date of the BIOS creation; check if the real time clock is set correctly.

10. Determine the current time; determine the month when the BIOS was created; check if the PC has a direct memory access controller.

11. Determine the type of microprocessor and the type of PC; check if the battery is good.

12. Determine the current date; determine the year the BIOS was created and the number of connected printers.

13. Determine the type of IBM; the size of extended memory and the number of connected serial ports.

14. Determine the type of video controller, its mode; check if coprocessor is installed; determine the month the BIOS was created.

15. Determine the current month; determine the number of connected printers and the date the BIOS was created.

16. Determine the type of IBM; the size of extended memory; check if the PC has a direct memory access controller.

17. Determine the current date, the year the BIOS was created and the number of connected printers.

18. Determine the type of microprocessor and the date of BIOS creation; check if the real time clock is set correctly.

19. Determine the type of IBM; determine the type of video controller, its mode; check if coprocessor is installed.

20. Determine the current time; the size of main memory on the system board and the type of microprocessor.

2.6. Content of the report

2.6.1. The topic of the laboratory work.

2.6.2. Purpose of work.

2.6.3. Individual task.
2.6.4. Program text.
2.6.5. The results of the program.
2.6.6. Conclusions.

Laboratory work 3

Topic: Organization of the work of the PC keyboard

Purpose of work: Studying the organization and principles of the keyboard and the acquisition of practical skills in controlling the keyboard microprocessor.

3.1. Topics for preliminary work

Organization and principle of operation of the PC keyboard.

3.2. Work description

3.2.1. Principles of the keyboard. In fig. 3.1 is a simplified keyboard layout.

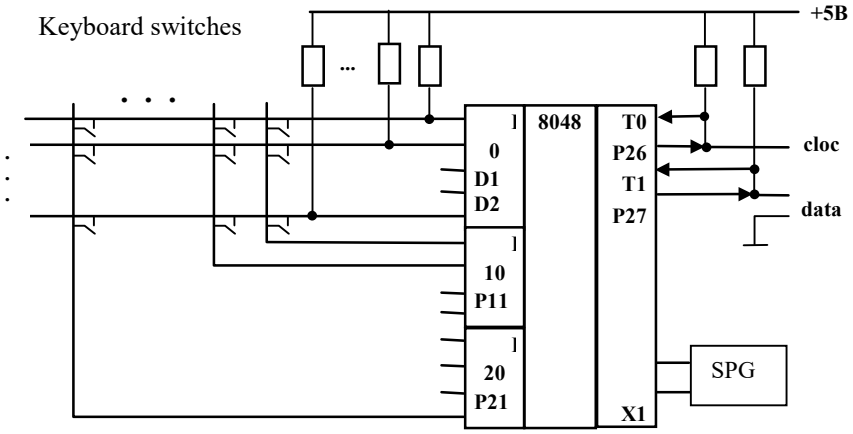

Figure 3.1 - Keyboard layout

A specialized microprocessor Intel 8042 on the motherboard is used as a keyboard controller, and on the keyboard itself - Intel 8048, which has two ports - input and output. The input port (X) is connected to the horizontal lines of the matrix, and the output (Y) is connected to the vertical ones. All horizontal lines are also connected via resistors to the + 5V power supply.

By setting the voltage level on each of the vertical lines in turn, corresponding to a logical zero, the Intel 8048 keyboard controller asks the state of the horizontal lines. If there are no pressed keys, the voltage level on all horizontal lines

corresponds to logic 1 (since all these lines are connected to the + 5V power supply through resistors). If the user presses a key, the corresponding vertical and horizontal lines will be closed. When the processor sets a logical zero on this vertical line, the voltage level on the horizontal line will also correspond to a logical zero.

As soon as a logical zero level appears on one of the horizontal lines, the keyboard processor records the keystroke.

The number of the fixed key is uniquely associated with the pinout of the keyboard matrix and does not directly depend on the designations applied to the surface of the keys. This number is called the Scan Code. Similar actions are performed when the user unpresses the previously pressed key. Note that the scan code of the pressed key differs from the scan code of the unpressed key by one in the high bit (i.e. more by 128).

The word “scan” emphasizes that the keyboard processor is scanning the keyboard to find the key pressed.

Having received the scan code from the Intel 8048 keyboard, the Intel 8042 sends an interrupt request (Int 09h) to the central processor and sends the scan code to port 60h.

If you press and hold the key, the keyboard will enter autorepeat mode. In this mode, the code of the pressed key is automatically sent to the central processor after a certain period of time, called the autorepeat period.

3.2.2. Keyboard ports.

To work with the keyboard, the PC uses ports with addresses 60h and 64h.

Port 64h is used to write commands to the 8042 controller (if the command contains a data byte, it is transmitted through port 60h), during reading, port 64h contains the status word 8042. Since the 8048 controller has no direct connection to the system bus, commands are sent to it through the controller 8042, for which both command and data are transmitted through port 60h. During reading, port 60h contains the scan code of the last key pressed.

3.2.3. 8048 keyboard controller commands.

The 8042 controller serves not only the keyboard, but also other computer systems (for example, it is used to return the central processor from the protected mode of operation to the real one). He carries out all his actions by executing the appropriate commands.

To send a command, you must first make sure that its internal command queue is empty (if a command consists of several bytes, the check is performed for each byte). This can be done by reading status word 8042 from port 64h. Bit number 1 must be zero (bits are numbered from 0). After the program waits for the 8042

controller to be ready, it can send it a command by writing it to the corresponding port with the address 64h or 60h (this is done for each command byte). The 8048 controller uses the following commands to control the keyboard, which are sent to port 60h:

3.2.3.1. The command for setting the characteristics of the autorepeat mode.

The command consists of two bytes. First byte: command code - F3h; the second byte defines the characteristics of the mode, the bits of which have the following purpose:

- 4-0: number of repetitions per second (see table 3.1);
- 6-5: initial delay before autorepeat generation in ms (00 = 250, 01 = 500, 10 = 750, 11 = 1000);
- 7: reserved, must be 0.

Table 3.1 – Autorepeat parameters

Constant	Frequency	Constant	Frequency	Constant	Frequency
00h	30.0	0Bh	10.9	16h	4.3
01h	26.7	0Ch	10.0	17h	4.0
02h	24.0	0Dh	9.2	18h	3.7
03h	21.8	0Eh	8.6	19h	3.3
04h	20.0	0Fh	8.0	1Ah	3.0
05h	18.5	10h	7.5	1Bh	2.7
06h	17.1	11h	6.7	1Ch	2.5
07h	16.0	12h	6.0	1Dh	2.3
08h	15.0	13h	5.5	1Eh	2.1
09h	13.3	14h	5.0	1Fh	2.0
0Ah	12.0	15h	4.6		

The autorepeat period determines the number of scan code transmissions generated by the keyboard processor per second.

Initially, when the system is initialized, the delay period for turning on the autorepeat mode is set by BIOS modules to 500 ms with an autorepeat period equal to 10 repeates per second.

3.2.3.2. Command to control the keyboard LEDs.

The command consists of two bytes. First byte: command code - EDh; To turn the LEDs on or off, after the command code (EDh), a data byte is sent via port 60h, the bits of which have the following purpose (Table 3.2):

Table 3.2 - The meaning of the LED control byte

Bit	Value
0	1 – turn on ScrollLock;
1	1 - turn on NumLock;
2	1 - turn on CapsLock;
3-7	not used

3.2.3.3. The command to return to the values set by the BIOS. The command code is F6h.

3.3. Execution order

3.3.1. According to the individual task:
- set the appropriate period and delay of autorepeat;
- blink the corresponding LED.

3.3.2. By reading the scan code from port 60h, determine the key, the scan code of which matches the student number in the journal. Determine the scan code for releasing this key. During determining scan codes, you can use the algorithm shown in Fig. 3.2.

3.3.3. After completing paragraphs. 3.1 and 3.2 restore the characteristics of the keyboard to its original state.

3.4. Programming features

3.4.1. In Pascal.

Use the predefined *Port* array to read data from a port and write data to a port. For example, to read from port 64h, use the expression b: = Port [$ 64], and write to port 60h, use the expression Port [$ 60]: = b, where b is a byte variable.

3.4.2. In the C.

To access the ports of the PC, the read and write functions of the port are used, which are stored in the <dos.h> library. The library is connected by the directive

#include <dos.h>

after which, for example, to write the value of b to port 60h, the following expression is used:

outportb (0x60, b);
To read from port 64h, the following expression is used:
b = intportb (0x64);
where: b is a variable of type char.

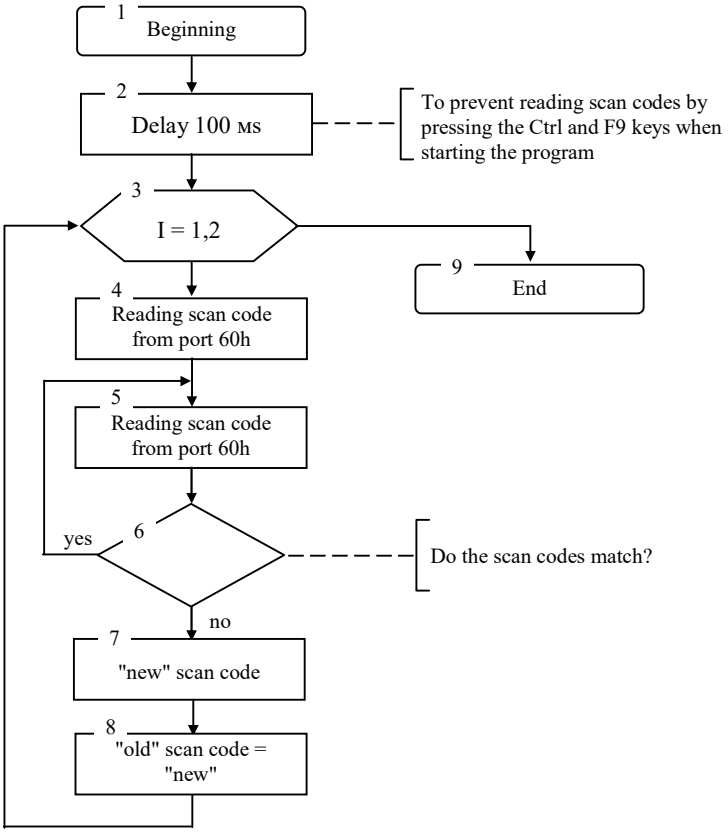

Figure 3.2 - Algorithm for determining scan codes for pressing and releasing

3.5. Individual tasks

Select individual tasks in accordance with the table. 3.3:

Table 3.3 - Variants for individual tasks

Number in journal	Autorepeat period	Autorepeat delay, (мs)	LED name
1	30,0	250	NumLock
2	26,7	500	CapsLock
3	24,0	750	ScrollLock
4	20,0	1000	NumLock
5	2,0	250	CapsLock
6	3,0	500	ScrollLock
7	30,0	750	NumLock
8	26,7	1000	CapsLock
9	24,0	250	ScrollLock
10	20,0	500	NumLock
11	2,0	750	CapsLock
12	3,0	1000	ScrollLock
13	5,0	250	NumLock
14	6,0	500	CapsLock
15	8,0	750	ScrollLock
16	10,0	1000	NumLock

3.6. Content of the report

3.6.1. The topic of the laboratory work.
3.6.2. Purpose of work.
3.6.3. Individual task.
3.6.4. Program text.
3.6.5. The results of the program.
3.6.6. Conclusions.

Laboratory work 4

Topic: PC keyboard buffer

Purpose of work: Studying the organization of the keyboard buffer and acquiring practical skills in determining the ASCII and scan codes of keyboard keys stored in the buffer

4.1. Topics for preliminary work

-Organization of the keyboard buffer.
-Formation of ASCII codes.

4.2. Work description

The program written in BIOS ROM to process interrupt 09h, caused by the keyboard controller when a key is pressed (released), reads the scan code from port 60h and, if it is the scan code of the switch key (right Shift, left Shift, Ctrl, Alt, ScrollLock, NumLock , CapsLock and Insert), the contents of memory cells 0040: 0017h and 0040: 0018h, storing the states of the toggle keys, are changed. In this case, nothing is written to the keyboard buffer (the exception is the Insert key).

Table 4.1 shows the format of the byte at 0040: 0017h, and in table. 4.2 - at 0040: 0018h.

Table 4.1 - Format of the byte at 0040: 0017h

Bit	Value, when bit=1
0	Right Shift key pressed
1	Left Shift key pressed
2	Ctrl key pressed
3	Alt key pressed
4	ScrollLock mode is on
5	NumLock mode is on
6	CapsLock mode is on
7	Insert mode is on

Table 4.2 - Format of the byte at 0040:0018h.

Bit	Value, when bit=1
0	Left Shift keys pressed together with Ctrl
1	Left Shift keys pressed together with Alt
2	SysReq key pressed
3	Ctrl keys pressed together with NumLock
4	ScrollLock mode is on
5	NumLock mode is on
6	CapsLock mode is on
7	Insert mode is on

The states of the switch keys can be changed by software.

If this is the scan code of a key that is not a switch, then in accordance with its scan code and the state of the switch keys, either the one-byte ASCII code of this key is generated, which, together with the scan code, is written to the keyboard buffer, or the two-byte extended code is the first which byte is 0, and the second byte in most cases matches the scan code. These two bytes are also written to the keyboard buffer. When the key is released, nothing is written to the buffer and only the state of the switch keys changes.

The 09h interrupt service program also monitors some key combinations.Table 4.3 shows these combinations and the actions performed by the interrupt handler when they are detected:

Table 4.3 - Key functions

Key combination	Actions performed
Ctrl-Alt-Del	System reset
Ctrl-NumLock	Putting the machine into waiting state
Shift-PrtSc	Printing of screen contents (interrupt 05h handling)
Ctrl-Break	End of program execution

The structure of the keyboard buffer is shown in Fig. 4.1. The keyboard buffer is located at addresses 0040: 001h - 0040: 003Ch of RAM and allows accumulating data (ASCII and scan codes) about 15 keystrokes.

Cell 0040: 001Ah (start indicator) stores the address of the first entered character, and cell 0040: 001Ch (tail indicator) stores the address of the first free cell after the last entered character.

Addresses are stored as an offset byte (the second byte of the cell is not used). If the buffer is empty, then the content of the start pointer matches the content of the tail pointer

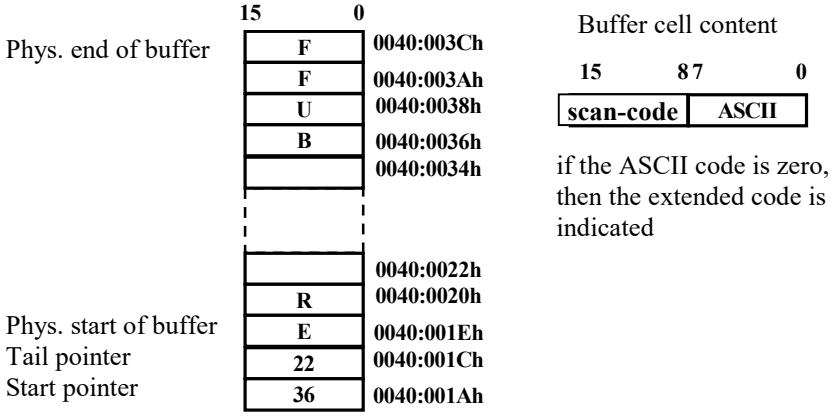

Figure 4.1 - Keyboard buffer structure

Writing to the buffer is carried out at the address of the tail pointer, while the contents of the pointer are increased by 2. Reading from the buffer is performed at the address of the start pointer, while the contents of the pointer also increase by 2 (if the pointer contains the address of the last cell of the buffer - 3Ch, then instead of increasing by 2 it is necessary to write the address of the beginning of the buffer - 1Eh).

4.3. Execution order

Write a program that implements the algorithm for reading ASCII and scancodes from the keyboard buffer shown in fig. 4.2.

In accordance with the individual task, determine the ASCII and scan code of the required keys.

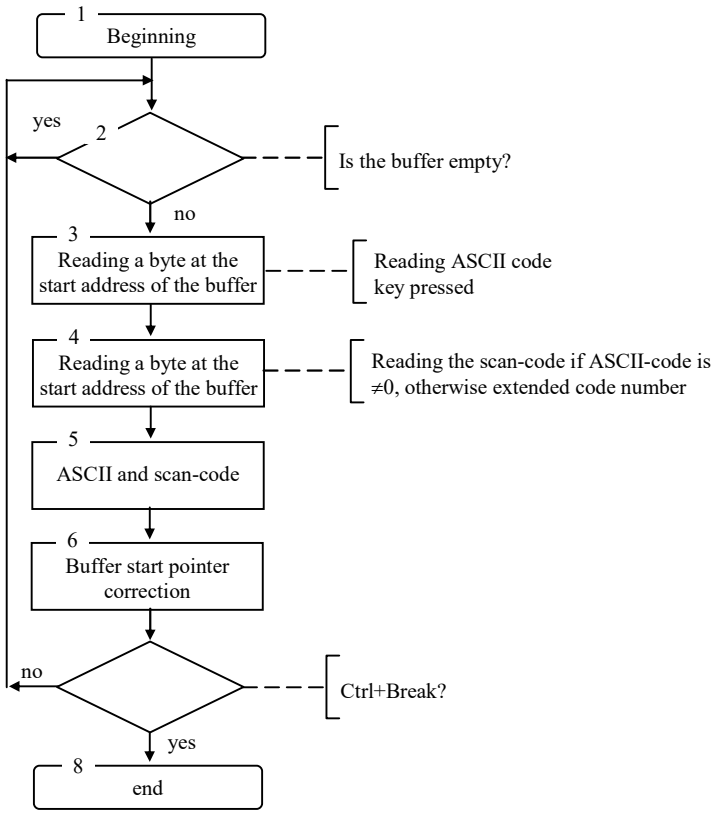

Figure 4.2 - Algorithm for reading the keyboard buffer

4.4. Programming features

4.4.1. In Pascal.

To read data from the keyboard buffer, use the predefined array Mem. For example, the expression Mem [$ 0040: $ 001C] is used to read the contents of the tail pointer, and Mem [$ 0040: Mem [$ 0040: $ 001A]] is used to read the ASCII code from the beginning of the buffer.

4.4.2. In C.

To read data from the keyboard buffer, far pointers, which are declared in the

program as follows:

char far *uk;

For example, to read the contents of a tail pointer, use the expression

uk = (char far *) 0x0040001C;

b = * uk;

and to read ASCII code from the beginning of the buffer

uk = (char far *) 0x0040001A;

b = * uk;

where: b is a variable of type char.

4.5. Individual tasks

Determine the ASCII code of the Russian letter, the number of which in the alphabet coincides with the student's number in the journal.

Determine the scan code for this key.

4.6. Content of the report

4.6.1. The topic of the laboratory work.
4.6.2. Purpose of work.
4.6.3. Individual task.
4.6.4. Program text.
4.6.5. The results of the program.
4.6.6. Conclusions.

Laboratory work 5

Topic: System timer. Sound generation

Purpose of work: Studying the functions of the system timer and acquiring practical skills in working with a timer and a speaker when generating sound.

5.1. Preliminary work topics

The structure and purpose of the ports of the configuration chip and the system timer.

5.2. Work description

5.2.1. System timer.

All IBM computers contain a device called a system timer. This device is connected to the IRQ0 interrupt request line and generates an INT 8h interrupt approximately 18.2 times per second (the exact value is 1193180/65535 times per second).

During initialization, the BIOS sets up its own timer interrupt handler. Each time this program increments by 1 the current value of a 4-byte variable located in the BIOS data area at 0040: 006Ch - the timer tick counter. If this counter overflows (i.e. more than 24 hours have passed since the timer started), 1 is set in cell 0040: 0070h.

The standard timer interrupt handler also monitors the operation of the floppy drive motors. If more than 2 seconds have passed since the last access to the floppy disk drive, the interrupt handler turns off the engine (the cell with the address 0040: 0040h contains the time remaining until the engine turns off).

The last action that the timer interrupt handler performs is the INT 1Ch interrupt call. After the system initialization, the vector INT 1Ch points to the IRET command, i.e. nothing gets executed. The user program can install its own handler for this interrupt in order to perform any periodic action.

It should be noted that the INT 1Ch interrupt is called before the interrupt controller is reset, therefore, while the INT 1Ch interrupt is being executed, all hardware interrupts are disabled. To reset the interrupt controller, write the value 20h to port 20h.

The timer is usually implemented on an Intel 8253 chip (for IBM PC and IBM XT) or 8254 (for IBM AT and IBM PS / 2)

5.2.2. Composition and operating modes of the timer chip.

Timers 8253 and 8254 consist of three independent channels. Each channel contains registers:

- the state of the RS channel (8 bits);
- control word PSW (8 bits);
- buffer register OL (16 bits);
- counter register CE (16 bits);
- register of conversion constants CR (16 bits).

Timer channels are connected to external devices using three lines:

GATE - control input;

CLOCK - clock frequency input;

OUT - timer output.

The CE counter register operates in subtraction mode. Its content decreases on the falling edge of the CLOCK signal, provided that the logic 1 level is set at the GATE input. Depending on the timer operation mode, when the CE counter reaches zero, the OUT output signal changes in one way or another.

The OL buffer register is intended for storing the current contents of the CE counter register without stopping the counting process. After memorizing, the buffer register is available to the program for reading.

The conversion constants register CR can be loaded into the counter register if required in the current mode of the timer operation.

There are 6 modes of timer operation. They are divided into three types:

• modes 0 and 4 - one-time execution of functions;

• modes 1 and 5 - work with restart;

• modes 2 and 3 - work with autoload.

In the one-time execution mode, before the start of counting, the contents of the CR constant register are rewritten into the CE counter register by the CLOCK signal, if the GATE signal is set to 1. Subsequently, the contents of the CE register are decreased as the CLOCK signals arrive.

The counting process can be suspended if a logic level 0 is applied to the GATE input. If then 1 is fed to the GATE input, the counting will continue. To repeat the execution of the function, a new load of the CR register is required, i.e. re-programming the timer.

With restart operation, the timer does not need to be reprogrammed to perform the same function. On the edge of the GATE signal, the value of the constant from the CR register is rewritten into the CE register, even if the current operation has not been completed.

In autoload mode, the CR register is automatically overwritten in the CE register upon completion of the count. The signal at the OUT output appears only when the GATE input has a logic level 1. This mode is used to create programmable

pulse generators and generators of square-wave pulses (meanders).

All three timer channels are used in IBM PC / XT / AT / PS2 computers.

Channel 0 is used by the operating system to generate time slots. This channel operates in Mode 3 and is used as a pulse generator with a frequency of approximately 18.2 Hz. It is these pulses that trigger the INT 8h hardware interrupt.

Channel 1 is used to regenerate the contents of the computer's dynamic memory, so it is better not to touch it. The output line of the OUT channel is connected to a direct memory access (DMA) chip, and its pulse causes the DMA to refresh the memory.

Channel 2 is connected to a loudspeaker (speaker) of the computer and can be used to generate various sounds or music, or as a random number generator. The channel uses timer mode number 3.

5.2.3. Timer programming.

The timer corresponds to 4 I/O ports with the following addresses: 40h - channel 0; 41h - channel 1; 42h - channel 2; 43h - control register. The byte format of the control register (control word) is shown in Table. 5.1.

Table 5.1 - Format of the control register

Bits	Values
0	0 - binary data; 1 - data in the form of BCD (binary decimal).
3-1	mode number: 000 - mode 0 (timer interrupt); 001 - mode 1 (programmable waiting multivibrator); X01 - mode 2 (programmable pulse generator); X11 - mode 3 (meander generator); 100 - mode 4 (software-triggered one-shot); 101 - mode 5 (hardware-triggered one-shot).
5-4	type of action: 00 - transfer the counter values to the buffer (read on the fly); 01 - read / write only high byte; 10 - read / write only low byte; 11 - read / write first low byte then high.
7-6	channel number: 00 - channel 0; 00 - channel 0; 01 - channel 1; 10 - channel 2; 11 - RBC command code (reverse reading).

The low bit of the byte determines the format of the constant used for counting. In BCD mode, the constant is in the range 1 –9999.

To program the timer channel, you must perform the following sequence of actions:

• write the control word to the control register at address 43h;

• send the required counter value to the channel port (addresses 40h - 42h), and the low byte is sent first, and then the high byte of the counter value.

Immediately after this, the timer channel begins to perform the required function.

5.2.4. Generation of sounds and music.

Sounds and music on a PC can be played in two ways:

• using a timer;

• without using a timer.

In any case, when an electrical signal of a certain level arrives at the speaker, the speaker membrane is retracted, and when the level of the electrical signal changes, it is pushed out. Thus, the periodic electrical signal vibrates the speaker membrane, i.e. generating sound of the same frequency.

Note that in all IBM models, with the exception of the IBM PCjr, which has (in addition to the speaker) a special sound generator, the sound volume does not change.

5.2.4.1. Speaker control using a timer.

One of the most common uses of a timer is to generate sounds and play music. The timer allows you to play music in the background, i.e. while the program is running, music may sound. As noted above, channel 2 of the 8254 is connected to the speaker. However, the speaker is not simply connected to the OUT of channel 2. Port 61h is also used to drive the speaker (Figure 5.1).

The low bit of this port, when set to 1, enables the channel to work, i.e. generation of impulses for the speaker. Additionally, bit 1 of port 61h is used to drive the speaker. If this bit is set to 1, pulses from channel 2 of the timer will be able to pass to the speaker.

The system pulse generator (SPG), regardless of the type and performance of the IBM computer, generates pulses of the same frequency - 1,193,180 Hz.

Thus, to turn on the sound, you need to do the following:

-to program channel 2 of the timer to the required frequency (i.e. load into the channel counter register a value equal to the ratio of the frequency of the SPG and the required sound frequency);

-to turn on sound, set the two low digits of port 61h to 1.

Since the remaining bits of port 61h are used for other purposes, they should be left unchanged.

To turn off the sound, you need to clear the two low bits of port 61h (again, without changing the state of the remaining bits).

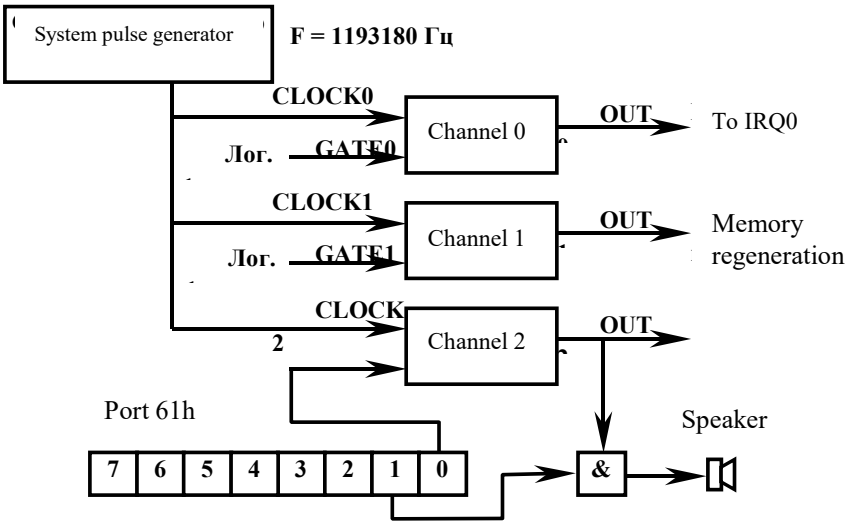

Figure 5.1 - Timer scheme

5.2.4.2. Speaker control without timer.

To generate sound without a timer, clear the low bit of port 61h (thereby disabling the operation of channel 2 of the timer) and controlling bit 1 of this port, i.e. setting this bit to 1, then to 0, generate impulses for the speaker. The pitch of the generated sound will correspond to the period of the pulses. Note that this method can generate pulses of any duty cycle, which gives more opportunities for creating various sound effects.

5.2.4.3. Play music.

A melody, as you know, consists of notes, separated or not separated by pauses. During playing a melody, it is necessary for each note to program accordingly channel 2 of the timer and turn on the speaker (using port 61h) for a certain time equal to the duration of the note. The program should then turn off the speaker and pause, if necessary, before playing the next note. Table 5.2 shows the frequencies for notes of the 2nd octave. For other octaves, when the pitch is lowered or raised, the frequency values must be divided or multiplied by two.

Table 5.2. Frequencies of notes of the second octave

Note	Frequency Hz	Note	Frequency Hz
C	267,7	F- sharp	370,0
C- sharp	277,2	G	392,0
D	293,7	G- sharp	415,3
E- sharp	311,1	A	440,0
D	329,6	A- sharp	466,2
F	349,2	B	493,9

Smooth transitions of tones are made by continuously changing the frequency. This sound effect can be made more expressive by slightly decreasing the duration of each tone as the sound rises, or slightly lengthening the duration as it decreases.

You can use the speaker control feature to create various effects that, in particular, make video games very interesting and engaging. All sound effects are based on varying the frequency of the sound - often in the most unusual way.

For example, to create a siren sound effect, you must vary the frequency of the sound between the two end points. The pitch should change from low to high and then decrease from high to low.

To simulate the sound of the explosion, which is used in many video games, you can modify the sound of the siren so that it only generates a sound downward frequency.

5.3. Execution order

Play a "melody" according to an individual task.

To do this it's necessary:

• program channel 2 of the timer in mode 3 to the desired frequency (determined by the note, see Table 5.2);

• using port 61h, set the required duration of the sounding of the note and the duration of the pause (the duration of the note in seconds is specified in an individual task after the name of the note).

Play a sound effect without using a timer.

5.4. Programming features

5.4.1. In Turbo-Pascal language.

To highlight the low and high bytes of a word, it is convenient to use the Lo and Hi functions:

Lo (w) - allocates the low byte of the word w;

Hi (w) - allocates the high byte of the word w, where w: word.

Use the predefined Port array to write data to the port. For example, to write

the low byte of the kd variable to the channel 2 counter register, you can use the expression Port [$ 42]: = Lo (kd), where kd is a word variable that sets the frequency division of the system generator by the required frequency.

To set some bits of a word or byte without changing the rest of the bits, use the OR operation with a mask containing 1 in the required bits and 0 in the rest; to reset - an AND operation with a mask containing 0 in the required bits and 1 in the rest.

To perform a software delay, use the Delay procedure described in the Crt module. The procedure argument is in ms, for example, Delay (1000) means a delay of 1 second.

5.4.2. In the Turbo-C language.

To write data to a port, use the port write functions that are stored in the <dos.h> library. The library is connected by the directive

#include <dos.h>

after which, for example, to write the low byte of the kd variable into the channel 2 counter register, you can use the expression

outportb (0x42, kd);

where kd is an int type variable that sets the frequency division of the system generator by the required frequency.

To set some bits of a word or byte without changing the rest of the bits, use the | operation with a mask containing 1 in the required bits and 0 in the rest; to reset - the & operation with a mask containing 0 in the required bits and 1 in the rest.

To perform a program delay, use the Delay () function described in the stdlib.h library, which is connected by the directive

#include <stdlib.h>

5.5. Individual tasks

Execute 5.3. Ringtone variants are shown below. After the note, its duration is indicated in seconds.

1. C (1), D (2), E (3).
2. C (3), A (2), G (1).
3. C sharp (5), G (1), D (3).
4. E (2), C (3), A sharp (1)
5. F (3), G-sharp (1), F (3).
6. F-sharp (2), D (4), B (2).
7. G (1), A-sharp (1), A (2).

8. A (3), C sharp (1), A sharp (3).
9. G-sharp (2), Bi (3), E (1).
10. A-sharp (1.5), F (1), C (2).
11. B (0.5), C sharp (3), A sharp (0.5).
12. C (2.5), F (1), G (2).
13. F (2), F-sharp (2), G (2).
14. D-sharp (2.5), D (2.5), C-sharp (2.5).
15. E (1), D (2), C (1).
16. D (1.5), A-sharp (2), D-sharp (1).
17. G (2), A (1), C sharp (0.5).
18. A (4), D-sharp (1), B (0.5).
19. D (3), E (0.5), F (0.9).
20. B (0.5), C (1.3), A (1.5).

5.6. Content of the report

5.6.1. The topic of the laboratory work.
5.6.2. Purpose of work.
5.6.3. Individual task.
5.6.4. Program text.
5.6.5. The results of the program.
5.6.6. Conclusions.

Laboratory work 6

Topic: Work of the PC video system in text mode

Purpose of the work: Studying the features of the functioning of a video system in text mode and the acquisition of practical skills in working with a video monitor in this mode.

6.1. Topics for preliminary work

Organization of video memory in text mode.
The structure of the video adapter.

6.2. Work description

All video systems use a buffer, which is a dual-port memory on the video controller (adapter) board, into which the microprocessor writes data for the image on the screen. The screen is periodically refreshed by scanning this data from the display side. The size and location of the buffer in the address space depends on the type of video controller and screen mode. When multiple screen images are stored in the buffer, each individual screen image is referred to as a display page.

6.2.1. Monochrome adapter.

Has 4K bytes of memory on the board, starting at address B0000h (i.e. B000: 0000h). This memory is sufficient to store one 80-character page of text.

6.2.2. Color Graphics Adapter (CGA).

The CGA has 16K bytes of on-board memory starting at B8000h memory address. This is enough to display one graphic screen, or to display four to eight screens of text, depending on the number of characters per line - 40 or 80.

6.2.3. Graphics adapters (EGA, VGA and SVGA).

EGA VGA and SVGA adapters work in a similar way in text mode. The adapter memory, in addition to being used as a video buffer, also stores character shape descriptions. The starting address of the video buffer is programmable, so the buffer with addresses A000h - BFFFh is used for enhanced graphics modes, and starting from addresses B000h and B800h - for modes compatible with monochrome and color text modes, respectively.

The number of pages available depends on both the screen mode and the

amount of available memory. Due to their complexity, EGA, VGA and SVGA have a 16K byte ROM that replaces and expands the BIOS procedures for operating the terminal. The beginning of the ROM area is the address C000: 0000h.

6.2.4. The content of the buffer in text mode.

In text modes, the following correspondence is established between the memory of the video controller and the image on the screen. At the beginning of the memory, data is written about the character located on the first line in the left corner, then data about the remaining characters of the first line, then data about the characters of the second line starting from the left, and so on.

During displaying text, different video systems work in the same way. 4000 bytes are allocated for the screen, so there are 2 bytes for each of the 2000 screen positions (25 lines × 80 characters). The first byte contains the ASCII code of the character. The display hardware converts the ASCII code number into its associated character and sends an image to the screen. The second byte (the attribute byte) contains information about how the given character should be output. For a monochrome display, it sets whether a given character is underlined, highlighted, or negative, or a combination of these attributes. In color systems, the attribute byte sets the foreground and background colors of the symbol, as well as the indication of its blinking.

Writing data directly to the monitor buffer significantly improves the display speed.

6.2.5. Video Character Attributes.

When the display is set to text mode in any of the video systems, two bytes of memory are allocated to each character position on the screen. The first byte contains the ASCII code of the character, and the second contains the video attribute of the character. The color adapter can display in color both the symbol and the entire area allotted to this symbol (background color).

The monochrome adapter is limited to black and white only, but it can generate underlined characters, which the color adapter cannot.

EGA can do everything that other systems do, and much more. In particular, on the improved display, it displays underlined color characters, since the 8-14 character display matrix provides such an opportunity.

In fig. 6.1 shows the format of the video attribute byte

7	6	5	4	3	2	1	0
M	R	G	B	I	R	G	B
*	Background color			Symbol color			

Figure 6.1 - Video Attribute Byte

R - red color entry: 0 = not included; 1 = included
G - green color entry: 0 = not included; 1 = included
B - blue color entry: 0 = not included; 1 = included
I - intensity: 0 = dim symbol; 1 = symbol is bright;
M - blinking: 0 = symbol does not blink; 1 = symbol is blinking.
* - a mode is possible when this bit will determine not the blinking of the symbol, but the intensity of the background color.

Both the symbol color and the background color are defined as the sum of the three primary colors - red, green, and blue. All possible colors are given in table 6.1.

Table 6.1 - Colors of symbols and background

	I R G B	Color	
Symbol color	0 0 0 0	Black	Background color
	0 0 0 1	Blue	
	0 0 1 0	Green	
	0 0 1 1	Cyan	
	0 1 0 0	Red	
	0 1 0 1	Magenta	
	0 1 1 0	Brown	
	0 1 1 1	Grey	
	1 0 0 0	Dark grey	
	1 0 0 1	Bright blue	
	1 0 1 0	Light green	
	1 0 1 1	Light blue	
	1 1 0 0	Light red	
	1 1 0 1	Light pink	
	1 1 1 0	Yellow	
	1 1 1 1	White	

Monochrome characters use attribute bytes in a slightly different way. As with color attributes, bits 0-2 set the color of the character, and bits 4-6 set the background. These colors can only be white and black.

Normal mode is white on black when bits 0-2 are set to 111 and bits 4-6 are set to 000. A negative image is created with inverse bit values. Characters are output

with increased brightness when bit 3 is set to 1. In all cases, setting bit 7 to 1 causes characters to flash.

6.2.5. Video controller registers.

In all video systems, the CRTC video terminal controller is based on the Motorola 6845 chip (EGA VGA and SVGA use custom chips based on the 6845).

Microcircuit 6845 has 25 control registers, numbered from 0 to 18h. The first 10 registers fix the horizontal and vertical parameters of the display. These registers are automatically set by the BIOS when the display mode is changed. We do not recommend experimenting with these registers, as you can spoil the terminal. Registers are 8 bits in size, but some are linked in pairs to store 16-bit values. The pairs of registers Ah – Bh and Eh – Fh set the shape and location of the cursor. The Ch – Dh pair controls the start address of the display.

Register 14h specifies which scan line in the character string is used for the underline.

All 25 registers are accessed through the same port, whose address for the monochrome adapter is 3B5h, for the color adapter and PCjr it is 3D5h. EGA VGA and SVGA use one of these two addresses depending on whether a color or monochrome monitor is attached to it. To write to the register, you must first send the number of the required register to the address register located in port 3B4h (3D4h for color). Then the next byte which sent to the port with the address 3B5h (3D5h) will be written to this register.

CGA and MDA have three more registers that are important to programmers. They have addresses 3B8h, 3B9h and 3BAh for monochrome and 3D8h, 3D9h and 3DAh - for color adapter. The first sets the screen mode, the second is mainly related to setting the screen colors, and the third provides useful information about the display status.

EGA VGA and SVGA distribute these functions between the attribute controller chip (port address 3C0h) and two graphics controller chips (port addresses 3CCh-3CFh). The attribute controller contains 16 EGA registers, numbered 00h to 0Fh. These registers can hold 6-bit color codes when an enhanced color display is connected, so any 16 colors from a set of 64 can be used.

6.2.6. Display modes.

Possible modes are shown in table 6.2.

Table 6.2 - Graphics adapter modes

Mode	Type	Format	Character size	Amount of colors	Video memory type
0	TXT	40x25	8x8*	16/8 (Shades)	4-layer
1	TXT	40x25	8x8*	16/8	4-layer
2	TXT	80x25	8x8*	16/8 (Shades)	4-layer
3	TXT	80x25	8x8*	16/8	4-layer
4	Gr	320x200	8x8	4	Linear
5	Gr	320x200	8x8	4 (Shades)	Linear
6	Gr	640x200	8x8	2	Linear
7	TXT	80x25	9x14*	3 (B/W/Bold)	Linear
08h-0Ch	reserve				
0Dh	Gr	320x200	8x8	16	4-layer
0Eh	Gr	640x200	8x8	16	4-layer
0Fh	Gr	640x350	8x14	3 (B/W/Bold)	Linear
10h	Gr	640x350	8x14	4 or 16	4-layer
llh	Gr	640x480	8x16	2	Linear
12h	Gr	640x480	8x16	16	4-layer
13h	Gr	320x200	8x8	256	Linear
14h-7Fh	BIOS custom extension modes for VGA and SVGA				

Function 0 of interrupt 10h sets the display mode. AL should contain the mode number according to table. 6.2.

6.3. Work order

Set the text (80×25) display mode.

In accordance with the individual task, display words with the required video attributes in a given place on the screen.

6.4. Programming features

6.4.1. In Turbo-Pascal language.

When using a software interrupt, you must:

• connect the Dos module, which describes the Intr procedure and the type of the Registers variable;

• declare a variable of this type, for example, reg: Registers;

• address the microprocessor registers as reg.al;

• call interrupt procedure 10h as follows:

Intr ($ 10, reg).

The video controller buffer in text mode can be represented as an array:

buff:array[0..3999] of byte absolute $B800:$0000.

Then, to display a character in the i-th row and in the j-th column, write the ASCII code of the character at the address buff [i * 160 + j * 2], and write the byte of the video attribute at the next address.

If words are specified as a string of characters *s* of type string, then the expression ord (s [k]) can be used to determine the ASCII code of the k-th letter of the string.

6.4.2. In the Turbo-C language.

When using a software interrupt, you must:
• connect the dos library, which describes the int86 procedure and the type of mixture REGS with the directive

#include <dos.h>

- declare the REGS as an operator

union REGS in, out

where:
in - is the name of the structure of the input registers
out – is the name of the structure of the output registers
8-bit registers are addressed as in.h.al or out.h.ah,
16-bit registers are addressed as in.x.ax or out.x.ax

• call interrupt procedure 10h as follows:

int86(0x10,&in,&out);

Accessing the video controller buffer in text mode is similar to accessing RAM and ROM cells using far pointers declared

char far * uk;

Then, to enter the starting address of the video buffer, you must write:

uk = (char far *) 0xB8000000;
and to display a character in the i-th line and in the j-th column, write

* (uk + i * 160 + j * 2) = kod;
* (uk + i * 160 + j * 2 + 1) = attr;

where *kod* and *attr* are char variables describing the ASCII code and character attribute, respectively.

6.5. Individual tasks

1. The first two words is red on a white background; the third word is high intensity blue; the fourth word is green flashing on red.
2. The first three words are blue on a red background; second word is high intensity.
3. The second word is magenta on a black background; the fifth word flashes; the seventh word is white of low intensity.
4. The first three words are blue on a white background; the fourth word is high intensity red; the sixth word is green flashing.
5. Seventh word is white on black background; the fifth word is high intensity red; the first three words are flash blue.
6. Two words are cyan color on a red background; the third word flashes; first word is high intensity brown.
7. The first two words are high intensity blue; the third word flashes; the fourth word is green on a red background.
8. First word is black on a white background; the fourth word blinks in red on a green background; the sixth word is brown high intensity.
9. The second word is pink of high intensity; the third word is cyan on a green background; the fifth word flashes.
10. The first two words are low intensity blue; the third word flashes; fourth word is red on a white background.
11. Even words are blue on pink; odd - red on black; the last fifth word flashes.
12. The first word is black on a white background; the fourth word is green on a red background; the rest are blue on a black background.

Display the specified words on the screen in a row and in a column, the numbers of which coincide with the student's number in the journal.

6.6. Content of the report

6.6.1. The topic of the laboratory work.

6.6.2. Purpose of work.
6.6.3. Individual task.
6.6.4. Program text.
6.6.5. The results of the program.
6.6.6. Conclusions.

Laboratory work 7

Topic: Cursor control, border color, palette registers, creation of special characters

Purpose of work: Obtaining practical skills in cursor control, border color and palette registers by programming adapter registers, creating special characters.

7.1. Topics for preliminary work

The purpose of the registers of the video adapter.
Description of symbols in text mode.

7.2. Work description

7.2.1. Cursor control.

The cursor serves two purposes. First, it serves as a pointer to where on the screen the program statements are sending their output. Second, it provides a visible point of reference on the screen for the user of the program. For the second use only, the cursor should be visible. When the cursor is invisible (off), it still points to the position of the screen. This is important because any screen output supported by the operating system starts at the current cursor position.

The cursor is generated by the 6845 display controller chip. This chip has registers that set the size and position of the cursor. The 6845 chip only provides a flickering cursor, although there are software ways to create a flicker-free cursor. The blink rate of the cursor cannot be changed. In graphics modes, the cursor is not displayed, although characters are positioned on the screen by the same cursor setting procedures as in text modes.

When the video system is operating in a mode that allows multiple display pages, each page has its own cursor, and when switching between pages, the cursor position is restored to the position it occupied when the last page was accessed. Some display modes allow up to 8 display pages and their corresponding cursor positions are stored in a set of eight 2-byte variables in the BIOS data area starting at 0040: 0050h. In each variable, the low byte contains the column number, counting from 0, and the high byte contains the row number, also counting from 0. When the number of pages is less than 8, variables located in lower memory addresses are used.

7.2.1.1. Cursor coordinates control (positioning).

The cursor can be set to absolute coordinates or coordinates relative to its

current position. Absolute coordinates can vary within 25 lines and 80 (sometimes 40) columns.

Registers 0Eh and 0Fh of the 6845 store the cursor position. The address register has a 3D4h port (data is entered into 3D5h port). The high byte is stored in the 0Eh register, the low byte is in the 0Fh register.

You can change their value and the cursor will move to the appropriate screen position, but DOS and BIOS screen interrupts will ignore your setting and return the cursor to its old position. This is because each time these interrupts are called, they restore the cursor registers using the 2-byte value stored in the BIOS data area. In this area, starting at 0040: 0050h, there can be up to eight such values, giving the current cursor position for each of the display pages. The first position corresponds to page 0, the second to page 1, and so on. The least significant byte of each variable contains the column number and the major row number. Columns and rows are both numbered from zero. The low-level procedure must modify these values as well to change the state of the cursor completely.

The cursor position is stored in registers 0Eh and 0Fh as a number from 0 to 1999, which corresponds to 2000 (25-80) screen positions. Do not confuse this numbering system with video buffer positions from 0 to 3999, where each character is followed by another attribute byte (to get an equivalent pointer to the cursor position, you need to shift the video buffer pointer 1 bit to the right). We also draw your attention to the fact that there is no need to swap the high and low bytes: in the register 0Eh - high, and 0Fh - low.

The cursor functions completely independently of the video memory. This means that when addressing directly to the display memory, the software must coordinate the movement of the cursor with the insertion of a new character into the buffer.

Programs sometimes read and save the current cursor position so that they can temporarily move the cursor to the command line and then return it to its original position. The procedure for reading the cursor position (from registers and the BIOS data area) is similar to the procedure for setting a cursor.

7.2.1.2. Cursor shape control.

The cursor can vary in thickness from a thin line to the maximum size allowed for a character. It is built from short horizontal line segments, the top of which is called the start line of the cursor, and the bottom is called the end line. For a monochrome display, there are 14 lines for each character, numbered from 0 to 13 starting from the top. Gaps between characters are provided by the top two lines and the bottom three lines. Most of the characters are on lines 2-10, although the tails of some characters reach lines 12 and 13, while the underscore is one twelfth line.

On a 200-line color display, only 8 lines are allocated for each character, and

the character is drawn on the top seven lines. These 8 lines are numbered 0 through 7 starting from the top, and the normal cursor is formed by one line 7. (Note that there is no underline on the color display, since using line 7 to do this would cause the characters to merge with those below them.) The high-resolution color display uses a 14-line monochrome display when it is in resolution mode and an 8-line display when it is in one of the color graphics modes.

The cursor can be formed by any combination of adjacent lines. For a monochrome display, it takes up all the space reserved for a character when the start line is 0 and the end line is 13 (for a graphic display, the end line value should be 7). If the values of the start and end lines are the same, then a single-line cursor appears. If the ending line number is less than the starting line number, the cursor becomes invisible.

The BIOS stores a 2-byte variable at 0040: 0060h that contains the current values of the start and end lines. The first byte contains the value of the ending string, and the second contains the starting one.

7.2.1.3. Creation of alternative cursor types.

All operating system interrupts associated with output to the screen use the cursor. You can change the shape of the cursor or make the cursor invisible by setting an address greater than 2000 or setting the starting line number greater than the ending line number. Alternative cursor types are possible when displaying to the screen using direct memory mapping methods. In this case, the "true" cursor is turned off, since it will not address characters to a specific position in the video buffer. Instead, a "fake" cursor is created using an attribute byte.

The most efficient method is to set the negative output attribute for the character pointed to by the cursor. For black and white screens, ASCII code 112 should be used as this attribute. Another way is to make the character pointed to by the cursor blink. In this case, simply add 128 to the current value of the attribute to make the character start blinking, and subtract 128 to stop blinking. The third way is to set the character to underscore mode (ASCII code 1). Finally, in programs that use the command line, consider using a special graphic character that follows the last character on the command line, such as the arrows displayed with ASCII codes 17 or 27.

Note that when a program receives input in multiple modes, you can help identify the current mode by using a special type of cursor.

7.2.2. Color control of the border of the screen (border).

The border of the character screen (outside the address area) may have a different color from the background color of the center of the screen. Any of 16

colors for CGA or any of 64 colors for EGA can be used. For CGA color graphics, bits 0-3 of port 3D9h (color select register) set the border color when the screen is in text mode.

For the EGA adapter, the border color is specified by register 11h of the adapter attribute block. Any of 64 colors can be set when an enhanced graphic display is connected. To set the border color in EGA, follow these steps:

• read port 3DAh to initialize the address register;

• set the register number 11h by writing it to the 3C0h port;

• set the color by sending a data byte to port 3C0h (6 bits are analyzed);

• enable displaying on the screen by setting bit 5 of the block address register to 1 (sending code 20h to port 3C0h). Border color change is used in keyboard drivers to indicate switching between Russian and Latin alphabets.

7.2.3. Creation of special characters.

Only a monochrome adapter cannot display characters of the form specified by the programmer himself. The color adapter supports 128 user-defined characters, PCjr 256, and EGA support 1024, of which 512 are simultaneously available. For a color adapter, the ROM-BIOS contains data to paint only the first 128 characters of the ASCII set (numbered 0 through 127). The next 128 characters are not available to you until you create them using the technique described here.

7.2.3.1. CGA adapter.

The symbols for the graphics adapter and PCjr are described using an 8 * 8 dot matrix. The data for each character is contained in eight bytes. Each byte contains a setting for points in one row, starting from the top, with the most significant bit (number 7) corresponding to the leftmost point in the row. When the corresponding bit is 1, the dot is highlighted. To describe a character, you must determine the correct bit sequences for the eight bytes and place them in sequential memory cells.

All 128 characters together require 1024 bytes, although it is not at all necessary that all characters be described. A special interrupt vector (constant pointer in low memory addresses) indicates the address of the first byte of the first extended character, i.e. character number 128. When code 128 is sent to the character position in the video buffer, the first eight bytes are scanned and output. If the character number is 129, then bytes 9 through 16 are output, and so on. The number of this interrupt vector is 1Fh and it is located at address 0000: 007Ch. Put the offset value in the low word (low byte first) and the segment address in the high word (low byte first) and change the 8 bytes of any character description.

Note that it is possible to define symbols with larger code numbers without having to allocate memory for symbols with lower numbers; you just need the

vector to point to some address that is less than the address of the beginning of the block containing data for describing characters.

7.2.3.2. EGA, VGA, SVGA adapters.

For EGA, the picture is a little more complicated, albeit more flexible. When the text mode is initialized, one of the two character sets (8×8 or 8×14), depending on the display resolution, is copied from the EGA ROM to bit plane 2 of the video buffer. This part of the buffer is divided into blocks, and the standard set of characters is placed in block 0. EGA defines four blocks for describing characters, in VGA, SVGA - 8 blocks. Each block contains a description of 256 characters (ASCII codes 0-255). Regardless of the size of the character description matrix, 32 bytes are allocated for each character, therefore each block of characters is 8 KB (32 * 256). When there is more than one block of characters, bit 3 of the attribute byte determines from which block the data will be taken to describe the character.

Which of the blocks will be needed depends on the setting of the bits of the character set selection register, the port address of which is 3C5h.

First, you need to write 3 to port 3C4h to indicate the required register, and then write a data byte to 3C5h. For EGA, bits 1.0 give the character block number, which is taken when bit 3 is 0, and bits 3.2 give the character block number when bit 3 of the attribute byte is 1; for VGA, SVGA, respectively, bits 1,0,4 determine the block number when bit 3 is 0, and bits 3,2,5 - when bit 3 is 1. In the case of equal block numbers, there is no possibility of using two character sets, but you can set any set. In the standard BIOS settings, these bits are all 0, so only block 0 is used. However, nothing can stop you from placing your characters in any position you want in any block.

To do this, follow these steps:

• write 6 to the 3CEh port (the address of the multipurpose register of the graphics controller);

• write 1 to the 3CFh port (Transfer of the video system to the graphic mode);

• write 2 to port 3C4h (address of the bit plane mask register);

• write 4 (0100b) to port 3C5h (allow access to the 2nd bit plane);

• write 4 to port 3C4h (address of the video memory usage mode register);

• write 6 to port 3C5h (canceling the odd / even mode, which concatenated bit planes 0-1 and 2-3 in text mode);

After allowing writing, taking into account the block number (Blok), ASCII character code (Kod) and the number of bytes in the character description, *N* bytes are changed, starting from the address

Adr = A000h:Blok*2000h+kod*32.

After that, it is necessary to restore the registers and transfer the video system to text mode:

• write 4 to port 3C4h (mode register address);
• write 2 to port 3C5h (even / odd mode setting);
• write 2 to port 3C4h (address of the bit plane mask register);
• write 3 (0011b) to port 3C5h (allow access to 0 and 1-st bit planes);
• write 6 to port 3CEh (multipurpose register address);
• write Eh to port 3CFh (transfer to text mode).

The content of *N* bytes is determined by the required character image.

Fig. 7.1 shows a handwritten image of the letter "u" for the matrix (8×14).

Please note that blank lines must be left above, below and to the right (or left) of the character image to separate characters in a line and between lines. This letter is described by the following 14 bytes: 00, 00, 88, 88, 88, 88, 88, 88, 98, AA, 44, 00, 00, 00.

If you changed the standard character set (Blok = 0), then you can restore it from ROM at any time.

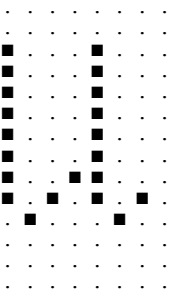

Figure 7.1 - Image of the letter “u”

For color adapter and PCjr, use interrupt 21h function 25h to change the 1 Fh interrupt vector. When entering DS: DX must point to the first byte of the data block.

The EGA has an 11h interrupt 10h function that manipulates the character set. This feature can be very complex when used to create special screen modes, but its main purpose is quite simple. There are four sub-functions.

When AL is 0, user-defined data is transferred from memory to a character description block of the 2nd bit plane.

When AL is 1 or 2, the data sets for characters 8×14 and 8×8, respectively, are copied from ROM to the character description block.

For VGA, when AL is 4, the 8×16 character set is copied from ROM to the character block. When AL is 3, the function sets the block assignment in the character set select register, as described above. In the latter case, you just need to put the corresponding data in BL and call the function.

To load data from ROM, put the block number in BL and execute the function. To load your data, you need ES: BP to point to it, the number of transmitted characters must be in CX, the offset (character number) in the block must be in DX, the number of bytes per character in BH, and the block number in BL. Then call interrupt 10h.

You can get information about the status of an existing character set using subfunction 30h (AL register) of 11h (AH register) function of interrupt 10h. This subfunction has no input registers, and when returned, CX contains the number of bytes allocated for each character, and DX contains how many rows of dots the character occupies on the screen (both of these parameters depend on both the character size and the vertical screen resolution).

7.3. The order of the work.

7.3.1. Determine the ASCII codes of the characters that make up the last name and first name of the student.

7.3.2. Set the byte descriptions of the specified characters in handwriting and the description of the student's personal hieroglyph.

7.3.3. Create a program that performs the following actions.

7.3.3.1. Sets the screen border color according to the individual task.

7.3.3.2. Moves the cursor to the position where the line number corresponds to the student's month of birth and the column number corresponds to the student's journal number.

7.3.3.3. Reshapes the cursor by changing the starting and ending line numbers.

7.3.3.4. Determines the number of bytes per character for describing a character.

7.3.3.5. Modifies the byte descriptions of the student's first and last name characters.

7.3.3.6. Displays the last name and first name of the student in handwritten letters at the specified cursor position, ending the line with personal hieroglyphs.

7.3.3.7. Changes the background color by programming the 0 register of the palette.

7.4. Features of programming.

7.4.1. In Turbo-Pascal language.

To access the PC ports, the predefined arrays *Port* and *PortW* are used. For example, to write the address of the bit plane register to port 3C4h, use the expression Port [$ 3C4]: = 2.

When using a software interrupt, you must:

• connect the Dos module, which describes the *Intr* procedure and the type of

the *Registers* variable;

• declare a variable of this type, for example, reg: Registers;

• address the microprocessor registers as reg.al;

• call interrupt procedure 10h as follows:

```
Intr($10,reg).
```

It is convenient to represent the description of the symbol image using a typed constant. For example, the description of the symbol shown in Fig. 7.1 is as follows:

```
const
 ms:array[1..14] of byte=($00,$00,$88,$88,$88,$88,$88,
 $88,$98,$AA,$44,$00,$00,$00);
```

If words are specified as a string of characters s of type *string*, then the expression ord (s [k]) can be used to determine the ASCII code of the *k*-th letter of the string.

The video controller buffer block can be represented as an array:

```
block:array[0..32*256] of byte absolute $A000:$0000.
```

7.4.2. In the Turbo-C language.

To access the PC ports, the port read and write functions are used, which are stored in the <dos.h> library. The library is connected by the directive

```
#include < dos.h>
```

after that, for example, to write the value 0Eh to port 3D5h, the following expression is used:

```
outportb(0x3D5,0x0E);
```

During using a software interrupt, you must:

• connect the Dos library, in which the int86x procedure and the REGS mixture type are written by the directive:

```
#include <dos.h>
```

• declare variables:

```
union REGS in,out,sr;
```

address the microprocessor registers as in.h.ah, in.x.ax;
address segment registers as sr.es;
to access the BP register, you need to declare the structure

```
struct REGPACK inb;
```

then access the register as inb.r_bp;
call interrupt routine 10h as follows:

```
int86x(0x10,&in,&out,&sr).
```

It is convenient to present the description of the symbol image during initializing the array. For example, the description of the symbol shown in Fig. 7.1 is as follows:

```
char ms[14]={0x00,0x00,0x88,0x88,0x88,0x88,0x88,
 0x88,0x98,0xAA,0x44,0x00,0x00,0x00};
```

Accessing the video controller buffer is similar to accessing RAM cells using far pointers declared

```
char far * uk;
```

Then, to enter the starting address of the video buffer, you must write:

```
uk=( char far *)0xA0000000;
```

7.5. Individual tasks

Select the parameters of the generated image according to the second digit of the student's journal number.

Table 7.1 - Variants for individual tasks

Parameters	Student's journal number									
	0	1	2	3	4	5	6	7	8	9
w_str	0	1	2	3	4	5	6	7	8	9
k_str	13	12	11	10	8	9	10	11	12	13
col_fon	30	31	32	33	34	35	36	37	38	39
col_for	10	9	8	7	6	5	4	3	2	1

where: *n_str* and *k_str* are the starting and ending lines of the cursor position in

familiarity.
col_fon - background color (palette register 0)
col_for - screen border color.

7.6. Content of the report

7.6.1. The topic of the laboratory work.
7.6.2. Purpose of work.
7.6.3. Individual task.
7.6.4. Program text.
7.6.5. The results of the program.
7.6.6. Conclusions.

Laboratory work 8

Topic: Work in graphic mode. Screen control, color. Point Image

Purpose of work: Acquisition of practical skills in working with a video monitor in graphic mode.

8.1. Topics for preliminary work
Purpose and modes of operation of the adapter in the graphics mode.
Organization of the video buffer in graphic mode.

8.2. Work description

The color graphics adapter has three graphics modes, PCjr has six and EGA has seven, VGA, SVGA are many undocumented modes. Memory requirements vary significantly for the available modes, depending on the screen resolution and the number of colors. In its enhanced graphics modes, EGA uses display memory in a very different way than other video systems, but it faithfully emulates their memory methods in three common modes.

8.2.1. CGA color graphics adapter and PCjr system.

Let's look at the color adapter and PCjr system first. Two colors (black and white) only require one bit of memory for each point on the screen. Four colors take 2 bits, and 16 colors take 4 (8-color modes are not used because the three bits required to represent them cannot be conveniently accommodated in byte bits). There are 200 vertical points for all modes. Low resolution (used only on PCjr) outputs 160 horizontal dots, medium resolution is twice that (320 pixels), and high resolution is twice that (640 pixels).

In two- and four-color modes, PCjr has the ability to select any of the 16 available colors. The color adapter is more limited. In two-color mode, it is always limited to white and black, and in four-color mode, only the background color can be selected from 16 colors, while the base color only needs to be taken from two predefined palettes. Palette 0 contains brown, green, and red, and palette 1 contains cyan, magenta, and white.

In contrast to text data in modes 4-6 and 8-Ah, graphic data is divided into parts on the video page. In most modes, data is split into two parts, with the first half of the buffer containing data for the even lines of the screen, and the second for the odd ones (lines are numbered from the top of the screen down).

8.2.2. Graphic adapters EGA, VGA, SVGA.

The EGA adapter provides the ability to work in various video modes in conjunction with color or monochrome monitors with digital inputs. In addition, the adapter provides the ability to work with a light pen. The adapter can operate in several graphics modes (4 bit planes are used) and has the ability to load fonts into the video memory in alphanumeric modes.

VGA, SVGA adapters have analog outputs (the signal level of each analog output R, G, B sets the intensity of the corresponding color component), their architecture is a development of the EGA architecture, to which a digital-to-analog converter unit is added, therefore VGA, SVGA emulate all EGA modes, and also have their own graphics modes of high resolution and quality of color reproduction, for the implementation of which requires significant amounts of video memory. The block diagram of the video adapters is shown in Figure 8.1.

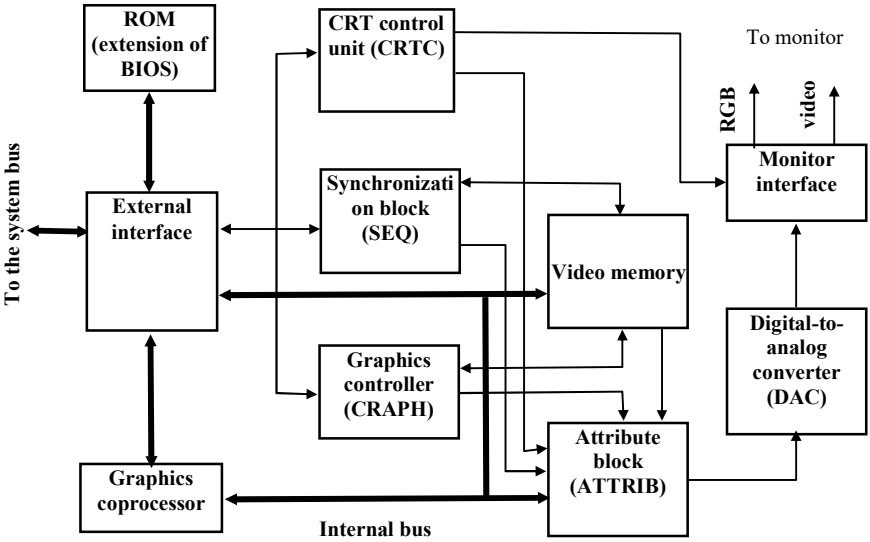

Figure 8.1 - Video adapter architecture

8.2.2.1. Main components.

The *CRT Controller* controls the horizontal and vertical synchronization signals, the start address of the output in the video buffer, the position and shape of

the cursor, and more.

The *synchronization block* (Sequencer) generates clock signals and signals to synchronize access to video memory. This device provides the ability to access the video memory from the processor at specially allocated times in the interval between the time intervals required to access the video memory during the regeneration of the image on the display screen. The same block contains registers for writing data to the bit planes.

The *Graphics Controller* directs data from memory to the attribute controller and to the processor.

In graphics modes, data from video memory is sent to the attribute controller chip sequentially. In text modes, data is sent in parallel, bypassing the graphics controller. To quickly change the image on the display screen, the hardware provides the ability to write 32 bits of data in one memory cycle (8 bits for each plane), and additional logic allows the processor to write data to the display memory without adhering to byte boundaries.

The *Attribute controller* performs the first stage of color indexing in text and graphics modes, i.e. expands the 4-bit color code obtained from the video memory (color index) to a 6-bit code, which is written in one of the 16 registers of the palette. The flicker and underline control actions are performed with the same microcircuit. The controller receives data from the video buffer and converts it into control signals supplied to the monitor input.

A digital-to-analog converter (DAC) appeared in the VGA adapter, performs the second stage of color indexing in text and graphics modes, i.e. extends the 8-bit color code obtained from the attribute controller or directly from the video memory (color index) to an 18-bit code (6 bits for each color component - R, G, B), which is written in one of the 256 PEL registers. Color control signals are input to the monitor in analog form along the R, G, and B lines.

Video buffer (display buffer - also called video memory or adapter memory). The video buffer is a dual-port memory, therefore it is available both from the video adapter (reading information and displaying it to the monitor) and from the processor side in write and read modes.

In the address space of the processor, 2 memory segments are allocated for the video buffer - A0000h and B0000h. Since modern graphics modes of high resolution and quality of color rendition require significantly large amounts of video memory, this video memory is mapped to the area of higher addresses. The video buffer addressing from the processor side depends on the video modes: in monochrome modes - it starts with the segment address B0000h; in color text messages - from the address B8000h; in color graphics - from the address A0000h.

The basic input / output system of the video adapter (BIOSV) is located in the memory of a special ROM installed on the adapter board. BIOSV integrates with the system base I / O system. This is where the fonts used to generate characters and the

video adapter control programs are located. The starting address of the ROM is C0000h.

8.2.2.2. Working in graphics mode 16/64 (10h)

For the 10H graphics mode, the memory is organized in the form of 4-bit planes, each of which is organized in the same way as for the high-resolution black-and-white mode: when a data byte is sent to a specific video buffer address, each bit corresponds to a point on the screen, and they describe a horizontal the line segment and bit 7 corresponds to the leftmost point. Four such bit planes are written, corresponding to the same addresses, in the video buffer. This assigns 4 bits to each pixel, which allows 16 colors to be described.

EGA can use the first stage of color indexing using palette registers. In this case, 64 colors become available, the encoding for which is rgbRGB. Components r, g, b - weak intensities 1/3 of max, RGB - normal intensities 2/3 of max. Various combinations create 64 shades. As always, 111111b is white and 000000b is black.

VGA, SVGA can use the second stage of color indexing using PEL registers, which allows you to specify 2^{18} shades.

8.2.2.3. Purpose of registers.

Synchronization block registers

The purpose of the registers of the synchronization block is presented in table. 8.1.

Address register - a register pointing to one of the registers of the synchronization block located at address 3C4h. This register is loaded with the binary number of the synchronization block register, which will be written to. The register numbers placed in the address register are presented in the "Index" field of the table. 8.1, after which, at address 3C5h, information is written / read into the selected register.

Table 8.1 - Registers of the synchronization block

№	Name	Port	Index	Mode
1	Address	3C4h	-	W
2	Reset	3C5h	00h	W
3	Clocking mode	3C5h	01h	W
4	Map mask	3C5h	02h	W
5	Character map select	3C5h	03h	W
6	Memory mode	3C5h	04h	W

The registers of the CRT control unit are presented in table. 8.2.

Table 8.2 - Purpose of the registers of the CRT control unit

№	Name	Port	Index	Mode EGA
1	Address	3?4h	-	W
2	Horizontal total	3?5h	00h	W
3	Horizontal display enable end	3?5h	01h	W
4	Start horizontal blank	3?5h	02h	W
5	End horizontal blank	3?5h	03h	W
6	Start horizontal retrace	3?5h	04h	W
7	End horizontal retrace	3?5h	05h	W
8	Vertical total	3?5h	06h	W
9	Overflow	3?5h	07h	W
10	Preset row scan	3?5h	08h	W
11	Max scan line	3?5h	09h	W
12	Cursor start	3?5h	0Ah	W
13	Cursor end	3?5h	0Bh	W
14	Start address high	3?5h	0Ch	R/W
15	Start address low	3?5h	0Dh	R/W
16	Cursor location high	3?5h	0Eh	R/W
17	Cursor location low	3?5h	0Fh	R/W
18	Vertical retrace start	3?5h	10h	W
19	Light pen high	3?5h	10h	R
20	Vertical retrace end	3?5h	11h	W
21	Light pen low	3?5h	11h	R
22	Vertical display end	3?5h	12h	W
23	Offset	3?5h	13h	W
24	Underline location	3?5h	14h	W
25	Start vertical blank	3?5h	15h	W
26	End vertical blank	3?5h	16h	W
27	Mode control	3?5h	17h	W
28	Line compare	3?5h	18h	W
? = B in monochrome mode and D in color mode				

Experiments with the registers of the CRT control unit, which form the image of the frame, can lead to damage to the video terminal.

The registers of the graphics controller are presented in table. 8.3.

Table 8.3 - Purpose of the registers of the graphics controller

N	Name	Port	Index
1	Graphics 1 Position	3CC	-
2	Graphics 2 Position	3CA	-
3	Graphics 1 & 2 Address	3CE	-
4	Set/Reset	3CF	00
5	Enable Set/Reset	3CF	01
6	Color Compare	3CF	02
7	Data rotate	3CF	03
8	Read Map Select	3CF	04
9	Mode	3CF	05
10	Miscellaneous	3CF	06
11	Color Don't Care	3CF	07
12	Bit Mask	3CF	08

The attribute controller registers are shown in Table 8.4.

Table 8.4 - Purpose of the attribute controller registers

№	Name	Port	Index
1	Address Register	3C0h	
2	Palette Registers	3C0h	00 - 0Fh
3	(Mode Control Register	3C0h	10h
4	Overscan Color Register	3C0h	11h
5	Color Plane Enable Register	3C0h	12h
6	Horisontal Pel Panning Register	3C0h	13h
7	Color selection register (VGA only)	3C0h	14h

8.2.2.4. Drawing points.

Considering the organization of graphic information in the video buffer, the output of one point implies changing individual bits of memory. The two, four, and sixteen color modes require one, two, and four bits to be changed to set the characteristics of a single point, respectively. These operations require huge amounts of CPU time, so graphics software is usually very slow. Careful thought often allows all the bits of a byte to be set at once, rather than referring to the same byte 4 or 8 times.

The Ch function of interrupt 10h sets the point. DX contains a row and CX contains a column. They are counted from 0. The color code is placed in AL. Note that the contents of AX will be destroyed when the interrupt is executed.

While the palette color is placed in the low-order bits of AL, the high-order bit

also matters. If it is equal to 1, then an exclusive OR operation is performed on the color with the current color. Recall that the exclusive OR operation sets a bit only if only one of the two compared bits is set. If both compared bits are 1 or both are 0, then the result is 0.

At a low level, we have the ability to directly access the video buffer (memory mapping). First, you must calculate the offset of point a - inside the buffer and b - inside the byte containing the bits associated with this point. After that, bit operations will ensure the appropriate setting.

In Dh, Eh and 10h modes, the memory is divided into 4 bit planes. Each plane is organized in the same way as for the high definition black and white mode of the color adapter, where a data byte is sent to a specific video buffer address, each bit corresponds to a horizontal line segment and bit 7 corresponds to the leftmost point. Four such bit planes are output, referring to the same addresses in the video buffer. This leads to the fact that each point is described by four bits (giving 16 colors), with each bit in a separate byte of a separate bit plane.

But how can you write 4 different bytes of data located at the same address? The answer to this question is that you are not sending four bytes in sequence to this address. Instead, one of three write modes allows all 4 bytes to be changed based on one byte of data received from the processor. The effect of the data sent by the processor depends on the setting of several registers, including two mask registers, which determine which bits and in which bit planes the bits are changed.

To understand these registers, we must first understand the four latch register. They contain data for the four bit planes at the position that was last accessed. (Note that the term "bit plane" is used both for the entire video buffer area and for the 1-byte buffer temporarily stored in the latch register). When the processor sends data to a specific address, this data can change or completely change the data of the latch register, and subsequently, it is the data from the latch register that is written to the video buffer. How the processor data affects the latch register depends on the write mode as well as the setting of some other registers. When reading an address from the video buffer, the latch registers are filled with four bytes from four bit planes at the given address. The latch registers are easy to manipulate, performing various logical operations with their contents, which allows you to arrange various graphic tricks.

The bit mask register (see graphics controller block register 08h) and the mask register (see synchronization block register 02h) of the card act on the latch registers, protecting certain bits or bit planes from being changed by data from the processor. Setting a bit in the bit mask register to 0 masks this bit in all four bit planes, making the corresponding point unavailable for change. However, since the hardware operates in byte terms, the actually "unchangeable" bits are overwritten in four bit planes. The data for these masked bits is stored in the latch registers, so the program must make sure that the current contents of the latch registers are at the

correct address. Therefore, before writing to the video buffer, you must first read from it. Bits 0-3 of the map mask register correspond to bit planes 0-3. The upper 4 bits of the register are not used. When bits 0-3 are 0, the corresponding bit planes do not change during write operations. The data sent to the video buffer can be modified and logically processed along with the data stored in the latch registers using the data rotation register (register 03h of the graphics controller block).

Bits 0-2 of the rotation register determine the number of bits by which the processor data is cyclically shifted to the left when writing data to the video buffer. To write data without shifting, this bit must be set to 0. When writing, set the logical function of the processor data with the data of the latch registers.

The meaning of the bits:

00 - data does not change

01 - logical "AND"

10 - logical "OR"

11 - exclusive "OR"

The shift operation is performed before the logical operation.

This property is used differently in different recording modes.

Three write modes and two read modes are set by the mode setting register (see graphics controller block register 05h). The write mode is set in bits 0 and 1, as a number from 0 to 2. Bit 2 must be equal to 0, as well as bits 4-7.

Bit 3 sets one of two read modes from the video buffer.

This bit can be 0 or 1. BIOS EGA sets write mode to 0, read mode to 0.

8.2.2.4.1. Write mode 0.

In the simplest case, write mode 0 copies processor data into each of the four bit planes. Suppose, for example, 11111111b is sent to a specific video buffer address and all bits and all bit planes are enabled (ie, nothing is masked by the mask registers described above). Then every bit in all four planes will be set to 1, so the bitstring for each of the corresponding points will be 1111b. This means that eight dots will be output in color 15, which initially corresponds to white, although the palette registers allow for any of the valid colors in fact.

Now consider the same situation when sending the value 00001000b. The bitstring for point 3 will be 1111b, and for the rest 0000b, which is black (initially). Therefore, in this case, only point 3 will appear on the screen, the remaining 7 points will be turned off.

If you send the palette code of the desired color to the map mask register, the register will mask certain bit planes so that the desired color is reproduced. The above discussion dealt with the simultaneous output of eight points.

Well, how to display fewer points? In this case, of course, it is necessary to save the existing data for some points, and in order for this to be possible, the

current content of this address is stored in the latch registers. The bit mask register is then used to mask those points that should not be changed. If a bit in this register is set to 0, then the data received from the processor for that bit is ignored and the data stored in the latch registers is used instead. The filling of the latch registers changes when the color resolution register is non-zero (register 01h of the graphics adapter block), bits 0-3 of which allow filling the byte of the selected bit planes with values from the corresponding bits of the color register (register 00h of the graphics adapter block). In this case, the processor data values (card masks are ignored).

8.2.2.4.2. Write Mode 1.

Write mode 1 is for special applications. In this mode, the current contents of the latch register are written to the specified address. As a reminder, the latch registers are filled with a read operation. This mode is very useful for fast data transfer during screen shift operations. The bit mask register and the map mask register do not affect this operation. It also doesn't matter what data the processor sends - the contents of the latch registers are written to memory unchanged.

8.2.2.4.3. Write mode 2.

Write Mode 2 provides an alternative way to set individual points. The processor sends data that has only the least significant 4 bits, considered as a color (palette register index). We can say that this bitstring is inserted across the planes. The string is duplicated for all eight points associated with a given address, until the bit mask register prevents certain points from being changed. The card mask register is active, as in write mode 0. Of course, the processor must send a full byte, but only the least significant 4 bits.

8.2.2.4.4. Read mode 0.

The processor reads data (8 points) from the gate register of the plane that is allowed by the plane selection register for reading (bits 0-2 of the 04h register of the graphics controller block). To read colors (palette register codes) of all 4 bit planes, it is necessary to sequentially allow reading and reading all planes at one address.

8.2.2.4.5. Read mode 1.

As a result of the read operation, only those bits in the byte for which the color matches the color specified in bits 0-3 of the color comparison register (register 02h of the graphics adapter block) will be set to 1.

8.2.2.5. Register programming.

Each of the adapter chips has an address register and several data registers. The address register is used as a pointer to a particular data register. The address register is a write-only register into which the index of the selected data register can be placed by the processor using the OUT instruction.

The data registers of each adapter chip are accessible through the corresponding I / O port. Access to various data registers is carried out by first entering the required data register into the address register of the index, followed by issuing an OUT command from the processor side to enter the required value into it. The registers of the attribute controller differ, where the address register and the data register have one port with the address 3C0h. Inside the controller, the addressing is organized in such a way that every time after writing to the 3C0h port, a switch is made: the address register - the data register corresponding to the value of the address register and vice versa. To initiate the switching process, the address register - the data register (first access to the address register, and then to the data register, etc.), it is necessary to perform a read operation from the port with the address 3BAh or 3DAh. After performing a read operation, the first access to port 3C0h will be access to the address register of the attribute controller. After loading the address register, the next output command to port 3C0h will write the required value to the corresponding data register of the attribute controller. Execution of this command again makes the address register available for writing and the process can be continued.

After writing data to the registers of the attribute controller, it is necessary to set 5 bits of the address register of the controller to 1 (output resolution), i.e. send 20h to port 3C0h.

The video adapter supports standard BIOS graphics functions. It is possible to output a point using the Ch function of interrupt 10h. On input, DX must contain the line number and CX the column number, both are counted from 0. The color code is placed in AL. The content of AX changes when the interrupt is executed.

8.3. Work order

8.3.1. Write a graphics display program that does the following:

8.3.1.1. Puts the video system into graphics mode 10h using interrupt 10.

8.3.1.2. Sets the specified background color by programming register 00h of the palette.

8.3.1.3. Outputs 16 lines of points at different values of the adapter registers. The starting address of the i-th line of the video buffer is calculated in the 10h mode by the expression

adp=A000:0000+80*i

8.3.1.4. Repeats the output of the next 16 lines with the same parameters as the previous ones, but taking into account the data shift in the rotation register.

8.3.1.5. Repeats the output of the next 16 lines with the same parameters, but taking into account the logical operation.

8.3.1.6. The rest of the screen after the acquired image is filled with a byte, the row and column numbers of which correspond to the student number in the journal. Use the recording mode 1.

8.3.1.7. In the resulting image, "extinguish" the color of the first line by entering the background code into the corresponding register of the palette.

8.3.1.8. Perform hardware horizontal screen shift by changing the start address.

8.3.1.9. Return to text mode.

8.3.2. Write, debug and save to perform work #9 the *put_pix* procedure for reading a byte and output in one memory access up to 8 points with the specified parameters

put_pix (x, y, color, bit_m, log_op)

where: x, y - coordinates of the video memory byte

color - color code

bit_m - bitmask value

log_op - the code of the logical operation with the read value, which is set in the rotation register.

8.4. Features of programming.

8.4.1. In Turbo-Pascal language.

To access video memory, the predefined array *Mem* is used. For example, to read a byte at address A000: 0000h, use the expression b: = Mem [$ A000: $ 0000] (b is a byte variable).

The predefined *Port* array is used to refer to the PC ports. For example, to write the 3Dh value to port 3C0h, use the expression

Port [$ 3C0]: = $ 3D;

to read from port 3DAh, use the expression

data: = Port [$ 3DA], where data is a byte variable.

8.4.2. In the Turbo-C language.

To access video memory, far pointers are used, which are declared in the program as follows:

char far * uk; - to work with bytes.

To read the byte at address A000: FFF5h, use the expression

uk = (char far *) 0xA0000000;

b = * uk; where b is a variable of type char;

To access the PC ports, the port read and write functions are used, which are stored in the <dos.h> library. The library is connected with the command

#include <dos.h>

after which, for example, to write the 3Dh value to port 3C0h, the following expression is used:
outportb (0x3C0,0x3d);

To read from port 3DAh, the following expression is used:
data = intportb (0x3DA), where data is a variable of the char type;

8.5. Individual task.
Select the parameters from Table 8.5 by 2 digit of the journal number.

Table 8.5 - Variants for individual tasks

Parameters	Journal number									
	0	1	2	3	4	5	6	7	8	9
Starting line number	100	140	200	80	120	150	60	180	210	40
Background color	1A	23	31	18	2B	35	13	2F	39	1D
1st line color	A	1	B	3	C	5	6	D	9	8
Color increment of the i-th row	1	2	4	5	8	7	9	6	3	A
Visible points mask	61	63	67	82	86	8E	72	76	7E	E2
Data shift	1	2	3	4	5	6	7	1	2	3
Logical operation	AND	OR	XOR	AND	OR	XOR	AND	OR	XOR	AND

8.6. Content of the report

8.6.1. The topic of the laboratory work.
8.6.2. Purpose of work.
8.6.3. Individual task.
8.6.4. Program text.
8.6.5. The results of the program.
8.6.6. Conclusions.

Laboratory work 9

Topic: Line Drawing, Shading

Purpose of work: Acquisition of practical skills in creating drawings.

9.1. Topics for preliminary work
Point display in graphic mode.

9.2. Work description.

Line drawing functions are basic graphics programs and are used to display lines in a given color by specifying their start and end coordinates. While drawing vertical horizontal lines is straightforward, the more difficult task is creating functions that draw lines along the diagonals. For example, what points make up the line drawn from 0,0 to 80,120?

One approach to developing line drawing functions uses the relationship between X and Y offsets. To show this approach in action, draw a line between 0.0 and 5.10. The X offset is 5 and the Y offset is 10. The ratio is 1/2. It will be used to determine the coefficient of dependence by which the X and Y coordinates should change when drawing lines. In this case, this means that the x-coordinate is incremented by half the amount of change in the y-coordinate. The beginner programmer often uses this method when developing line drawing functions. While the approach is mathematically correct and easy to understand, it requires floating point math to work properly and to avoid serious rounding errors. This means a sharp decrease in performance, if the math coprocessor is not included in the system. For this reason, this method is rarely used.

The most common method for drawing lines involves using the Bresenheim algorithm. Although it is also based on the relationship between X and Y distances, it does not require division or floating point calculations. Instead, the relationship between the X and Y coordinate values is represented indirectly through a series of additions and subtractions. The main idea of the Bresenheim algorithm is to record the average error values between the ideal position of each point and the position on the display screen where it is actually displayed.

The error between the ideal and the actual position of the point arises due to the limited capabilities of the technical means. In fact, there are no displays with infinitely high resolution, and therefore the actual position of each point on the line requires the best approximation. In each iteration of the line drawing cycle, two variables X_err and Y_err are called, which increase depending on the change in the values of the X and Y coordinates, respectively.

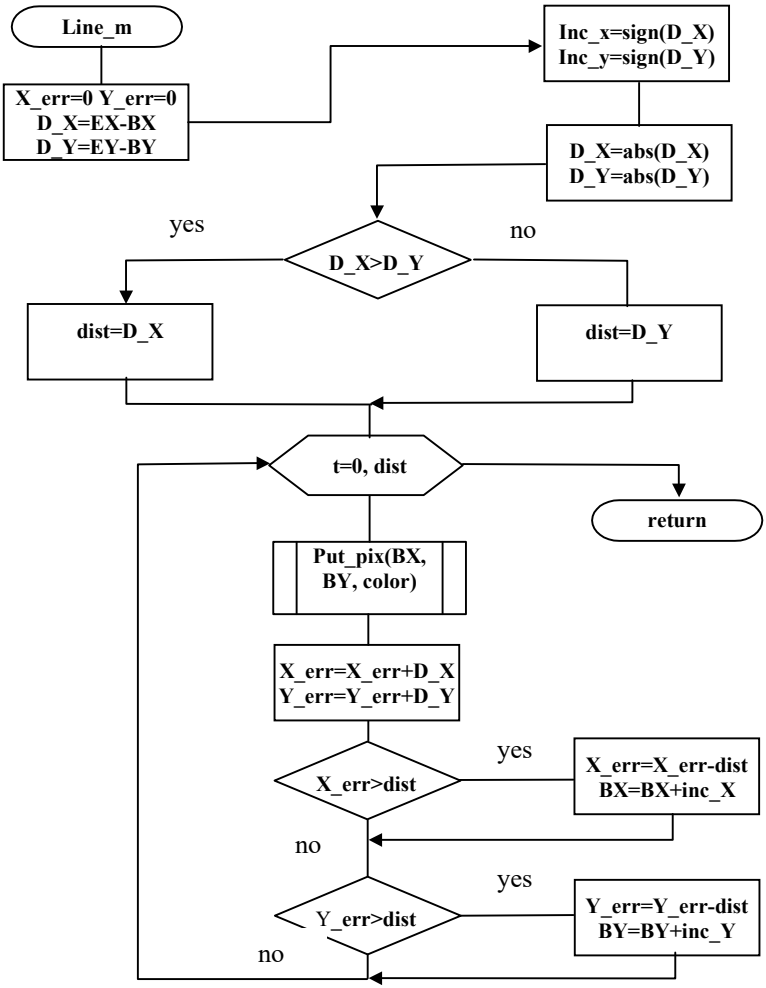

Figure 9.1 - Algorithm for drawing lines (Bresenheim)

When the error reaches a certain value, it is reset to its original position and the corresponding coordinate counter is incremented. This process continues until the line is completely drawn. In this algorithm, the following designations are accepted: BX, BY - coordinates of the starting point; EX, EY - coordinates of the end point; D_X, D_Y - maximum distance at the corresponding coordinate; c - color of the

output point; inc_X, inc_Y - direction of movement along the corresponding coordinate.

As you can see from the algorithm, when moving along the larger coordinate (dist), the increment along this coordinate and the image of the point is always performed, while along the other coordinate only when the condition is fulfilled - err> = dist.

If you have functions for drawing lines, it is not difficult to create functions for drawing rectangles. The example shown here draws rectangles in a given color by creating coordinates for two opposite corners.

```
/* drawing a rectangle */
void box(startx,starty,endx,endy,color_code)
int startx,starty,endx,endy,color_code;
{
        line(startx,starty,endx,starty,color_code);
        line(startx,starty,startx,endy,color_code);
        line(startx,endy,endx,endy,color_code);
        line(endx,starty,endx,endy,color_code);
}
```

In order to paint up a rectangle, it is required to write to each raster point inside the rectangle. In this case, it is necessary in a cycle to draw in a given color a series of horizontal lines shifted by one unit along the *y* coordinate - the bar_m() function.

Careful thought helps to eliminate the unnecessary sluggishness inherent in many programs for filling areas for a graphics screen. When padding is based on simple calculations that are performed in turn for each point, time-consuming bit operations are required. A more economical code can determine whether all bit positions of a specific videobuffer byte should be the same color, and when this condition is met, a predefined value is assigned to this byte, which sets all points to the correct color. In this case, there is no need to repeat operations on the same byte, each time setting the bits for only one of the points, information about which contains this byte.

When filling a rectangle with lines, only the first and last byte of each line can be incomplete (unless BX or EX is a multiple of 8). When shading with horizontal lines, it makes no sense to use the line_m () procedure, which implements the Bresenheim algorithm, you just need to determine the byte address in the video memory, which corresponds at the moment to 8 output points with the initial coordinates X and Y, and X is a multiple of 8:

adr=A000:0000h + X/8 +Y*80.

Work #7 explains how to create a description of a symbol in the form of a matrix of 8×8 points, having the form you require. Although such symbols can only

be displayed in standard symbolic positions, their use can greatly facilitate the filling of charts. A pattern that fills all 8×8 points can be drawn in an interval of several rows and columns, filling the area much faster than is achieved with a point-by-point sketch.

When shading a rectangle in the central part, it is advisable to use pseudo-graphic characters with ASCII codes 219, 220, 221, 222, 223, which fill everything, familiarity (219), lower half (220), left (221), right (222) and upper (223). When drawing a rectangle, it also makes no sense to use the Bresenheim algorithm, since it consists of horizontal and vertical segments. The image of horizontal segments is performed similarly to the above method of shading with horizontal lines, and the image of vertical segments is reduced to the output in a cycle of one byte with a given quotient (with the current line of 1 pixel, the mask contains one 1) and an increment of the address by 80 - the transition to the next line. This is modification of the image of an oblique line.

During the operation of the Bresenheim algorithm, in the case of D_X> D_Y, several points are displayed in each line. If, before block 9, the algorithm parses the BX value with unchanged BY and appends 1 to the bit mask. Then, when using the procedure for outputting a byte with a given mask, developed in the laboratory, it is accessed only in the case of a multiplicity of 8 of the BX value (transition to the next video memory address) or changing BY (transition to the next line). If D_Y> D_X and the line thickness is 2, 3, etc. pixel in one call to put_pix (), you can output the corresponding number of points (if they refer to the same video memory address).

9.3 Order of work.

9.3.1. Write a program that uses the put_pix () procedure developed in work 8 and performs the following actions:

9.3.1.1. Puts the system into graphics mode 10h.

9.3.1.2. Draws a rectangle with parameters corresponding to the individual task (rest_m ()).

9.3.1.3. Paints up a rectangular area with parameters corresponding to the individual task (bar_m ()).

9.3.1.4. Draws an oblique line using a modification of Bresenheim's algorithm with parameters corresponding to the individual task (line_m ()).

9.3.1.5. Returns the system to text mode.

9.4. Features of programming.

9.4.1. In Turbo-Pascal language.

To access video memory, the predefined array Mem is used. For example, to read a byte at address A000: 0000h, use the expression b: = Mem [$ A000: $ 0000]

(b is a byte variable).

The predefined Port array is used to refer to the PC ports. For example, to write the 3Dh value to port 3C0h, use the expression

Port [$ 3C0]: = $ 3D;

to read from port 3DAh, use the expression

data: = Port [$ 3DA], where data is a byte variable.

9.4.2. In the Turbo-C language.

To access video memory, far pointers are used, which are declared in the program as follows:

char far * uk; - to work with bytes.

To read the byte at address A000: FFF5h, use the expression

uk = (char far *) 0xA0000000;

b = * uk; where b is a variable of type char;

To access the PC ports, the port read and write functions are used, which are stored in the <dos.h> library. The library is connected with the command

#include <dos.h>

after which, for example, to write the 3Dh value to port 3C0h, the following expression is used:

outportb (0x3C0,0x3d);

To read from port 3DAh, the following expression is used:

data = intportb (0x3DA), where data is a variable of the char type;

9.5. Individual task.

For the second digit of the student's number, select the parameters according to table. 9.1.

Table 9.1 - Variants for individual tasks

Parameters	Student's journal number									
	0	1	2	3	4	5	6	7	8	9
Xn1	30	0	10	30	50	60	40	30	20	0
Yn1	60	0	20	50	10	40	30	10	40	50
Xk1	260	200	250	280	300	310	210	240	290	190
Yk1	120	150	140	130	120	110	100	150	140	130
L1	5	3	4	2	4	3	5	4	3	4
color	C	1	3	4	5	6	7	8	A	B
Xn2	350	400	410	450	430	460	360	330	420	340
Yn2	0	60	50	40	10	30	40	10	50	20
Xk2	580	500	600	550	630	560	680	510	520	530
Yk2	150	120	130	140	150	100	110	120	130	140
Color2	7	D	E	F	1	2	3	4	5	6
Xn2	10	30	0	20	30	40	60	50	100	90
Yn2	320	160	360	140	310	120	290	180	280	150
Xk2	630	500	400	600	580	530	420	510	620	460
Yk2	150	300	160	310	140	290	120	280	180	320
Color3	1	8	9	A	B	C	D	E	F	2
L3	3	5	4	3	4	5	3	4	2	4

where: Xn1, Yn1, Xk1, Yk1 - coordinates of the upper left (index n) and right (index k) corner of the rectangle; L1 - line width in the rectangle; color1 - color of the rectangle; Xn2, Yn2, Xk2, Yk2, color2 - corresponding shading parameters. Xn3, Yn3, Xk3, Yk3, L3, color3 - coordinates, thickness and color of the oblique line.

9.6. Content of the report

9.6.1. The topic of the laboratory work.
9.6.2. Purpose of work.
9.6.3. Individual task.
9.6.4. Program text.
9.6.5. The results of the program.
9.6.6. Conclusions.

LITERATURE

1 Поворознюк А. І. Архітектура комп'ютерів. Архітектура мікропроцесорного ядра та системних пристроїв: Навчальний посібник. Ч.1. – Харків: НТУ "ХПІ" , 2014. – 355 с.

2 Поворознюк А. І. Архітектура комп'ютерів. Архітектура зовнішньої пам'яті, відеосистеми та зовнішніх інтерфейсів: Навчальний посібник. Ч.2. – Харків: НТУ "ХПІ", 2014. – 296 с.

3. Methodological instructions for course project of the course Computer architecture. / Povoroznyuk A. I., Povoroznyuk O. A., Filatova G.E. – Kharkiv: NTU "KhPI", 2023. – 42 p.

4. Joseph D. Dumas II. Computer Architecture. Fundamentals and Principles of Computer Design. – CRC Press – 2018 – 400 P.

5. Chopra Rajiv. Computer Architecture and Organization (A Practical Approach).– S. Chand Publishing – 2018 – 197 P.

6. James L. Antonakos Pentium Microprocessor. – Pearson 2020 – 539 P.

7. Harvey G. Cragon. Computer Architecture and Implementation. – Cambridge University Press – 2021 – 318 P.

8. Jim Ledin, Dave Farley. Modern Computer Architecture and Organization: Learn x86, ARM, and RISC-V architectures and the design of smartphones, PCs, and cloud servers. – Packt Publishing – 2022 – 666 P.

Educational edition

POVOROZNYUK Anatoly
POVOROZNYUK Oksana

Laboratory workshop on the course

COMPUTER ARCHITECTURE

The work was recommended to the publication by prof. M.Y. Zapolovsky

In the author's edition

Plan 2024, item 138