Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

**Программное средство «Эмулятор работы операционной системы DOS»**

БГУИР КП I–40 01 01 009 ПЗ

Выполнил

студент: гр. 251003 Жук Я. С.

Проверил: Шостак Е. В.

Минск 2023

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Лапицкая Н.В. 2023г.

ЗАДАНИЕ

по курсовому проектированию

Студенту *Жук Яне Сергеевне*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Тема работы *Эмулятор работы операционной системы DOS*\_\_\_\_\_\_\_\_ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Срок сдачи законченной работы *22.12.2023г.* ,

3. Исходные данные к работе *Среда разработки FASM Windows. Язык программирования Assembler.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*1 Анализ литературных источников* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2 Постановка задачи*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*3 Разработка программного средства\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*4 Тестирование и проверка работоспособности программного средства\_\_\_\_*

*5 Руководство по установке и использованию программного средства \_\_\_\_\_*

*Заключение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Список использованных источников\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Приложения* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*Схема алгоритма в формате А1*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Консультант по курсовой работе *Шостак Е.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

7.Дата выдачи задания *08.09.2023г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объема работы):

*Раздел 1. Введение к 15.09.2023г. – 10 % готовности работы;\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 2 к 15.10.2023г. – 30% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 3 к 15.11.2023г. – 60% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 4 к 20.11.2023г. – 80% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 5.Заключение. Приложения к 25.11.2023г. – 90% готовности работы;*

*оформление пояснительной записки и графического материала к 10.12.2023г. – 100% готовности работы.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Защита курсового проекта с 10.12.2023г. по 26.12.2023г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

РУКОВОДИТЕЛЬ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шостак Е.В.*

*(подпись)*

Задание принял к исполнению *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Жук Я. С. 08.09.2023г.*

*(дата и подпись студента)*

**содержание**

[Введение 5](#_30j0zll)

[1 Анализ литературных источников 6](#_1fob9te)

[1.1 История 6](#_3znysh7)

[1.2 Анализ прототипов 6](#_2et92p0)

[1.2.1 DOSBox. (2002г.) 6](#_tyjcwt)

[1.2.2 DOSEMU(1992) 7](#_3dy6vkm)

[2 Постановка задачи 8](#_1t3h5sf)

[2.1 Описание функциональных требований 8](#_4d34og8)

[2.2 Необходимые ресурсы для курсовой работы 8](#_2s8eyo1)

[3 Разработка программного средства 9](#_17dp8vu)

[3.1 Разработка эмулятора и анализ архитектуры х86 9](#_3rdcrjn)

[3.1.1 Размер команд 9](#_26in1rg)

[3.1.2 Анализ инструкций 9](#_lnxbz9)

[3.1.3 MODR/M 10](#_35nkun2)

[3.2 Разработка эмулятора и анализ среды MS-DOS 12](#_1ksv4uv)

[3.2.1 Эмуляция BIOS 12](#_44sinio)

[3.2.2 Эмуляция программных прерываний MS-DOS 12](#_2jxsxqh)

[3.2.3 Эмуляция работы сервисов 13](#_z337ya)

[3.2.4 Эмуляция PSP и загрузки программ в RAM 13](#_3j2qqm3)

[3.2.5 Эмуляция работы некоторых функций MS-DOS 13](#_1y810tw)

[3.3 Разработка графического интерфейса 13](#_4i7ojhp)

[3.3.1 Реализация интерфейса 13](#_2xcytpi)

[3.4 Разработка основного ЦОС 15](#_1ci93xb)

[4 Тестирование и проверка работоспособности програмного средства 17](#_3whwml4)

[4.1 Тестирование корректности работы приложения 17](#_2bn6wsx)

[4.2 Итоги тестирования 19](#_qsh70q)

[5 Руководство по установке и использованию программного средства 20](#_3as4poj)

[5.1 Минимальные системные требования 20](#_1pxezwc)

[5.2 Установка программного обеспечения 20](#_49x2ik5)

[Заключение 22](#_2p2csry)

[Список использованных источников 23](#_147n2zr)

[Приложение А 24](#_3o7alnk)

[Приложение Б 27](#_23ckvvd)

# Введение

Операционная система MS-DOS стала первым крупным продуктом, давшим рывок ныне одному из главных гигантов в разработке ПО – компании Microsoft. Данная операционная система была выбрана компанией IBM для своих новых компьютеров семейства IBM-PC. Это семейство стало одним из первых массовых персональных компьютеров в истории, именно с них компьютер перестал ассоциироваться с серьёзными производствами, университетами или научными исследованиями, а стал полноценной утилитарной вещью.

Однако с момента выпуска MS-DOSа прошло более 40 лет. Уже успела полностью измениться архитектура процессоров – произошёл переход от классической 16-битной архитектуры к 32, а затем и к 64-битной. Также не стоит забывать, что классический х86 процессор работал в реальном режиме, в котором программа могла обращаться к любой части адресуемого пространства и свободно оттуда читать и записывать данные. В современных компьютерах же используется множество механизмов защиты данных, будь то кольца защиты, страничная адресация, виртуализация адресов. Также не стоит забывать и о переходе от проводных механизмов прерываний к сообщениям.

Всё это не даёт возможности работать большинству тех программ, которые были написаны под MS-DOS. Ранее компания Intel поддерживала некую среду, в которой можно было исполнять COM-овские файлы, однако постепенно от этого отказываются в силу того, что очень много инструкций являются ныне устаревшими и они тянутся тяжким грузом для современных процессоров.

Данное программное средство создаёт среду, в которой приложения, которые были написаны под MS-DOS, смогут выполняться ровно так, как задумывалось автором, но на современном процессоре и современной операционной системе.

# Анализ литературных источников

## История

MS-DOS – первый по настоящему прорывной продукт для компании Microsoft. Однако данная операционная система не была разработана компанией с нуля, они купили права на использование и доработку уже существовавшей операционной системы 86-DOS, разработанной компанией  [Seattle Computer Products](https://ru.wikipedia.org/w/index.php?title=Seattle_Computer_Products&action=edit&redlink=1).

MS-DOS был достаточно долгоиграющим продуктом, ведь его обновляли вплоть до 2000-го года, полноценная поддержка её продолжалась до 2001-го года. За чуть менее чем 20 лет успело выйти 8 глобальный версий MS-DOS (последняя версия имела индекс 8.0, однако начиная с версий 7.0 являлась скорее составляющей новых продуктов, таких как Windows 95, Windows 98 и Windows Me).

Программы, написанные под архитектуру IBM PC компанией Microsoft активно поддерживались и во время Wimdows XP, и во время Windows 7, однако после неё постепенно начали отказываться от обработки файлов формата COM. Ниже предоставлен пример того, как отреагирует Windows 11 на попытку запуска файла формата COM на исполнение:

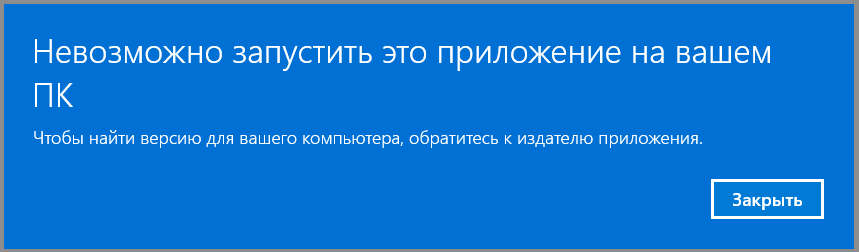


Рисунок 1 – реакция Операционной Системы Windows на попытку открытия файла lab3.com

## Анализ прототипов

### DOSBox. (2002г.)

DOSBox– эмулятор среды DOS для PC, на которых программы, написанные под DOS, могут выполняться некорректно, либо же вовсе не могут быть выполнены. Данный эмулятор был написан на языке программирования C++.

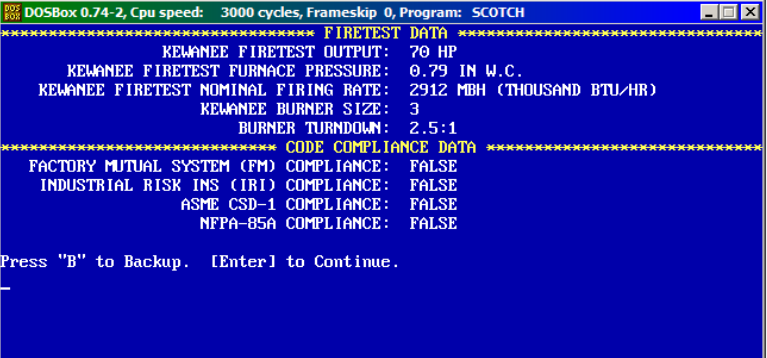


Рисунок 1.1 – DOSBOX. (2002)

### DOSEMU(1992)

DOSEMU — ПО для создания слоя совместимости для запуска [MS-DOS](https://ru.wikipedia.org/wiki/MS-DOS) систем и их клонов таких как [FreeDOS](https://ru.wikipedia.org/wiki/FreeDOS), а также DOS совместимого ПО под [GNU](https://ru.wikipedia.org/wiki/GNU)/[Linux](https://ru.wikipedia.org/wiki/Linux_(%D1%8F%D0%B4%D1%80%D0%BE)) на компьютерах [x86](https://ru.wikipedia.org/wiki/X86) архитектуры ([IBM PC](https://ru.wikipedia.org/wiki/IBM_PC)-совместимые компьютеры).

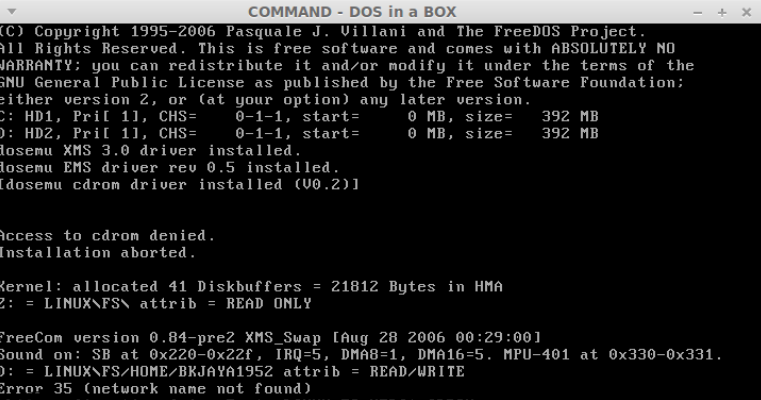


Рисунок 1. 2 – DOSEMU (1992)

# Постановка задачи

## Описание функциональных требований

В результате обзора аналогов был сделан вывод, что основой для решения поставленной задачи будет DOSEMU (1992). В процессе разработки должны быть реализованы базовые функции эмулятора:

* Возможность записи/удаления символов в командной строке;
* Возможность исполнения файлов, находящихся в заданной директории;
* Возможность выполнять некоторые базовые команды DOSа (cd…);
* Эмуляция работы счётчика команд;
* Эмуляция работы всех однобайтовых инструкций;
* Эмуляция работы некоторых двухбайтовых инструкций   
  (80h-83h, FEh-FFh, F6-F7);
* Эмуляция работы BIOS;
* Эмуляция работы шины;
* Эмуляция работы некоторых базовых прерываний DOS-а;
* Эмуляция звука при переполнении буфера;
* Организация двух видеорежимов (03h, 13h).

## Необходимые ресурсы для курсовой работы

Для разработки программного средства будет использоваться язык программирования Assembly. Среда разработки – FASM Windows. Для работы с графическим интерфейсом и звуком используется Windows API.

# Разработка программного средства

## Разработка эмулятора и анализ архитектуры х86

### Размер команд

Особенностью х86 процессора можно назвать то, что его архитектура является ярким представителем CISC (Complex Instruction Set Command). Исходя из этого, первой сложностью, которая возникает при проектировке эмулятора, является нефиксированная длина одной инструкции, варьирующаяся на поддерживаемой версии процессора от 1 до 6 байт.

### Анализ инструкций

Каждый первый прочитываемый байт для процессора – код инструкции. Некоторое число байт за инструкцией отвечают за конкретизацию операндов.

В Intel x86 существуют однобайтовые, двухбайтовые и трёхбайтовые инструкции. Механизм кодировки двухбайтовых и трёхбайтовых инструкций состоит в том, что при некоторых зарезервированных значениях анализируемого байта необходимо прочитать ещё один байт, в котором будет либо определена инструкция, тогда получается двухбайтовая инструкция, либо же полученное значение так же будет зарезервировано и для конкретизации инструкции нужно будет прочитать третий байт.

В данном эмуляторе реализованы однобайтовые и некоторые двухбайтовые инструкции.

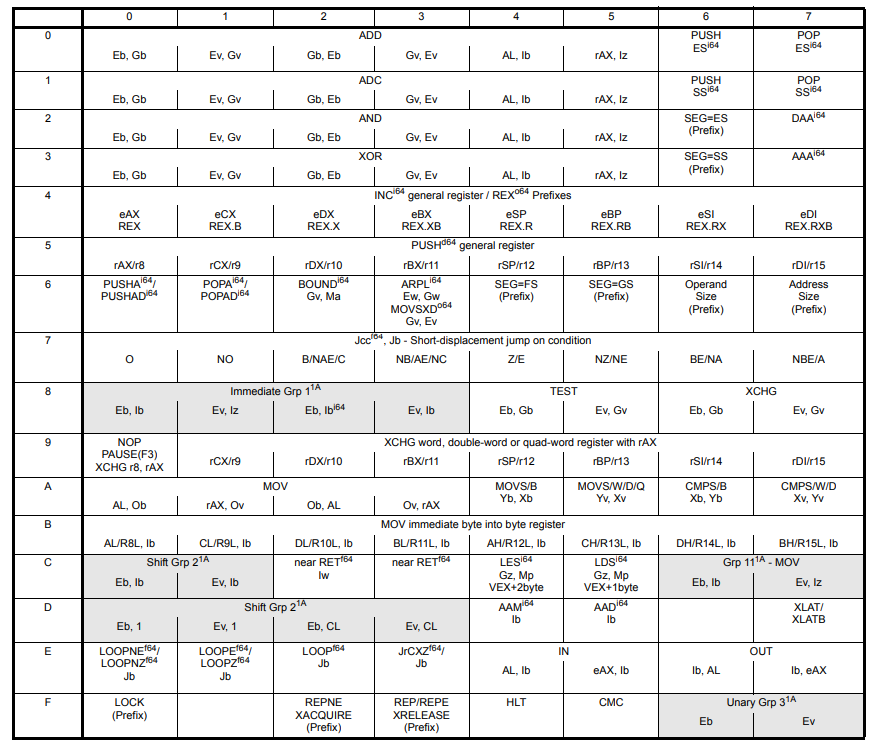


Рисунок 2.2 – пример таблицы кодировки первых байт инструкций в процессоре Intel x86

### MODR/M

MODR/M Byte- это байт, следующий за байтами, которые определяют то, какая инструкция должна выполняться, и в котором закодированы два операнда, участвующие в этой инструкции. MODR\M byte содержит в себе 3 поля:

* MOD(7-6 биты) – в нём кодируется тип второго операнда (если значение данного поля лежит в диапазоне от 0 до 2, то в нём кодируется операнд в памяти, если же значение поля равно 3, то регистр)
* REG(5-3 биты) – в нём кодируется первый операнд-регистр общего назначения (в зависимости от размера операндов, который кодируется в инструкции, это поле может значить как один из восьми возможных 16битных регистров общего назначения, так и на одну из 8битных частей 16битных регистров общего назначения).
* R/M(2-0 биты) – в этом поле уточняется, какие регистры используются для определения второго операнда.

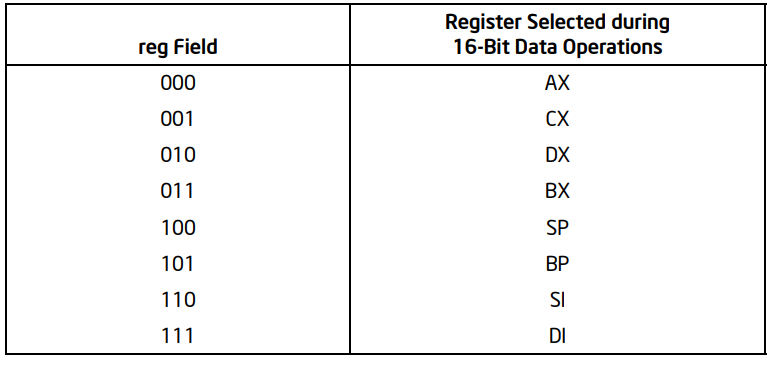


Рисунок 2.3 – кодировка поля REG в MODR/M байте

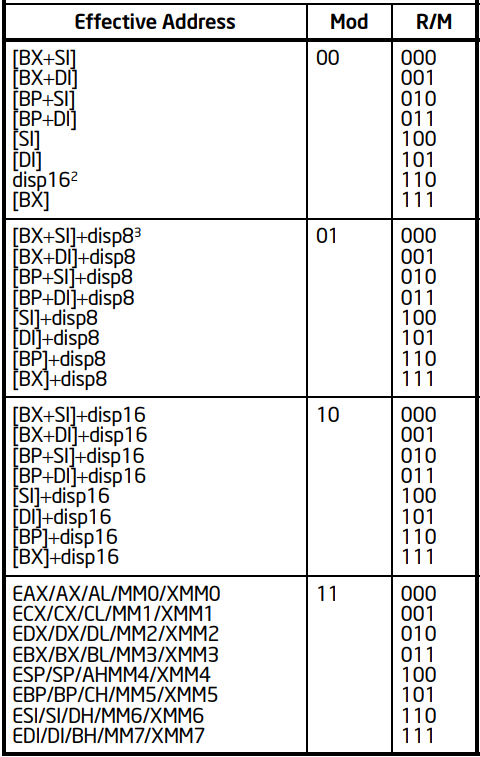


Рисунок 2.4 – кодировка полей R\M и Mode в MODR/M байте

## Разработка эмулятора и анализ среды MS-DOS

### Эмуляция BIOS

Одной из важных задач было сделать минимальную эмуляцию работы BIOS, а конкретно обработку клавиатурного ввода и обработку системного таймера. Кроме того, в следствие того, что при работе с данными видами прерываний происходит заполнение / изменение такой «структуры» (указателей и самого клавиатурного буфера, счетчика и т.п.), как BIOS Data Area, её составление также необходимо. Так как взаимодействие пользователя с программным средством организовывается посредством WinAPI, то суть обработки клавиатурного ввода заключается в том, чтобы:

* Создать корректный ЦОС;
* Выявлять нажатие на такие клавиши, как SHIFT, ALT, CTRL и т.д., для выставления флагов в BDA (BIOS Data Area);
* Выявлять нажатие расширенных клавиш;
* Выявлять нажатие обычных клавиш;
* Заносить в клавиатурный буфер скан-код и ASCII-код нажатой обычной клавиши;
* Изменять значения указателей клавиатурного буфера («головы» и «хвоста»);
* Отлавливать переполнение клавиатурного буфера и производить реакцию на него;
* Корректно обрабатывать чтение из клавиатурного буфера;
* Корректно обрабатывать нажатие на клавиши SPACE и BACKSPACE.

### Эмуляция программных прерываний MS-DOS

У MS-DOS своих прерываний не много, конкретно 20-2fh, из которых лишь 21h содержит большое количество функций, однако помимо него необходимо прописать и 20h, которое используется для выхода из программы, в котором в силу организации связи с эмуляцие й процессора, прописана остановка передачи управления этому самому процессору. Из прерывания 21h были прописаны следующие функции, которые также затрагивают и эмулируемые 16-битные регистры:

* 00h;
* 01h;
* 02h;
* 08h;
* 09h;
* 0ah.

### Эмуляция работы сервисов

Помимо прерываний BIOS’a и DOS’a стояла необходимость реализовать следующие сервисы:

* 10h – для взаимодействия с графической памятью и видеорежимами;
* 16h – для взаимодействия с клавиатурным буфером.

### Эмуляция PSP и загрузки программ в RAM

Так как в данной реализации эмулировалось взаимодействие операционной системы с такими расширением файлов, как .COM, необходимо было загружать содержимое исполняемого файла по смещению 100h, в котором заполнить такую «структуру», как PSP (Program Segment Prefix), а также настроить эмулируемые регистры, эмулируемый стек и некоторый регистр, который будет указывать на место в эмулируемой памяти, с которой эмулируемому процессору необходимо будет производить «прочитку» и «выполнение».

### Эмуляция работы некоторых функций MS-DOS

Так как данная операционная система содержит большое количество функций, было решено написать только некоторые и самые необходимые, а именно:

* «CD»;
* «DIR»;
* «CLS».

## Разработка графического интерфейса

### Реализация интерфейса

Интерфейс программного средства создавался при помощи взаимодействия с Windows API 32, а конкретно с битмапами и окнами.

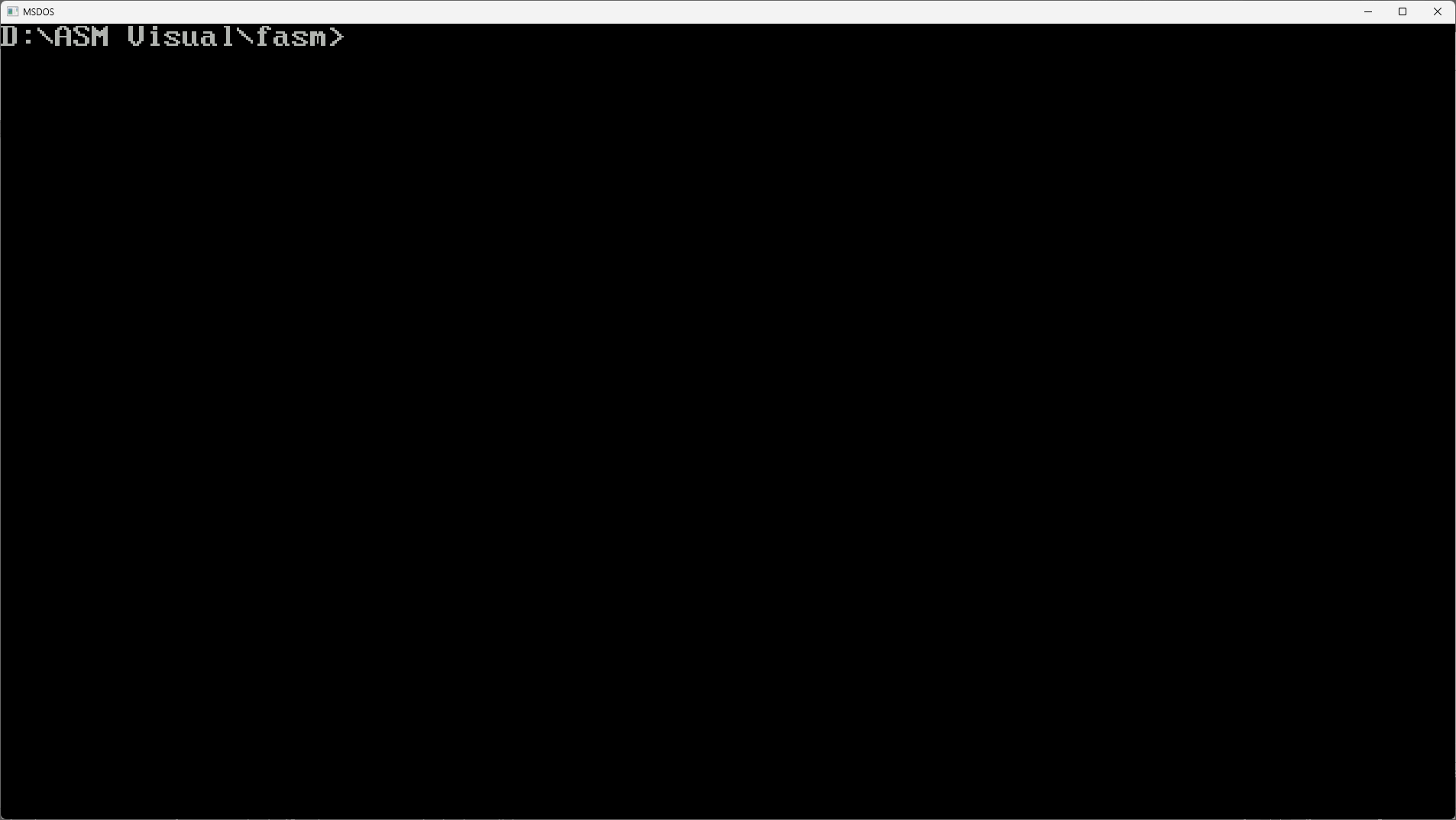


Рисунок 3.1.1 – Окно программного средства DOS в момент запуска программы

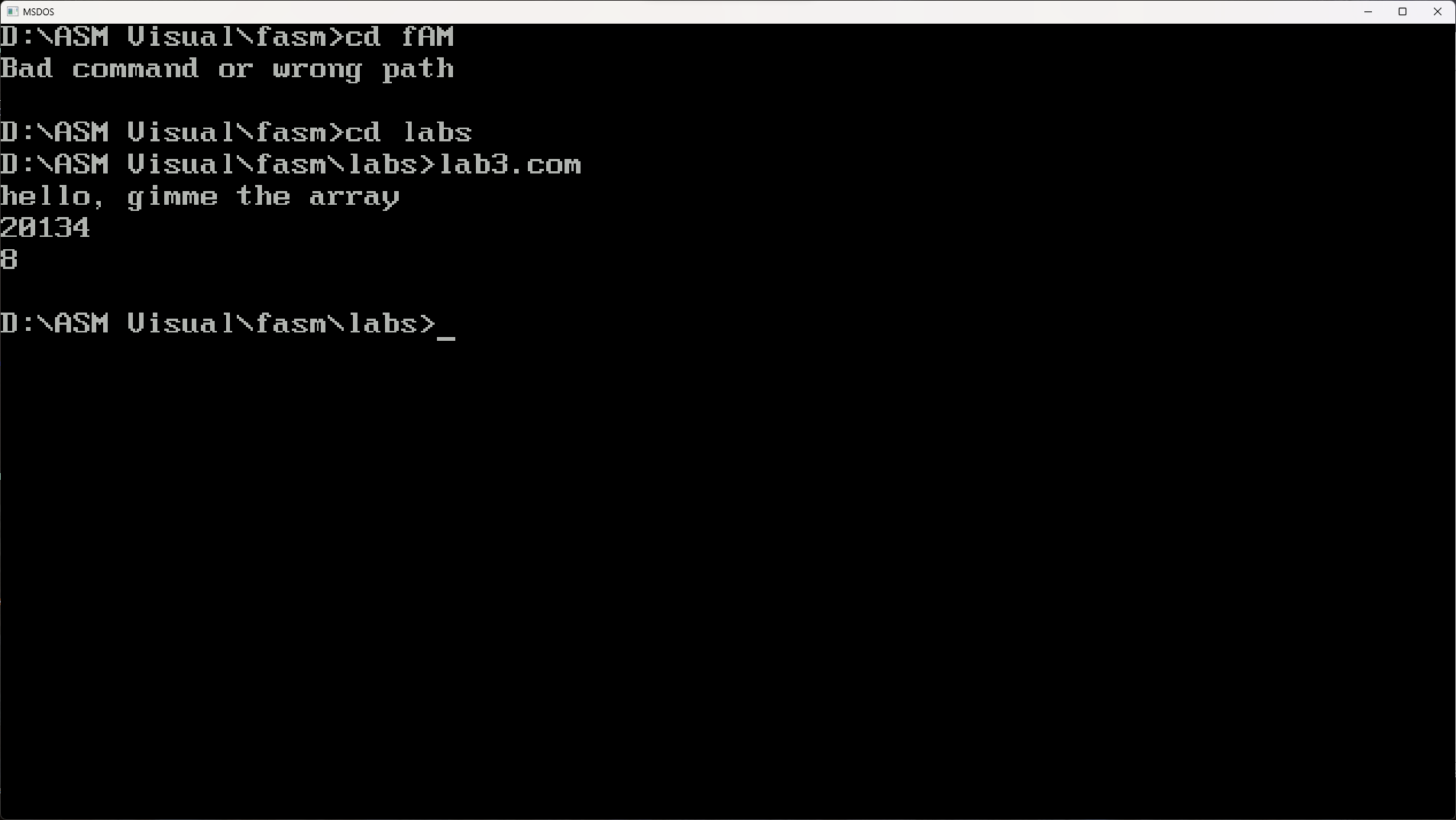


Рисунок 3.1.2– Окно программного средства DOS после ввода нескольких команд



Рисунок 3.1.3 – Окно программного средства DOS после запуска программы, изменяющей текущий видеорежим

## Разработка основного ЦОС

Работа с Windows API 32 подразумевает взаимодействие с множеством приходящих окну сообщений, например WM\_PAINT, WM\_KEYDOWN. Конкретно на этом взаимодействии построена связь эмулируемых ОС и процессора при наличии / отсутствии загруженного исполняемого файла; нажатия и работы пользователя с окном и работы программного средства. Кроме того, каждый второй раз передачи управления эмулируемым процессором операционной системе должна происходить перерисовка, тем самым эмулируется «бесконечный» цикл перерисовки в компьютерах.

Суть данного цикла обработки сообщений строится на использовании самих функций WinAPI для проверки наличия сообщений в очереди и их получении / переводе и некоторых флагов:

* isWaitingInput – если какое-либо прерывание дожидается ввода пользователя;
* isProgram – загружен ли исполняемый файл и происходит ли его выполнение (по инструкции за «такт»).

Ниже предоставлен код реализованного ЦОСа:

message\_loop:

invoke PeekMessage, mesg, dword[hWnd], 0, 0,

PM\_NOREMOVE

cmp eax, 0

jz .noMessageInQueue

invoke GetMessage, mesg, dword[hWnd], 0, 0

cmp eax, 0

je end\_d

invoke TranslateMessage, mesg

invoke DispatchMessage, mesg

jmp message\_loop

.noMessageInQueue:

cmp [isWaitingInput], 1

je message\_loop

cmp [isProgram], 1

jne message\_loop

call cpuTact

mov dword[IPpointer],esi

inc byte[amountCPU]

cmp byte[amountCPU], 2

jne message\_loop

invoke InvalidateRect, [hWnd], 0, 0

jmp message\_loop

end\_d:

invoke ExitProcess, 0

# Тестирование и проверка работоспособности програмного средства

## Тестирование корректности работы приложения

Таблица 4.1 – Тестирование корректности работы приложения

| № | Тестируемая функциональ-ность | Последовательн-ость действий | Ожидаемый результат | Полученный результат |
| --- | --- | --- | --- | --- |
| 1 | Проверка отклика консоли на пользовательский ввод | Запустить приложение, нажать на произвольную клавишу клавиатуры | Отрисовка нажатого символа | Тест пройден |
| 2 | Проверка опции переключения курсора | Запустить приложение, нажать на 'Ins' | Курсор станет не горизонтальным, а вертикальным, либо наоборот | Тест пройден |
| 3 | Проверка возможности выхода из приложения | Запустить приложение, нажать на кнопку Закрыть | Закрытие окна, завершение работы приложения | Тест пройден |
| 4 | Обработка команд операционной системы | Запустить приложение, ввести одну из команд, например cd, ввести путь к папке, в которую пользователь желает перейти | Изменение текущего директория | Тест пройден |
| 5 | Обработка неверного пользовательского ввода | Запустить приложение, ввести неверное выражение | Вывод строки “bad command or wrong path” | Тест пройден |
| 6 | Запуск лабораторной работы | Запустить приложение ввести название файла с лабораторной работой | Корректный результат работы лабораторной работы и возвращение управления пользователю | Тест пройден |

Продолжение таблицы 4.1

| 7 | Проверка возможности перехвата пользовательского прерывания | Запустить приложение, открыть файл int70h.com, нажать на символ 'а' | Отрисовка VGA-палитры | Тест пройден |
| --- | --- | --- | --- | --- |
| 8 | Проверка возможности перехода в другой видеорежим | Запустить приложение, открыть файл int70h.com, нажать на символ 'а' | Отрисовка VGA-палитры | Тест пройден |
| 9 | Проверка правильности отображения оригинальной палитры | Запустить приложение, открыть файл int70h.com, нажать на символ 'а' | Отрисовка VGA-палитры | Тест пройден |
| 10 | Проверка корректности возврата из пользовательского прерывания | Запустить приложение, открыть файл int70h.com, нажать на символ 'а', после отрисовки палитры нажать кнопку enter | Возврат в исходный видеорежим с сохранением истории ввода | Тест пройден |
| 11 | Проверка корректности работы команды cls | Запустить приложение, после ввода некоторых команд ввести cls | Очищение консоли | Тест пройден |
| 12 | Проверка корректности работы команды dir | Запустить приложение, ввести команду dir | Вывод данных о хранящихся в данной поддиректории файлов | Тест пройден |

Продолжение таблицы 4.1

| 13 | Проверка возможности запуска лабораторных работ с подпрограммами | Запустить приложение, открыть файл lab6.com, ввести 2 произвольных числа | Получение результата деления первого на второе число, а также остатка от деления. Передача управления пользователю | Тест пройден |
| --- | --- | --- | --- | --- |
| 14 | Проверка возможности переполнения буфера | Запустить приложение, зажать произвольную клавишу. | При наличии на строке более 128 символов, на каждый новый зажатый символ издаётся звук частотой в 783 гц. | Тест пройден |

## Итоги тестирования

Подводя итог, отмечу, что программа отвечает заданным функциональным требованиям, наблюдается стабильность в работе. Вопросов к эстетической части не имеется.

# Руководство по установке и использованию программного средства

## Минимальные системные требования

Программное средство должно функционировать на персональных компьютерах со следующими характеристиками:

* процессор: AMD Fx-6300 или лучше;
* оперативная память: 2 GB 1600 MHz DDR3 или лучше;
* место на жестком диске: 1 Mb;
* видеоадаптер: DirectX 9 или более поздняя версия.

Программное средство должно функционировать в окружении операционной системы Windows 10 / 11.

## Установка программного обеспечения

Для корректной работы программы необходимо распаковать архив в удобную вам папку, затем нажать ПКМ на исполняемый .exe файл, и выбрать «Создать ярлык», после чего этот ярлык поместить в удобное для вас место.

При запуске программы вы увидите главное и единственное окно (рис. 5.1).

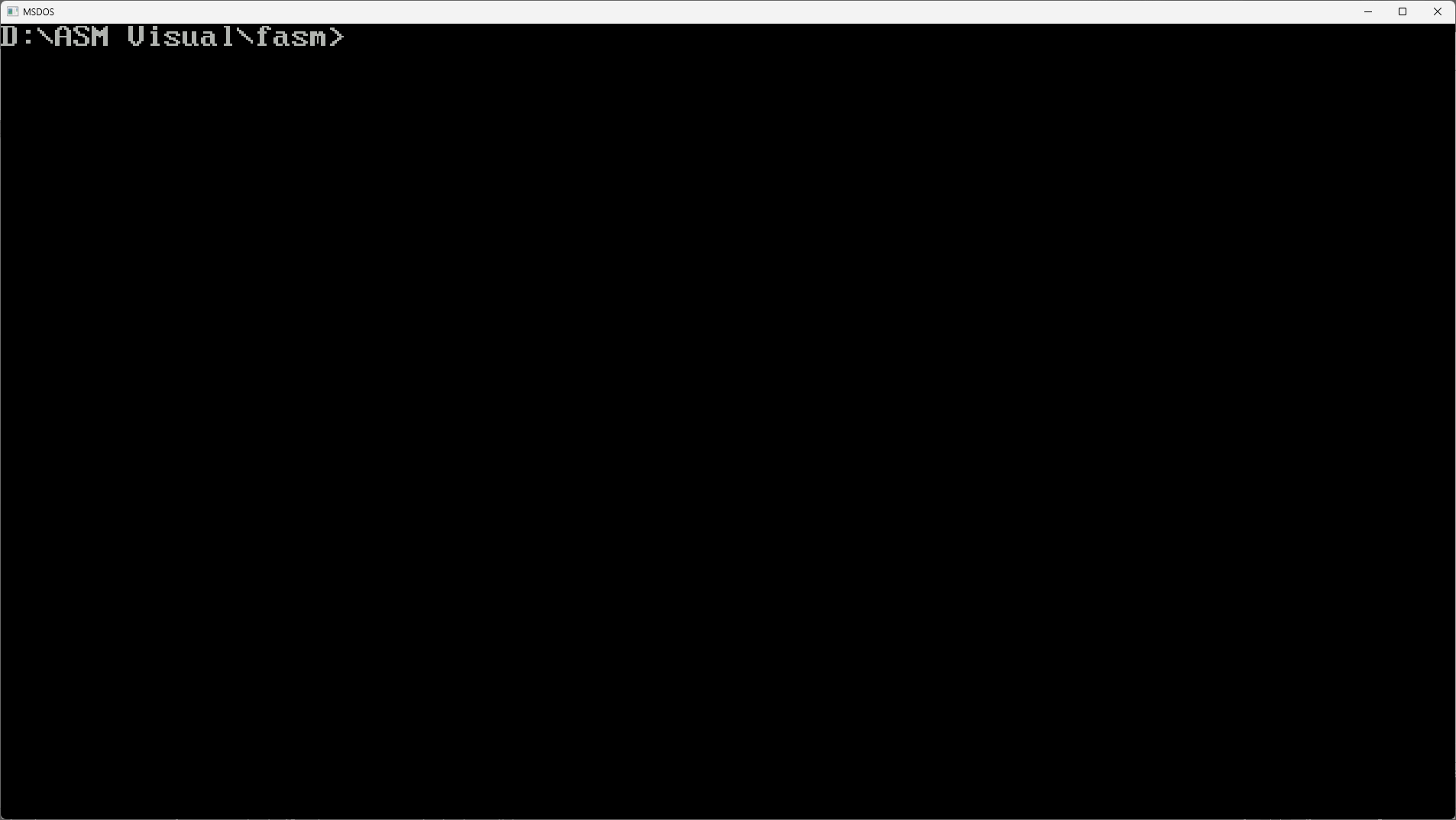
****

Рисунок 5.1 – Окно программного средства DOS

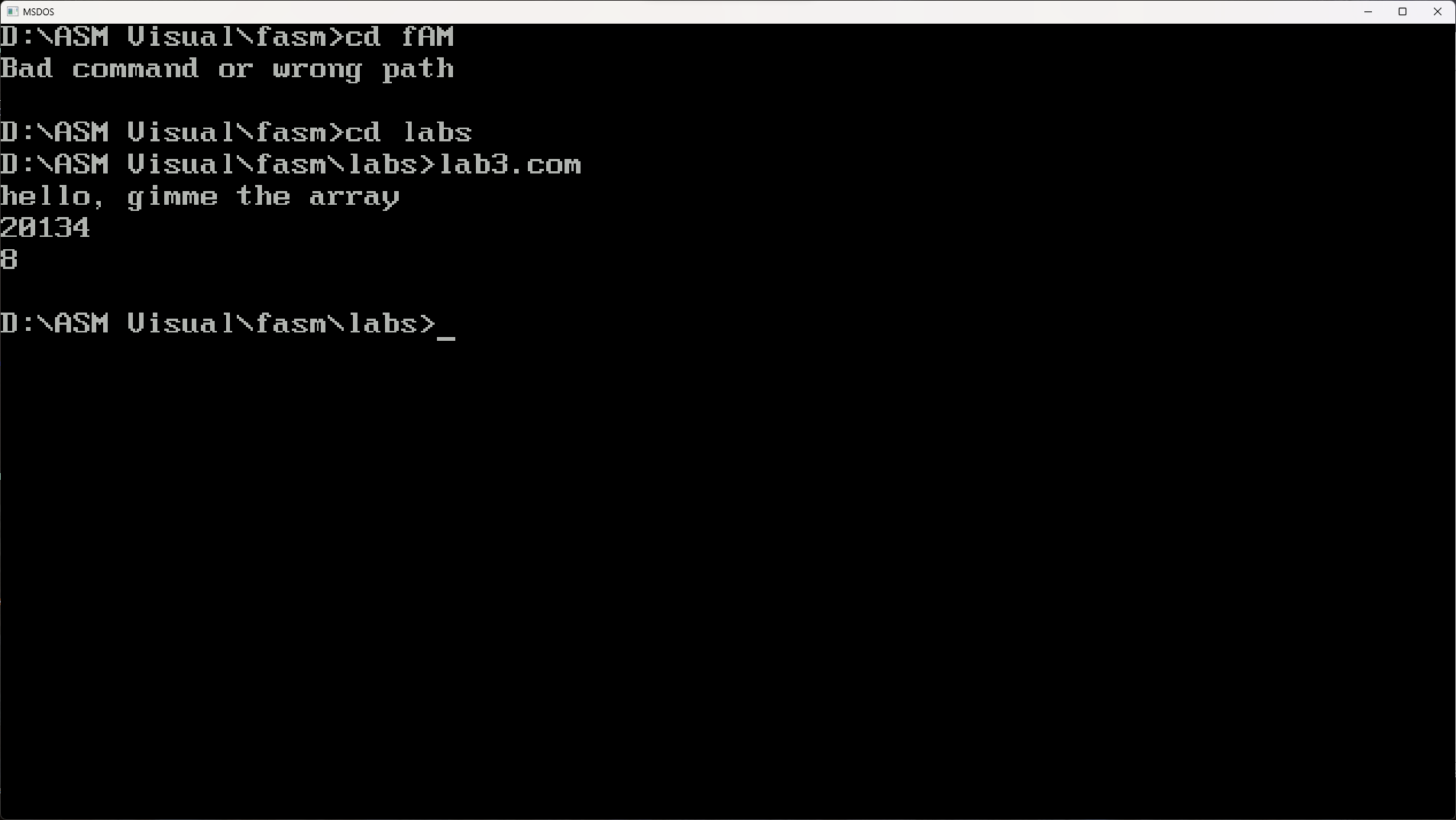


Рисунок 5.2 – Окно программного средства DOS после ввода нескольких команд



Рисунок 5.3 – Окно программного средства DOS после запуска программы, изменяющей текущий видеорежим

# Заключение

В результате выполнения данной работы было разработано программное средство — эмулятор "MS-DOS", с помощью которого возможен запуск приложений, написанных под архитектуру классических х86 процессоров.

Опыт, полученный в ходе работы, включает в себя взаимодействие напрямую с WinAPI, в ходе которого был получен опыт разработки собственной консоли на основе стандартных окон, изучена и решена проблема перерисовки окна, обработки пользовательского ввода, эмуляции поведения командной строки. Также с помощью WinAPI была реализована возможность работы с несколькими видеорежимами, осуществляемая с помощью дополнительного битмапа с палитрой, подгружаемой из файла.

Также стоит отметить работу напрямую с документацией Intel, в ходе которой были изучены таблицы кодировки однобайтовых инструкций, двухбайтовых инструкций, способы кодирования операндов в ModR\M байте, кодировка инструкций в ModR\M байте у инструкций, в которых одним из операндов являлись immediate values. Также был решён вопрос с корректной работой счётчика команд у инструкций переменной длины и у инструкций условного и безусловного переходов.

Проанализировав результаты работы программы, можно сделать вывод о ее корректной функциональности. Разработанный эмулятор успешно выполняет все технические требования, установленные в техническом задании. Это свидетельствует о выполнении поставленных целей и достижении ожидаемого функционального результата.

# Список использованных источников

[1] Оношко, Д. Е. Основы разработки операционных систем : учебно-методическое пособие / Д. Е. Оношко, В. В. Бахтизин. – Минск : БГУИР, 2022. – 123 с.

[2] Intel® 64 and IA-32 Architectures Software Developer’s Manual – 3555 с.

[3] IBM PC/Леванцевич В.А. –2018. 83 с.

[4] Гук, М. Процессоры Pentium II, Pentium Pro и просто Pentium / М. Гук. – СПб. : Питер Ком, 1999. – 288 с.

[5] Кузелин, М. О. Современные семейства ПЛИС фирмы Xilinx : справ. пособие / М. О. Кузелин, Д. А. Кнышев, В. Ю. Зотов. – М. : Горячая линия-Телеком, 2004. – 440 с.

[6] Технические средства диагностирования : справочник / В.В. Клюев [и др.]. – М. : Машиностроение, 1989. – 672 с.

[7] Embedded Microcontrollers : Databook / Intel Corporation. – Santa Clara, Ca, 1994.

[8] Newland, D.E. Mechanical Vibration Analysis and Computation / D.E. Newland – New York: Dover Publications, – 2006 – 608 p.

[9] Таненбаум Э., Бос Х. - Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.: ил. — (Серия «Классика computer science»).

[10] MSDN – Microsoft. – Режим доступа: https://docs.microsoft.com/en/us/welcome-to-docs

# Приложение А



Рисунок А.1 – Схема алгоритма настройки эмулируемого стека



Рисунок А.2 – Схема алгоритма функции CLS



Рисунок А.3 – Схема алгоритма обработки клавиши BACKSPACE

# Приложение Б

; ---S E C T I O N O F C O D E---

.code:

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* INTERAPTIONS \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int08h:

ret

int10h:

push ebx

push edx

mov dl, [GeneralRegs+1]

cmp dl, 0fh

je .h0f

cmp dl, 00h

jnz int21h.end\_INT

.h00:

mov dl, [GeneralRegs]

cmp dl, 3

jne .Mode13

mov [currVideoMode], btm\_03h

jmp int21h.end\_INT

.Mode13:

mov [currVideoMode], btm\_13h

jmp int21h.end\_INT

.h0f:

mov edx, [currVideoMode]

mov dl, byte[edx]

mov byte[GeneralRegs], dl

mov byte[GeneralRegs+2\*3+1], 0

mov edx, [currVideoMode + 1 + BITMAPINFOHEADER.biWidth]

mov byte[GeneralRegs+1], dl

jmp int21h.end\_INT

int16h:

jmp int21h.end\_INT

int20h:

mov [isProgram], 0

mov eax, 00000400h

mov edi, RAM

mov ecx, 255

rep stosd

mov dword[RAM+400h], 9Bh

mov byte[RAM+419h], 0

mov word[RAM+41Ah], 41Eh

mov word[RAM+41Ch], 41Eh

mov [repaintFull], 1

invoke InvalidateRect, dword[hWnd], 0, 0

call newLine

ret

int21h:

push ebx

push edx

mov dl, [GeneralRegs+1]

cmp dl, 2ch

je .h2c

cmp dl, 01h

je .h01

cmp dl, 02h

je .h02

cmp dl, 08h

je .h08

cmp dl, 09h

je .h09

cmp dl, 0ah

je .h0a

cmp dl, 0ch

je .h0c

cmp dl, 2ah

je .h2a

cmp dl, 00h

jne .end\_INT

.h00:

jmp .end\_INT

.h01

mov byte[GeneralRegs], 0

mov dx, [RAM+41Ah]

cmp dx, [RAM+41Ch]

ja .read

.waiting:

mov byte[isWaitingInput], 1

jmp .end\_INT

.read:

xor eax, eax

call readSymbol

mov [GeneralRegs], al

cmp byte[GeneralRegs+1], 08h

jz .end\_h08

.echoIt:

call newLine

mov byte[currPos], 0

stdcall DrawSymb, eax, 0, 8

jmp .end\_INT

.h02: ; sends the char to the StOutput; handles backspace

xor edx, edx

mov dl, [GeneralRegs+4]

cmp dl, 8

jne .noBackspace

stdcall prevPosition

jmp .end\_INT

.noBackspace:

stdcall DrawSymb, edx, 0, 8

jmp .end\_INT

.h08:

jmp .h01

.end\_h08:

stdcall DrawSymb, eax, 0, 8

jmp .end\_INT

.h09:

xor eax, eax ; --calculating address of DS:DX

mov ax, word[SegmentRegs+6]

shl eax, 4

xor edx, edx

mov dx, word[GeneralRegs+4]

add eax, edx

add eax, RAM

stdcall outputString, eax, 0

cmp [currPos], 0

je .end\_INT

mov ax, [currPos]

sub ax, 2

mov [startPos], ax

jmp .end\_INT

.h0a:

xor eax, eax ; --calculating address of DS:DX

mov ax, word[SegmentRegs+6]

shl eax, 4

xor edx,edx

mov dx, word[GeneralRegs+4]

add eax, edx

add eax, RAM

mov edx, eax

.readSymb:

call readSymbol

cmp eax, 0

je .wait

stdcall DrawSymb, eax, 0, 8

movzx ebx, byte[edx+1]

mov byte[edx+ebx+2], al

inc byte[edx+1]

mov bl, byte[edx+1]

cmp bl, byte[edx]

je .readyString

jmp .readSymb

.wait:

; inc edx

; mov byte[edx], bl

mov [isWaitingInput], 1

jmp .end\_INT

.readyString:

movzx ebx, byte[edx+1]

mov byte[edx+ebx+2], 13

jmp .end\_INT

.h0c:

jmp .end\_INT

.h2a:

jmp .end\_INT

.h2c:

; <<43H CREATE>>

.end\_INT:

pop edx

pop ebx

jmp calculateNewAdress

int22h:

ret

int23h:

ret

int24h:

ret

int25h:

ret

int43h:

ret

calculateNewAdress:

push eax

; push esi

push edx

movzx eax, word[SegmentRegs+2\*2] ; SS

shl eax, 4

movzx esi, word[GeneralRegs+4\*2]; SP

add eax, RAM ;SS\*16+SP+RAM

add eax, esi

movzx esi, word[eax+2]

shl esi, 4

movzx edx, word[eax+0]

add esi, edx

add esi, RAM ;esi

mov [IPpointer], esi

mov bx, word[eax+4]

mov [FLAGSr], bx

add word[GeneralRegs+4\*2], 6

pop edx

; pop esi

pop edx

ret

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* WINMAIN \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

WinMain:

;PREPARING OPERATING SYSTEM'S MEMORY

mov eax, 00000400h ; --loading the interrupt vector table with specific value

mov edi, RAM

mov ecx, 255

rep stosd

mov dword[RAM+400h], 9Bh

mov byte[RAM+419h], 0

mov word[RAM+41Ah], 41Eh

mov word[RAM+41Ch], 41Eh

invoke CreateFile, \_fontName, GENERIC\_READ, FILE\_SHARE\_READ, NULL,\ ; --loading the font

OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL

mov [hFontFile], eax

invoke ReadFile, dword[hFontFile], fontMem, fontMemSize, fontReaded, NULL

invoke CloseHandle, dword[hFontFile]

invoke CreateFile, \_paletteName, GENERIC\_READ, FILE\_SHARE\_READ, NULL,\ ; --loading the font

OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL

mov [hFontFile], eax

invoke ReadFile, dword[hFontFile], btm\_13h.palette, colorsFile, fontReaded, NULL

invoke CloseHandle, dword[hFontFile]

mov esi, btm\_13h.palette

mov ecx,255

loopStart:

mov ebx, [esi]

mov dword[swapper],ebx

mov bl, byte[swapper+0]

mov bh, byte[swapper+2]

mov byte[swapper+2],bl

mov byte[swapper+0],bh

mov ebx, dword[swapper]

mov dword[esi], ebx

add esi,4

loop loopStart

invoke GetSystemMetrics, SM\_CXSCREEN ; --finding screen resolution for creating a window on whole user screen

mov [screenWidth], eax

invoke GetSystemMetrics, SM\_CYSCREEN

mov [screenHeight], eax

invoke GetModuleHandle, 0 ; --creating window

mov [wcex.hInstance], eax

invoke RegisterClassEx, wcex

invoke CreateWindowEx, 0, dword[wcex.lpszClassName], \_title,\

WS\_OVERLAPPEDWINDOW + WS\_VISIBLE, 0,\

0, dword[screenWidth], dword[screenHeight],\

0, 0, dword[wcex.hInstance], 0

mov dword[hWnd], eax

invoke GetDC, dword[hWnd]

mov dword[hDC], eax

invoke SetTimer, dword[hWnd], IDT\_INSERT, 500, NULL

; MESSAGE LOOP

message\_loop:

invoke PeekMessage, mesg, dword[hWnd], 0, 0, PM\_NOREMOVE

cmp eax, 0

jz .noMessageInQueue

invoke GetMessage, mesg, dword[hWnd], 0, 0

cmp eax, 0

je end\_d

invoke TranslateMessage, mesg

invoke DispatchMessage, mesg

jmp message\_loop

.noMessageInQueue:

cmp [isWaitingInput], 1

je message\_loop

cmp [isProgram], 1

jne message\_loop

call cpuTact

mov dword[IPpointer],esi

inc byte[amountCPU]

cmp byte[amountCPU], 2

jne message\_loop

invoke InvalidateRect, [hWnd], 0, 0

jmp message\_loop

end\_d:

invoke ExitProcess, 0

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* WINDOWPROC \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

proc WindowProc uses edx ebx ecx esi edi, hwnd, wmsg, wparam, lparam

locals

ps PAINTSTRUCT

rcClient RECT

endl

;cmp [wmsg], WM\_SYSKEYDOWN ; --analys key (namely, alt + smth)

;je .inputKey

cmp [wmsg], WM\_KEYDOWN ; --analys key (namely, not alphabet and digits)

je .inputKey

;cmp [wmsg], WM\_KEYUP ; --analys key for flags only

;je .changeFlags

cmp [wmsg], WM\_CHAR ; --analys key (namely, ASCII symbols)

je .inputTextASCII

cmp [wmsg], WM\_TIMER ; --insert cursor

je .paintInsert

cmp [wmsg], WM\_PAINT ; --"cleaning" window

je .paintWindow

cmp [wmsg], WM\_DESTROY

je .destroyWindow

; START OF PROCCESING SOME KIND OF MESSAGE

.defProc:

invoke DefWindowProc, dword[hwnd], dword[wmsg], dword[wparam], dword[lparam]

ret

.inputKey: ; KEY/SYSKEYDOWN

xor ecx, ecx

cmp [wparam], VK\_RSHIFT ; -- analys shift keys and alt for keyboard buffer

jz .movKeyboardFlags

inc cl

cmp [wparam], VK\_LSHIFT

je .movKeyboardFlags

inc cl

cmp [wparam], VK\_CONTROL

je .movKeyboardFlags

inc cl

cmp [wparam], VK\_MENU

je .movKeyboardFlags

inc cl

cmp [wparam], VK\_SCROLL

je .movKeyboardFlags

inc cl

cmp [wparam], VK\_CAPITAL

je .movKeyboardFlags

inc cl

cmp [wparam], VK\_INSERT

je .changeInsert

cmp [wparam], VK\_CANCEL ;Ctrl+BREAK

call int23h

cmp [wparam], VK\_BACK

je .clearSymbol

cmp [wparam], VK\_DELETE

je .clearSymbol

cmp [wparam], VK\_RETURN

je .pressedEnter

jmp end\_wp

.movKeyboardFlags:

mov al, 0000001b

rol al, cl

or byte[RAM + 417h], al

jmp end\_wp

.changeInsert:

push ebx

mov bl, byte[insertType]

xor bl, -1 ; changing state

mov byte[insertType], bl

jmp end\_wp

.clearSymbol:

cmp [isProgram], 0

je .notProgram

cmp [isWaitingInput], 1

jne .stopClear

cmp byte[GeneralRegs+1], 01h

jz .h01Clear

.clearPlace:

movzx eax, [currPos]

cmp [startPos], ax

jnb .stopClear

stdcall nextPosition

stdcall clearPos

stdcall clearPos

cmp byte[GeneralRegs+1], 0ah

jnz .changeBP

xor ebx, ebx ; --calculating address of DS:DX

mov bx, word[SegmentRegs+6]

shl ebx, 4

xor edx,edx

mov dx, word[GeneralRegs+4]

add ebx, edx

add ebx, RAM

call readSymbol

cmp eax, 0

jz .noSymb

movzx edx, byte[ebx+1]

mov byte[ebx+edx+1], al

stdcall DrawSymb, eax, 0, 8

;cmp word[RAM+41Ah], 41eh

;jz end\_wp

;sub word[RAM+41Ah], 2 ; <<MAKE CHECK OF OVERFLOW>>

jmp end\_wp

.noSymb:

dec byte[ebx+1]

.changeBP:

cmp word[RAM+41Ah], 41eh

jz end\_wp

sub word[RAM+41Ah], 2

cmp word[RAM+41Ch], 41eh

jz end\_wp

sub word[RAM+41Ch], 2

jmp end\_wp

.isNewLine:

movzx eax, word[ptrBufferInput]

add ax, word[startPos]

cmp ax, 79

ja .clearPlace

jmp .stopClear

.notProgram:

movzx eax, [currPos]

sub eax, 2

cmp ax, [startPos]

jna .isNewLine

stdcall nextPosition

stdcall clearPos

stdcall clearPos

dec [ptrBufferInput]

jmp end\_wp

.h01Clear:

mov byte[GeneralRegs], 0

jmp .changeBP

.stopClear:

jmp end\_wp

.pressedEnter:

cmp [isProgram], 0

jnz .program

cmp [ptrBufferInput], 0

jz end\_wp

stdcall nextPosition

stdcall clearPos

stdcall newLine

stdcall findCommand

mov al, 0

mov ecx, 128

mov edi, bufferInput

rep stosb

mov [ptrBufferInput], 0

jmp end\_wp

.program:

cmp [isWaitingInput], 1

jne end\_wp

stdcall nextPosition

stdcall clearPos

.int21h:

xor edx, edx

mov dl, [GeneralRegs+1]

cmp dl, 08h

jnz .nextF

mov [isWaitingInput], 0

call newLine

.nextF:

cmp dl, 01h

jne .h0a

.h01:

; call newLine

; mov [currPos], 0

; movzx edx, byte[GeneralRegs]

; ; stdcall DrawSymb, edx, 0, 8

; mov [isWaitingInput], 0

jmp end\_wp

.h0a:

.readyString:

xor eax, eax ; --calculating address of DS:DX

mov ax, word[SegmentRegs+6]

shl eax, 4

xor edx,edx

mov dx, word[GeneralRegs+4]

add eax, edx

add eax, RAM

xor ebx, ebx

movzx ebx, byte[eax+1]

mov byte[eax+ebx+2], 13

mov [isWaitingInput], 0

jmp end\_wp

jmp end\_wp

.inputTextASCII: ; CHAR ~I hope, that immediately after KEYDOWN is CHAR~

invoke KillTimer, dword[hWnd], IDT\_INSERT

cmp byte[lparam+24], 0 ; --checking is this an extended key

je .end\_defProc

cmp byte[wparam], 0Dh ; --skip ENTER

je .end\_it

cmp byte[wparam], 8 ; --skip BACKSPACE

je .end\_it

cmp [isProgram], 0

je .notProg

stdcall incReadPointer, 1

cmp eax, 0

jnz .end\_defProc

xor ebx, ebx

movzx ebx, word[RAM+41Ah]

movzx eax, byte[wparam] ; --setting ASCII code of non-extended symb

mov byte[RAM+ebx+1], al

cmp [isWaitingInput], 1

jne .changePointer

stdcall incReadPointer, 0

.i21h:

mov dl, [GeneralRegs+1]

cmp dl, 0ah

je .ha

.h1:

cmp byte[GeneralRegs], 0

jne .isWritten

call readSymbol

mov [GeneralRegs], al

cmp byte[GeneralRegs+1], 08h

jz .changePointer

stdcall DrawSymb, eax, 0, 8

mov [isWaitingInput], 0

jmp .changePointer

;stdcall prevPosition

.isWritten:

invoke Beep, 783, 270

jmp .changePointer

.ha:

xor eax, eax ; --calculating address of DS:DX

mov ax, word[SegmentRegs+6]

shl eax, 4

xor edx,edx

mov dx, word[GeneralRegs+4]

add eax, edx

add eax, RAM

mov ebx, eax

mov dl, [eax]

dec dl

cmp byte[eax+1], dl

jnae .below

.above:

invoke Beep, 783, 270

jmp .changePointer

.below:

inc byte[ebx+1] ; --inc the actual length

movzx edx, byte[ebx+1]

call readSymbol

mov byte[ebx+1+edx], al ; --adding new symbol

stdcall DrawSymb, eax, 0, 8

;stdcall prevPosition

;mov dl, al

;xor ebx, ebx

;mov bl, byte[ebx+1]

;add ebx, ebx

;xor eax, eax

.changePointer: ; --checking is this an end of the buffer

;mov ax, [RAM+41Ch] ; --checking isn't it the end of keyboard buffer

;sub ax, 41Eh

;add ax, 2

;mov dl, 32

;div dl

;cmp ah, 0

;je .goToStart

;add word[RAM+41Ch], 2

; mov ax, [RAM+41Ah] ; --checking is this an overflow

; cmp ax, [RAM+41Ch]

; jae .goToStart

;invoke Beep, 783, 270

; either goToStart or do not write to buffer until it's read an character

; jmp .end\_defProc

; .goToStart:

; mov word[RAM+41Ch], 41Eh

jmp .end\_defProc

.notProg:

cmp [ptrBufferInput], 127

jnae .notWhole

invoke Beep, 783, 270

jmp .end\_defProc

.notWhole:

xor eax, eax

xor ebx, ebx

movzx eax, word[wparam]

movzx ebx, byte[ptrBufferInput]

mov word[bufferInput+ebx\*2], ax

stdcall DrawSymb, eax, 0, 8

inc byte[ptrBufferInput]

jmp .end\_defProc

.end\_defProc:

invoke SetTimer, dword[hWnd], IDT\_INSERT, 500, NULL

mov [insertState], -1

jmp .defProc

.end\_it:

invoke SetTimer, dword[hWnd], IDT\_INSERT, 500, NULL

mov [insertState], -1

jmp end\_wp

.changeFlags: ; KEYUP

mov cl, 0

cmp [wparam], VK\_RSHIFT ; --analys shift keys and alt for keyboard buffer

jz .movKeyboardFlag

inc cl

cmp [wparam], VK\_LSHIFT

je .movKeyboardFlag

inc cl

cmp [wparam], VK\_CONTROL

je .movKeyboardFlag

inc cl

cmp [wparam], VK\_MENU

je .movKeyboardFlag

inc cl

cmp [wparam], VK\_SCROLL

je .movKeyboardFlag

inc cl

cmp [wparam], VK\_CAPITAL

je .movKeyboardFlag

inc cl

cmp [wparam], VK\_INSERT

je .movKeyboardFlag

jmp end\_wp

.movKeyboardFlag:

mov al, 1111110b

rol al, cl

and byte[RAM + 417h], al

jmp end\_wp

.paintInsert: ; TIMER

cmp [isProgram], 0

jne .programI

.paintI:

call paintIns

jmp end\_wp

.programI:

cmp [isWaitingInput], 1

jz .paintI

jmp end\_wp

.paintWindow: ; PAINT

lea edx, [rcClient]

invoke GetClientRect, [hWnd], edx

lea ebx, [ps]

invoke BeginPaint, [hWnd], ebx

;mov edx, btm\_03h.header

mov edx, [currVideoMode]

inc edx

mov ecx, [edx + BITMAPINFOHEADER.biHeight]

not ecx

mov ebx, edx

add ebx, sizeof.BITMAPINFOHEADER

add ebx, 1024

mov ebx, btm\_03h.videoModeMem

cmp byte[edx-1], 3

je .paintWind

mov ebx, [btm\_13h.videoModeMem]

.paintWind:

invoke StretchDIBits, [hDC], 0, 0, [rcClient.right], [rcClient.bottom], \

0, 0, [edx + BITMAPINFOHEADER.biWidth], ecx,\

ebx, edx, 0, SRCCOPY

invoke EndPaint, [hWnd], ebx

cmp byte[repaintFull], 1

jnz .end\_pw

cmp [isProgram], 1

jz .end\_pw\_p

invoke GetCurrentDirectory, 79, startString ; <<MAKE EVERY TIME IT GOES TO THE NEX LINE (ALMOST)>>

invoke GetCurrentDirectory, 0, 0

sub eax, 2

mov [startPos], ax

mov byte[startString+eax\*2+2], '>'

stdcall outputString, startString, 1

.end\_pw\_p:

mov [repaintFull], 0

.end\_pw:

xor eax, eax

jmp end\_wp

.destroyWindow: ; DESTROY

invoke PostQuitMessage, NULL

invoke ExitProcess, 0

xor eax, eax

end\_wp:

ret

endp

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* FUNCTIONS \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; INCREASING THE HEAD OF THE KEYBOARD BUFFER

proc incReadPointer uses ebx edx, isChecking

mov ax, [RAM+41Ah] ; getting head

sub ax, 41Eh

add ax, 2 ; 'inc' it

mov dl, 32

div dl

cmp ah, 0 ; is it the end of the buffer

jz .isTailEqu

xor eax, eax

cmp [isChecking], 1

jz .end\_ir

add word[RAM+41Ah], 2 ; if it is all okey with inc of the head

jmp .end\_ir

.isTailEqu:

mov ax, [RAM+41Ah]

cmp ax, [RAM+41Ch]

jz .toStartBuf

invoke Beep, 783, 270 ; it is the end of buf end the head isn't equ the tail

mov eax, 1

jmp .end\_ir

.toStartBuf:

xor eax, eax

cmp [isChecking], 1

jz .end\_ir

mov word[RAM+41Ah], 41Eh

.end\_ir:

ret

endp

; LOAD EXECUTABLE PROGRAM INTO THE RAM

proc loadProgram uses eax ebx edx edi

locals

file\_path du 120 dup(0)

file\_handle dd ?

endl

lea edi, [file\_path]

mov esi, bufferInput

movzx edx, [ptrBufferInput]

.loadPath:

movsw

dec edx

cmp edx, 0

jnz .loadPath

lea edi, [file\_path]

invoke CreateFile, edi, GENERIC\_READ, FILE\_SHARE\_READ, \

0, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, 0

mov dword[file\_handle], eax

cmp eax, INVALID\_HANDLE\_VALUE

je .noFile

.settingRegs:

mov word[SegmentRegs], 8000h

mov word[SegmentRegs+2\*1], 8000h

mov word[SegmentRegs+2\*2], 8000h

mov word[SegmentRegs+2\*3], 8000h

mov dword[IPpointer], 80000h+100h+RAM

.loadingIntoMemory:

invoke ReadFile, [file\_handle], dword[IPpointer], fileSize, fontReaded, NULL

invoke CloseHandle, [file\_handle]

.settingStack:

xor eax, eax

movzx eax, word[SegmentRegs+2\*2]

shl eax, 4

add eax, RAM+0FFFEh

mov word[eax], 0

mov word[GeneralRegs+2\*4], 0FFFEh

mov [isProgram], 1

.settingPSP:

mov esi, [IPpointer]

mov word[esi-100h], 020cdh

jmp .end\_lp

.noFile:

call outputError

.end\_lp:

ret

endp

; READ SYMBOL FROM KEYBOARD BUFFER

readSymbol:

push ebx

push edx

xor ebx, ebx

xor eax, eax

mov bx, word[RAM+41Ch]

cmp bx, word[RAM+41Ah]

je .clearBuffer

mov bl, byte[RAM + ebx + 1] ; --getting a symbol

add word[RAM+41Ch], 2

mov ax, word[RAM+41Ch] ; --checking isn't it the end of keyboard buffer

sub ax, 41Eh

mov dl, 32

div dl

cmp ah, 0

jne .end\_r

.goToStart:

mov word[RAM+41Ch], 41Eh

.end\_r:

movzx eax, bl

.clearBuffer:

pop edx

pop ebx

ret

; PAINT INSERT CURSOR

paintIns:

push ebx

mov bl, byte[insertState]

xor bl, -1 ; changing state

mov byte[insertState], bl

cmp bl, -1

jnz .drawInsert

stdcall nextPosition

stdcall clearPos ; if it's neccessary to clear

jmp .end\_p

.drawInsert:

mov dx, word[currPos]

mov bh, byte[currLine]

mov edi, btm\_03h.videoModeMem

xor eax, eax ; --calculate videoModeMem address

mov al, bh

mov cl, 80

mul cl

shl eax, 4

add ax, dx

add edi, eax

cmp [insertType], 0

jne .wideInsert

mov ecx, 17

.loadMem:

xor eax, eax

mov al, 10000000b

stosb

add edi, 79 ; Going to the next line

loop .loadMem

jmp .end\_of\_paint

.wideInsert:

mov ecx, 14

.loadMemNull:

xor eax, eax

stosb

add edi, 79 ; Going to the next line

loop .loadMemNull

mov ecx, 2

.loadMemBottom:

xor eax, eax

mov al, 11111111b

stosb

add edi, 79 ; Going to the next line

loop .loadMemBottom

.end\_of\_paint:

stdcall nextPosition

invoke InvalidateRect, [hWnd], 0, 0

; stdcall DrawSymb, 0B3h, 0, 8

stdcall prevPosition

.end\_p:

pop ebx

ret

; CLEAR SYMBOL

proc clearPos uses ebx edx ecx

;cmp word[RAM+41Ah], 41Eh

;jz .preparing

;dec word[RAM+41Ah]

.preparing:

stdcall prevPosition

mov dx, word[currPos]

mov bh, byte[currLine]

mov edi, btm\_03h.videoModeMem

xor eax, eax ; --calculate videoModeMem address

mov al, bh

mov cl, 80

mul cl

shl eax, 4

add ax, dx

add edi, eax

mov ecx, 17

xor eax, eax

.loadMem:

stosb

add edi, 79 ; --going to the next line

loop .loadMem

invoke InvalidateRect, dword[hWnd], 0, 0

ret

endp

; NEXT POSITION

proc nextPosition uses ebx edx

mov dx, word[currPos]

mov bh, byte[currLine]

inc dx

cmp dx, 79

jna .noChangeLine

xor dx, dx ; Set 0 to currPos

inc bh ; Inc currLine

cmp bh, 25

jne .noRepaintFull

mov byte[repaintFull], 1

xor bh, bh

.noRepaintFull:

mov byte[currLine], bh

.noChangeLine:

mov word[currPos], dx

ret

endp

; PREV POSITION

proc prevPosition uses ebx edx

mov dx, word[currPos]

mov bh, byte[currLine]

cmp dx, 0

jne .noChangeLine

mov dx, 79 ; Set 79 to currPos

dec bh ; Dec currLine

cmp bh, 0

jne .noRepaintFull

; mov byte[repaintFull], 1

.noRepaintFull:

mov byte[currLine], bh

jmp .end\_pp

.noChangeLine:

dec dx

.end\_pp:

mov word[currPos], dx

ret

endp

; DRAW SYMBOL

proc DrawSymb uses ebx edx ecx esi edi, symb, clrBack, clrSymb

mov dx, word[currPos]

mov bh, byte[currLine]

mov edi, btm\_03h.videoModeMem

mov esi, fontMem

xor eax, eax ; --calculate videoModeMem address

mov al, bh

mov cl, 80

mul cl

shl eax, 4

add ax, dx

add edi, eax

xor eax, eax ; --calculagte fontMem address

mov eax, [symb]

shl eax, 4

add esi, eax

mov ecx, 17

.loadMem:

;xor al, al

movsb

add edi, 79 ; Going to the next line

loop .loadMem

stdcall nextPosition

invoke InvalidateRect, [hWnd], 0, 0

ret

endp

; GOING TO THE NEXT LINE

newLine:

mov word[currPos], 0

mov word[startPos], 0

inc byte[currLine]

cmp byte[currLine], 24

jne .noRepaintFull

mov byte[repaintFull], 1

.noRepaintFull:

ret

; OUTPUTTING STRING AT THE SCREEN

proc outputString uses ecx ebx edx, stringL, isUnicode

mov ecx, [stringL]

cmp [isUnicode], 1

jnz .isAscii

mov edx, 2

jmp .drawSymbol

.isAscii:

mov edx, 1

.drawSymbol:

cmp byte[ecx], '$'

jz .end\_o

cmp word[ecx], 0

jz .end\_o

cmp byte[ecx], 10

jne .noNewLine

cmp byte[ecx+edx], 13

jne .noNewLine

call newLine

add ecx, edx

jmp .end\_o

.noNewLine:

xor ebx, ebx

mov bl, byte[ecx]

stdcall DrawSymb, ebx, 0, 8

add ecx, edx ;

jmp .drawSymbol

.end\_o:

ret

endp

; RECOGNITION OF DOS COMMANDS FROM KEYBOARD BUFFER <<MAKE CASE INSENSITIVE>> <<WITH ANOTHER END OF COMM AND LIST OF COMMS>>

proc findCommand uses esi edi edx

mov esi, commNames

sub esi, 2

mov eax, -1 ; No command

.start:

add esi, 2

mov edi, bufferInput

xor dx, dx

.goTrought:

cmp word[esi], 0 ; is it the end of a command

jz .isFind

cmp dx, [ptrBufferInput] ; is it the end of the buffer

jae .nextCommand

inc dx

cmpsw

jz .goTrought

sub edi, 2

.isFind:

inc eax

cmp word[edi], ' '

jz .end\_f

cmp word[edi], 0

jz .end\_f

.nextCommand:

cmp dword[esi], 0 ; If it's not an end of the list of commands

jz .noCommand

cmp word[esi], 0 ; If it's not an end of curr command

jz .start

add esi, 2

jmp .nextCommand

.noCommand:

stdcall loadProgram

ret

.end\_f:

mov [ptrBufferInput], dx

cmp eax, 14

jz .noCommand

mov edx, commTable

shl eax, 2

add edx, eax

mov edx, [edx]

call edx

ret

endp

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \* COMMANDS \*

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CD\_comm:

stdcall CD\_

ret

DIR\_comm:

stdcall DIR\_

ret

; CD

proc CD\_ uses eax edx ebx ecx edi esi

locals

operand du 101 dup(0)

endl

lea edi, [operand]

mov esi, bufferInput

mov ebx, esi

mov edx, edi

add si, word[ptrBufferInput]

add si, word[ptrBufferInput]

add si, 2

.gettingOperand:

movsw

cmp word[esi], 0

je .setDir

cmp edi, [edx+100]

je .setDir

cmp esi, [ebx+100]

jnz .gettingOperand

.setDir:

mov byte[edi], '\'

mov word[edi+1], 0

invoke SetCurrentDirectory, edx

cmp eax, 0

jne .moved

call outputError

jmp .end\_c

.moved:

mov ecx, 128

mov eax, '$'

mov edi, startString

rep stosb

invoke GetCurrentDirectory, 79, startString

invoke GetCurrentDirectory, 0, 0

sub eax, 2

mov [startPos], ax

mov byte[startString+eax\*2+2], '>'

stdcall outputString, startString, 1

.end\_c:

ret

endp

CLS\_comm:

push edi

push ecx

push eax

mov edi, btm\_03h.videoModeMem

mov ecx, bmSize

xor eax, eax

rep stosb

invoke InvalidateRect, [hWnd], 0, 0

mov [repaintFull], 1

mov [currPos], 0

mov [currLine], 0

pop eax

pop ecx

pop edi

ret

proc DIR\_ uses eax ebx edx edi

locals

operand du 101 dup(0)

fileSizeOutput du 50 dup(0)

hFind dd 0

endl

lea edi, [operand]

invoke GetCurrentDirectory, 99, edi

invoke GetCurrentDirectory, 0, 0

shl eax, 1

add edi, eax

sub edi, 2

mov byte[edi], '\'

mov byte[edi+1], 0

add edi, 2

mov byte[edi], '\*'

mov byte[edi+1], 0

lea ebx, [hFind]

lea ecx, [operand]

invoke FindFirstFile, ecx, ffd

mov [ebx], eax

.gettingFiles:

mov edx, [ffd.dwFileAttributes]

and edx, FILE\_ATTRIBUTE\_DIRECTORY

cmp edx, 0

je .notDir

stdcall outputString, ffd.cFileName, 1

stdcall outputString, \_dirOutput, 1

stdcall newLine

jmp .whilePart

.notDir:

stdcall outputString, ffd.cFileName, 1

stdcall outputString, \_fileOutput, 1

lea edx, [fileSizeOutput]

cmp [ffd.nFileSizeHigh], 0

je .onlyLowPart

cinvoke wsprintfW, edx, \_formatBytes, [ffd.nFileSizeHigh]

lea edx, [fileSizeOutput]

stdcall outputString, edx, 1

.onlyLowPart:

cinvoke wsprintfW, edx, \_formatBytes, [ffd.nFileSizeLow]

lea edx, [fileSizeOutput]

stdcall outputString, edx, 1

stdcall outputString, \_bytesOutput, 1

stdcall newLine

.whilePart:

lea ebx, [hFind]

invoke FindNextFile, [ebx], ffd

cmp eax, 0

jne .gettingFiles

mov [repaintFull], 1

ret

endp

outputError:

stdcall outputString, errorIndefComm, 1

stdcall newLine

stdcall newLine

mov [repaintFull], 1

ret

**ВЕДОМОСТЬ ДОКУМЕНТОВ**

| Обозначение | | | | Наименование | | | | | Дополнительные сведения | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | Текстовые документы | | | | |  | |
|  | | | |  | | | | |  | |
| БГУИР КП I- 40 01 01 009 ПЗ | | | | Пояснительная записка | | | | | 56 с. | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | | Графические документы | | | | |  | |
|  | | | |  | | | | |  | |
| ГУИР. 251003 009 СА | | | | Функция обновления состояния игрока | | | | | Формат А1 | |
|  | | | | Схема алгоритма | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  |  |  |  |  | БГУИР КП I- 40 01 01 009 ПЗ | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм | Лист | № докум. | Подп. | Дата | «Эмуляция работы операционной системы DOS»  Ведомость курсовой  работы | Литера | | | Лист | Листов |
| Разраб. | | Жук Я. С. |  | 26.12.23 | Т |  |  |  | 56 |
| Пров. | | Шостак Е.В. |  | 26.12.23 | Кафедра ПОИТ  гр. 251003 | | | | |