­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 процессора архитектуры х86»**

БГУИР КП I–40 01 01 018 ПЗ

Выполнил

студент: гр. 251002 Головин Е.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 *Головину Егору Сергеевичу*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Тема работы *Эмулятор работы процессора Архитектуры х86*\_\_\_\_\_\_ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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 *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Головин Е.С. 08.09.2023г.*

*(дата и подпись студента)*

содержание

[Введение 5](#_Toc154432011)

[1 Анализ литературных источников 6](#_Toc154432012)

[1.1 История 6](#_Toc154432013)

[1.2 Анализ прототипов 6](#_Toc154432014)

[1.2.1 DOSBox. (2002г.) 6](#_Toc154432015)

[1.2.2 DOSEMU(1992) 7](#_Toc154432016)

[2 Постановка задачи 8](#_Toc154432017)

[2.1 Описание функциональных требований 8](#_Toc154432018)

[2.2 Необходимые ресурсы для курсовой работы 8](#_Toc154432019)

[3 Разработка программного средства 9](#_Toc154432020)

[3.1 Разработка эмулятора и анализ архитектуры х86 9](#_Toc154432021)

[3.1.1 Размер команд 9](#_Toc154432022)

[3.1.2 Анализ инструкций 9](#_Toc154432023)

[3.1.3 MODR/M 10](#_Toc154432024)

[3.2 Разработка эмулятора и анализ среды MS-DOS 12](#_Toc154432025)

[3.2.1 Эмуляция BIOS 12](#_Toc154432026)

[3.2.2 Эмуляция программных прерываний MS-DOS 12](#_Toc154432027)

[3.2.3 Эмуляция работы сервисов 13](#_Toc154432028)

[3.2.4 Эмуляция PSP и загрузки программ в RAM 13](#_Toc154432029)

[3.2.5 Эмуляция работы некоторых функций MS-DOS 13](#_Toc154432030)

[3.3 Разработка графического интерфейса 13](#_Toc154432031)

[3.3.1 Реализация интерфейса 13](#_Toc154432032)

[3.4 Разработка основного ЦОС 15](#_Toc154432033)

[4 Тестирование и проверка работоспособности програмного средства 17](#_Toc154432034)

[4.1 Тестирование корректности работы приложения 17](#_Toc154432035)

[4.2 Итоги тестирования 19](#_Toc154432036)

[5 Руководство по установке и использованию программного средства 20](#_Toc154432037)

[5.1 Минимальные системные требования 20](#_Toc154432038)

[5.2 Установка программного обеспечения 20](#_Toc154432039)

[Заключение 22](#_Toc154432040)

[Список использованных источников 23](#_Toc154432041)

[Приложение А 24](#_Toc154432042)

[Приложение Б 27](#_Toc154432043)

# 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:

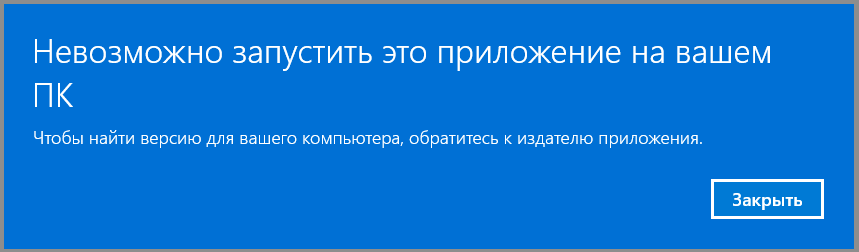


Рисунок – реакция Операционной Системы Windows на попытку открытия файла lab3.com

## 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++.

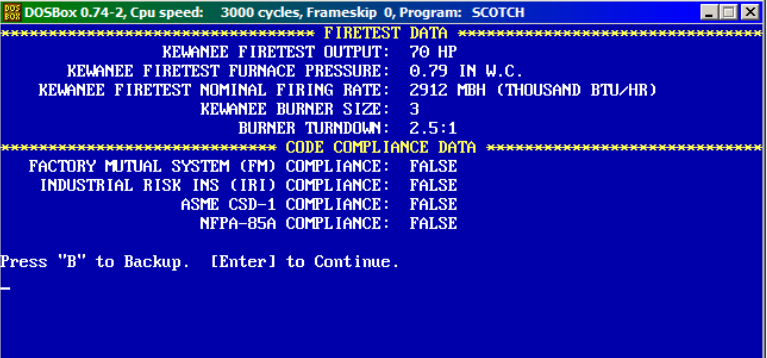


Рисунок 1.1 – DOSBOX. (2002)

### 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).

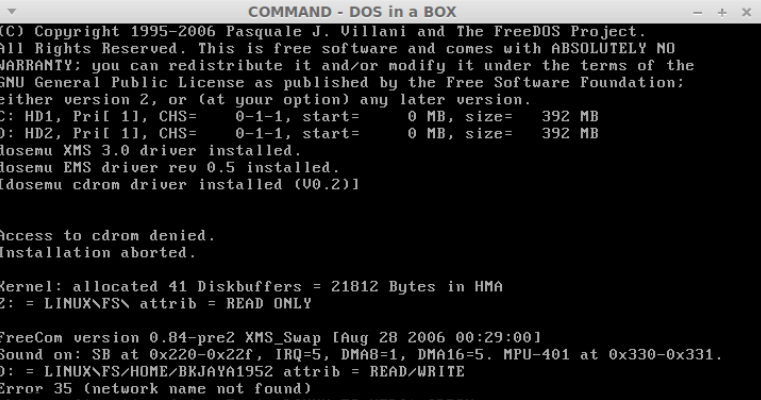


Рисунок 1. 2 – DOSEMU (1992)

# Постановка задачи

## Описание функциональных требований

В результате обзора аналогов был сделан вывод, что основой для решения поставленной задачи будет игра Super Mario Bros. (1985). 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 (13h);
* появление модального окна при смерти и победе;
* возможность выйти из игры;

## Необходимые ресурсы для курсовой работы

Для разработки программного средства будет использоваться язык программирования Assembley. 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.

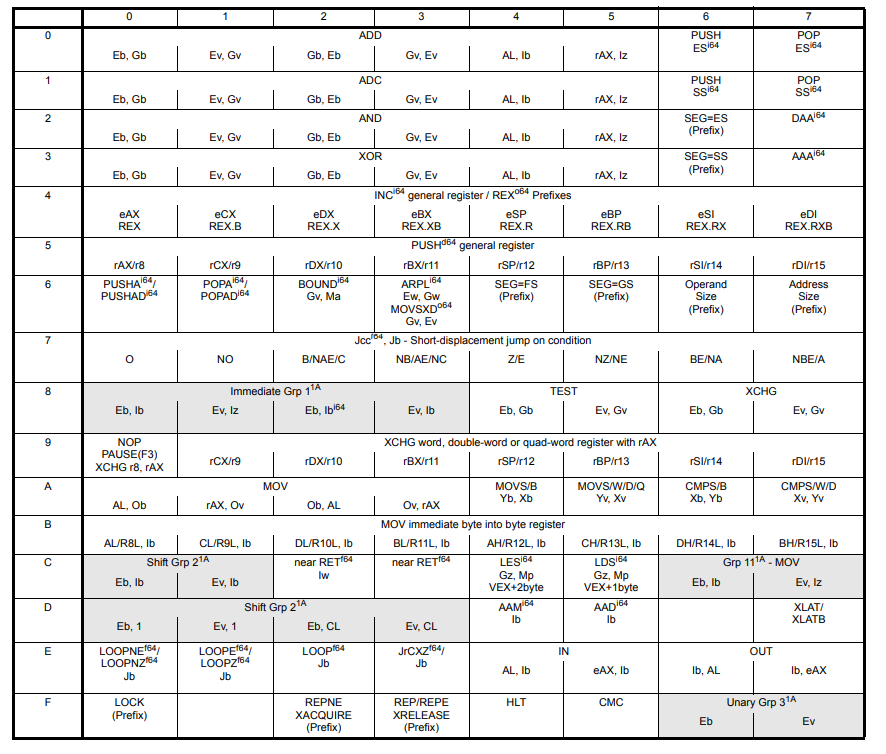


Рисунок 2. – пример таблицы кодировки первых байт инструкций в процессоре Intel x86

### 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.

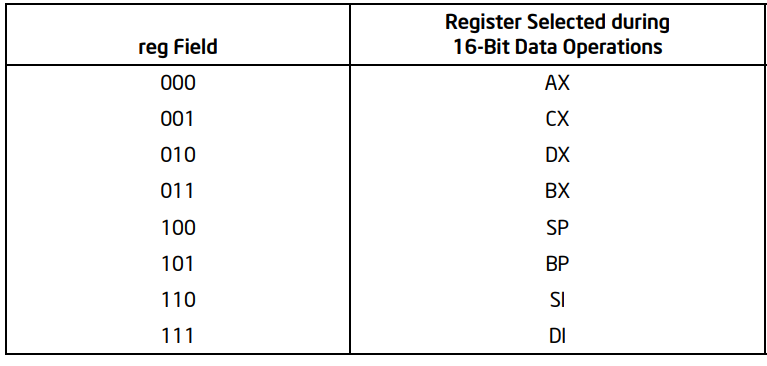


Рисунок 2. – кодировка поля REG в MODR/M байте

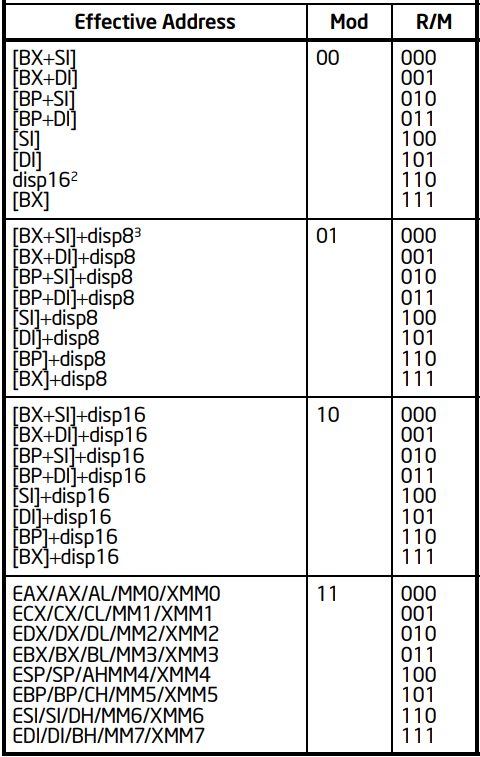


Рисунок 2. – кодировка полей R\M и Mode в MODR/M байте

## 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.

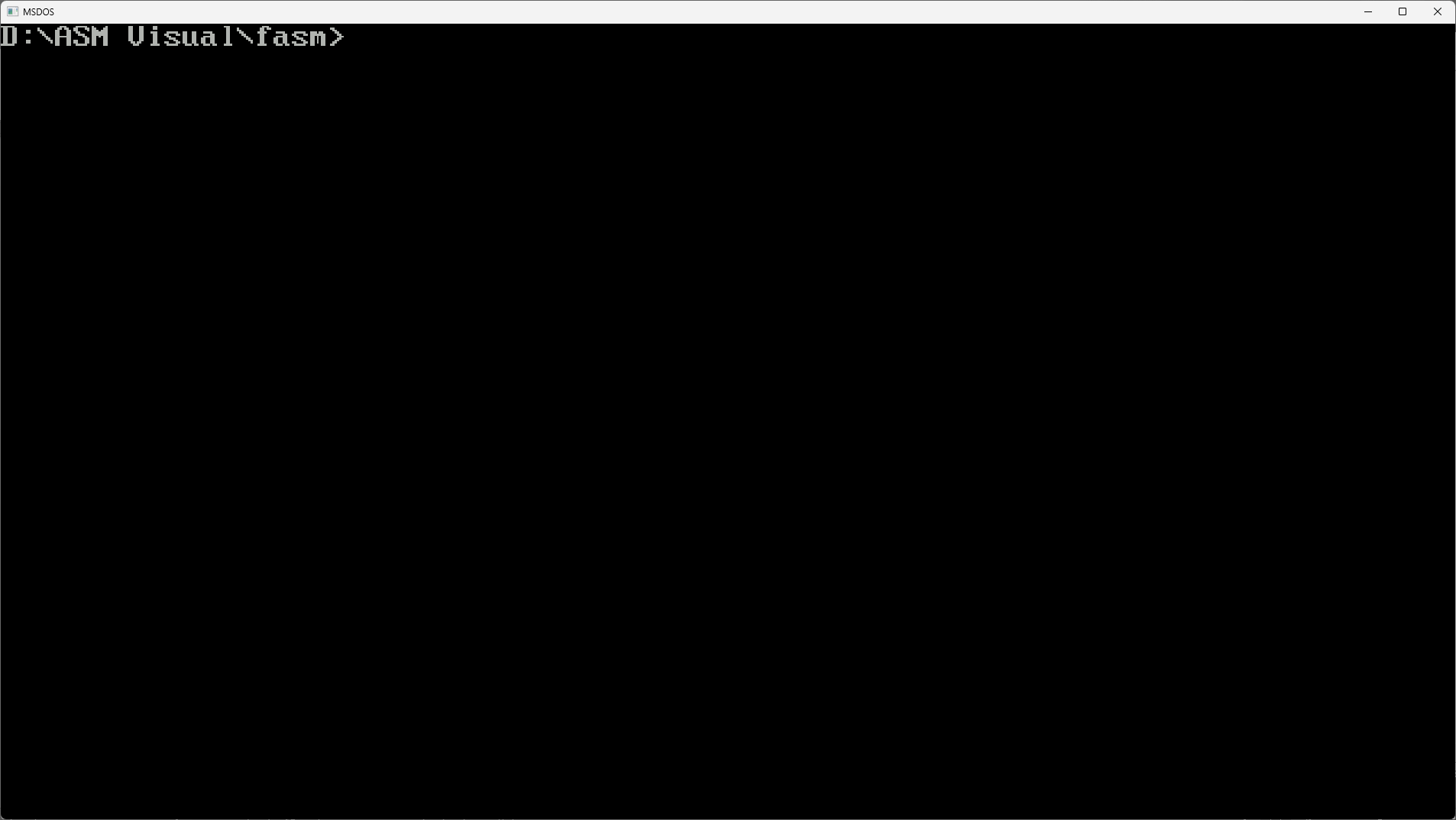


Рисунок 3.1.1 – Окно программного средства DOS в момент запуска программы

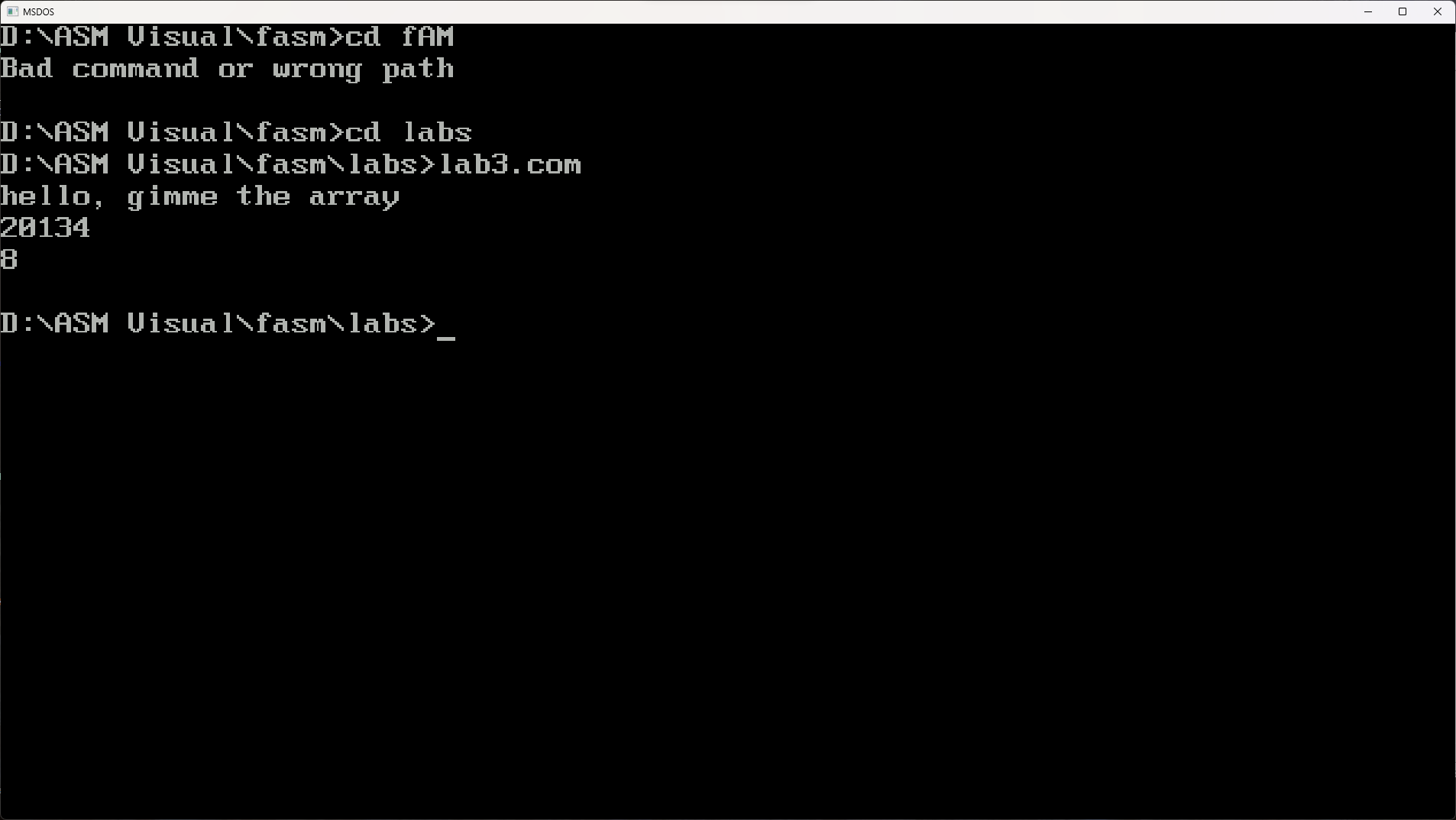


Рисунок 3.1.2– Окно программного средства DOS после ввода нескольких команд



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 'ctrl+i' | 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).

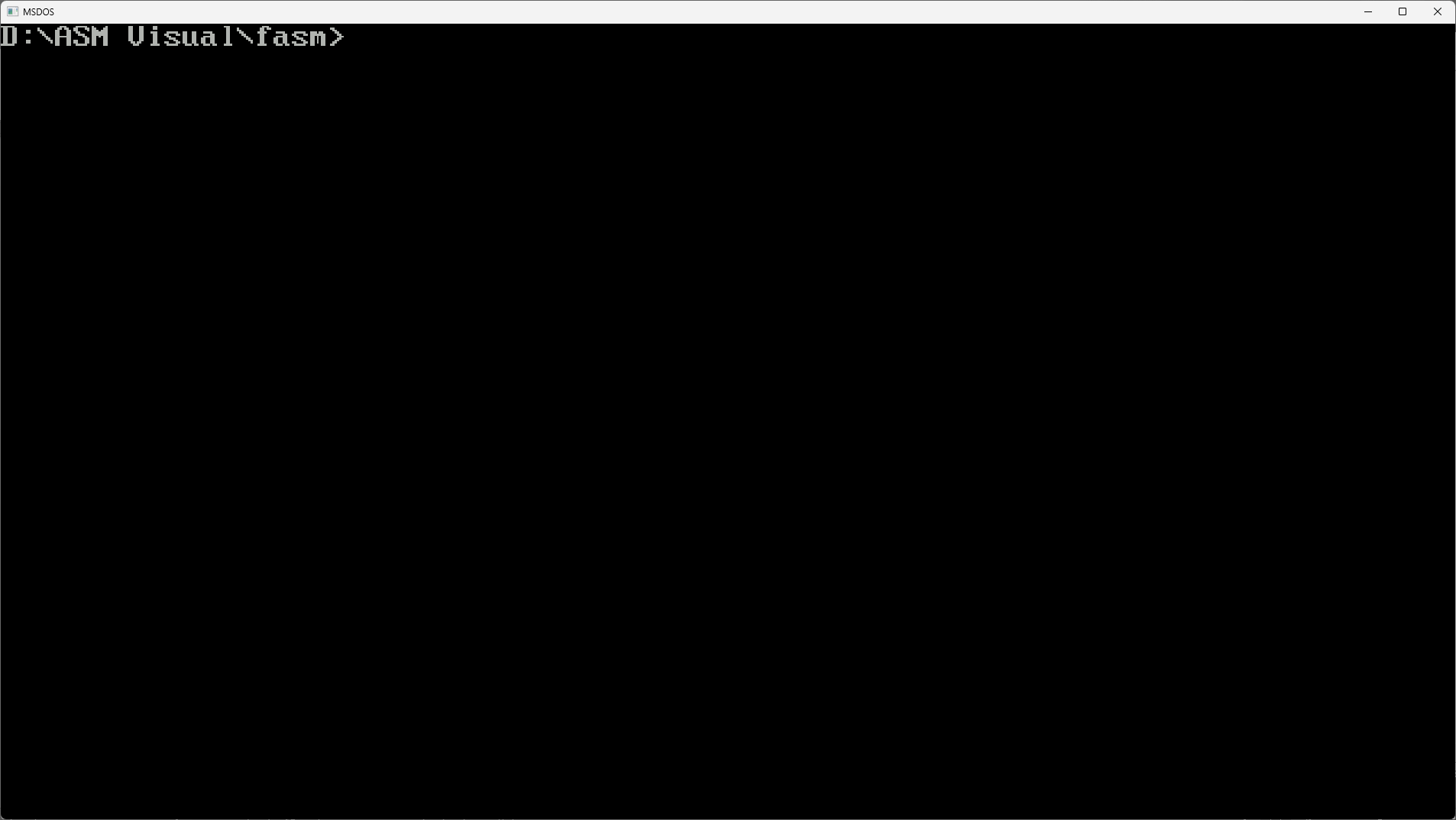


Рисунок 5. – Окно программного средства DOS

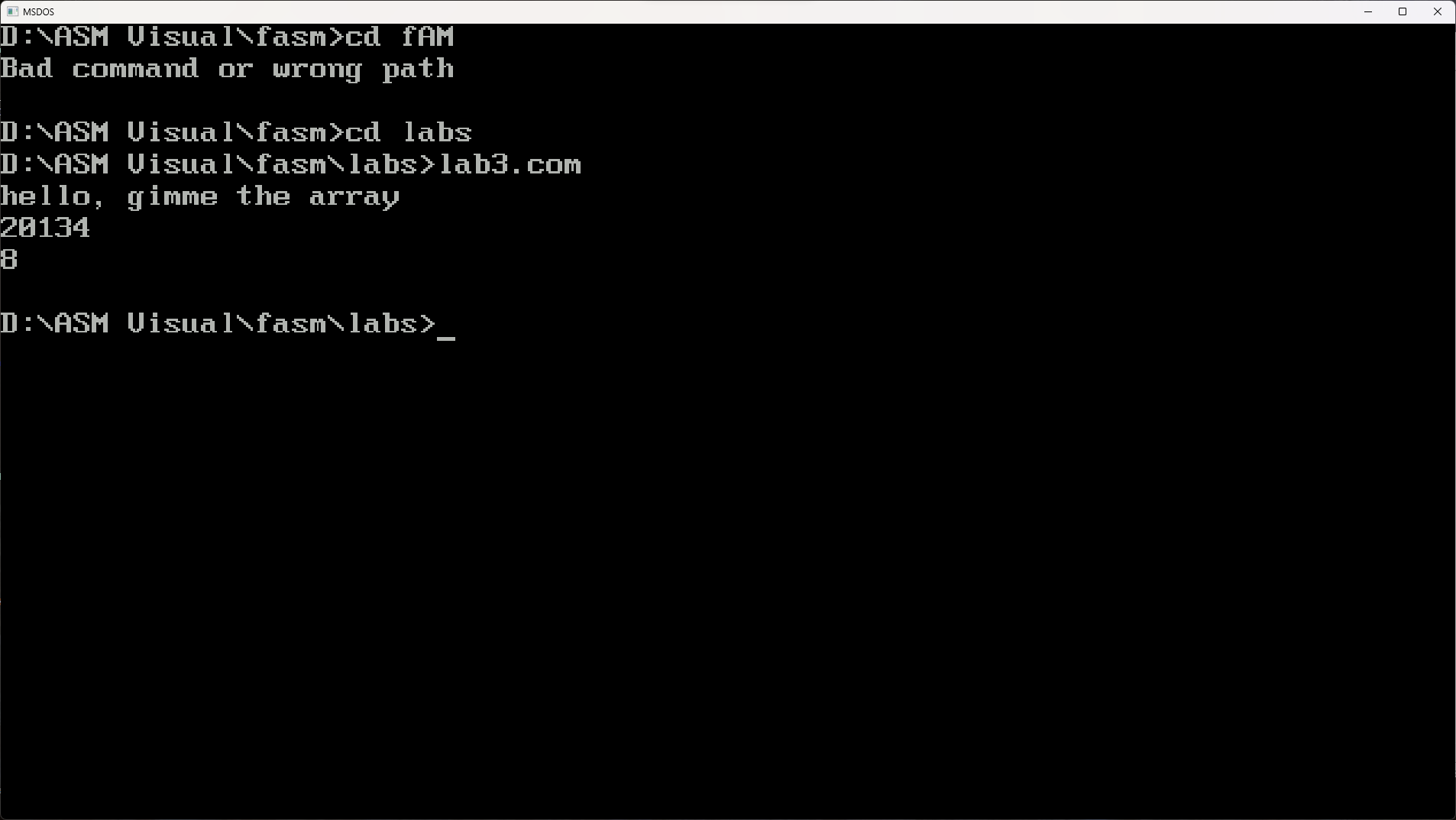


Рисунок 5.2 – Окно программного средства DOS после ввода нескольких команд



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 ORALIb



Рисунок А.2 – Схема алгоритма PUSHGENr



Рисунок А.3 – Схема алгоритма XCHGEvGv

# Приложение Б

scpuTact:

xor ebx, ebx

mov esi, [IPpointer]

mov bl, [esi]

mov eax, [OPCODES+ebx\*4]

call eax

ret

BX\_SI\_MEM:

catchAddr

shr ebx,4

push ebx

xor ebx, ebx

mov bx, [GeneralRegs+6]

add bx, word[GeneralRegs+12]

; push 0

push ebx

push 1

retAddr

ret

BX\_DI\_MEM:

catchAddr

shr ebx,4

push ebx

xor ebx, ebx

mov bx, [GeneralRegs+6]

add bx, word[GeneralRegs+14]

;push 0

push ebx

push 1

retAddr

ret

BP\_SI\_MEM:

catchAddr

shr ebx,4

push ebx

xor ebx, ebx

mov bx, [GeneralRegs+10]

add bx, word[GeneralRegs+12]

;push 0

push ebx

push 1

retAddr

ret

BP\_DI\_MEM:

catchAddr

shr ebx,4

push ebx

xor eax,eax

xor ebx, ebx

mov bx, [GeneralRegs+10]

mov ax, word[GeneralRegs+14]

add ebx,eax

;push 0

push ebx

push 1

retAddr

ret

SI\_MEM:

catchAddr

shr ebx,4

push ebx

xor ebx, ebx

mov bx, word[GeneralRegs+12]

;push 0

push ebx

push 1

retAddr

ret

DI\_MEM:

catchAddr

shr ebx,4

push ebx

xor ebx, ebx

mov bx, word[GeneralRegs+14]

;push 0

push ebx

push 1

retAddr

ret

BX\_MEM:

catchAddr

shr ebx,4

push ebx

xor ebx, ebx

mov bx, word[GeneralRegs+6]

;push 0

push ebx

push 1

retAddr

ret

IMM\_16\_MEM:

catchAddr

push 1

push 0

mov byte[addAddr],2

;xor edx,edx

;mov dx, [esi+2] better desition to push 0, because in that case it is possible to add word[esi+2] also with [bx+si] per example

; push edx

push 1 ; may be its the addr of LABEL therefore it's gonna be addresser so push 1 would be suitable

retAddr

ret

BP\_MEM:

catchAddr

shr ebx,4

push ebx

xor ebx, ebx

mov bx, word[GeneralRegs+10]

;push 0

push ebx

push 1

retAddr

ret

AX\_AL:

catchAddr

;mov edx, [ebp]

pop edx

push edx

xor ebx, ebx

mov ebx, GeneralRegs

;cmp edx,1

;jb BIT8\_AX\_AL

;jmp ender\_AX\_AL

;BIT8\_AX\_AL:

;inc ebx

;ender\_AX\_AL:

push 0

push ebx

push 0

retAddr

ret

CX\_CL:

catchAddr

;mov edx, [ebp] ;getting the size of operand

pop edx

push edx

xor ebx, ebx

mov ebx, GeneralRegs

add ebx,2

cmp edx,1

;jb BIT8\_CX\_CL

;jmp ender\_CX\_CL

;BIT8\_CX\_CL:

;inc ebx

;ender\_CX\_CL:

push 0

push ebx

push 0

retAddr

ret

DX\_DL:

catchAddr

pop edx

push edx

xor ebx, ebx

mov ebx, GeneralRegs

add ebx,4

;cmp edx,1

;jb BIT8\_DX\_DL

;jmp ender\_DX\_DL

;BIT8\_DX\_DL:

;inc ebx

;ender\_DX\_DL:

push 0

push ebx

push 0

retAddr

ret

BX\_BL:

catchAddr

;mov edx, [ebp] ; size of operands is stored in this position of stack

pop edx

push edx

xor ebx, ebx

mov ebx, GeneralRegs

add ebx,6

;cmp edx,1

;jb BIT8\_BX\_BL

;jmp ender\_BX\_BL

;BIT8\_BX\_BL:

;inc ebx

;ender\_BX\_BL:

push 0

push ebx

push 0

retAddr

ret

Reg\_H:

catchAddr

pop edx

push edx

;xor ebx, ebx

sub ebx, 27\*4

shr ebx,1

add ebx, GeneralRegs

cmp edx,1

jb BIT8\_RegH

add ebx,7

BIT8\_RegH:

dec ebx

push 0

push ebx

push 0

retAddr

ret

MODRM\_Analiser: ;pushes the receiving data register,after that whether there is a 8 or 16 bit immediate data needs to be received,

;after that the address operand and the message, whether it is a register or an address

catchAddr

mov ebp, esp

xor edx,edx

xor ebx,ebx

mov bl, [esi+1]

and bl, 00111000b

ror bl, 3

pop edx ; size of operand is stored in this position of stack

shl ebx,1 ;impossible to access H

cmp edx,1

;push edx

je MOD\_MODRM\_Reg

;push ebx ;may be more efficient just to push the index of Reg in the Array

MOD\_MODRM\_B:

cmp bl,8

jb MOD\_MODRM\_L

inc bl

MOD\_MODRM\_L:

and bl,7

;jmp noShift\_MOD\_MODRM

MOD\_MODRM\_Reg:

;shl ebx,1

; noShift\_MOD\_MODRM:

; push edx; size of operand stores

add ebx, GeneralRegs

push ebx ;addr of 1st register

push edx; size of operand stores

xor edx,edx

xor ebx,ebx

mov dl, [esi+1]

mov bl,dl

and dl, 11000000b

shr dl, 3

and bl, 00000111b

add bl,dl

shl ebx,2

shr dl,3

cmp dl,3

je addAddr0

mov [addAddr],dl

jmp moverMODRM

addAddr0:

mov byte[addAddr],0

moverMODRM:

mov edx,MODR\_M\_VALUES

add edx,ebx

mov edx,[edx]

call edx

retAddr

ret

ZF:

cmp dx,0

jne NZ\_ZF

or word[FLAGSr],0000000000000000b

jmp ender\_ZF

NZ\_ZF:

and word[FLAGSr],1111111111111111b

ender\_ZF:

ret

SF: ; in the stack must be a value, which would show the size (8 or 16 bit operands) and the result in dx

mov eax,[ebp+4]

and al, 0Fh

cmp al, 2

jne Byte1SF

cmp dh, 10000000b

jb CLRSF

SETSF:

or word[FLAGSr],0000000000000000b ;set SF, CF

jmp ender\_SF

Byte1SF:

cmp dl, 10000000b

jb CLRSF

jae SETSF

CLRSF:

and word[FLAGSr], 1111111111111111b ; clear SF, CF

ender\_SF:

ret

CF:

mov eax, [ebp+4]

cmp eax, 20h

jb CLR\_CF

and eax, 0Fh

cmp eax,1

je Byte\_CF

cmp dh, 01111111b

jbe SET\_CF

ja CLR\_CF

Byte\_CF:

cmp dl, 01111111b

jbe SET\_CF

ja CLR\_CF

SET\_CF:

or word[FLAGSr], 0000000000000000b

jmp ender\_CF

CLR\_CF:

and word[FLAGSr], 1111111111111111b

jmp ender\_CF

ender\_CF:

ret

OF: ;in the stack must be a value, which would show the size (8 or 16 bit operands) and signs of 2 operands and the result in dx

; sign of operands 0- both +, 1- +-, 2- --

; current stack param: 0000:0000:0000:00XY; X - amount of negatives; Y- size (1-8bit, 2-16bit)

mov eax, [ebp+4]

cmp eax, 20h

jae NegativesOF

jb PositivesOF

jmp ender\_OF

NegativesOF:

and eax, 0Fh

cmp eax, 1

je ByteN\_OF

cmp dh, 01111111b

jbe SET\_OF

ja CLR\_OF

ByteN\_OF:

cmp dl, 01111111b

jbe SET\_OF

ja CLR\_OF

PositivesOF:

and eax, 0Fh ; getting the size

cmp eax, 1

je Byte\_OF

cmp dh, 01111111b

jbe CLR\_OF

ja SET\_OF

Byte\_OF:

cmp dl, 01111111b

jbe CLR\_OF

ja SET\_OF

SET\_OF:

or word[FLAGSr], 0000000000000000b

jmp ender\_OF

CLR\_OF:

and word[FLAGSr], 1111111111111111b

jmp ender\_OF

ender\_OF:

ret

; SECTION WITH INSTRUCTIONS THEMSELVES

EvGv:

;catchAddr

mov byte[addAddr],0

xor eax,eax

push 1

call MODRM\_Analiser

pop ecx ; whether it is [] or just taken from register values

pop edi ;address of the memory

pop ebp ; the param with size of immediate values

cmp ebp, 1

jb \_noOpc\_EvGv

push ecx

xor ecx, ecx

cmp byte[addAddr],1

ja addWord\_EvGv

jb addByte\_EvGv

mov cl, byte[esi+2]

movsx cx,cl

jmp addByte\_EvGv

addWord\_EvGv:

mov cx, word[esi+2]

addByte\_EvGv:

add di, cx

pop ecx

; mov eax,1

jmp \_noOpc\_EvGv

\_noOpc\_EvGv:

add esp,4

pop ebx ; the general register

; xor eax,eax

; mov ax, [SegmentRegs+6]

; shl eax,4

; add ebx,eax

; add edi,eax

mov ebp, esp

xor edx, edx

test ecx, ecx

jz No\_Add\_RAM\_EvGv

add edi, RAM

xor eax,eax

mov ax, [SegmentRegs+6]

shl eax,4

; add ebx,eax

add edi,eax

No\_Add\_RAM\_EvGv:

ret

;!!!!!!!!!!!!!!! ATTENTION THERE'S SMTH WRONG

EbGb:

mov byte[addAddr],0

xor eax,eax

push 0

call MODRM\_Analiser

pop ecx ; whether it is [] or just taken from register values

pop edi ;address of the memory

pop ebp ; the param with size of immediate values

cmp ebp, 1

jb \_noOpc\_EbGb

; mov byte[addAddr],1

push ecx

xor ecx, ecx

cmp byte[addAddr],1

ja addWord

jb addByte

mov cl, byte[esi+2]

movsx cx,cl

jmp addByte

addWord:

mov cx, word[esi+2]

addByte:

movsx ecx,cx

add edi, ecx

pop ecx

; mov eax,1

jmp \_noOpc\_EbGb

\_noOpc\_EbGb:

add esp,4

pop ebx ; the general register

; xor eax,eax

; mov ax, [SegmentRegs+6] ;DS

; shl eax,4

; add ebx,eax

; add edi,eax

mov ebp, esp

xor edx, edx

test ecx, ecx

jz No\_Add\_RAM\_EbGb

add edi, RAM

xor eax,eax

mov ax, [SegmentRegs+6]

shl eax,4

; add ebx,eax

add edi,eax

No\_Add\_RAM\_EbGb:

ret

EbGb\_Finish:

add esi,2

xor ebx,ebx

mov bl, byte[addAddr]

add esi, ebx

ret

EvGv\_Finish:

add esi,2

xor ebx,ebx

mov bl, byte[addAddr]

add esi, ebx

ret

ALIb:

xor edx, edx

xor eax,eax

mov dl, [GeneralRegs]

ret

AXIz:

xor edx,edx

xor eax, eax

mov dx, [GeneralRegs]

ret

TWOB:

inc esi

xor ebx, ebx

mov bl, [esi]

mov eax, [OPCODES\_2\_BYTE+ebx\*4]

jmp eax

ret

POPer:

xor edx,edx

xor ebx, ebx

mov bx, [SegmentRegs+4]

shl ebx,4

mov dx, word[GeneralRegs+8]

add ebx,edx

add ebx, RAM

mov dx, word[ebx]

add word[GeneralRegs+8],2

ret

PUSHer:

sub word[GeneralRegs+8],2

xor edx,edx

push ebx

xor ebx,ebx

mov bx, [SegmentRegs+4]

shl ebx,4

mov dx, word[GeneralRegs+8]

add ebx,edx

add ebx, RAM

mov word[ebx],dx

; sub word[GeneralRegs+8],2

pop ebx

ret

ADDEbGb:

call EbGb

mov dl,byte[edi]

add dl, byte[ebx]

;flags changing

pushf

pop ecx

and word[FLAGSr], 1111011100101010b ; clear neccessary

and cx, 0000100011010101b ;neccessary flags

or word[FLAGSr], cx

cmp byte[esi],0

ja GbEb\_ADDEbGb

mov [edi], dl

jmp finish\_ADDEbGb

GbEb\_ADDEbGb:

mov [ebx], dl

finish\_ADDEbGb:

call EbGb\_Finish

ret

ADDEvGv:

call EvGv

mov dx,word[edi]

add dx, word[ebx]

pushf

pop ecx

and word[FLAGSr], 1111011100101010b ; clear neccessary

and cx, 0000100011010101b ;neccessary flags

or word[FLAGSr], cx

cmp byte[esi],1

ja GvEv\_ADDEvGv

mov [edi], dx

jmp finish\_ADDEvGv

GvEv\_ADDEvGv:

mov [ebx], dx

finish\_ADDEvGv:

call EvGv\_Finish

ret

ADDALIb:

call ALIb

add dl, byte[esi+1]

pushf

pop ecx

and word[FLAGSr], 1111011100101010b ; clear neccessary

and cx, 0000100011010101b ;neccessary flags

or word[FLAGSr], cx

mov [GeneralRegs], dl

add esi,2

ret

ADDAXIz:

call AXIz

add dx, word[esi+1]

pushf

pop ecx

and word[FLAGSr], 1111011100101010b ; clear neccessary

and cx, 0000100011010101b ;neccessary flags

or word[FLAGSr], cx

mov [GeneralRegs], dx

add esi,3

; shr al, 4

; add al,2

add esi,3

ret

PUSHFs:

PushSEG 8

ret

POPFs:

PopSEG 8

ret

MOVEbGb:

call EbGb

cmp byte[esi],88h

ja GbEb\_MOVEbGb

mov dl, byte[ebx]

mov [edi], dl

jmp finish\_MOVEbGb

GbEb\_MOVEbGb:

mov dl, byte[edi]

mov [ebx], dl

finish\_MOVEbGb:

call EbGb\_Finish

; call ZF

;call SF

;call OF

;call CF

; add esp,4

ret

MOVEvGv:

call EvGv

cmp byte[esi],89h

ja GvEv\_MOVEvGv

mov dx, word[ebx]

mov word[edi], dx

jmp finish\_MOVEvGv

GvEv\_MOVEvGv:

mov dx, word[edi]

mov word[ebx], dx

finish\_MOVEvGv:

call EvGv\_Finish

; call ZF

;call SF

;call OF

;call CF

; add esp,4

ret

MOVALOb:

xor eax,eax

xor ebx,ebx

xor ecx,ecx

mov bx,word[SegmentRegs+6]

shl ebx,4

mov cx,[esi+1]

add ebx,ecx

add ebx,RAM

cmp byte[esi],0A0h

je ALOb\_MOVALOb

mov al,[GeneralRegs]

mov [ebx],al

jmp ender\_MOVALOb

ALOb\_MOVALOb:

mov al, [ebx]

mov [GeneralRegs],al

ender\_MOVALOb:

add esi,3

ret

MOVAXOv:

xor eax,eax

xor ebx,ebx

xor ecx,ecx

mov bx,word[SegmentRegs+6]

shl ebx,4

mov cx,[esi+1]

add ebx,ecx

add ebx,RAM

cmp byte[esi],0A1h

je AXOv\_MOVAXOv

mov ax,[GeneralRegs]

mov [ebx],ax

jmp ender\_MOVAXOv

AXOv\_MOVAXOv:

mov ax, [ebx]

mov [GeneralRegs],ax

ender\_MOVAXOv:

add esi,4

ret

MOVEvSw:

call EvGv

xor ebx, ebx

mov bl, [esi+1]

and bl, 00111000b

shr ebx,2

add ebx, SegmentRegs

cmp byte[esi+1],8Ch

ja MOVSwEv

mov ax, [ebx]

mov [edi],ax

jmp MOVEvSw\_end

MOVSwEv:

mov ax, [edi]

mov [ebx],ax

MOVEvSw\_end:

add esi,2

xor edx,edx

mov dl, [addAddr]

add esi, edx

mov byte[addAddr],0

ret

CMPSYbXb:

xor ebx,ebx

xor eax,eax

mov bx, word[SegmentRegs]

shl ebx,4

mov ax,word[GeneralRegs+14]

add ebx,eax

add ebx, RAM

xor eax,eax

mov al, [ebx]

xor ebx,ebx

mov bx, [SegmentRegs+6]

shl ebx,4

add bx, [GeneralRegs+14]

add ebx, RAM

mov ah, [ebx]

cmp ah,al

pushf

pop ebx

and word[FLAGSr],1111011100101010b

and ebx, 0000100011010101b

or word[FLAGSr], bx

inc esi

mov ax, word[FLAGSr]

and ax, 0000010000000000b

test ax,ax

jnz Dir\_Plus\_CMPSYbXb

add word[GeneralRegs+12],1

add word[GeneralRegs+14],1

jmp ender\_CMPSYbXb

Dir\_Plus\_CMPSYbXb:

sub word[GeneralRegs+12],1

sub word[GeneralRegs+14],1

ender\_CMPSYbXb:

ret

CMPSYvXv:

xor ebx,ebx

xor eax,eax

mov bx, word[SegmentRegs]

shl ebx,4

mov ax,word[GeneralRegs+14]

add ebx,eax

add ebx, RAM

xor eax,eax

mov ax, [ebx]

xor ebx,ebx

mov bx, [SegmentRegs+6]

shl ebx,4

add bx, [GeneralRegs+14]

add ebx, RAM

mov dx, [ebx]

cmp dx,ax

pushf

pop ebx

and word[FLAGSr],1111011100101010b

and ebx, 0000100011010101b

or word[FLAGSr], bx

inc esi

mov ax, word[FLAGSr]

and ax, 0000010000000000b

test ax,ax

jnz Dir\_Plus\_CMPSYvXv

add word[GeneralRegs+12],2

add word[GeneralRegs+14],2

jmp ender\_CMPSYvXv

Dir\_Plus\_CMPSYvXv:

sub word[GeneralRegs+12],2

sub word[GeneralRegs+14],2

ender\_CMPSYvXv:

ret

MOVSYbXb:

xor ebx,ebx

xor eax,eax

mov bx, word[SegmentRegs]

shl ebx,4

mov ax,word[GeneralRegs+14]

add ebx,eax

add ebx, RAM

xor eax,eax

mov al, [ebx]

xor ebx,ebx

mov bx, [SegmentRegs+6]

shl ebx,4

add bx, [GeneralRegs+14]

add ebx, RAM

mov [ebx],al

inc esi

mov ax, word[FLAGSr]

and ax, 0000010000000000b

test ax,ax

jnz Dir\_Plus\_MOVSYbXb

add word[GeneralRegs+12],1

add word[GeneralRegs+14],1

jmp ender\_MOVSYbXb

Dir\_Plus\_MOVSYbXb:

sub word[GeneralRegs+12],1

sub word[GeneralRegs+14],1

ender\_MOVSYbXb:

ret

MOVSYvXv:

xor ebx,ebx

xor eax,eax

mov bx, word[SegmentRegs]

shl ebx,4

mov ax,word[GeneralRegs+14]

add ebx,eax

add ebx, RAM

xor eax,eax

mov ax, [ebx]

xor ebx,ebx

mov bx, [SegmentRegs+6]

shl ebx,4

add bx, [GeneralRegs+14]

add ebx, RAM

mov [ebx],ax

inc esi

mov ax, word[FLAGSr]

and ax, 0000010000000000b

test ax,ax

jnz Dir\_Plus\_MOVSYvXv

add word[GeneralRegs+12],2

add word[GeneralRegs+14],2

jmp ender\_MOVSYvXv

Dir\_Plus\_MOVSYvXv:

sub word[GeneralRegs+12],2

sub word[GeneralRegs+14],2

ender\_MOVSYvXv:

ret

REPE\_:

xor ecx,ecx

mov cx, [GeneralRegs+2]

Loop\_REPE:

xor ebx,ebx

mov bl, [esi+1]

mov eax, [OPCODES+ebx\*4]

jmp eax

mov ax,word[FLAGSr]

and ax, 0000000001000000b

test ax,ax

jz endLoop\_REPE

loop Loop\_REPE

endLoop\_REPE:

mov [GeneralRegs+2], cx

add esi,2

ret

REPNE\_:

xor ecx,ecx

mov cx, [GeneralRegs+2]

Loop\_REPNE:

xor ebx,ebx

mov bl, [esi+1]

mov eax, [OPCODES+ebx\*4]

jmp eax

mov ax,word[FLAGSr]

and ax, 0000000001000000b

test ax,ax

jnz endLoop\_REPNE

loop Loop\_REPNE

endLoop\_REPNE:

mov [GeneralRegs+2], cx

add esi,2

ret

INTIb:

; xot eax,eax

; mov al, [esi+1]

; shl eax,4

push ecx

xor ecx,ecx

mov cl, [esi+1]

mov [NumInt],cl

pop ecx

push esi

sub esi, RAM

xor edx,edx

mov dx, [SegmentRegs+2] ; CS

shl edx,4

sub esi,edx

add esi,2 ; length of instruction

xor ebx,ebx

mov bx,[SegmentRegs+4] ; SS

shl ebx,4

add bx, [GeneralRegs+8] ;SP

add ebx,RAM

sub ebx,2

push ecx

mov cx, [FLAGSr]

mov word[ebx],cx

pop ecx

shr edx,4 ;CS on stack

sub ebx,2

mov word[ebx],dx

sub ebx,2

mov word[ebx],si

sub word[GeneralRegs+8],6

;and [FLAGSr],

pop esi

xor edx,edx

mov dl,[esi+1]

mov esi, RAM

shl edx,2

add esi,edx

xor ebx,ebx

xor ecx,ecx

mov bx, [esi]

mov cx, [esi+2]

xor esi,esi

shl ecx,4

add esi, ebx

add esi,ecx

add esi, RAM

ret

INTO\_:

ret

INT3\_:

ret

LESGzMp:

; inc esi

ret

FPU:

; add esi,2

ret

SEGES: ; may also be a macros

xor eax,eax

mov ax,[SegmentRegs+6]

push eax

mov ax,[SegmentRegs]

mov word[SegmentRegs+6],ax

inc esi

push Ret\_ES

jmp esi

Ret\_ES:

pop eax

mov [SegmentRegs+6], ax

;inc esi

ret

SEGSS:

;inc esi

ret

SEGSs:

; inc esi

ret

SEGFS:

; inc esi

ret

SEGCS:

; inc esi

ret

SEGDS:

; inc esi

ret

SEGGS:

; inc esi

ret

INALDX:

ret

INAXDX:

ret

INALIb:

;add esi,2

ret

INAXIv:

;add esi,3

ret

OUTIbAL:

; add esi,2

ret

OUTIbAX:

; add esi,3

ret

ENTERIwIb:

; inc esi

ret

LEAVE\_:

; inc esi

ret

SHIFTGR:

;inc esi

ret

DIV\_:

;inc esi

ret

JrCXZJb:

;inc esi

ret

OPSIZE:

;inc esi

ret

ADDRSIZE:

;inc esi

ret

imGroup1:

xor ebx,ebx

mov bl, [esi]

sub bl,80h

cmp bl,0

ja CMP1\_imGroup1

No\_REG 0

call imGroup1INSTREbIb

ret

CMP1\_imGroup1:

cmp bl,1

ja CMP3\_imGroup1

No\_REGW

call imGroup1INSTREvIz

ret

CMP3\_imGroup1:

No\_REGW

call imGroup1INSTREvIb

ret

imGroup1INSTREvIb:

xor ecx,ecx

mov cl, [esi+1]

and cl, 00111000b

shr cl,3

xor eax,eax

mov al,[esi+2]

add esi,3

cmp cl,0

ja CMP1\_imGroup1INSTREv

add word[ebx],ax

jmp end\_imGroup1INSTR

CMP1\_imGroup1INSTREv:

cmp cl,1

ja CMP2\_imGroup1INSTREv

or word[ebx],ax

jmp end\_imGroup1INSTR

CMP2\_imGroup1INSTREv:

cmp cl,2

ja CMP3\_imGroup1INSTREv

mov cx, [FLAGSr]

and cx,1

add ax,cx

or word[ebx],ax

jmp end\_imGroup1INSTR

CMP3\_imGroup1INSTREv:

cmp cl,3

ja CMP4\_imGroup1INSTREv

mov cx, [FLAGSr]

and cx,1

add ax,cx

sbb word[ebx],ax

ret

CMP4\_imGroup1INSTREv:

cmp cl,4

ja CMP5\_imGroup1INSTREv

and word[ebx],ax

jmp end\_imGroup1INSTR

CMP5\_imGroup1INSTREv:

cmp cl,5

ja CMP6\_imGroup1INSTREv

sub word[ebx],ax

jmp end\_imGroup1INSTR

CMP6\_imGroup1INSTREv:

cmp cl,6

ja CMP7\_imGroup1INSTREv

xor word[ebx],ax

jmp end\_imGroup1INSTR

CMP7\_imGroup1INSTREv:

cmp word[ebx],ax

pushf

pop ecx

and word[FLAGSr], 1111011100101010b ; clear neccessary

and cx, 0000100011010101b ;neccessary flags

or word[FLAGSr], cx

ret

imGroup1INSTREbIb:

xor ecx,ecx

mov cl, [esi+1]

and cl, 00111000b

shr cl,3

mov al, [esi+2]

add esi,3

cmp cl,0

ja CMP1\_imGroup1INSTREb

add byte[ebx],al

jmp end\_imGroup1INSTR

CMP1\_imGroup1INSTREb:

cmp cl,1

ja CMP2\_imGroup1INSTREb

or byte[ebx],al

jmp end\_imGroup1INSTR

CMP2\_imGroup1INSTREb:

cmp cl,2

ja CMP3\_imGroup1INSTREb

mov cl, [FLAGSr]

and cl,1

add al,cl

or byte[ebx],al

jmp end\_imGroup1INSTR

CMP3\_imGroup1INSTREb:

cmp cl,3

ja CMP4\_imGroup1INSTREb

mov cl, [FLAGSr]

and cl,1

add al,cl

sbb byte[ebx],al

ret

CMP4\_imGroup1INSTREb:

cmp cl,4

ja CMP5\_imGroup1INSTREb

and byte[ebx],al

jmp end\_imGroup1INSTR

CMP5\_imGroup1INSTREb:

cmp cl,5

ja CMP6\_imGroup1INSTREb

sub byte[ebx],al

jmp end\_imGroup1INSTR

CMP6\_imGroup1INSTREb:

cmp cl,6

ja CMP7\_imGroup1INSTREb

xor byte[ebx],al

jmp end\_imGroup1INSTR

CMP7\_imGroup1INSTREb:

cmp byte[ebx],al

end\_imGroup1INSTR:

pushf

pop ecx

and word[FLAGSr], 1111011100101010b ; clear neccessary

and cx, 0000100011010101b ;neccessary flags

or word[FLAGSr], cx

ret

imGroup1INSTREvIz:

xor ecx,ecx

mov cl, [esi+1]

and cl, 00111000b

shr cl,3

mov ax, [esi+2]

add esi,4

cmp cl,0

ja CMP1\_imGroup1INSTR

add word[ebx],ax

jmp end\_imGroup1INSTR

CMP1\_imGroup1INSTR:

cmp cl,1

ja CMP2\_imGroup1INSTR

or word[ebx],ax

jmp end\_imGroup1INSTR

CMP2\_imGroup1INSTR:

cmp cl,2

ja CMP3\_imGroup1INSTR

mov cx, [FLAGSr]

and cx,1

add ax,cx

or word[ebx],ax

jmp end\_imGroup1INSTR

CMP3\_imGroup1INSTR:

cmp cl,3

ja CMP4\_imGroup1INSTR

mov cx, [FLAGSr]

and cx,1

add ax,cx

sbb word[ebx],ax

ret

CMP4\_imGroup1INSTR:

cmp cl,4

ja CMP5\_imGroup1INSTR

and word[ebx],ax

jmp end\_imGroup1INSTR

CMP5\_imGroup1INSTR:

cmp cl,5

ja CMP6\_imGroup1INSTR

sub word[ebx],ax

jmp end\_imGroup1INSTR

CMP6\_imGroup1INSTR:

cmp cl,6

ja CMP7\_imGroup1INSTR

xor word[ebx],ax

jmp end\_imGroup1INSTR

CMP7\_imGroup1INSTR:

cmp word[ebx],ax

pushf

pop ecx

and word[FLAGSr], 1111011100101010b ; clear neccessary

and cx, 0000100011010101b ;neccessary flags

or word[FLAGSr], cx

ret

.data:

; 0 1 2 3 4 5 6 7 8 9 A B C D E F

OPCODES: dd ADDEbGb, ADDEvGv, ADDEbGb, ADDEvGv, ADDALIb, ADDAXIz, PUSHEs, POPEs, OREbGb, OREvGv, OREbGb, OREvGv, ORALIb, ORAXIz, PUSHCs, TWOB, \

ADCEbGb, ADCEvGv, ADCEbGb, ADCEvGv, ADCALIb, ADCAXIz, PUSHSs, POPSs, SBBEbGb, SBBEvGv, SBBEbGb, SBBEvGv, SBBALIb, SBBAXIz, PUSHDs, POPDs, \

ANDEbGb, ANDEvGv, ANDEbGb, ANDEvGv, ANDALIb, ANDAXIz, SEGES, DAA\_, SUBEbGb, SUBEvGv, SUBEbGb, SUBEvGv, SUBALIb, SUBAXIz, SEGCS, DAS\_, \

XOREbGb, XOREvGv, XOREbGb, XOREvGv, XORALIb, XORAXIz, SEGSS, AAA\_, CMPEbGb, CMPEvGv, CMPEbGb, CMPEvGv, CMPALIb, CMPAXIz, SEGDS, AAS\_, \

INCGenR, INCGenR, INCGenR, INCGenR, INCGenR, INCGenR, INCGenR, INCGenR, DECGenR, DECGenR, DECGenR, DECGenR, DECGenR, DECGenR, DECGenR, DECGenR, \

PUSHGenR, PUSHGenR, PUSHGenR, PUSHGenR, PUSHGenR, PUSHGenR, PUSHGenR, PUSHGenR, POPGenR, POPGenR, POPGenR, POPGenR, POPGenR, POPGenR, POPGenR, POPGenR, \

PUSHA\_, POPA\_, BOUNDGvMa, ARPLEwGw, SEGFS, SEGGS, OPSIZE, ADDRSIZE, PUSHIz, IMULGvEvIz, PUSHIb, IMULGbEbIb, INSYb, INSYz, OUTSXb, OUTSXz, \

JccO, JccNO, JccB, JccNB, JccZ, JccNZ, JccNA, JccA, JccS, JccNS, JccP, JccNP, JccL, JccNL, JccNG, JccG, \

imGroup1, imGroup1, imGroup1, imGroup1, TESTEbGb, TESTEvGv, XCHGEbGb, XCHGEvGv, MOVEbGb, MOVEvGv, MOVEbGb, MOVEvGv, MOVEvSw, LEAGvM, MOVEvSw, imGrop1A, \

NOP\_, XCHGAxGen, XCHGAxGen, XCHGAxGen, XCHGAxGen, XCHGAxGen, XCHGAxGen, XCHGAxGen, CBW\_, CWD\_, fCALLAp, fWAIT\_, PUSHFv, POPFv, SAHF\_, LAHF\_, \

MOVALOb, MOVAXOv, MOVALOb, MOVAXOv, MOVSYbXb, MOVSYvXv, CMPSYbXb, CMPSYvXv, TESTALIb, TESTAXIz, STOSYbAL, STOSYwAX, LODSAlXb, LODSAXXw, SCASALYb, SCASAXYv, \

MOV\_L\_Ib, MOV\_L\_Ib, MOV\_L\_Ib, MOV\_L\_Ib, MOV\_H\_Ib, MOV\_H\_Ib, MOV\_H\_Ib, MOV\_H\_Ib, MOV\_W\_Iv, MOV\_W\_Iv, MOV\_W\_Iv, MOV\_W\_Iv, MOV\_W\_Iv, MOV\_W\_Iv, MOV\_W\_Iv, MOV\_W\_Iv, \

Shifts, Shifts, RETne, RETne, LESGzMp, LDSGzMp, MOVEbIb, MOVEvIz, ENTERIwIb, LEAVE\_, fRET\_, fRET\_, INT3\_, INTIb, INTO\_, IRET\_, \

Shifts, Shifts, Shifts, Shifts, AAMIb, AADIb, SHIFTGR, XLAT\_, FPU, FPU, FPU, FPU, FPU, FPU, FPU, FPU, \

LOOPNEJb, LOOPEJb, LOOPJb, JrCXZJb, INALIb, INAXIv, OUTIbAL, OUTIbAX, nCALLJz, nJMPJz, fJMPAp, sJMPJb, INALDX, INAXDX, OUTDXAL, OUTDXAX, \

SHIFTGR, SHIFTGR, REPNE\_, REPE\_, HLT\_, CMC\_, F6\_, F6\_, CLC\_, STC\_, CLI\_, STI\_, CLD\_, STD\_, FE\_, FF\_

MODR\_M\_VALUES: dd BX\_SI\_MEM, BX\_DI\_MEM, BP\_SI\_MEM, BP\_DI\_MEM, SI\_MEM, DI\_MEM, IMM\_16\_MEM, BX\_MEM,\

BX\_SI\_MEM, BX\_DI\_MEM, BP\_SI\_MEM, BP\_DI\_MEM, SI\_MEM, DI\_MEM, BP\_MEM, BX\_MEM, \

BX\_SI\_MEM, BX\_DI\_MEM, BP\_SI\_MEM, BP\_DI\_MEM, SI\_MEM, DI\_MEM, BP\_MEM, BX\_MEM, \

AX\_AL, CX\_CL, DX\_DL, BX\_BL, Reg\_H, Reg\_H, Reg\_H, Reg\_H

retValPointer: dd 0

retVal: times 4 dd ?

; AX CX DX BX SP BP SI DI

GeneralRegs: dw 8 dup(0h) ; may accure to be very optimising desicion

; ES CS SS DS FS GS

SegmentRegs: dw 6 dup(0h)

FLAGSr: dw 0000010000000010b

addAddr: db 0

NumInt: db 0

prevESP\_: dd ?

ВЕДОМОСТЬ ДОКУМЕНТОВ

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | | Дополнительные сведения | |
|  | | | | Текстовые документы | | | | |  | |
|  | | | |  | | | | |  | |
| БГУИР КП I- 40 01 01 018 ПЗ | | | | Пояснительная записка | | | | | 56 с. | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | | Графические документы | | | | |  | |
|  | | | |  | | | | |  | |
| ГУИР. 251002 018 СА | | | | Функция обновления состояния игрока | | | | | Формат А1 | |
|  | | | | Схема алгоритма | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  | | | |  | | | | |  | |
|  |  |  |  |  | БГУИР КП I- 40 01 01 018 ПЗ | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм | Лист | № докум. | Подп. | Дата | «Эмулятор процессора х86»  Ведомость курсовой  работы | Литера | | | Лист | Листов |
| Разраб. | | Головин Е.С. |  | 26.12.23 | Т |  |  | 56 | 56 |
| Пров. | | Шостак Е.В. |  | 26.12.23 | Кафедра ПОИТ  гр. 251002 | | | | |