

Санкт-Петербургский политехнический университет Петра Великого  
Институт компьютерных наук и технологий

**Кафедра компьютерных систем и программных технологий**

**Отчет о лабораторной работе**

**Курс:** Проектирование ОС и компонентов

**Тема:** Загрузчики

Выполнил студент группы 13541/3

\_\_\_\_\_ Д.В. Круминьш  
(подпись)

Преподаватель

\_\_\_\_\_ Е.В. Душутина  
(подпись)

Санкт-Петербург  
2018 г.

# **Содержание**

<b>1 Задание</b>	<b>3</b>
<b>2 Сведения о системе</b>	<b>3</b>
<b>3 Выполнение работы</b>	<b>4</b>
3.1 Анализ назначения и структуры загрузчиков . . . . .	4
3.1.1 Первичный загрузчик . . . . .	4
3.1.2 Вторичный загрузчик . . . . .	6
3.2 Реализация простейшего первичного загрузчика . . . . .	7
3.2.1 Алгоритм программы . . . . .	7
3.2.2 Компиляция кода . . . . .	8
3.2.3 Поготовка виртуальной машины VMware . . . . .	8
3.2.4 Тестирование . . . . .	8
3.3 Реализация на языке С . . . . .	9
3.4 Реализация поиска и загрузки вторичного загрузчика . . . . .	11
3.4.1 Подготовка образа . . . . .	11
3.5 Реализация мультизагрузчика . . . . .	14
3.5.1 Подготовка накопителя . . . . .	15
3.5.2 Запись Ubuntu 16.04 . . . . .	16
3.5.3 Запись DrWeb LiveDisk 9.00 . . . . .	17
3.5.4 Запись мордочки . . . . .	17
3.5.5 Модификация первичного загрузчика . . . . .	20
3.5.6 Тестирование . . . . .	20
3.6 Первичный загрузчик ОС . . . . .	24
3.6.1 Алгоритм работы . . . . .	24
3.6.2 Подготовка . . . . .	24
3.6.3 Тестирование . . . . .	26
<b>Вывод</b>	<b>28</b>
<b>Список литературы</b>	<b>29</b>
<b>Приложения</b>	<b>30</b>

## 1 Задание

1. В загрузочный сектор поместить программу вывода на экран произвольного сообщения. Программу написать на ассемблере (предпочтительно встраиваемом в стандартную конфигурацию) и на С (при необходимости допустимы ассемблерные вставки), убедиться в работоспособности обоих вариантов. См. пример кода ниже. В качестве носителя предпочтителен выбор флэш. Начать можно с экспериментов над виртуальной дискетой, как показано ниже.
2. Создать первичный загрузчик для виртуального, а затем реального носителя, (пример показан для виртуальной дискеты и ФС FAT12/16), который будет находить файл программы на носителе и загружать ее на выполнение.
3. **Создать мультизагрузчик**, обеспечивающий варианты выбора загружаемой на исполнение программы Для эксперимента можно использовать несколько собственных программ, включая программы из п.1 и 2.
4. **Разработать первичный загрузчик ОС**  
Это задача аналогичная предыдущей (п.2) с той разницей, что в качестве исполняемого файла, искомого на носителе выступает файл с ядром ОС.

## 2 Сведения о системе

Работа производилась на реальной системе, со следующими характеристиками:

Элемент	Значение
Имя ОС	Майкрософт Windows 10 Pro (Registered Trademark)
Версия	10.0.16299 Сборка 16299
Установленная оперативная память (RAM)	16,00 ГБ
Процессор	Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz, 2496 МГц, ядер: 4, логических процессоров: 4

Таблица 1: Сведения о системе

Для выполнения работы(проверка первичного, вторичного загрузчика) использовалась **VMware Workstation 12 pro (12.5.7 build-5813279)**

## **3 Выполнение работы**

### **3.1 Анализ назначения и структуры загрузчиков**

При включении вычислительной машины(ВМ), в ее оперативной памяти нет никакой программы для исполнения. Поэтому аппаратно предусмотрено, что после включения вычислительной машины управление осуществляется базовой системой ввода-вывода – **BIOS**.

Инициализируется начальная самопроверка аппаратного обеспечения **POST** (Power-On Self-Test), в ходе которой происходит настройка низкоуровневых аппаратно-зависимых параметров системы.

В случае успешного прохождения тестирования BIOS ищет на всех доступных носителях первичный загрузчик операционной системы. BIOS считывает первый сектор жесткого диска – посредством прерывания int 19h и размещает его в памяти по адресу 0000:7C00h[1]. После этого производится проверка, что данный сектор соответствует разметке **MBR**, и если это так, то передает туда управление. Если же сектор не соответствует MBR, то производится загрузка первого сектора с другого носителя и аналогичная проверка.

#### **3.1.1 Первичный загрузчик**

Первичный загрузчик **MBR**(Master Boot Record) – это программа, расположенная в первом секторе (512 байт) одного из физических носителей, основной функцией которого является определение активного логического раздела, размещение в памяти и передача управления на соответствующий загрузочный сектор **VBR**(Volume Boot Record). Функционирование MBR не зависит от типа файловых систем логических разделов.

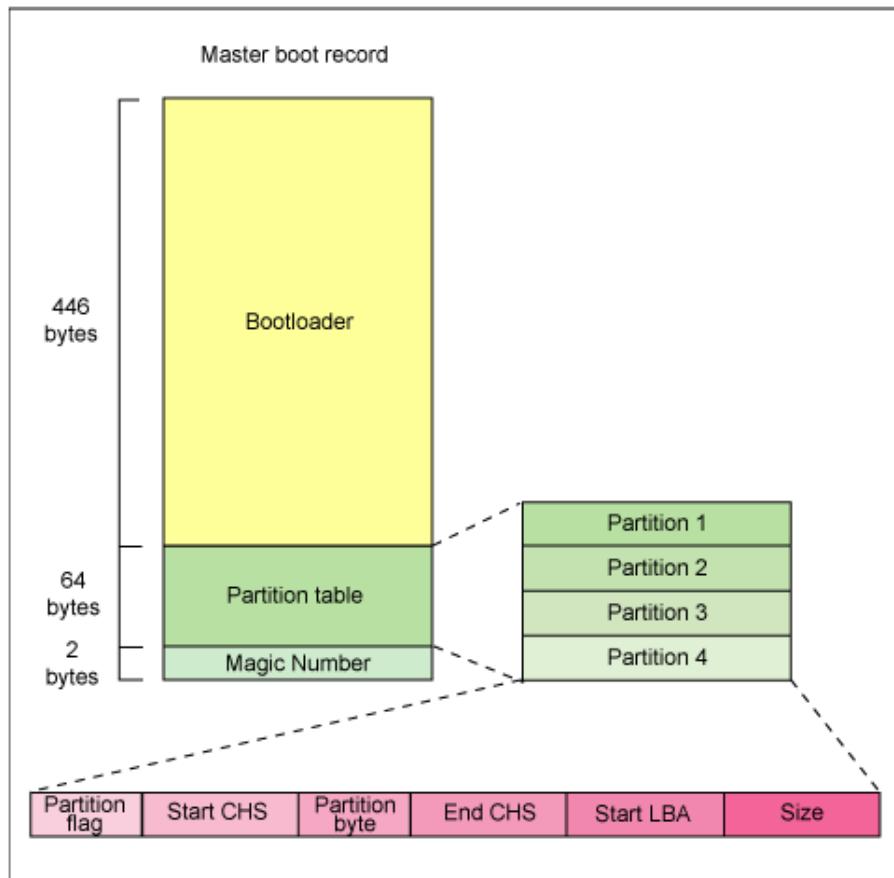


Рис. 1: Структура MBR

Запись состоит из следующих полей:

### Bootloader

Код начальной загрузки. Команды, используемые для определения местоположения и загрузки загрузочной записи раздела (VBR) с активного (загрузочного) раздела.

### Partition Table

Главная таблица разделов. Таблица, состоящая из четырех 16-байтовых записей для четырех первичных логических разделов или трех первичных и одного расширенного разделов. Каждый первичный раздел определяет один логический диск, а расширенный раздел может быть разбит на несколько логических дисков. Каждая запись таблицы разделов имеет следующий формат:

Смещение (байт)	Размер (байт)	Запись
0	1	Boot Indicator
1-3	3	Start CHS
4	1	Partition-type

5-7	3	End CHS
8-11	4	Starting Sector
12-15	4	Partition Size

Таблица 2: Структура 16-байтной записи[2]

1. **Boot Indicator.** Флаг типа раздела (активный, неактивный, некорректный и т.д.)
2. **Start CHS.** Адрес первого сектора раздела в формате CHS (Cylinder, Head, Sector). Представляет собой кортеж из трёх координат(цилиндр, головка, сектор). Цилиндр - совокупность дорожек (треков) заданного радиуса на всех пластинах (и всех сторонах пластин). Головка - номер считывающей головки (соответствует номеру пластины, с каждой поверхности считывает только одна головка). Сектор - градус угла поворота диска.
3. **Partition-type.** Тип файловой системы, расположенной на разделе. Например **0Bh** означает систему **FAT32**.
4. **End CHS.** Адрес последнего сектора раздела в формате CHS.
5. **Starting Sector.** Адрес первого сектора раздела в формате LBA (Logical Block Addressing). Формат аналогичен CHS, за исключением того что позволяет обращаться к разделам находящимся за пределами 7.8 ГБ и до 2.2 ТВ.
6. **Partition Size.** Число секторов раздела.

**Примечание:** в текущий момент идет отказ от CHS в пользу LBA, причем размер LBA увеличили с 32 бит до 48, что позволяет работать с максимальной емкостью в 144 петабайта[3].

### Magic Number

Двухбайтовая сигнатура **55AAh**, используемая для идентификации MBR.

#### 3.1.2 Вторичный загрузчик

Вторичный загрузчик - программа расположенная в загрузочном секторе активного логического раздела **VBR**(Volume Boot Record).

Задачей вторичного загрузчика является поиск ядра в определенной файловой системе на разделе диска, загрузка ее в память и передача управления ядру ОС. Для этого

вторичный загрузчик должен уметь работать с файловой системой, расположенной на данном разделе.

Смещение (hex формат)	Размер (байт)	Запись
0	3	JMP на загрузчик
03	8	Название системы
0B	25	Блок параметров BIOS
24	26	Дополнительный блок параметров BIOS
3E	448	Загрузчик
1FE	2	Сигнатура 55AAh

Таблица 3: Структура VBR для ФС FAT

Итогом работы первичного и вторичного загрузчика является размещение ядра операционной системы в оперативной памяти, и далее, готовность к выполнению уже более высокоуровневых функций.

## 3.2 Реализация простейшего первичного загрузчика

Код первичного загрузчика приведен далее. Программа демонстрирует основную структуру первичного загрузчика и простейший принцип функционирования – в данном случае простой вывод приветственного сообщения.

Код основан на статье с хабрахабра[4] и приведен в приложении 1.

### 3.2.1 Алгоритм программы

Программа размещается при считывании в оперативной памяти начиная с адреса 0x7C00h. Затем с использованием прерывания видео-сервиса BIOS 10h производится очищение экрана и установка видеорежима (сервис установки видеорежима; для возможности вывода цветного текста). После этого осуществляется циклический посимвольный вывод строки буфера, содержащей тестовое сообщение

### 3.2.2 Компиляция кода

Для этого поставим yasm(<http://yasm.tortall.net/Download.html>) и выполним команду:

```
vsyasm.exe mbr.asm -f bin -o mbr
```

Листинг 1: Компиляция ассемблерного кода

После этого был получен файл **mbr** размером в 512 байт. Дополнительно файлу было задано расширение **.iso**.

### 3.2.3 Поготовка виртуальной машины VMware

Была создана пустая виртуальная машина. В её настройках была добавлен виртуальная дискета в которой был указан файл **mbr.iso**.

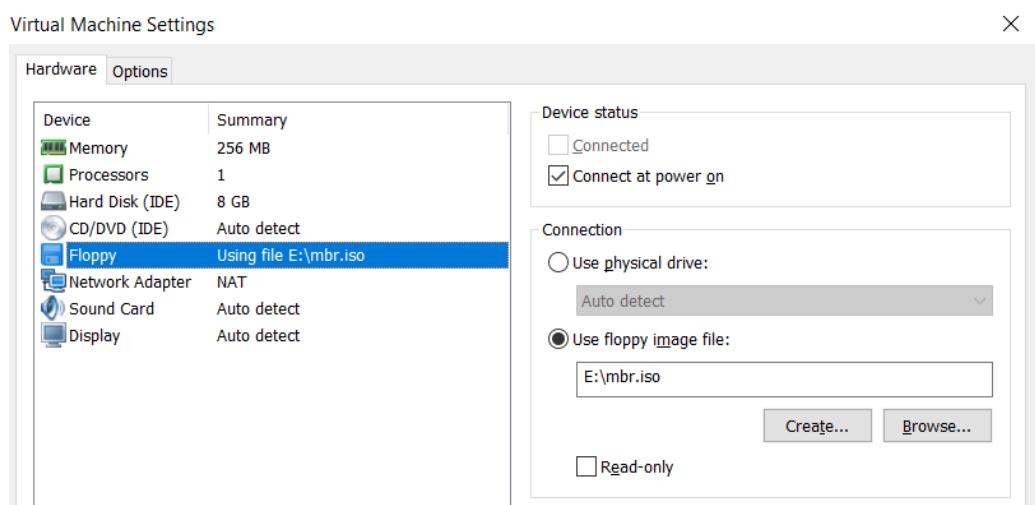


Рис. 2: Виртуальная дискета

### 3.2.4 Тестирование

Запускаем виртуальную машину. На экране появится надпись:

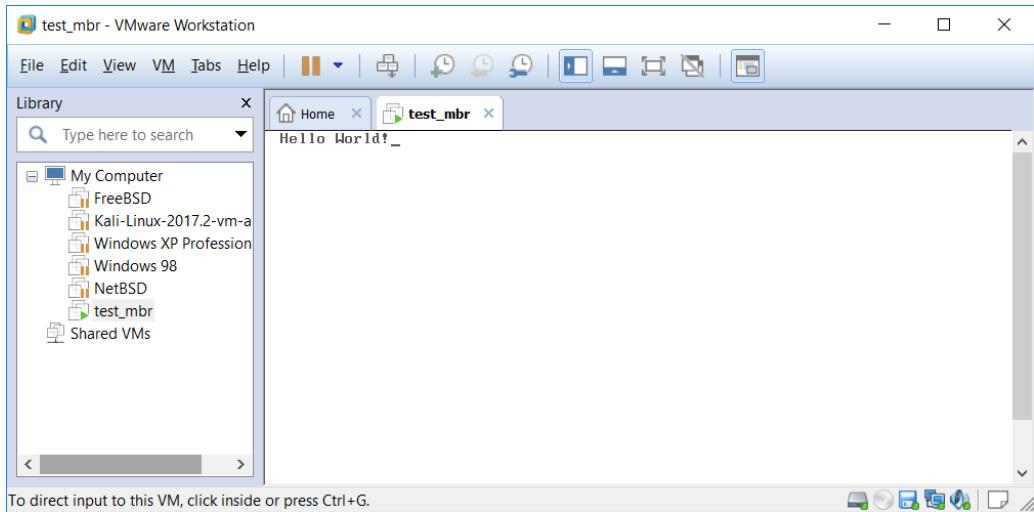


Рис. 3: Запуск первичного загрузчика

Как и ожидалось на экран вывелоось приветственное сообщение.

### 3.3 Реализация на языке С

На языке С, был реализован простейший первичный загрузчик.

```
#define SECTOR_SIZE 512
#define BOOTLOADER_TYPE
#define BOOTLOADER_RUNNING_ADDRESS 0x8000
#define BOOTLOADER_LOAD_ADDRESS 0x8000
#define BOOTLOADER_START_SECTOR 1

asm( ".code16gcc\n" );

#define __NOINLINE __attribute__(( noinline ))
#define __REGPARM __attribute__(( regparm(3)))
#define __NORETURN __attribute__(( noreturn))

void __NOINLINE __REGPARM print(const char *s) {
    while(*s) {
        asm volatile (
            "movb    $0xE, %%ah\n"
            "movb    %b0, %%al\n"
            "int     $0x10"
            :
            : "al" (*s++) : "ah", "bx");
    }
}
```

```
void __NORETURN __attribute__((section("__start"))) main() {
    print("Hello , MBR!\r\n");
    goto *((void *) BOOTLOADER_RUNNING_ADDRESS);
}
```

Листинг 2: mbr.c

Данный файл можно разделить на 3 области:

1. Область с различными **define**, где определяется размер сектора, адрес для загрузки, загружаемый сектор, а также некоторые макросы для используемых далее функций;
2. Функция **print**, которая печатает на экран некоторый текст, который был передан в данную функцию в качестве параметра;
  - Реализация вывода аналогична прошлой, печать символов происходит до появления пустого символа.
3. Функция **main**, которая и вызывает функцию печати.

Дополнительно, для удобства компиляции был создан **Makefile** с необходимыми ключами компиляции.

```
boot: boot.c linker.ld
    @gcc ${CFLAGS} -c -Os -nostdinc -nostdlib -nodefaultlibs -fno-builtin -WI
    ↳ ,--gc-sections -fno-stack-protector -ffreestanding -m32 -Wall -Werror -l.
    ↳ -o boot.o boot.c
    @ld -melf_i386 -static -Tlinker.ld -nostdlib --nmagic -o boot.elf boot.o
    @objcopy -O binary boot.elf boot.MBR

clean:
    rm -rf boot.elf boot.MBR boot.o
```

Листинг 3: Makefile

Для компиляции программы, необходимо выполнить команду **make**.

Тестирование будет произведено с использованием Qemu версии 2.5.0.

Выполним команду **qemu-system-x86\_64 boot.MBR**, после чего откроет окно QEMU:

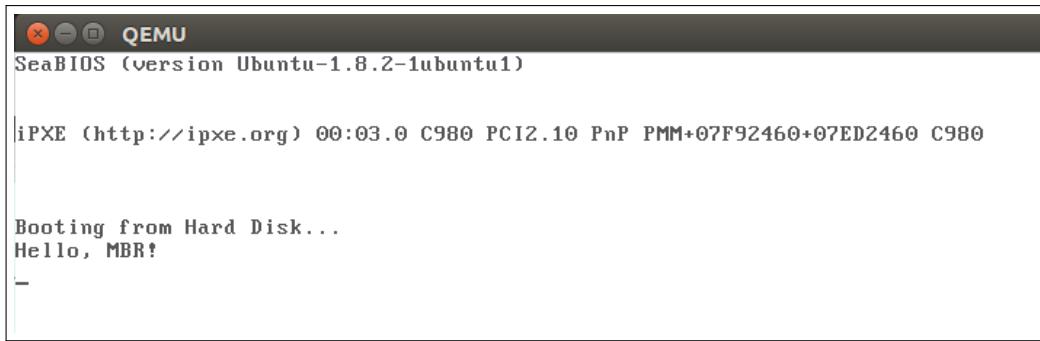


Рис. 4: Окно QEMU

Сообщение было успешно выведено.

### 3.4 Реализация поиска и загрузки вторичного загрузчика

В данной части работы проводится реализация первичного загрузчика, который находит необходимый файл на устройстве с определенной файловой системой. Также была выполнена реализация вторичного загрузчика, код которого загружается первичным загрузчиком для выполнения.

В качестве файловой системы выбрана FAT-12. Код загрузчиков основан на статье[5] и приведен в приложении 2.

Первичный загрузчик осуществляет поиск файла с определенным заранее известным именем(bootor). Если первичный загрузчик находит его, то производит его загрузку в определенное место в оперативной памяти, после чего передает туда управление.

Вторичный загрузчик проверяет, что был загружен по определенному адресу, после чего выводит приветственное сообщение.

#### 3.4.1 Подготовка образа

Для получения исполняемых файлов необходимо их скомпилировать:

```
vsyasm.exe bootor.asm -f bin -o BOOTOR  
vsyasm.exe fat12.asm -f bin -o fat12
```

Листинг 4: Компиляция ассемблерного кода

Далее с помощью программы **UltraISO** был создан образ **loader\_1.iso** в корень которого был добавлен файл **BOOTOR**.

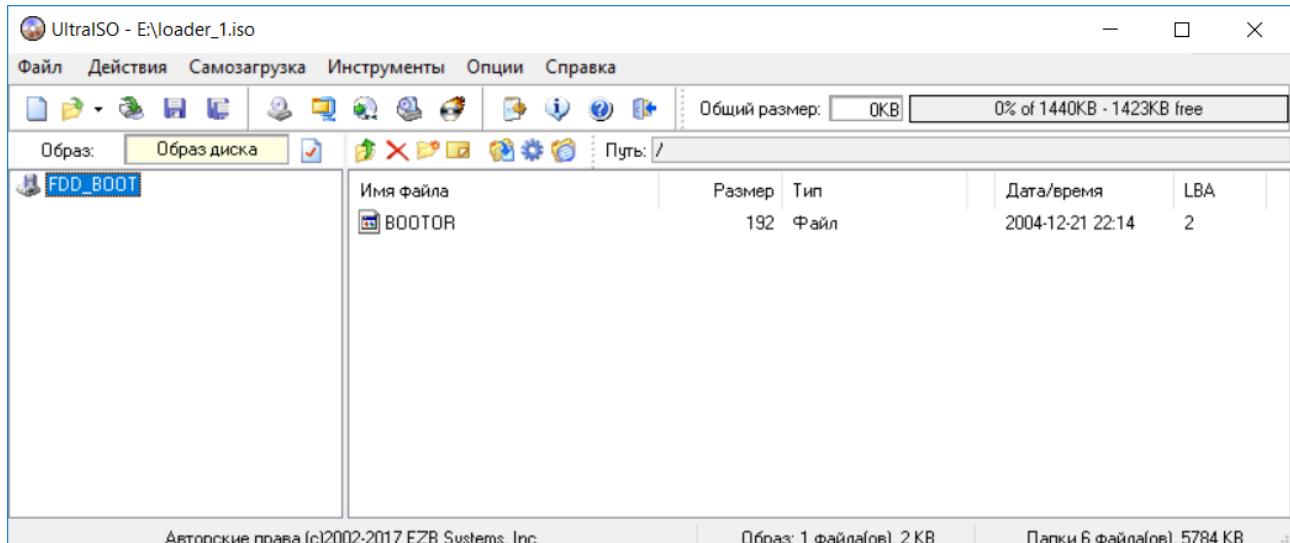


Рис. 5: Содержимое loader\_1.iso

Далее с помощью редактора **FlexHEX** был открыт бинарный код файлов **loader\_1.iso** и **fat12**.

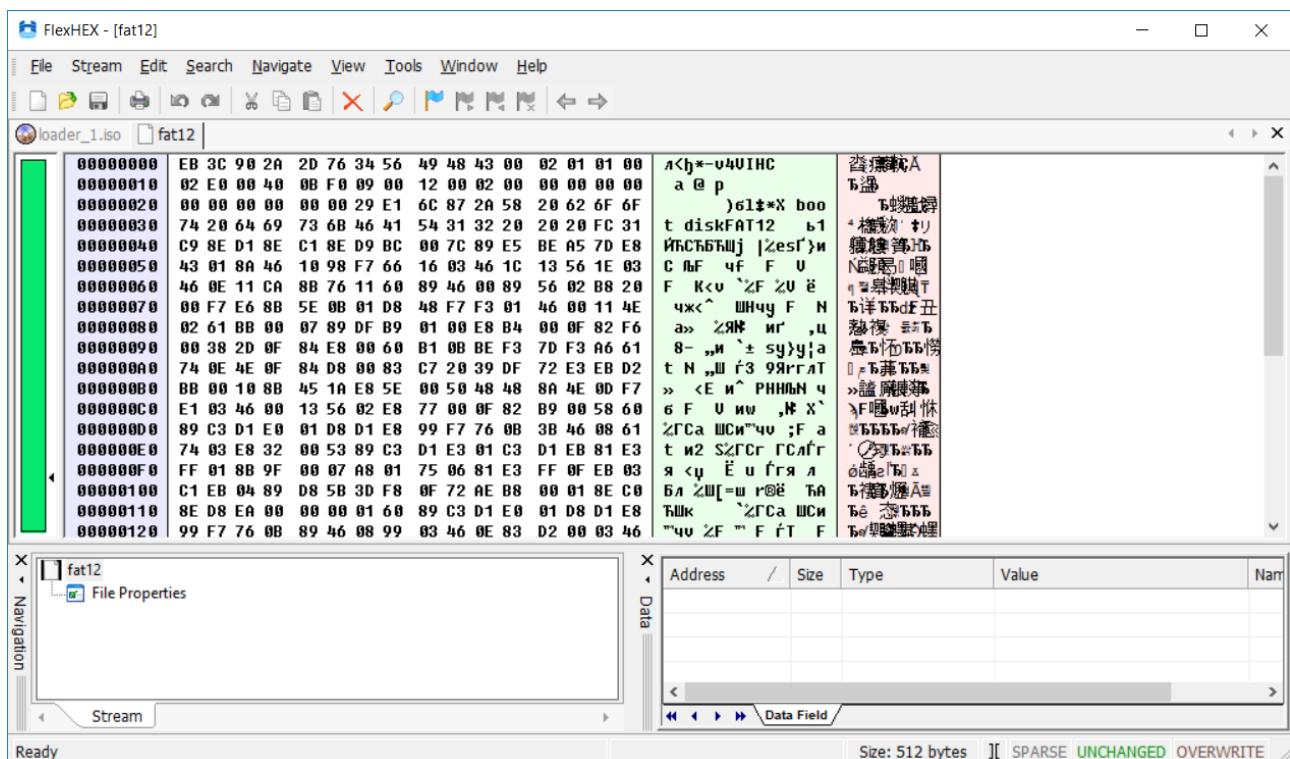


Рис. 6: Бинарный код fat12

В данном случае необходимо скопировать первые 512 байт(первый сектор) файла fat12 в начало файла loader\_1.iso.

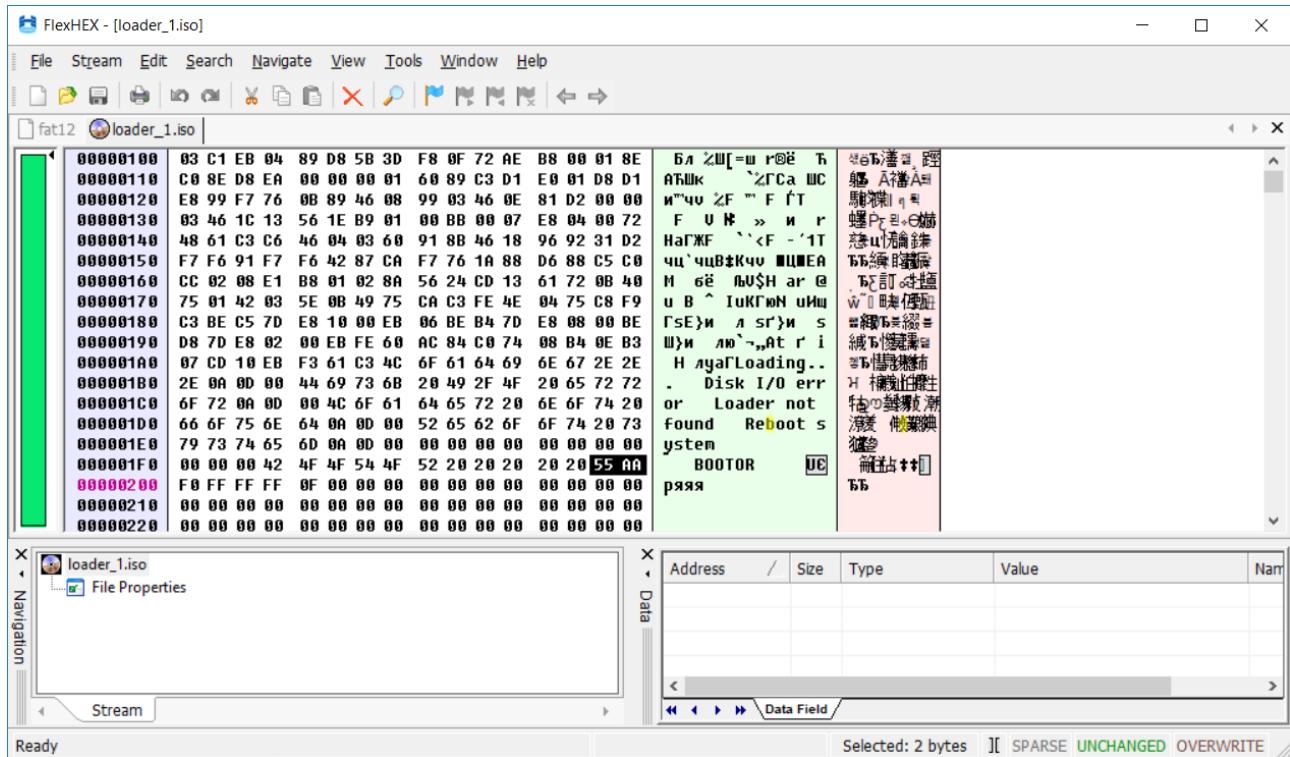


Рис. 7: Бинарный код loader\_1.iso

Последние два байта соответствуют сигнатуре **55AAh**.

Была создана пустая виртуальная машина, далее в её настройках был добавлен виртуальная дискета в которой был указан файл **loader\_1.iso**.

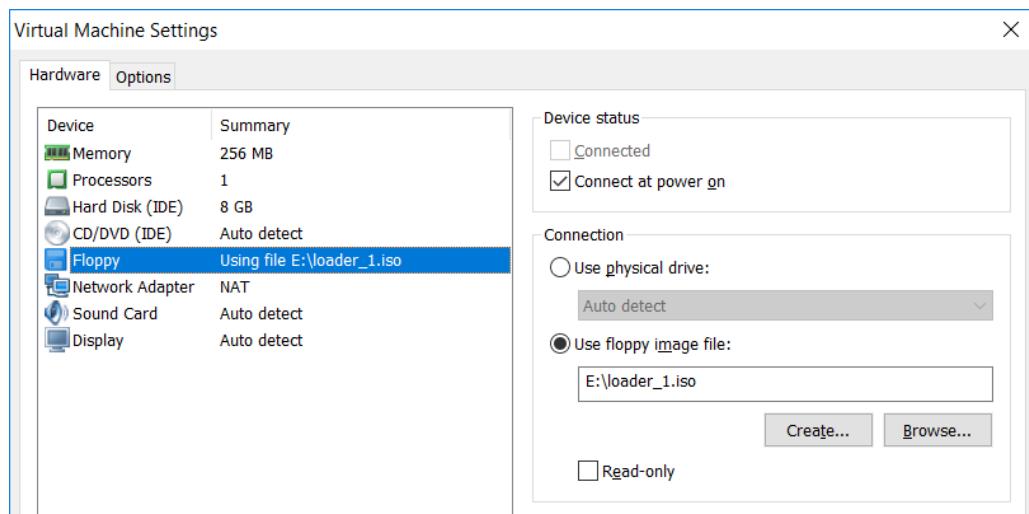


Рис. 8: Виртуальная дискета

Далее запускаем виртуальную машину. В случае успеха было выведено приветственное сообщение вторичным загрузчиком.

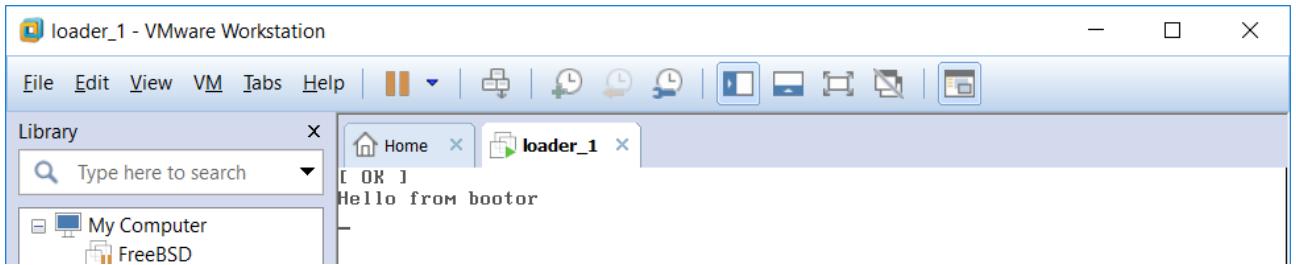


Рис. 9: Приветственное сообщение вторичного загрузчика

В случае, если например удалить файл bootor из образа, то будет выведено сообщение об ошибке.

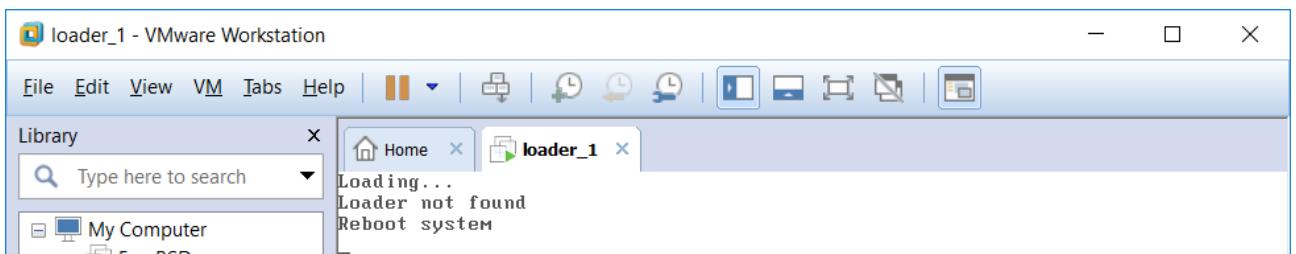


Рис. 10: Сообщение об ошибке

В результате работы удалось выполнить поиск определенного файла и передачи ему управления. Управление передавалось простейшему коду, который выводил приветственное сообщение. Но если производить поиск, загрузку и передачу управления не произвольному файлу, а ядру ОС, то будет произведена ее полноценная загрузка.

### 3.5 Реализация мультизагрузчика

Разметка MBR позволяет описать до 4 первичных разделов, каждый из которых может быть активным – то есть может содержать вторичный загрузчик. Также имеется возможность создать расширенный раздел, в котором можно описать ещё 4 раздела, но в данном случае, для упрощения расширенный раздел не анализируется.

Для загрузки, выступают следующие программы:

1. Ubuntu 16.04;
2. DrWeb livedisk 9.00;
3. Вывод на экран "мордочки".

Алгоритм мультизагрузчика:

- Для избежание проблемы того, что вторичный загрузчик будет скопирован на место первичного загрузчика, необходимо заранее скопировать первичный загрузчик с адреса **7C00h** в безопасную область памяти, в данном случае, безопасной областью выбран адрес **60h**;
- Поиск активных разделов и вывод их номеров, для этого анализируются все четыре записи таблицы разделов, на предмет наличия записи активного раздела - 80h;
- Ожидание ввода номера раздела с клавиатуры. Для чтения нажатой клавиши используется прерывание BIOS **int 16h**, после чего производится сравнение ASCII кода введенной клавиши с номерами активных разделов;
- Загрузка первого сектора(вторичного загрузчика) выбранного активного раздела по адресу 7C00h и передача управления по этому адресу. Загрузка первого сектора производится с помощью прерывания BIOS **int 13h**.

Исходный код мультизагрузчика, приведен в приложении 3, в листинге 10.

### 3.5.1 Подготовка накопителя

Работа будет производится на реальном устройстве - USB флэшке емкостью 32 Гб. Средствами, встроенной в Windows утилиты - **управление дисками**, устройство было размечено следующим образом:

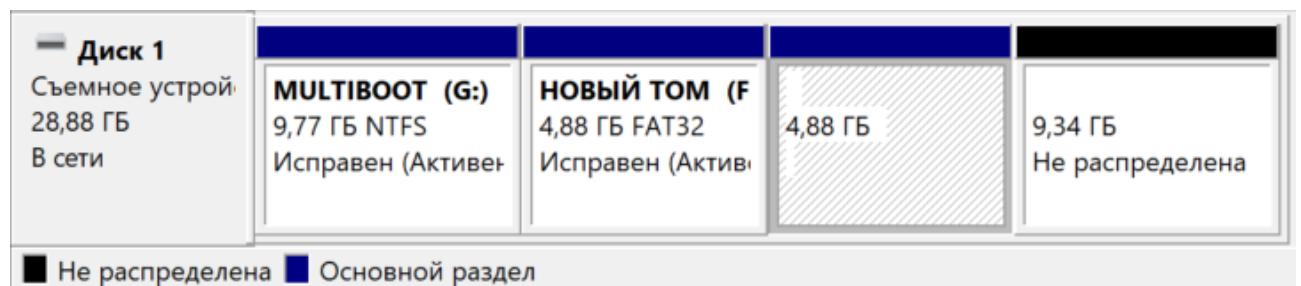


Рис. 11: Разметка носителя

Где:

- Первый раздел(G) с ФС **NTFS** - для Ubuntu;
- Второй раздел(F) с ФС **FAT32** - для DrWeb;
- Третий раздел, без ФС - для "мордочки".

### 3.5.2 Запись Ubuntu 16.04

Основная проблема с корректностью запуска ОС, связана с корректностью записи на носитель программы загрузчика. Например, при записи DrWeb livedisk, никаких проблем не возникло, так как программа и предназначалась для подобных целей. То в данном случае, столкнулся со следующими проблемами:

- Большинство программ для записи были не в состоянии записать образ на конкретный раздел, большинство из них предлагало перезаписать в USB-накопитель.
- Оставшиеся программы, некорректно записывали ОС, в следствии чего, вторичный загрузчик сообщал о том что не может найти необходимые файлы.

Путем проб и ошибок, выбор пал на программу **YUMI – Multiboot USB Creator**, которая использует в виде вторичного загрузчика - модифицированный загрузчик syslinux.

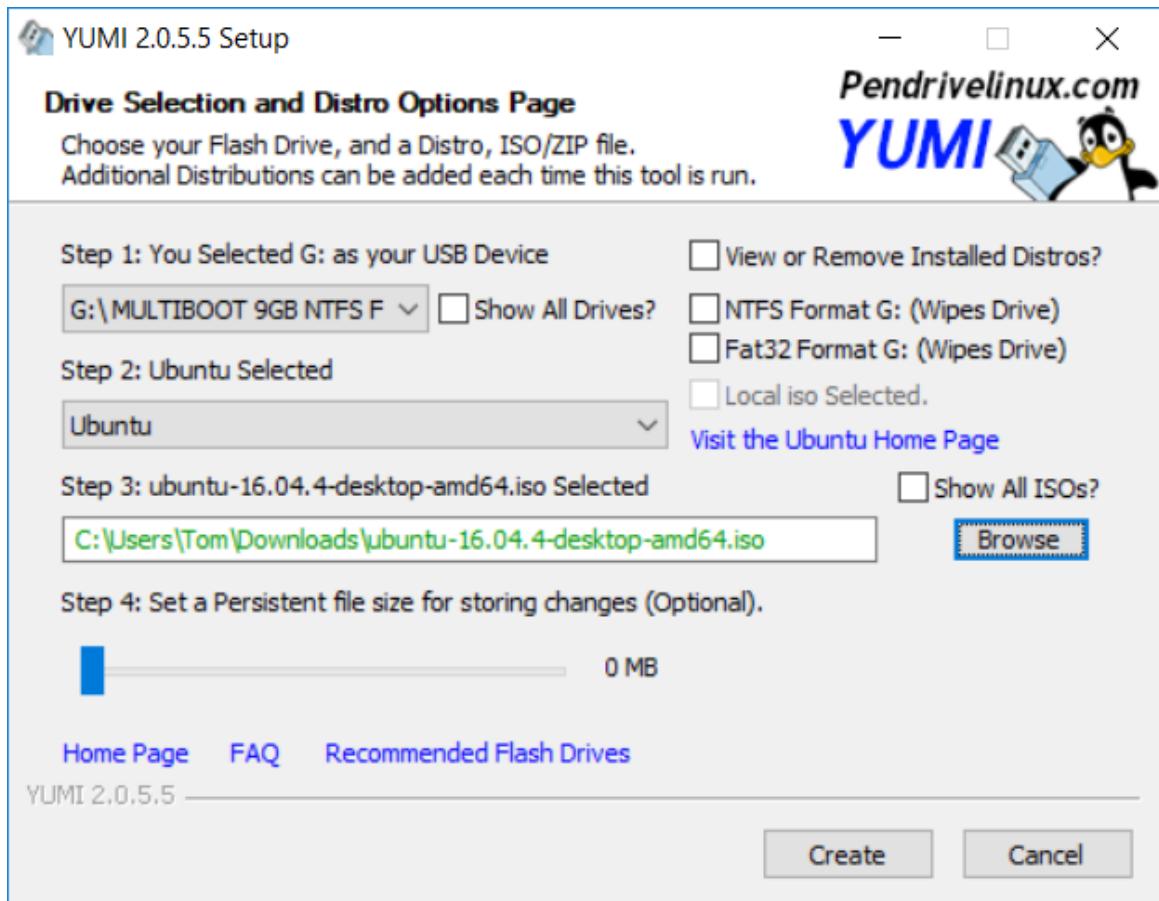


Рис. 12: Подготовка к записи ОС, с использованием Yumi

### 3.5.3 Запись DrWeb LiveDisk 9.00

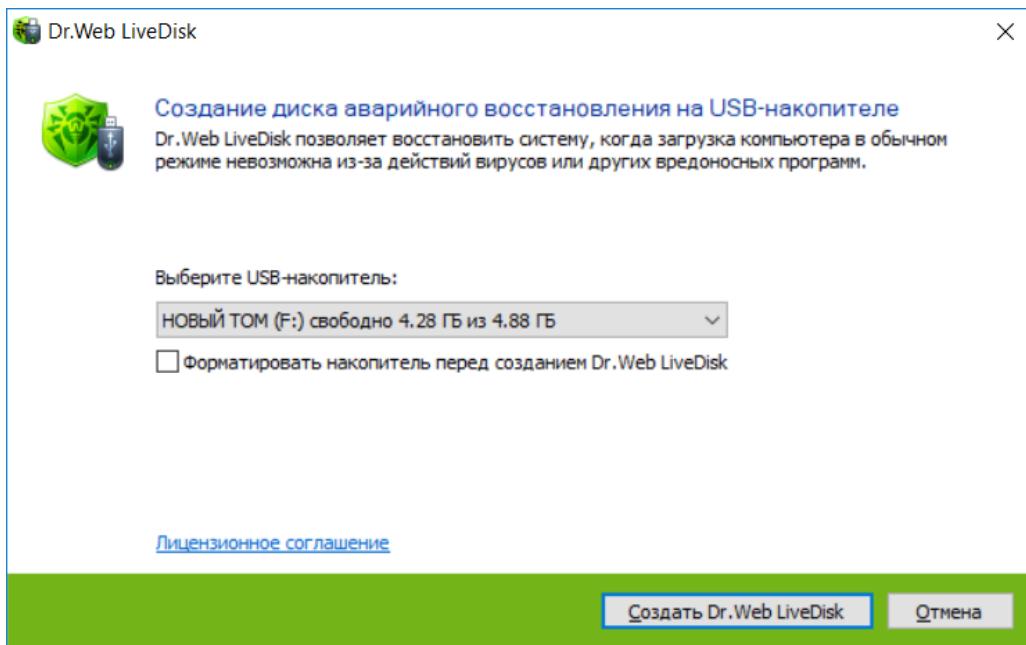


Рис. 13: Запись DrWeb

После записи, раздел на который была произведена установка станет активным.

### 3.5.4 Запись мордочки

В качестве "мордочки", будет выступать следующие изображение:

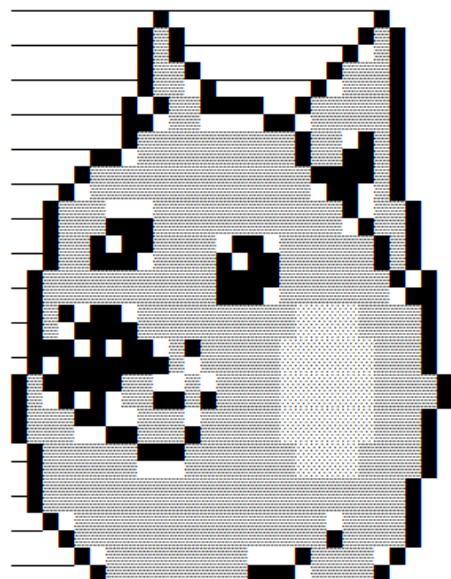


Рис. 14: "Мордочка"

Программа, для вывода "мордочки" расположена в приложении 3, листинге 8, где каждый выводимый символ задан в виде ASCII кода символа.

Для получения ASCII кодов символов, на языке **Python** был написан специальный конвертер, который представлен в приложении 3, листинге 9.

Алгоритм которого заключается в посимвольно чтении исходного файла с изображением, где для каждого символа, используя словарь(символ - код) выводится его код.

После компиляции ассемблерного кода, с помощью редактора **HxD** открываем его, а также 0 сектор USB-накопителя.

Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	
0000000000	B8 C0 07 8E D8 31 F6 B8 60 00 8E C0 31 FF B9 FF	С.А.Мицё.АляНя
0000000010	00 F3 A5 EA 18 06 00 00 B8 F4 00 E8 C0 00 B8 OE	.уГк...ёф.иA.ё.
0000000020	01 E8 BA 00 8B 36 A4 01 B7 80 B1 FF 80 F9 03 74	.иё..<6я.·ВяБшт
0000000030	2F 83 C6 10 FE C1 26 8A 1C 38 FB 75 EF E8 02 00	/fЖ.юБ&.8ыши..
0000000040	EB EA B8 28 01 E8 96 00 BF A1 01 00 0D B8 A1 01	лкё(.и-.иУ...ёУ..
0000000050	E8 8B 00 28 0D BF A6 01 B5 00 01 CF C6 05 01 C3	и..(.и..и..ПЖ..Г
0000000060	B8 43 01 E8 78 00 BF A6 01 8B 36 A4 01 B4 00 CD	ëС.их.и!.<6я.г.Н
0000000070	16 3C 30 75 05 83 C6 10 EB 21 3C 31 75 05 83 C6	.<Оu.íЖ.л!<lu.íЖ
0000000080	20 EB 18 3C 32 75 05 83 C6 30 EB 0F 83 C6 40 3C	л.<2u.íЖ0l.íЖ@<
0000000090	33 74 08 B8 6D 01 E8 45 00 EB CB B4 00 2C 30 01	3t.ëm.иE.лНг.,0.
00000000A0	C7 80 3D 00 74 ED B4 41 BB AA 55 CD 13 72 0A D1	ЗВ=.thгA»EUH.г.C
00000000B0	E9 73 06 81 FB 55 AA 74 08 B8 86 01 E8 1F 00 CD	йз.ГыUet.ët.и..Н
00000000C0	18 B4 42 BF AA 01 83 C7 08 83 C6 08 66 8B 1C 66	.гBie.í3.íЖ.f<.f
00000000D0	89 1D BE AA 01 CD 13 72 E0 EA 00 7C 00 00 56 53	%.sE.H.rак. ..VS
00000000E0	89 C3 31 F6 B4 0E 8A 00 3C 0A CD 10 74 03 46 EB	кГlцг.Ь.<.Н.t.Fл
00000000F0	F5 5B 5E C3 2A	*[^Г*****.Si
000000100	2A 0D 0A 53 69	*****.Si
000000110	6D 70 6C 65 20 6D 75 6C 74 69 62 6F 6F 74 20 20	mple multiboot
000000120	6C 6F 61 64 65 72 0D 0A 46 69 6E 64 20 62 6F 6F	loader..Find boo
000000130	74 61 62 6C 65 20 70 61 72 74 69 74 6F 6E 73	table partitions
000000140	3A 0D 0A 53 65 6C 65 63 74 20 70 61 72 74 20 74	:..Select part t
000000150	6F 20 62 6F 6F 74 20 66 72 6F 6D 20 28 70 72 65	o boot from (pre
000000160	73 73 20 30 20 2E 2E 20 33 29 0D 0A 57 72 6F	ss 0 ... 3)..Wro
000000170	6E 67 20 63 68 6F 69 73 65 2E 20 54 72 79 20 61	ng choise. Try a
000000180	67 61 69 6E 0D 0A 61 20 64 69 73 6B 20 72 65 61	gain..a disk rea
000000190	64 20 65 72 72 6F 72 20 6F 63 63 75 72 65 64 0D	d error occurred.
0000001A0	0A 30 0D 0A AE 01 00 00 00 10 00 01 00 00 00 7C	.0...@.....
0000001B0	00 00 00 00 00 00 00 00 89 8F 52 5A 00 00 80 20	.....%ЦRZ..Ђ
0000001C0	21 00 07 FE FF FF 00 08 00 00 00 80 38 01 80 FE	!.юяя....Ђ8.Ђю
0000001D0	FF FF 0C FE FF FF 00 88 38 01 00 40 9C 00 80 FE	яя.юяя.€8..@њ.Ђю
0000001E0	FF FF 0C FE FF FF 00 C8 D4 01 00 40 9C 00 00 00	яя.юяя.ИФ..@њ...
0000001F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA	.....Ue

Рис. 15: 0 сектор USB-накопителя

Зеленой рамкой выделен адрес, по которому начинается данный раздел. В данном случае это **01 D4 C8 00**(адрес читается задом наперед).

Если открыть калькулято в режиме программиста и ввести данный адрес, то его **DEC** значение будет означать номер сектора.



Рис. 16: Определение сектора

Перейдем по данному сектору, и скопируем в него данные из скомпилированный программы по выводу "мордочки".

Конечный результат, должен выглядеть следующим образом:

Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F		Сектор 30722048
3A9900000	B8 03 00 CD 10 8C C8 8E D8 BE 1C 7C FC B4 0E B7	..Н.ЫИШв. ъг..	
3A9900010	00 AC 84 C0 74 04 CD 10 EB F7 EB FE C4 C4 C4 C4	.~„At. Н.члюДДДД	
3A9900020	C4 C4 C4 C4 DC C4	ДДДДДДДДДДДДДДДД	
3A9900030	C4 C4 C4 DC 0D 0A C4 C4 C4 C4 C4 C4 C4 DB B1	ДДД..ДДДДДДДДД..	
3A9900040	DB C4 C4 C4 C4 C4 C4 C4 C4 DC DF B1 DB 0D	НДДДДДДДДДДДД..	
3A9900050	0A C4 C4 C4 C4 C4 C4 C4 DB B1 B1 DF DC C4 C4	.ДДДДДДДДы+яъДД	
3A9900060	C4 C4 C4 DC DF B1 B1 B1 DB 0D 0A C4 C4 C4 C4	ДДДДД..ДДДДД..	
3A9900070	C4 C4 C4 DB DC DF B1 B1 DF DF DF DC DC DF B1	ДДДД..ДДДД..	
3A9900080	B1 B1 B1 B1 DB 0D 0A C4 C4 C4 C4 DC DC DF B1	++..+..ДДДДД..	
3A9900090	B1 B1 B1 B1 B1 B1 B1 DB B1 B1 DC DB B1 DB	++++..+..+..	
3A99000A0	0D 0A C4 C4 C4 DC DF B1 B1 B1 B1 B1 B1 B1 B1	..ДДДД..	
3A99000B0	B1 B1 B1 B1 B1 DF DB DF B1 B1 B1 B1 B1 B1 B1	++..+..+..+..	
3A99000C0	B1 B1 B1 DC DC B1 B1 B1 B1 B1 B1 B1 B1 B1	++..+..+..+..	
3A99000D0	B1 B1 DF DC B1 B1 B1 OD 0A C4 C4 DB B1 B1 B1	++..+..+..+..	
3A99000E0	DB DF B1 B1 B1 B1 DC DF DB DC B1 B1 B1 B1 B1	..+..+..+..+..	
3A99000F0	DB B1 DB OD 0A C4 DB B1 B1 B1 B1 B1 B1 B1 B1	..+..+..+..+..	
3A9900100	B1 B1 DB DB DF B1 B1 B1 B1 B1 B1 DF DC DB	++..+..+..+..	
3A9900110	OD 0A C4 DB B1 DF DC DB DC B1 B1 B1 B1 B1 B1	..+..+..+..+..	
3A9900120	B1 B1 B1 B0 B0 B0 B0 B1 B1 B1 B1 DB OD 0A C4	++..+..+..+..+..	
3A9900130	DB DF B1 DC DC DB DB DC B1 DF B1 B1 B1 B1 B1	..+..+..+..+..	
3A9900140	B0 B0 B0 B0 B0 B1 B1 B1 B1 DB OD 0A DB B1 DF DB	..+..+..+..+..	
3A9900150	DF DB DF B1 B1 DC DC B1 DC B1 B1 B1 B0 B0 B0	..+..+..+..+..	
3A9900160	B0 B0 B0 B1 B1 B1 B1 DB OD 0A DB B1 B1 B1 DF DF	..+..+..+..+..	
3A9900170	DC DC B1 B1 B1 DC B1 B1 B1 B1 B0 B0 B0 B0	..+..+..+..+..	
3A9900180	B0 B1 B1 B1 DB OD 0A C4 DB B1 B1 B1 B1 B1 B1	..+..+..+..+..	
3A9900190	DF DF B1 B1 B1 B1 B1 B0 B0 B0 B0 B1 B1 B1	..+..+..+..+..	
3A99001A0	B1 DB OD 0A C4 DB B1 B1 B1 B1 B1 B1 B1 B1	..+..+..+..+..	
3A99001B0	B1 DB OD 0A	..+..+..+..+..	
3A99001C0	C4 C4 DF DC B1	..+..+..+..+..	
3A99001D0	B1 B1 B1 B1 DC B1 B1 B1 B1 DB OD 0A C4 C4 C4	..+..+..+..+..	
3A99001E0	DF DC B1 B1 B1 B1 B1 B1 DC DC DF B1	..+..+..+..+..	
3A99001F0	B1 B1 B1 DC DF 00 00 00 00 00 00 00 55 AA	..+..+..+..+..	

Рис. 17: 0 сектор третьего раздела

### 3.5.5 Модификация первичного загрузчика

Теперь необходимо заменить код в 0 секторе, на код мультизагрузчика, итоговый результат должен выглядеть следующим образом:

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0000000000	B8	C0	07	8E	D8	31	F6	B8	60	00	8E	C0	31	FF	B9	FF
0000000010	00	F3	A5	EA	18	06	00	00	B8	F4	00	E8	C0	00	B8	0E
0000000020	01	E8	BA	00	8B	36	A4	01	B7	80	B1	FF	80	F9	03	74
0000000030	2F	83	C6	10	FE	C1	26	8A	1C	38	FB	75	EF	E8	02	00
0000000040	EB	EA	B8	28	01	E8	96	00	BF	A1	01	00	0D	B8	A1	01
0000000050	E8	8B	00	28	0D	BF	A6	01	B5	00	01	CF	C6	05	01	C3
0000000060	B8	43	01	E8	78	00	BF	A6	01	8B	36	A4	01	B4	00	CD
0000000070	16	3C	30	75	05	83	C6	10	EB	21	3C	31	75	05	83	C6
0000000080	20	EB	18	3C	32	75	05	83	C6	30	EB	0F	83	C6	40	3C
0000000090	33	74	08	B8	6D	01	E8	45	00	EB	CB	B4	00	2C	30	01
00000000A0	C7	80	3D	00	74	ED	B4	41	BB	AA	55	CD	13	72	0A	D1
00000000B0	E9	73	06	81	FB	55	AA	74	08	B8	86	01	E8	1F	00	CD
00000000C0	18	B4	42	BF	AA	01	83	C7	08	83	C6	08	66	8B	1C	66
00000000D0	89	1D	BE	AA	01	CD	13	72	E0	EA	00	7C	00	00	56	53
00000000E0	89	C3	31	F6	B4	0E	8A	00	3C	0A	CD	10	74	03	46	EB
00000000F0	F5	5B	5E	C3	2A											
0000000100	2A	0D	0A	53	69											
0000000110	6D	70	6C	65	20	6D	75	6C	74	69	62	6F	6F	74	20	20
0000000120	6C	6F	61	64	65	72	0D	0A	46	69	6E	64	20	62	6F	6F
0000000130	74	61	62	6C	65	20	70	61	72	74	69	74	69	6F	6E	73
0000000140	3A	0D	0A	53	65	6C	65	63	74	20	70	61	72	74	20	74
0000000150	6F	20	62	6F	6F	74	20	66	72	6F	6D	20	28	70	72	65
0000000160	73	73	20	30	20	2E	2E	20	33	29	0D	0A	57	72	6F	
0000000170	6E	67	20	63	68	6F	69	73	65	2E	20	54	72	79	20	61
0000000180	67	61	69	6E	0D	0A	61	20	64	69	73	6B	20	72	65	61
0000000190	64	20	65	72	72	6F	72	20	6F	63	63	75	72	65	64	0D
00000001A0	0A	30	0D	0A	AE	01	00	00	00	10	00	01	00	00	7C	
00000001B0	00	00	00	00	00	00	00	00	89	8F	52	5A	00	00	80	20
00000001C0	21	00	07	FE	FF	FF	00	08	00	00	00	80	38	01	80	FE
00000001D0	FF	FF	0C	FE	FF	FF	00	88	38	01	00	40	9C	00	00	00
00000001E0	FF	FF	0C	FE	FF	FF	00	C8	D4	01	00	40	9C	00	00	00
00000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AA

Рис. 18: 0 сектор USB-накопителя

Зеленой рамкой выделены индикаторы активных разделов, в данном случае необходимо выставить значение **80h**.

### 3.5.6 Тестирование

После подключения USB - накопителя к компьютеру, необходимо зайти в меню выбора устройства, с которого необходимо произвести загрузку.

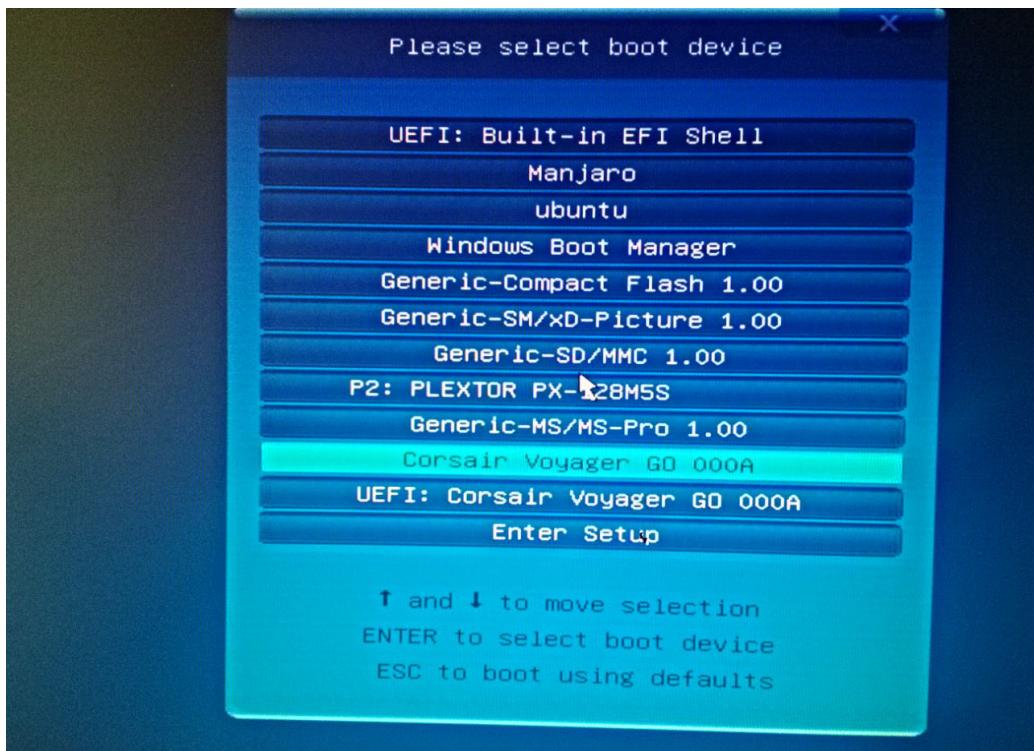


Рис. 19: Выбор устройства для загрузки

В данном случае это **Corsair Voyager GO 000A**.

Далее следует меню мультизагрузчика, который успешно определил три активных сектора.

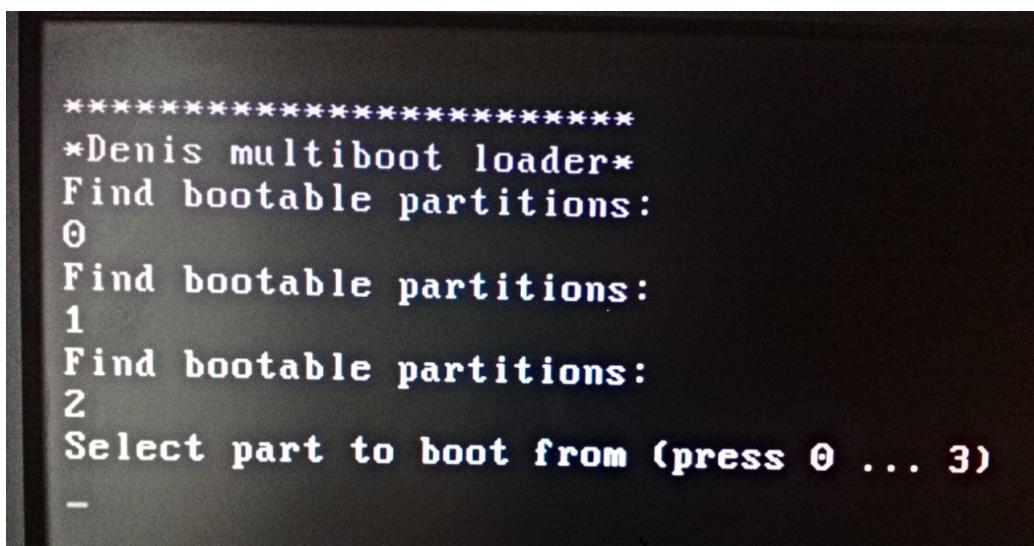


Рис. 20: Меню мультизагрузчика

Выберем 0 раздел, после чего будет загружен вторичный загрузчик Yumi.



Рис. 21: Меню Yumi

Выберем пункт **Linux Distributions**, после чего будет открыто окно выбора ОС.

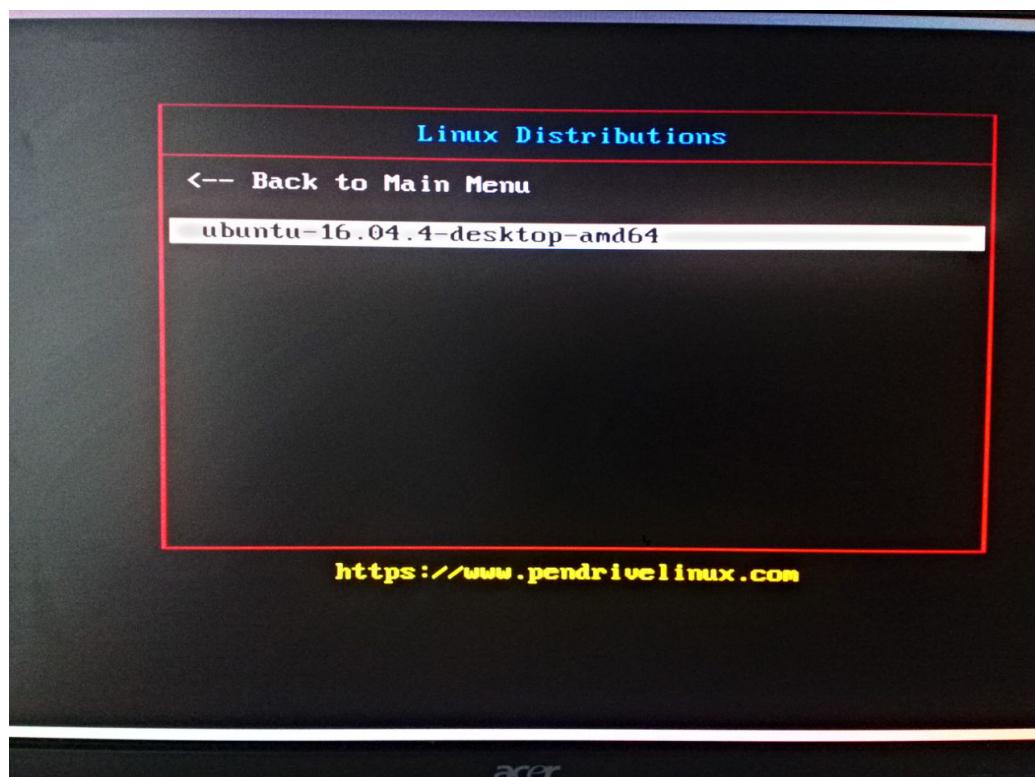


Рис. 22: Меню выбора ОС

При выборе Ubuntu, начнется загрузка ОС.

Если выбрать 1 раздел, то будет загружен DrWeb.

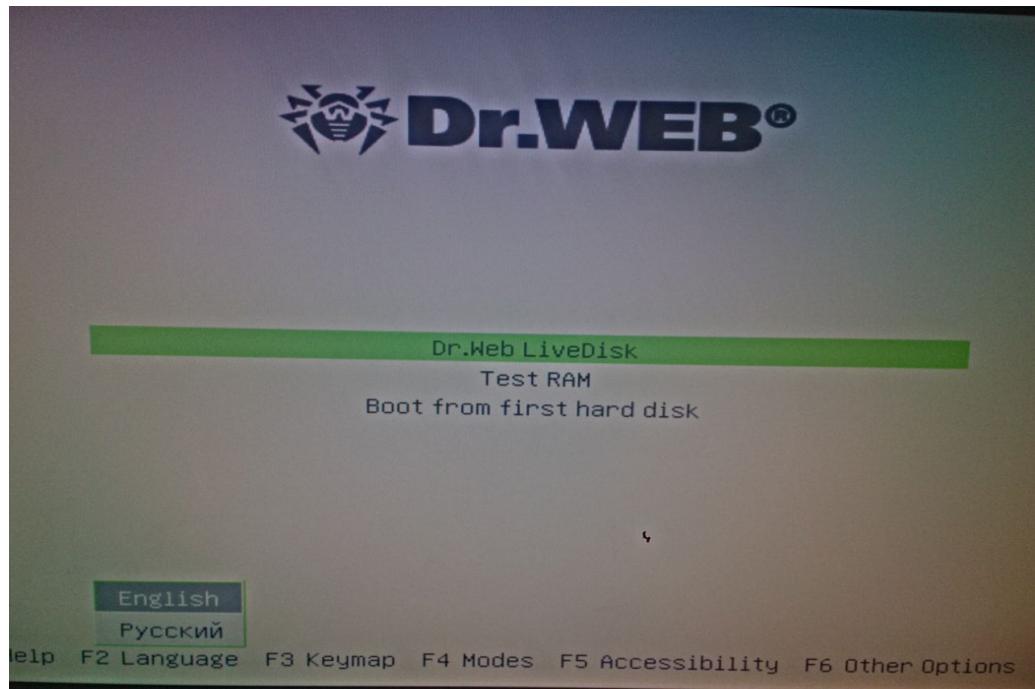


Рис. 23: Приветственный экран DrWeb

И наконец, если выбрать 2 раздел, то будет выведена "мордочка".

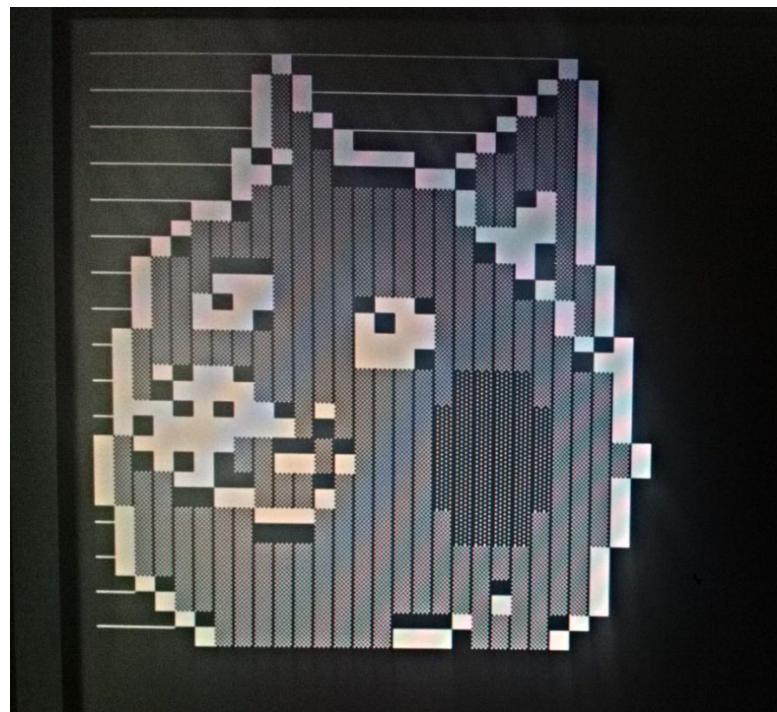


Рис. 24: "Мордочка"

## 3.6 Первичный загрузчик ОС

### 3.6.1 Алгоритм работы

В качестве ОС выступает ms dos версии 6.22, сам первичный загрузчик написан на основе кода из гитхаба[6] в котором был дизассемблирован загрузчик DOS.

Алгоритм работы:

1. Копирование таблицы параметров дискеты (DPT), на которую указывает вектор прерывания 1E;
2. Изменение копии DPT;
3. Изменение вектора прерывания, чтобы он указывал на измененную копию DPT;
4. Сброс контроллера дискеты – с помощью прерывания int 13;
5. Вычисление адреса корневого каталога дискеты;
6. Чтение первого секторы корневого каталога по адресу 0000:0500h;
7. Проверка, что первые два элемента в корневом каталоге – это файлы IO.SYS и MSDOS.SYS;
8. Чтение первых трех секторов IO.SYS по адресу 0070:0000h;
9. Передача управления на адрес 700h.

Исходный код приведен в приложении 4, листинге 11.

### 3.6.2 Подготовка

При создании загрузочной дискеты, необходимо корректно записать файлы операционной системы. В частности первыми файлами в корневой директории должны быть **IO.SYS** и **MSDOS.SYS**, в противном случае система не загрузится.

На рисунке далее, приведена корневая директория уже подготовленной дискеты:

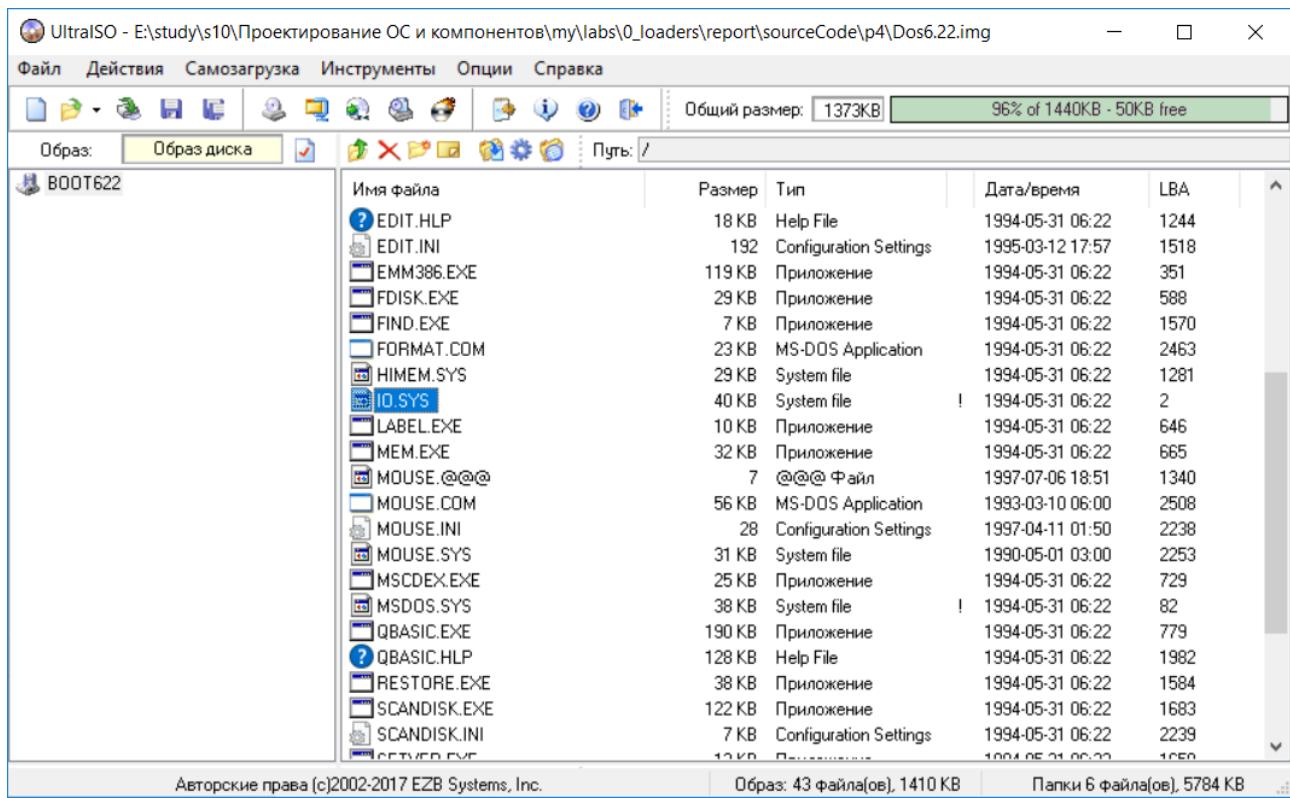


Рис. 25: Корень дискеты

После создания образа дискеты, в загрузочный сектор с помощь редактора **HxD** был занесен код загрузчика:

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
000000000	EB	3C	90	20	20	20	20	20	20	20	00	02	01	01	00	ÿ<þ .....
000000010	02	E0	00	40	0B	F0	09	00	12	00	02	00	00	00	00	.a.@.p.....
000000020	00	00	00	00	00	29	00	00	00	00	20	20	20	20	20	.....)....
000000030	20	20	20	20	20	46	41	54	31	32	20	20	20	FA	31	FAT12 Ѳ1
000000040	C0	8E	D0	BC	00	7C	16	07	BB	78	00	36	C5	37	1E	56 ARPj. ..»x.6E7.V
000000050	16	53	BF	3E	7C	B9	0B	00	FC	F3	A4	06	1F	C6	45	FE .Si> P..ъуи..ЖЕю
000000060	0F	8B	0E	18	7C	88	4D	F9	89	47	02	C7	07	3E	7C	FB <... €Mд‰G.В.> ы
000000070	CD	13	72	79	31	C0	39	06	13	7C	74	08	8B	0E	13	7C H.ry1A9.. t.<...
000000080	89	0E	20	7C	A0	10	7C	F7	26	16	7C	03	06	1C	7C	13 Ѯ.   . ч&. ... .
000000090	16	1E	7C	03	06	0E	7C	83	D2	00	A3	4D	7C	89	16	4F .. ... ѓT.JM ‰.O
0000000A0	7C	A3	49	7C	89	16	4B	7C	B8	20	00	F7	26	11	7C	8B  JI ќ.K ѓ .ч&. <
0000000B0	1E	0B	7C	01	D8	48	F7	F3	01	06	49	7C	83	16	4B	7C .. ..ШНчу..I ѓ.K
0000000C0	00	BB	00	05	8B	16	4F	7C	A1	4D	7C	E8	92	00	72	1D ..».<.О ЎM и'.г.
0000000D0	B0	01	E8	AC	00	72	16	89	DF	B9	0B	00	BE	E6	7D	F3 °.и-.г.ќЯР..sjу
0000000E0	A6	75	0A	8D	7F	20	B9	0B	00	F3	A6	74	18	BE	9E	7D  u.Ќ. Р..у t.sh}
0000000F0	E8	5F	00	31	C0	CD	16	5E	1F	8F	04	8F	44	02	CD	19 и_.ІАН.^Ц.ЦД.Н.
00000100	58	58	EB	E8	8B	47	1A	48	48	8A	1E	0D	7C	30	FF XXXли<G.ННЬ.. Оя	
00000110	F7	E3	03	06	49	7C	13	16	4B	7C	BB	00	07	B9	03	00 чг..I ..K »..R..
00000120	50	52	51	E8	3A	00	72	D8	B0	01	E8	54	00	59	5A	58 PRQи:.xШ°.иT.YZX
00000130	72	BB	83	C0	01	83	D2	00	03	1E	0B	7C	E2	E2	8A	2E r»ѓA.ѓT..... ввЂ.
00000140	15	7C	8A	16	24	7C	8B	1E	49	7C	A1	4B	7C	EA	00	00 .. љ.Ѕ <.I ЎK к..
00000150	70	00	AC	08	C0	74	29	B4	0E	BB	07	00	CD	10	EB	F2 p.-.At)ѓ.»..Н.лт
00000160	3B	16	18	7C	73	19	F7	36	18	7C	FE	C2	88	16	53	7C ;... s.ч6. юВ€.S
00000170	31	D2	F7	36	1A	7C	88	16	25	7C	A3	51	7C	F8	C3	F9 1Tч6. €.% JQ шГщ
00000180	C3	B4	02	8B	16	51	7C	B1	06	D2	E6	0A	36	53	7C	89 Гѓ.<.Q ±.Tж.6S %
00000190	D1	86	E9	8A	16	24	7C	8A	36	25	7C	CD	13	C3	0D	0A Стй.Ѕ љ6% Н.Г..
000001A0	4E	6F	6E	2D	53	79	73	74	65	6D	20	64	69	73	6B	20 Non-System disk
000001B0	6F	72	20	64	69	73	6B	20	65	72	72	6F	72	0D	0A	52 or disk error..R
000001C0	65	70	6C	61	63	65	20	61	6E	64	20	70	72	65	73	73 eplace and press
000001D0	20	61	6E	79	20	6B	65	79	20	77	68	65	6E	20	72	65 any key when re
000001E0	61	64	79	0D	0A	00	49	4F	20	20	20	20	20	53	59 ady...IO SY	
000001F0	53	4D	53	44	4F	53	20	20	53	59	53	00	00	55	AA SMSDOS SYS..UE	

Рис. 26: Загрузочный сектор дискеты

### 3.6.3 Тестирование

Для тестирования была создана виртуальная машина, в настройках которой была добавлена виртуальная дискета, в которой был указан файл дискеты - **Dos6.22.img**.

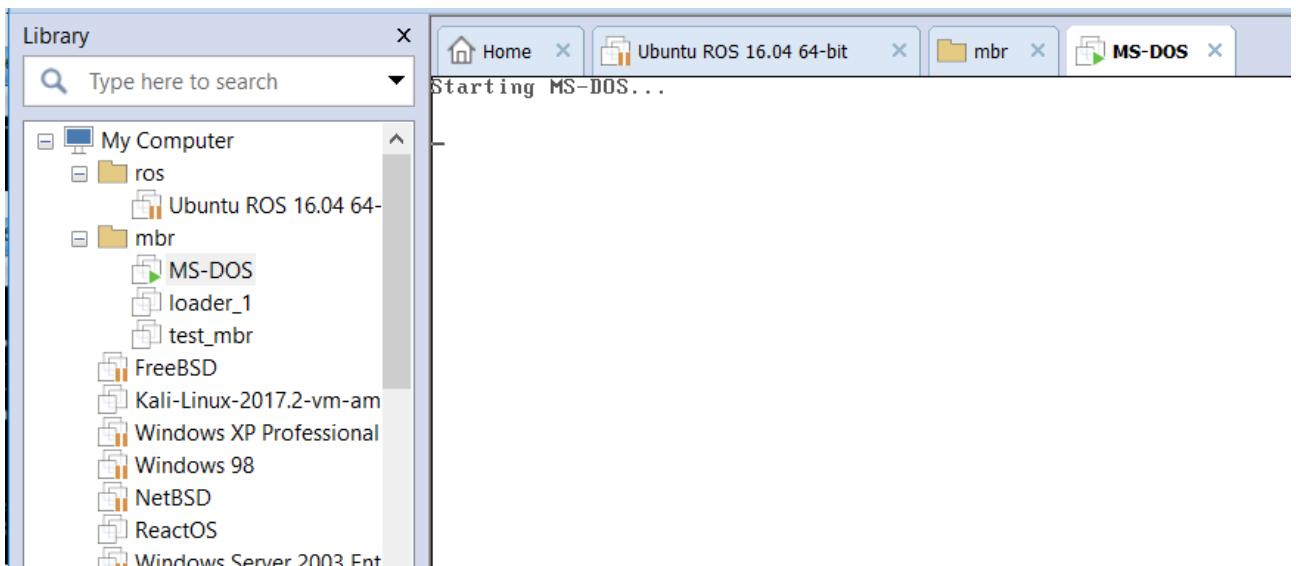


Рис. 27: Сообщение о загрузке MsDOS

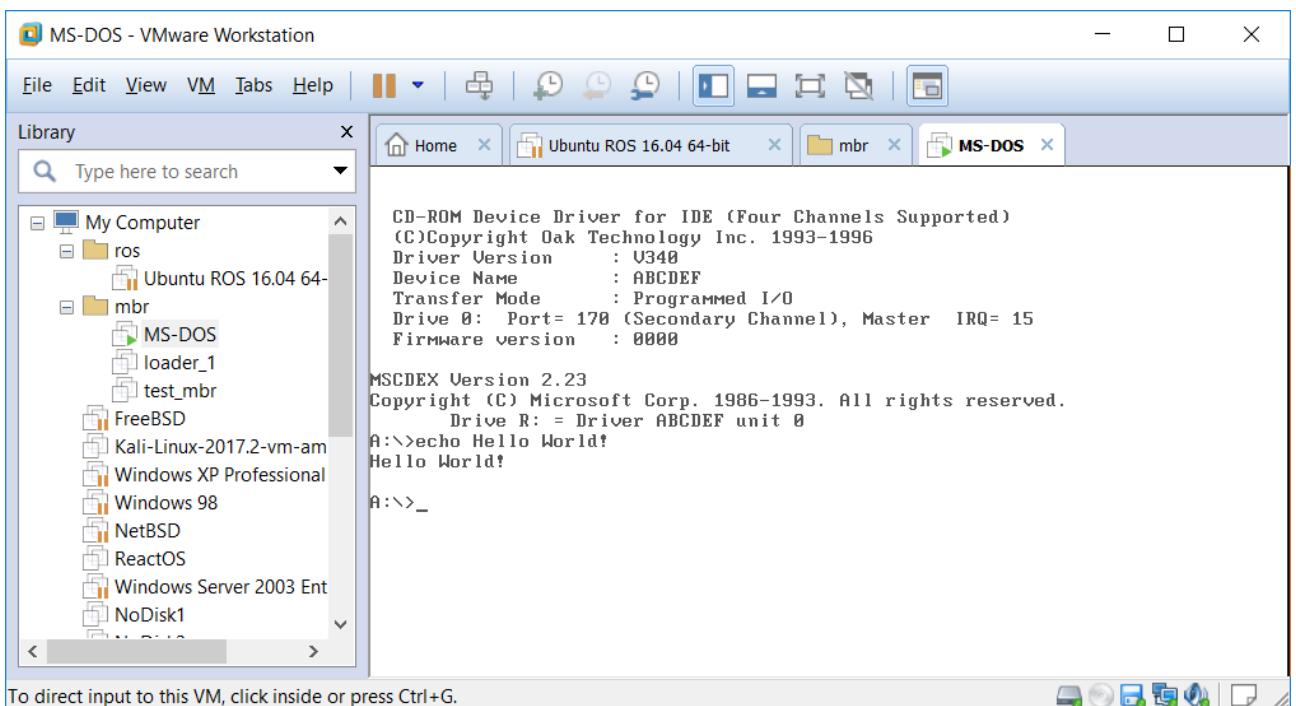


Рис. 28: Загруженная MsDOS

Образ дискеты также прикреплен к отчету.

## **Вывод**

В данной работе были анализированы первичный и вторичный загрузчики, которые впоследствии были реализованы, в том числе при реализации мультизагрузчика.

При реализации мультизагрузчика, корректная запись загружаемой программы, как оказалось, не такая простая задача. Дело в том, что вторичный загрузчик этих программ записывается некорректным образом. Для решения этой проблемы используются специализированные программы, но большинство из них неспособны записать программу(операционную систему) на какой-либо конкретный раздел накопителя, вместо этого предлагается лишь один раздел.

Подобная проблема также возникла при создании образа **MS-DOS**, где необходимо было разместить, файлы для загрузки системы, в определенной последовательности.

Реализованные в данной работе программы, демонстрирует основные принципы функционирования загрузчиков ОС, а также мультизагрузчиков, которые позволяют иметь несколько ОС на одном носителе.

## **Список литературы**

- [1] INT 19H: Bootstrap Loader. – URL: <http://webpages.charter.net/danrollins/techhelp/0243.htm> (дата обращения: 2018-03-04).
- [2] Partition Table Entry. – URL: <http://thestarman.pcministry.com/asm/mbr/PartTables.htm> (дата обращения: 2018-03-04).
- [3] Hexadecimal Numbers Significant to Drive/Partition Limits. – URL: <http://thestarman.pcministry.com/asm/6to64bits.htm> (дата обращения: 2018-03-04).
- [4] Пишем свою ОС: Выпуск 1. – URL: <https://habrahabr.ru/post/101810/> (дата обращения: 2018-03-07).
- [5] Пишем свой загрузочный сектор. – URL: [http://www.compdoc.ru/prog/asm/boot\\_sector/](http://www.compdoc.ru/prog/asm/boot_sector/) (дата обращения: 2018-03-07).
- [6] Disassembly of a Microsoft FAT12/16 volume boot record. – URL: <https://github.com/AceRoqs/Disassemblies/blob/master/fat16-vbr.asm> (дата обращения: 2018-04-30).

## Приложение 1

### Простейший первичный загрузчик

```
section .text
    use16
    org 0x7C00 ; наша программа загружается по адресу 0x7C00
start:
    mov ax, cs
    mov ds, ax ; выбираем сегмент данных

    mov si, message
    cld ; направление для строковых команд
    mov ah, 0x0E ; номер функции BIOS
    mov bh, 0x00 ; страница видеопамяти
puts_loop:
    lodsb ; загружаем очередной символ в al
    test al, al ; нулевой символ означает конец строки
    jz puts_loop_exit
    int 0x10 ; вызываем функцию BIOS
    jmp puts_loop
puts_loop_exit:
    jmp $ ; вечный цикл

message:
    db 'Hello World!',0
finish:
    times 0x1FE-finish+start db 0
    db 0x55, 0xAA ; сигнатура загрузочного сектора
```

Листинг 5: mbr.asm

## Приложение 2

### Первичный загрузчик для устройства с файловой системой FAT-12

```
; Общая часть для всех типов FAT
BS_jmpBoot:
jmp short BootStart ; Переходим на код загрузчика
nop
BS_OEMName db '*v4VIHC' ; 8 байт, что было на моей дискете, то и написал
BPB_BytsPerSec dw 0x200 ; Байт на сектор
BPB_SecPerClus db 1 ; Секторов на кластер
```

```

BPB_RsvdSecCnt dw 1      ; Число резервных секторов
BPB_NumFATs db 2       ; Количество копий FAT
BPB_RootEntCnt dw 224    ; Элементов в корневом каталоге (max)
BPB_TotSec16    dw 2880   ; Всего секторов или 0
BPB_Media      db 0xF0   ; код типа устройства
BPB_FATsz16   dw 9       ; Секторов на элемент таблицы FAT
BPB_SecPerTrk  dw 18     ; Секторов на дорожку
BPB_NumHeads   dw 2       ; Число головок
BPB_HiddSec    dd 0       ; Скрытых секторов
BPB_TotSec32   dd 0       ; Всего секторов или 0

; Заголовок для FAT12 и FAT16
BS_DrvNum     db 0       ; Номер диска для прерывания int 0x13
BS_ResNT      db 0       ; Зарезервировано для Windows NT
BS_BootSig     db 29h    ; Сигнатура расширения
BS_VolID       dd 2a876CE1h  ; Серийный номер тома
BS_VolLab      db 'X boot disk' ; 11 байт, метка тома
BS_FilSysType  db 'FAT12'   ; 8 байт, тип ФС
; Структура элемента каталога
struc  DirItem
    DIR_Name:  resb 11
    DIR_Attr:   resb 1
    DIR_ResNT:  resb 1
    DIR_CrtTimeTenth resb 1
    DIR_CrtTime:  resw 1
    DIR_CrtDate:  resw 1
    DIR_LstAccDate: resw 1
    DIR_FstClusHi: resw 1
    DIR_WrtTime:   resw 1
    DIR_WrtDate:   resw 1
    DIR_FstClusLow: resw 1
    DIR_FileSize:  resd 1
endstruc ;DirItem

; Наши не инициализированные переменные
; При инициализации они затрут не нужные нам поля заголовка FAT: BS_jmpBoot и ;
    ↳ BS_OEMName
struc  NotInitData
    SysSize:   resd 1 ; Размер системной области FAT
    fails:    resd 1 ; Число неудачных попыток при чтении
    fat:      resd 1 ; Номер загруженного сектора с элементами FAT
endstruc ;NotInitData

; По этому адресу мы будем загружать загрузчик

```

```

#define SETUP_ADDR 0x1000
; А по этому адресу нас должны были загрузить
#define BOOT_ADDR 0x7C00
#define BUF 0x500

BootStart:
    cld
    xor cx, cx
    mov ss, cx
    mov es, cx
    mov ds, cx
    mov sp, BOOT_ADDR
    mov bp, sp
    ; Сообщим о том что мы загружаемся
    mov si, BOOT_ADDR + mLoading
    call print

    mov al, [byte bp+BPB_NumFATs]
    cbw
    mul word [byte bp+BPB_FATsz16]
    add ax, [byte bp+BPB_HiddSec]
    adc dx, [byte bp+BPB_HiddSec+2]
    add ax, [byte bp+BPB_RsvdSecCnt]
    adc dx, cx
    mov si, [byte bp+BPB_RootEntCnt]
    ; dx:ax – Номер первого сектора корневого каталога
    ; si – Количество элементов в корневом каталоге
    pusha
    ; Вычислим размер системной области FAT = резервные сектора +
    ; все копии FAT + корневой каталог
    mov [bp+SysSize], ax ; осталось добавить размер каталога
    mov [bp+SysSize+2], dx
    ; Вычислим размер корневого каталога
    mov ax, 32
    mul si
    ; dx:ax – размер корневого каталога в байтах, а надо в секторах
    mov bx, [byte bp+BPB_BytsPerSec]
    add ax, bx
    dec ax
    div bx
    ; ax – размер корневого каталога в секторах
    add [bp+SysSize], ax ; Теперь мы знаем размер системной
    adc [bp+SysSize+2], cx ; области FAT, и начало области данных
    popa
    ; В dx:ax – снова номер первого сектора корневого каталога

```

```
; si – количество элементов в корневом каталоге
```

#### NextDirSector:

```
; Загрузим очередной сектор каталога во временный буфер  
mov bx, 700h ; es:bx – буфер для считываемого сектора  
mov di, bx ; указатель текущего элемента каталога  
mov cx, 1 ; количество секторов для чтения  
call ReadSectors  
jc near DiskError ; ошибка при чтении
```

#### RootDirLoop:

```
; Ищем наш файл  
; cx = 0 после функции ReadSectors  
cmp [di], ch ; byte ptr [di] = 0?  
jz near NotFound ; Да, это последний элемент в каталоге  
; Нет, не последний, сравним имя файла  
pusha  
mov cl, 11 ; длина имени файла с расширением  
mov si, BOOT_ADDR + LoaderName ; указатель на имя искомого файла  
rep cmpsb ; сравниваем  
popa  
jz short Found ; Нашли, выходим из цикла  
; Нет, ищем дальше  
dec si ; RootEntCnt  
jz near NotFound ; Это был последний элемент каталога  
add di, 32 ; Переходим к следующему элементу каталога  
; bx указывает на конец прочтенного сектора после call ReadSectors  
cmp di, bx ; Последний элемент в буфере?  
jb short RootDirLoop ; Нет, проверим следующий элемент  
jmp short NextDirSector ; Да последний, загрузим следующий сектор
```

#### Found:

```
; Загрузка загрузчика  
mov bx, SETUP_ADDR  
mov ax, [byte di+DIR_FstClusLow] ; Номер первого кластера файла  
; Загружаем сектор с элементами FAT, среди которых есть FAT[ax]  
; LoadFAT сохраняет значения всех регистров  
call LoadFAT
```

#### ReadCluster:

```
; ax – Номер очередного кластера  
; Загрузим его в память  
push ax  
; Первые два элемента FAT служебные
```

```

dec ax
dec ax
; Число секторов для чтения
; cx = 0 после ReadSectors
mov cl, [byte bp+BPB_SecPerClus] ; Секторов на кластер
mul cx
; dx:ax – Смещение кластера относительно области данных
add ax, [byte bp+SysSize]
adc dx, [byte bp+SysSize+2]
; dx:ax – Номер первого сектора требуемого кластера
; cx еще хранит количество секторов на кластер
; es:bx – конец прошлого кластера и начало нового
call ReadSectors ; читаем кластер
jc near DiskError ; Увы, ошибка чтения
pop ax ; Номер кластера
; Это конец файла?
; Получим значение следующего элемента FAT
pusha
; Вычислим адрес элемента FAT
mov bx, ax
shl ax, 1
add ax, bx
shr ax, 1
; Получим номер сектора, в котором находится текущий элемент FAT
cwd
div word [byte bp+BPB_BytsPerSec]
cmp ax, [bp+fat] ; Мы уже читали этот сектор?
popa
je Checked ; Да, читали
; Нет, надо загрузить этот сектор
call LoadFAT

```

Checked:

```

; Вычислим адрес элемента FAT в буфере
push bx
mov bx, ax
shl bx, 1
add bx, ax
shr bx, 1
and bx, 511 ; остаток от деления на 512
mov bx, [bx+0x700] ; а вот и адрес
; Извлечем следующий элемент FAT
; В FAT16 и FAT32 все немного проще :(
test al, 1

```

```

jnz odd
and bx, 0xFFFF
jmp short done

odd:
    shr bx, 4

done:
    mov ax, bx
    pop bx
    ; bx – новый элемент FAT
    cmp ax, 0xFF8    ; EOF – конец файла?
    jb ReadCluster ; Нет, читаем следующий кластер
    ; Наконецто– загрузили
    mov ax, SETUP_ADDR>>4    ; SETUP_SEG
    mov es, ax
    mov ds, ax
    ; Передаем управление, наше дело сделано :)
    jmp SETUP_ADDR>>4:0

LoadFAT ; proc
; Процедура для загрузки сектора с элементами FAT
; Элемент ах должен находиться в этом секторе
; Процедура не должна менять никаких регистров
    pusha
    ; Вычисляем адрес слова содержащего нужный элемент
    mov bx, ax
    shl ax, 1
    add ax, bx
    shr ax, 1
    cwd
    div word [byte bp+BPB_BytsPerSec]
    ; ax – смещение сектора относительно начала таблицы FAT
    mov [bp+fat], ax    ; Запомним это смещение, dx = 0
    cwd      ; dx:ax – номер сектора, содержащего FAT[?]
    ; Добавим смещение к первой копии таблицы FAT
    add ax, [byte bp+BPB_RsvdSecCnt]
    adc dx, 0
    add ax, [byte bp+BPB_HiddSec]
    adc dx, [byte bp+BPB_HiddSec+2]
    mov cx, 1    ; Читаем один сектор. Можно было бы и больше, но не быстрее
    mov bx, 700h    ; Адрес буфера
    call ReadSectors
    jc DiskError    ; Ошибочка вышла
    popa

```

```

    ret

; *****
; *      Чтение секторов с диска          *
; *****
; * Входные параметры:                  *
; * dx:ax      - (LBA) номер сектора      *
; * cx         - количество секторов для чтения  *
; * es:bx      - адрес буфера            *
; *****
; * Выходные параметры:                *
; * cx         - Количество не прочтенных секторов  *
; * es:bx      - Указывает на конец буфера      *
; * cf = 1     - Произошла ошибка при чтении      *
; *****

ReadSectors ;proc
next_sector:
    ; Читаем очередной сектор
    mov byte [bp+fails], 3 ; Количество попыток прочесть сектор

try:
    ; Очередная попытка
    pusha
    ; Преобразуем линейный адрес в CSH
    ; dx:ax = a1:a0
    xchg ax, cx ; cx = a0
    mov ax, [byte bp+BPB_SecPerTrk]
    xchg ax, si ; si = Scnt
    xchg ax, dx ; ax = a1
    xor dx, dx
    ; dx:ax = 0:a1
    div si ; ax = q1, dx = c1
    xchg ax, cx ; cx = q1, ax = a0
    ; dx:ax = c1:a0
    div si ; ax = q2, dx = c2 = c
    inc dx ; dx = Sector?
    xchg cx, dx ; cx = c, dx = q1
    ; dx:ax = q1:q2
    div word [byte bp+BPB_NumHeads] ; ax = C (track), dx = H
    mov dh, dl ; dh = H
    mov ch, al
    ror ah, 2
    or cl, ah
    mov ax, 0201h ; ah=2 – номер функции, al = 1 сектор

```

```

        mov dl, [byte bp+BS_DrvNum]
int 13h
popa
jc Failure ; Ошибка при чтении
; Номер следующего сектора
inc ax
jnz next
inc dx

next:
add bx, [byte bp+BPB_BytsPerSec]
dec cx ; Все сектора прочтены?
jnz next_sector ; Нет, читаем дальше

return:
ret

Failure:
dec byte [bp+fails] ; Последняя попытка?
jnz try ; Нет, еще раз
; Последняя, выходим с ошибкой
stc
ret
; ReadSectors    endp

;
; Сообщения об ошибках
NotFound: ; Файл не найден
        mov si, BOOT_ADDR + mLoaderNotFound
        call print
        jmp short die

DiskError: ; Ошибка чтения
        mov si, BOOT_ADDR + mDiskError
        call print
        ;jmp short die

die: ; Просто ошибка
        mov si, BOOT_ADDR + mReboot
        call print

_die: ; Бесконечный цикл, пользователь сам нажмет Reset
        jmp short _die
; Процедура вывода ASCIIZ строки на экран

```

```

; ds:si - адрес строки

print: ; proc
    pusha
print_char:
    lodsb    ; Читаем очередной символ
    test    al, al ; 0 - конец?
    jz short pr_exit ; Да конец
    ; Нет, выводим этот символ
    mov ah, 0eh
    mov bl, 7
    int 10h
    jmp short print_char ; Следующий
pr_exit:
    popa
    ret

#define endl 10,13,0

; Строковые сообщения
mLoading db 'Loading...', endl
mDiskError db 'Disk I/O error', endl
mLoaderNotFound db 'Loader not found', endl
mReboot db 'Reboot system', endl

; Выравнивание размера образа на 512 байт
times 499-($-$) db 0
LoaderName db 'BOOTOR' ; Имя файла загрузчика
BootMagic dw 0xAA55 ; Сигнатура загрузочного сектора

```

Листинг 6: fat12.asm

## Вторичный загрузчик

```

; Загрузочный сектор должен был загрузить нас
; по адресу 0:0x1000.

BOOTOR_MAGIC equ 0x12EF ; Сигнатура

org 0x1000
base:

; Определимся с регистрами:
    xor ax, ax
    mov es, ax
    mov ds, ax

```

```

        mov ss, ax
        mov sp, 0x1000

; Установим видеорежим, очистить экран
        mov ax, 3
        int 10h

; Вообщим о том что мы загрузились
        cmp word [BOOTOR_SIG], BOOTOR_MAGIC
        je near GoodSig

; Загрузочный сектор неправильно нас загрузил
        mov si, mFAIL
        call print
        mov si, mBadSig
        call print

die:    jmp short die

mBadSig db 'Major problem: Loader did not load kernel completely.',10,13
        db 'Reboot computer',10,13,0
mOK db '[ OK ]',10,13,0
mFAIL db '[FAIL]',10,13,0
mHello db 'Hello from bootor',10,13,0

;

print:
        cld
        pusha
.PrintChar:
        lodsb
        test al, al
        jz short .Exit
        mov ah, 0eh
        mov bl, 7
        int 10h
        jmp short .PrintChar
.Exit:
        popa
        ret
;

GoodSig:    mov si, mOK
            call print

```

```

    mov si, mHello
    call    print

    jmp die

;

BOOTOR_SIG dw BOOTOR_MAGIC
align    16

```

Листинг 7: bootor.asm

## Приложение 3

### Программа по выводу "мордочки" на экран

```

section .text
use16
org 0x7C00 ; наша программа загружается по адресу 0x7C00
start:
    mov ax, 3
    int 10h

    mov ax, cs
    mov ds, ax ; выбираем сегмент данных

    mov si, message
    cld ; направление для строковых команд
    mov ah, 0xE ; номер функции BIOS
    mov bh, 0x00 ; страница видеопамяти
puts_loop:
    lodsb ; загружаем очередной символ в al
    test al, al ; нулевой символ означает конец строки
    jz puts_loop_exit
    int 0x10 ; вызываем функцию BIOS
    jmp puts_loop
puts_loop_exit:
    jmp $ ; вечный цикл

message:
    db 196,196,196,196,196,196,196,196,196,196,196,196,196,196,196,
    ↳ 196,196,196,196,196,196,220,13,10,196,196,196,196,196,196,196,219,
    ↳ 177,219,196,196,196,196,196,196,196,196,196,220,223,177,219,13,10,

```

```

→ 196,196,196,196,196,196,196,219,177,177,223,220,196,196,196,196,
→ 196,220,223,177,177,219,13,10,196,196,196,196,196,196,219,220,
→ 223,177,177,223,223,223,223,220,220,223,177,177,177,177,177,219,13,10,
→ 196,196,196,196,196,220,220,223,177,177,177,177,177,177,177,177,177,
→ 219,177,177,220,219,177,219,13,10,196,196,196,220,223,177,177,177,177,
→ 177,177,177,177,177,177,177,177,177,223,219,219,223,177,219,13,10,
→ 196,196,219,177,177,220,220,220,177,177,177,177,177,177,177,177,177,
→ 177,177,177,223,220,177,177,219,13,10,196,196,219,177,177,219,220,219,
→ 223,177,177,177,177,220,223,219,220,177,177,177,177,177,219,177,219,
→ 13,10,196,219,177,177,177,177,177,177,177,177,177,177,219,219,219,
→ 223,177,177,177,177,177,177,177,223,220,219,13,10,196,219,177,223,220,
→ 219,219,220,177,177,177,177,177,177,177,177,177,177,176,176,176,177,
→ 177,177,177,219,13,10,196,219,223,219,220,219,219,220,177,223,
→ 177,177,177,177,177,176,176,176,176,176,177,177,177,219,13,10,219,
→ 177,223,219,223,219,223,177,177,220,220,177,220,177,177,177,177,176,176,
→ 176,176,176,176,177,177,177,177,219,13,10,219,177,177,177,223,223,220,
→ 220,177,177,177,220,177,177,177,177,176,176,176,176,176,176,177,177,
→ 177,219,13,10,196,219,177,177,177,177,177,223,223,223,177,177,177,
→ 177,177,177,177,176,176,176,176,177,177,177,177,219,13,10,196,219,177,
→ 177,177,177,177,177,177,177,177,177,177,177,177,177,177,177,177,177,
→ 177,177,177,177,219,13,10,196,196,223,220,177,177,177,177,177,177,177,
→ 177,177,177,177,177,177,177,177,220,177,177,177,177,219,13,10,196,
→ 196,196,196,223,220,177,177,177,177,177,177,177,177,220,220,220,
→ 223,177,177,177,177,220,223,0

```

```

finish:
    times 0x1FE-finish+start db 0
    db    0x55, 0xAA ; сигнитура загрузочного сектора

```

Листинг 8: doge.asm

### Алгоритм по преобразованию ascii символов в их код

```

f = open('ascii.txt', 'rU', encoding='utf8')
asciiDic = { ' ': "32,", "'': "196,", '\': "219,", '\"': "178,", '\"': "176,", '\n': "
→ 13,10,", '\"': "220,", '\"': "177,", '\"': "223,"}
while True:
    c = f.read(1)
    if not c:
        break
    print(asciiDic[c], end=' ')

```

Листинг 9: readFile.py

К сожалению, в отчет не удалось прикрепить файл, где расположена "мордочка" для вы-

вода на дисплей, из-за невозможности отображения дополнительных символов ascii, что так-же можно заметить в листинге выше.

## Мультизагрузчик

```
use16

===== Копируем тело загрузчика по адресу 0000:0600h =====
    mov ax, 7C00h ; по этому адресу мы загружены
    mov ds, ax      ; настраиваем регистры – стек, сегменты и т п..
    xor si, si

    mov ax, 60h ; по этому адресу копируемся
    mov es, ax
    xor di, di

    mov cx, 0FFh      ; сколько байт копировать

    rep movsw ; копируем

    jmp 0000:0618h ; Передаем управление на новое расположение кода –
    ↳ следующая за jmp инструкция

===== Приветствуем пользователя =====
    mov ax, hello_msg_1
    call print
    mov ax, hello_msg_2
    call print

===== Проверяем таблицу разделов =====
    mov si, [part_adr] ; начало таблицы разделов
    mov bh, 80h ; загрузочный раздел – признак

    mov cl, -1 ; настройка счетчика разделов
partitions_chek:
    cmp cl, 3 ; если уже было проверено 4 записи, выходим из цикла и
    ↳ переходим к обработке записей
    je partition_select

    add si, 10h ; настраиваем на адрес следующей записи таблицы
    inc cl ; увеличиваем число обработанных записей

    mov bl, [es:si] ; смотрим на признак раздела
    cmp bl, bh
```

```

jne partitions_chek           ; запись не является загрузочной

call partitions_process       ; запись загрузочная – выведем ее номер и
→ поясняющее сообщение
    ; [es:si] содержит адрес записи в таблице разделов
    ; cl – номер раздела

jmp partitions_chek           ; проверяем еще

===== Подпрограмма вывода информации об активном разделе
→ =====

partitions_process:
    mov ax, boot_part_msg ; поясняющее сообщение
    call print

    mov di, part_num      ; порядковый номер раздела от 0 до 3
    add [ds:di], cl
    mov ax, part_num
    call print
    sub [ds:di], cl

    mov di, boot_flags
    mov ch, 0
    add di, cx
    mov byte[ds:di], 1

    ret;
=====

partition_select:
    ; Обрабатываем пользовательский ввод с помощью прерывания 16
    mov ax, select_part_msg ; просьба выбрать раздел
    call print

choise:   mov di, boot_flags      ; настройка регистров для получения результата
→ ввода
    mov si, [part_adr]

    mov ah, 0
    int 16h                 ; ждем ввода

p0:    cmp al, 48                ; сравниваем коды нажатой клавиши с 0, 1, 2 и 3 –
→ определяем выбранный раздел
    jne p1
    add si, 10h

```

```

        jmp disk

p1:  cmp al, 49          ; выбран первый раздел?
      jne p2
      add si, 20h
      jmp disk

p2:  cmp al, 50          ; выбран второй раздел?
      jne p3
      add si, 30h
      jmp disk

p3:  add si, 40h          ; выбран третий раздел?
      cmp al, 51
      je disk

wrong_choise:           ; была введена ерунда
      mov ax, wrong_input_msg ; сообщаем об этом и ждем нового ввода
      call print
      jmp choise

disk:   mov ah, 0
        sub al, 48          ; сначала проверка, выбрал ли пользователь
        → действительно загрузочный раздел
        add di, ax
        cmp byte [ds:di], 0
        je wrong_choise
              ; по [es:si] содержится запись таблицы разделов
              ; о выбранном загрузочном диске

        mov ah, 41h          ; проверка поддержки диском расширенного режима (> 8 GB)
        ; dl содержит номер диска
        mov bx, 55AAh        ; проверяем; через прерывание
        int 13h
        jc ext_not_present_error ; ошибка при чтении – выводим сообщение
        shr cx, 1            ; считали один сектор?
        jnb ext_not_present_error
        cmp bx, 0AA55h       ; все хорошо – будем грузить в нужную область
        → памяти
        je read_boot_sect

ext_not_present_error:    ; ошибка при чтении диска
        mov ax, ext_not_pres_msg
        call print
        int 18h

```

```

read_boot_sect:
    mov ah, 42h      ; производим настройку регистров для чтения сектора
    → выбранного раздела
    mov di, DAP_structure
    add di, 8
    add si, 8
    mov ebx, [ds:si]
    mov [ds:di], ebx
    mov si, DAP_structure ; читаем диск
    int 13h
    jc ext_not_present_error

    jmp 0000:7C00h      ; передаем управление на вторичный загрузчик

;===== Подпрограмма вывода сообщений ======
print:
    push si
    push bx

    mov bx, ax
    xor si, si
    mov ah, 0Eh

p:   mov al, [bx + si]
    cmp al, 0Ah
    int 10h

    je end_print

    inc si
    jmp p

end_print:
    pop bx
    pop si
    ret

;=====
hello_msg_1 db '*****', 0Dh, 0Ah
hello_msg_2 db '*Denis multiboot loader*', 0Dh, 0Ah
boot_part_msg db 'Find bootable partitions:', 0Dh, 0Ah
select_part_msg db 'Select part to boot from', 0Dh, 0Ah
wrong_input_msg db 'Wrong choise. Try again', 0Dh, 0Ah
ext_not_pres_msg db 'a disk read error occured', 0Dh, 0Ah

```

## Листинг 10: MultiBoot.ASM

## **Приложение 4**

## Первичный загрузчик ОС

```
; Disassembly of a Microsoft FAT12/16 volume boot record, by Toby Jones.  
; Disassembled around 16 March 2001.  
  
; This code is valid for DOS boot sectors, though it is likely that modern  
; boot sectors simply display an error message without attempting to load IO.  
    → SYS.  
  
struc direntry  
    .Name          resb 8  
    .Extension     resb 3  
    .Attributes    resb 1  
    .Reserved      resb 10  
    .Time          resw 1  
    .Date          resw 1  
    .StartCluster   resw 1  
    .FileSize      resd 1  
endstruc  
  
;  
org 7c00h  
  
start:  
    jmp     short after_data  
    nop  
  
; BIOS Parameter Block —————  
  
%define MEDIA_DESCRIPTOR_FLOPPY144_DISK 0f0h  
  
struc bpb_fat16  
    .OEMName        resb 8  
    .BytesPerSector resw 1
```

```

    . SectorsPerCluster resb 1
    . ReservedSectors   resw 1
    . FATs              resb 1
    . RootEntries       resw 1
    . Sectors           resw 1
    . MediaDescriptor   resb 1
    . FATSectors        resw 1
    . SectorsPerTrack   resw 1
    . Heads              resw 1
    . HiddenSectors      resd 1
    . HugeSectors        resd 1
    . DriveNumber        resb 1
    . Reserved           resb 1
    . BootSignature      resb 1
    . VolumeID           resd 1
    . VolumeLabel         resb 11
    . FileSystemType     resb 8
endstruc

bpb:
istruc bpboff16
    at bpboff16.OEMName,          db  'MS-DOS'
    at bpboff16.BytesPerSector,   dw 200h
    at bpboff16.SectorsPerCluster, db 1
    at bpboff16.ReservedSectors,  dw 1
    at bpboff16.FATs,             db 2
    at bpboff16.RootEntries,      dw 0e0h
    at bpboff16.Sectors,          dw 0b40h
    at bpboff16.MediaDescriptor,  db MEDIA_DESCRIPTOR_FLOPPY144_DISK
    at bpboff16.FATSectors,       dw 9
    at bpboff16.SectorsPerTrack,  dw 12h
    at bpboff16.Heads,            dw 2
    at bpboff16.HiddenSectors,    dd 0
    at bpboff16.HugeSectors,      dd 0
    at bpboff16.DriveNumber,      db 0
    ; Reserved is used by the boot code to store the next head number to read
    ; from.
    at bpboff16.Reserved,         db 0
    at bpboff16.BootSignature,    db 29h ; Indicates the following three
    ; fields are valid.
    at bpboff16.VolumeID,         dd 0
    at bpboff16.VolumeLabel,       db  'VolumeLabel'
    at bpboff16.FileSystemType,   db 'FAT12'
iend

```

```

; Disk Parameter Table
struc dpt
    .HeadUnloadTime      resb 1
    .HeadLoadTime        resb 1
    .MotorDelayOff       resb 1
    .BytesPerSector      resb 1
    .SectorsPerTrack     resb 1
    .SectorGapLength     resb 1
    .DataLength          resb 1
    .FormatGapLength     resb 1
    .FormatByteValue     resb 1
    .HeadSettlingTime    resb 1
    .MotorDelayOn         resb 1
endstruc

struc bss_data
    .free_space_start    resd 1
    .root_dir_start      resd 1
    .cylinder             resw 1
    .sector               resb 1
endstruc

;

after_data:
    cli                      ; Set up a standard stack frame at 0:7c00.
    xor ax,ax
    mov ss,ax
    mov sp,start

    push ss                  ; The int 1eh vector is a pointer to the Disk
    → Parameter Table.
    pop es
    mov bx,1eh * 4            ; Real mode interrupt vector table (IVT)
    → entries are 4 bytes each.
    lds si,[ss:bx]

    push ds                  ; The Disk Parameter Table (DPT) is likely in
    → ROM,
    push si                  ; so copy it out to make it mutable.
    push ss
    push bx
    mov di,after_data         ; Copy DPT over the already executed boot
    → sector code.
    mov cx,dpt_size

```

```

cld
repz    movsb

push    es
pop     ds

; Update DPT with new parameters. Setting the IVT entry must be done with
; interrupts disabled.
mov     byte [di - dpt_size + dpt.HeadSettlingTime],0fh ; Set the head
; settling time to 15 ms.
mov     cx,[bpb + bpbfat16.SectorsPerTrack]           ; Store disk's
; sectors per track
mov     [di - dpt_size + dpt.SectorsPerTrack],cl       ; in new DPT.
mov     [bx + 2],ax                                     ; Set new DPT
; pointer in IVT entry
mov     word [bx],after_data                          ; to 0:after_data.

sti
int    13h                                         ; Reset disk with
; new DPT parameters.
jc     restart_boot

xor    ax,ax
cmp    [bpb + bpbfat16.Sectors],ax                 ; If the number of
; sectors is non-zero,
jz     .calc_root_directory_start                  ; then use that as
; the huge sector count.
mov    cx,[bpb + bpbfat16.Sectors]                ; The sector count
; and huge sector count are
mov    [bpb + bpbfat16.HugeSectors],cx            ; not otherwise
; used in the boot sector.

.calc_root_directory_start:
mov    al,[bpb + bpbfat16.FATs]                   ; Calculate the
; combined size of the FAT tables
mul    word [bpb + bpbfat16.FATSectors]          ; in sectors.
add    ax,[bpb + bpbfat16.HiddenSectors]         ; Add in the count
; of hidden and reserved sectors
adc    dx,[bpb + bpbfat16.HiddenSectors + 2]    ; to determine the
; start of the root directory.
add    ax,[bpb + bpbfat16.ReservedSectors]
adc    dx,0
mov    [bss_section + bss_data.root_dir_start],ax ; Save the start of
; the root directory. Also save
mov    [bss_section + bss_data.root_dir_start + 2],dx ; the start of free

```

```

→ space, which will be updated
mov      [bss_section + bss_data.free_space_start],ax      ; once the size of
→ the root directory is known.
mov      [bss_section + bss_data.free_space_start + 2],dx

mov      ax,direntry_size                                ; Calculate the
→ size of root directory in bytes.
mul      word [bpb + bpbo_fat16.RootEntries]

mov      bx,[bpbo + bpbo_fat16.BytesPerSector]           ; Calculate the
→ size of root directory in sectors (rounded up).
add      ax,bx
dec      ax
div      bx
add      [bss_section + bss_data.free_space_start],ax      ; Calculate
→ the start of free space, which begins
adc      word [bss_section + bss_data.free_space_start + 2],0      ;
→ immediately after the root directory.

mov      bx,500h
mov      dx,[bss_section + bss_data.root_dir_start + 2]      ; Get the
→ start of the root directory in CHS.
mov      ax,[bss_section + bss_data.root_dir_start]
call    lba_sector_to_chs
jc     restart_boot

mov      al,1                                         ; Read in one sector of the root directory
→ to 0:500.
call    read_sectors
jc     restart_boot

; Verify that the first two directory entries are IO.SYS and MSDOS.SYS.
mov      di,bx                                      ; Point to the first filename and ensure it
→ is IO.SYS.
mov      cx,8 + 3
mov      si,iosys_string
repz   cmpsb
jnz    restart_boot
lea     di,[bx + direntry_size] ; Point to the next filename and ensure it
→ is MSDOS.SYS.
mov      cx,8 + 3
repz   cmpsb
jz     load_io_sys        ; Load IO.SYS.

restart_boot:

```

```

        mov     si ,emsg           ; 'Non-System disk or disk error'...
        call    print_string       ; Print the error string.
        xor    ax ,ax
        int    16h                ; Wait for a keystroke.
        pop    si
        pop    ds
        pop    word [ si ]
        pop    word [ si + 2]
        int    19h                ; Reload the boot sector.
                                ; This call does not return.

;

stackfix_restart_boot:
        pop    ax
        pop    ax
        pop    ax
        jmp    restart_boot

;

; bx->Root directory entry of IO.SYS.

load_io_sys:
        ; This code requires that the first three sectors of IO.SYS be contiguous.
        mov    ax ,[bx + direntry.StartCluster]          ; Get the cluster
        ↳ start of IO.SYS.
        dec    ax
        dec    ax
        mov    bl ,[bpb + bpbofat16.SectorsPerCluster] ; Calculate the
        ↳ sector number of that cluster.
        xor    bh ,bh
        mul    bx
        add    ax ,[bss_section + bss_data.free_space_start] ; Clusters begin at
        ↳ the start of free space.
        adc    dx ,[bss_section + bss_data.free_space_start + 2]
        mov    bx,700h                         ; Set up to read in
        ↳ 3 sectors of IO.SYS to 0:700.
        mov    cx,3

read_next_io_sys_sector:
        push   ax                  ; Save logical sector number (dx:ax)
        ↳ and correct sector count (cx).
        push   dx
        push   cx

```

```

    call    lba_sector_to_chs          ; Convert logical sector to CHS.
    jb     stackfix_restart_boot

    mov     al,1                      ; Read in one sector of IO.SYS.
    call    read_sectors

    pop    cx
    pop    dx
    pop    ax

    jb     restart_boot             ; If read failed , then print error
    ↳ message and reboot.

    add    ax,1                      ; Calculate the
    ↳ next IO.SYS logical sector number.
    adc    dx,0
    add    bx,[bpb + bpbo_fat16.BytesPerSector]      ; Calculate the
    ↳ next read buffer address.
    loop   read_next_io_sys_sector           ; Loop until three
    ↳ sectors are read.

    ; Jump to IO.SYS, passing in parameters.
    mov    ch,[bpb + bpbo_fat16.MediaDescriptor]
    mov    dl,[bpb + bpbo_fat16.DriveNumber]
    mov    bx,[bss_section + bss_data.free_space_start]
    mov    ax,[bss_section + bss_data.free_space_start + 2]
    jmp    0070h:0000                 ; Jump to IO.SYS entry point.

;

; si->The string to print.
print_string:
    lodsb                         ; Get a character.
    or     al,al                   ; If the character is null then return.
    jz     conditional_return

    mov    ah,0eh
    mov    bx,7
    int    10h                     ; Print this character.
    jmp    print_string            ; Go do the next character.

; Convert logical sector to CHS ——————
; dx:ax->Logical sector number to convert.
lba_sector_to_chs:

```

```

        cmp      dx,[bpb + bpb_fat16.SectorsPerTrack]      ; Fail if LBA is too
        ↳ large.
        jnb     .error

        div      word [bpb + bpb_fat16.SectorsPerTrack]      ; Calculate the track
        ↳ number.
        inc      dl                                         ; Sectors are one based
        ↳ .
        mov      [bss_section + bss_data.sector],dl          ; Remainder is the
        ↳ starting sector number.

        xor      dx,dx                                     ; Calculate final head
        ↳ and track/cylinder.
        div      word [bpb + bpb_fat16.Heads]
        mov      [bpb + bpb_fat16.Reserved],dl            ; Save the head number.
        mov      [bss_section + bss_data.cylinder],ax        ; Save the track number
        ↳ .

        clc
        retn

.error:
        stc

conditional_return:
        retn

;

; al->The number of sectors to read.
; es:bx->Pointer to the read buffer.
read_sectors:
        mov      ah,2
        mov      dx,[bss_section + bss_data.cylinder]      ; Put the cylinder in ch,
        ↳ with the high
        mov      cl,6                                       ; bits (8-9) in the top of
        ↳ cl. The lower six
        shl      dh,cl                                     ; bits hold the starting
        ↳ sector number.
        or       dh,[bss_section + bss_data.sector]
        mov      cx,dx
        xchg    ch,cl
        mov      dl,[bpb + bpb_fat16.DriveNumber]        ; Put the head number in dh
        ↳ , and the drive number
        mov      dh,[bpb + bpb_fat16.Reserved]           ; in dl (80h is the first

```

```
    ↳ fixed disk).
    int      13h                      ; Read in the sectors.
    retn

; String Table ——————  
  
emsg:  
    db 0dh, 0ah, 'Non-System disk or disk error'  
    db 0dh, 0ah, 'Replace and press any key when ready', 0dh, 0ah, 0

iosys_string:  
    db 'IO      SYS'  
    db 'MSDOS   SYS'  
    dw 0  
  
    dw 0aa55h  
  
;  
  
; These structures are located after the BPB, and overlays the code that
; has already been executed by the time these structures are initialized.
absolute after_data
    resb dpt_size
bss_section resb bss_data_size
```

Листинг 11: dosBoot.asm