

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №1**  
**по дисциплине «Операционные системы»**  
**Тема: Исследование структур загрузочных модулей**

Студентка гр. 7381

\_\_\_\_\_

Процветкина А.В.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2019

### **Цель работы.**

Исследование различий в структурах исходных текстов модулей .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

### **Основные теоретические положения.**

Тип IBM PC хранится в байте по адресу 0F000:0FFFEh, в предпоследнем байте ROM BIOS. Соответствие байта типу IBM PC представлено в табл. 1.

Таблица 1 – Соответствие байта и типа IBM PC

Значение байта	Тип IBM PC
FF	PC
FE, FB	PC/XT
FC	AT
FA	PS2 модель 30
FC	PS2 модель 50/60
F8	PS2 модель 80
FD	PCjr
F9	PC Convertible

### ***План загрузки в память модулей .COM:***

При загрузке программы типа .COM регистр IP всегда инициализируется числом 100h, поэтому сразу за директивой org 100h должно стоять первое выполнимое предложение программы. После загрузки программы все 4 сегментных регистра указывают на начало единственного сегмента, т. е. фактически на начало PSP. Указатель стека автоматически инициализируется числом FFFEh. Таким образом, независимо от фактического размера программы, ей выделяется 64 Кбайт адресного пространства, всю нижнюю часть которого занимает стек.

### **Постановка задачи.**

Составить исходный .COM модуль, определяющий тип PC и версию системы. Получить "плохой".EXE модуль из программы, предназначенной для COM модуля, после чего построить "хороший" .EXE модуль выполняющий те же функции, что и отлаженный .COM модуль. Сравнить тексты полученных программ и модулей. Ответить на контрольные вопросы.

### **Выполнение работы.**

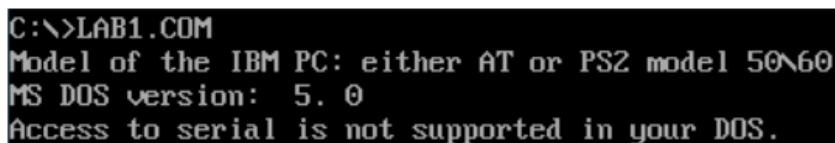
Был написан текст для .COM модуля, определяющий тип PC и версию системы. Ассемблерная программа считывает предпоследний байт ROM BIOS и после сравнения его с имеющимися данными выводит на экран либо идентифицированный тип PC, либо этот самый байт в шестнадцатеричном представлении.

Затем определяется версия системы с помощью вызова функции 30h прерывания 21h, которая имеет результатом следующий набор данных:

1. AL – номер основной версии
2. AH – номер модификации
3. BH – серийный номер OEM (original equipment manufacturer)
4. BL:CH – 24-х битовый серийный номер пользователя.

*Примечание:* для большинства версий DOS значения регистров BH и CH после вызова данной функции равны 0.

Содержимое регистров преобразуется в строковый формат, после чего полученная о системе информация выводится на экран, как показано на рис. 1.



```
C:\>LAB1.COM
Model of the IBM PC: either AT or PS2 model 50\60
MS DOS version: 5. 0
Access to serial is not supported in your DOS.
```

Рисунок 1 – Пример работы программы

Поскольку серийные номера OEM и пользователя недоступны в

эмулированной в работе системе DOS (значения регистров BX и CX равны 0), проверить полную работоспособность программы напрямую не представляется возможным, однако после изменения значений регистров в отладчике TD, можно убедиться в корректности выдаваемого результата, представленного на рис.3, для случайных значений из рис.2.

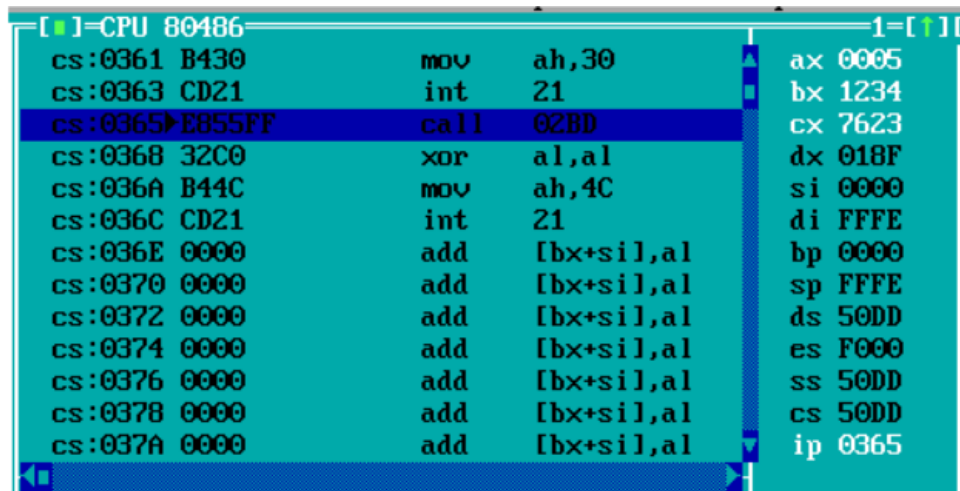


Рисунок 2 – Изменение значений регистров вручную

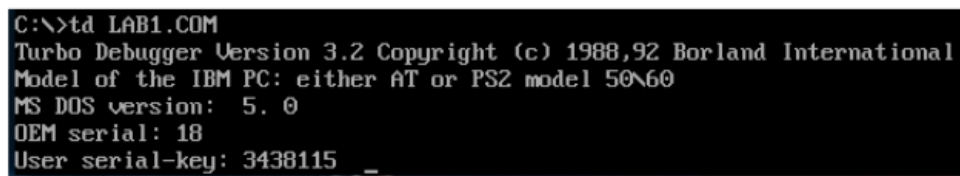


Рисунок 3 – Результат работы программы

Исходный код составленной программы представлен в приложении А.

### Ответы на контрольные вопросы.

*Отличия исходных текстов .COM и .EXE программ*

1. Сколько сегментов должна содержать .COM-программа?

*Ответ:* 1.

2. EXE-программа?

*Ответ:* Обязательно как минимум 1 – сегмент кода, логически их обычно 3: сегмент кода, данных и стека, однако и без двух последних .EXE-программы работают корректно.

3. Какие директивы должны обязательно быть в тексте .COM-программы?

*Ответ:* В программе обязательно должны присутствовать директивы `ORG`, `END`, `ASSUME`. При удалении последней из них наблюдаются следующие ошибки компиляции:



```

**Error** COM_SO~1.ASM(82) Near jump or call to different CS
**Error** COM_SO~1.ASM(87) Near jump or call to different CS
**Error** COM_SO~1.ASM(92) Near jump or call to different CS
**Error** COM_SO~1.ASM(97) Near jump or call to different CS
**Error** COM_SO~1.ASM(102) Near jump or call to different CS
**Error** COM_SO~1.ASM(107) Near jump or call to different CS
**Error** COM_SO~1.ASM(111) Near jump or call to different CS
**Error** COM_SO~1.ASM(136) Near jump or call to different CS
**Error** COM_SO~1.ASM(139) Near jump or call to different CS
**Error** COM_SO~1.ASM(141) Near jump or call to different CS
**Error** COM_SO~1.ASM(153) Near jump or call to different CS
**Error** COM_SO~1.ASM(157) Near jump or call to different CS
**Error** COM_SO~1.ASM(173) Near jump or call to different CS
**Error** COM_SO~1.ASM(178) Near jump or call to different CS
**Error** COM_SO~1.ASM(195) Near jump or call to different CS
**Error** COM_SO~1.ASM(201) Near jump or call to different CS
**Error** COM_SO~1.ASM(228) Near jump or call to different CS
**Error** COM_SO~1.ASM(236) Near jump or call to different CS
Error messages:      26
Warning messages:    None
Passes:              1
Remaining memory:    470k

```

Рисунок 4 – Ошибки компиляции при удалении директивы `ASSUME`

С помощью директивы `ASSUME` ассемблеру сообщается информация о соответствии между сегментными регистрами, и программными сегментами. Все сегменты сами по себе равноправны, для того чтобы использовать их как сегменты кода, данных или стека, необходимо предварительно сообщить транслятору об этом, для чего используют специальную директиву `ASSUME`. Эта директива сообщает транслятору о том, какой сегмент к какому сегментному регистру привязан.

4. Все ли форматы команд можно использовать в .COM-программе?

*Ответ:* Нет, не все. Например, использование директивы `seg` приводит к ошибкам, иллюстрируемых на рис. 5. COM-программа подразумевает наличие только одного сегмента, а значит, можно использовать только `near`-переходы, так как в `far`-переходах подразумевается использование нескольких сегментов.

Так же нельзя использовать команды, связанные с адресом сегмента, потому что адрес сегмента до загрузки неизвестен, так как в COM-программах в DOS не содержится таблицы настройки, которая содержит описание адресов, зависящих от размещения загрузочного модуля в ОП, потому что подобные адреса в нем запрещены.

```
C:\>tlink /t COM_S0~1.OBJ
Turbo Link Version 5.1 Copyright (c) 1992 Borland International
Fatal: Cannot generate COM file : segment-relocatable items present
```

Рисунок 5 – Ошибки компиляции при использовании директивы seg

### *Отличия форматов файлов .COM и .EXE модулей*

1. Какова структура файла .COM? С какого адреса располагается код?

*Ответ:* см. рис.6.



Рисунок 6 – Организация .COM-модуля

2. Какова структура файла "плохого" .EXE? С какого адреса располагается код? Что располагается с адреса 0?

*Ответ:* "Плохой" .EXE отличается от файла .COM при просмотре через FAR добавленным заголовком, который располагается с адреса 0. Заголовок содержит необходимую информацию для загрузки программы в память и специальную таблицу, необходимую для настройки ссылок на дальние сегменты программы (relocation table - таблица перемещения). Дело в том, что при создании COM программы весь код программы находится в одном сегменте. Чтобы такая программа корректно работала, ей не нужно знать, в каком сегменте располагается её код, имеет значение лишь адрес внутри сегмента (смещение). В EXE программе кодовых сегментов может

быть несколько и для обращения к коду другого сегмента (например, дальний вызов процедуры) нужно знать не только смещение внутри этого сегмента, но и его сегментный адрес. Но программа, не загруженная в память, не может знать этот сегментный адрес, так как заранее не известно, в какое место памяти операционная система поместит код программы, прочитанный из EXE файла. Например, операционная система может поместить код программы начиная с сегментного адреса 0x1000 (64Кб) или с 0x2000 (128Кб) и т.д. Для решения этой проблемы и служит таблица перемещения. Код начинается с адреса 300h (см. рис.7).

00000002E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000002F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000300:	E9 40 02 4D 6F 64 65 6C	20 6F 66 20 74 68 65 20	й@Model of the
0000000310:	49 42 4D 20 50 43 3A 20	24 50 43 0D 0A 24 50 43	IBM PC: \$PC\$PC
0000000320:	2F 58 54 0D 0A 24 65 69	74 68 65 72 20 41 54 20	/XT\$either AT
0000000330:	6F 72 20 50 53 32 20 6D	6F 64 65 6C 20 35 30 5C	or PS2 model 50\
0000000340:	36 30 0D 0A 24 50 53 32	20 6D 6F 64 65 6C 20 33	60\$PS2 model 3

Рисунок 7 – Вид "плохого" .EXE-модуля

3. Какова структура файла "хорошего" .EXE? Чем он отличается от файла "плохого" .EXE?

*Ответ:* В отличие от плохого, хороший EXE-файл не содержит директивы ORG 100h (которая выделяет память под PSP), поэтому код начинается с меньшего адреса. В "хорошем" .EXE-файле присутствует разбиение на сегменты. Есть стек. Структура "хорошего" .EXE приведена на рис.8.



Рисунок 8 – Структура "хорошего" .EXE-модуля

### *Загрузка .COM модуля в основную память*

1. Какой формат загрузки модуля .COM? С какого адреса располагается код?

*Ответ:* Для .COM-файла DOS автоматически определяет стек и устанавливает одинаковый общий сегментный адрес во всех четырех сегментных регистрах (начало PSP). Если для программы размер сегмента в 64К является достаточным, то DOS устанавливает в регистре SP адрес конца сегмента – FFFE. PSP заполняет по-прежнему система, но место под него в начале сегмента должен отвести программист. Код располагается с адреса 100h.

2. Что располагается с адреса 0?

*Ответ:* PSP.

3. Какие значения имеют сегментные регистры? На какие области памяти они указывают?

*Ответ:* Все сегментные регистры указывают на PSP.

4. Как определяется стек? Какую область памяти он занимает? Какие адреса?

*Ответ:* Значение регистра SP устанавливается так, чтобы он указывал на последнюю доступную в сегменте ячейку памяти. Таким образом программа занимает начало, а стек – конец сегмента.



### *Загрузка "хорошего".EXE модуля в основную память*

1. Как загружается "хороший" .EXE? Какие значения имеют сегментные регистры?

*Ответ:* Сначала формируется PSP, затем стандартная часть заголовка считывается в память, после чего загрузочный модуль считывается в начальный сегмент. DS и ES указывают на начало префикса программного сегмента. Регистры CS и SS получают значения, указанные компоновщиком.

2. На что указывают регистры ES и DS?

*Ответ:* на начало PSP.

3. Как определяется стек?

*Ответ:* В регистры SS и SP записываются значения, указанные в заголовке, а к SS прибавляется сегментный адрес начального сегмента.

4. Как определяется точка входа?

*Ответ:* Оператором `END start_procedure_name`. Эта информация хранится в заголовке модуля.

### **Выводы.**

В ходе лабораторной работы был написан .COM модуль, определяющий тип РС и версию системы. Из него получен "плохой" .EXE модуль, после чего построен "хороший". Файлы были сопоставлены и изучены. Были исследованы особенности загрузки каждого из модулей в память.