

**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

\_\_\_\_\_

Машина Ю.Д.

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.**

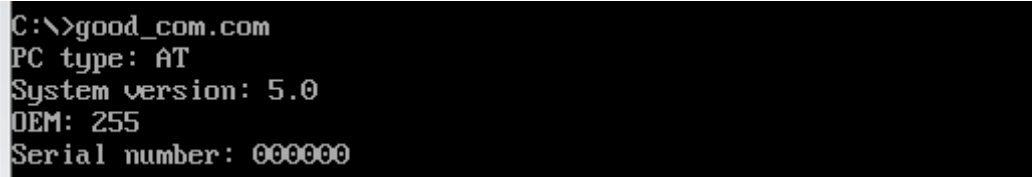
Был написан текст для .COM модуля, определяющий тип PC и версию системы. Ассемблерная программа считывает предпоследний байт ROM BIOS и после сравнения его с имеющимися данными выводит на экран либо идентифицированный тип PC, либо этот самый байт в шестнадцатеричном представлении.

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:\>good_com.com
PC type: AT
System version: 5.0
OEM: 255
Serial number: 000000
```

Рисунок 1 – Пример работы программы

Серийные номера OEM и пользователя недоступны в

эмулированной в работе системе DOS.

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 их удалении возникают ошибки компиляции.



```
**Error** good_com.asm(135) Near jump or call to different CS
**Error** good_com.asm(138) Near jump or call to different CS
**Error** good_com.asm(141) Near jump or call to different CS
**Error** good_com.asm(144) Near jump or call to different CS
**Error** good_com.asm(147) Near jump or call to different CS
**Error** good_com.asm(150) Near jump or call to different CS
**Error** good_com.asm(153) Near jump or call to different CS
**Error** good_com.asm(156) Near jump or call to different CS
**Error** good_com.asm(171) Near jump or call to different CS
**Error** good_com.asm(174) Near jump or call to different CS
**Error** good_com.asm(175) Near jump or call to different CS
**Error** good_com.asm(183) Near jump or call to different CS
**Error** good_com.asm(184) Near jump or call to different CS
**Error** good_com.asm(191) Near jump or call to different CS
**Error** good_com.asm(197) Near jump or call to different CS
**Error** good_com.asm(198) Near jump or call to different CS
**Error** good_com.asm(207) Near jump or call to different CS
**Error** good_com.asm(208) Near jump or call to different CS
Error messages: 37
Warning messages: None
Passes: 1
Remaining memory: 471k
```

Рисунок 2 – Ошибки компиляции при удалении директивы ASSUME

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. Также 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.

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

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

*Ответ:* см. рис.3.



Рисунок 3 – 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 (64Kб) или с 0x2000 (128Kб) и т.д. Для решения этой проблемы и служит таблица перемещения. Код начинается с адреса 300h (см. рис.7).

000002d0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
000002e0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
000002f0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00000300	e9 b9 01 50 43 20 74 79 70 65 3a 20 24 50 43 0d	йй.PC type: \$PC.
00000310	0a 24 50 43 2f 58 54 0d 0a 24 41 54 0d 0a 24 50	.\$PC/XT..\$AT..\$P
00000320	53 32 20 6d 6f 64 65 6c 20 33 30 0d 0a 24 50 53	S2 model 30..\$PS
00000330	32 20 6d 6f 64 65 6c 20 35 30 20 6f 72 20 36 30	2 model 50 or 60
00000340	0d 0a 24 50 53 32 20 6d 6f 64 65 6c 20 38 30 0d	..\$PS2 model 80.

Рисунок 4 – 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 5 – 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 PC 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.

## ПРИЛОЖЕНИЕ А

### КОД ИСХОДНОЙ ПРОГРАММЫ

```
TESTPC  SEGMENT
        ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING,
SS:NOTHING
        ORG 100H

START:  JMP BEGIN

PCtype db 'PC type: $'
PCtype_PC db 'PC',0DH,0AH,$'
PCtype_PCXT db 'PC/XT',0DH,0AH,$'
PCtype_AT db 'AT',0DH,0AH,$'
PCtype_PS2_30 db 'PS2 model 30',0DH,0AH,$'
PCtype_PS2_50_or_60 db 'PS2 model 50 or 60',0DH,0AH,$'
PCtype_PS2_80 db 'PS2 model 80',0DH,0AH,$'
PCtype_PCjr db 'PCjr',0DH,0AH,$'
PCtype_PC_Convertible db 'PC Convertible',0DH,0AH,$'

System_version db 'System version:  . ',0DH,0AH,$'
OEM db 'OEM:  ',0DH,0AH,$'
Serial_number db 'Serial number:      ',0DH,0AH,$'

;-----
TETR_TO_HEX PROC near
    and AL,0Fh
    cmp AL,09
    jbe NEXT
    add AL,07
NEXT:  add AL,30h
    ret
TETR_TO_HEX ENDP
;-----
BYTE_TO_HEX PROC near
;байт в AL переводится в два символа шестн. числа в AX
    push CX
    mov AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX; в AL старшая цифра
    pop CX          ; в AH младшая
    ret
```

BYTE\_TO\_HEX ENDP

;-----

WRD\_TO\_HEX PROC near

; перевод в 16 с/с 16-ти разрядного числа

; в AX – число, DI – адрес последнего символа

push BX

mov BH,AH

call BYTE\_TO\_HEX

mov [DI],AH

dec DI

mov [DI],AL

dec DI

mov AL,BH

call BYTE\_TO\_HEX

mov [DI],AH

dec DI

mov [DI],AL

pop BX

ret

WRD\_TO\_HEX ENDP

;-----

BYTE\_TO\_DEC PROC near

; перевод в 10с/с, SI – адрес поля младшей цифры

push AX

push CX

push DX

xor AH,AH

xor DX,DX

mov CX,10

loop\_bd: div CX

or DL,30h

mov [SI],DL

dec SI

xor DX,DX

cmp AX,10

jae loop\_bd

cmp AL,00h

je end\_l

or AL,30h

mov [SI],AL

end\_l: pop DX

pop CX

pop AX

ret

BYTE\_TO\_DEC ENDP

```

;-----
Print PROC near
    mov AH,09h
    int 21h
    ret
Print ENDP

```

```

Type_of_PC PROC near
    push ax
    lea dx,PCtype
    call Print
    mov ax,0F000h ;тип хранится в байте по адресу
0F000:0FFFEh (предпоследний байт ROM BIOS)
    mov es,ax
    mov ax,es:0FFFEh

    cmp al,0FFh ; PC
    je PC_type

    cmp al,0FEh ; PC/XT
    je PCXT_type

    cmp al,0FBh ; PC/XT ;(FE,FB)
    je PCXT_type

    cmp al,0FCh ;AT
    je AT_type

    cmp al,0FAh ;PS2 модель 30
    je PS2_30_type

    cmp al,0FCh ;PS2 модель 50 или 60
    je PS2_50_or_60_type

    cmp al,0F8h ;PS2 модель 80
    je PS2_80_type

    cmp al,0FDh
    je PCjr_type

    cmp al,0F9h
    je PC_Convertible_type

PC_type:

```

```

        lea dx,PCtype_PC
        jmp PrintType
PCXT_type:
        lea dx,PCtype_PCXT
        jmp PrintType
AT_type:
        lea dx,PCtype_AT
        jmp PrintType
PS2_30_type:
        lea dx,PCtype_PS2_30
        jmp PrintType
PS2_50_or_60_type:
        lea dx,PCtype_PS2_50_or_60
        jmp PrintType
PS2_80_type:
        lea dx,PCtype_PS2_80
        jmp PrintType
PCjr_type:
        lea dx,PCtype_PCjr
        jmp PrintType
PC_Convertible_type:
        lea dx,PCtype_PC_Convertible
        jmp PrintType

PrintType:
call    Print
pop ax
ret
Type_of_PC ENDP

```

```

SystemVersion PROC near
    mov ah,30h
    int 21h

```

; System version (AL- номер основной версии, AH - номер модификации)

```

    lea dx,System_version
    mov si,dx
    add si,16
    call BYTE_TO_DEC
    add si,3
    mov al,ah
    call BYTE_TO_DEC

```

call Print

; OEM (BH-сериный номер Original Equipment  
Manufacturer)

```
lea dx,OEM
mov si,dx
add si,7
mov al,bh
call BYTE_TO_DEC
call Print
```

; Serial number (BL:CX - 24-битовый серийный номер  
пользователя)

```
lea dx,Serial_number
mov di,dx
mov al,bl
call BYTE_TO_HEX
add di,15
mov [di],ax
mov ax,cx
mov di,dx
add di,20
call WRD_TO_HEX
call Print
```

```
ret
SystemVersion ENDP
```

; Код

BEGIN:

```
call Type_of_PC
call SystemVersion
xor AL,AL ;|
mov AH,4Ch ;| exit to dos
int 21H ;|
TESTPC ENDS
END START
```