

**МИНОБРНАУТИ 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

\_\_\_\_\_

Кортев Ю.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.

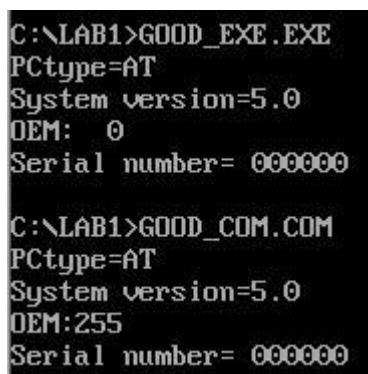
### **Постановка задачи.**

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.

Затем исходный код COM модуля переделан для корректной работы EXE модуля.



```
C:\LAB1>GOOD_EXE.EXE
PCtype=AT
System version=5.0
OEM: 0
Serial number= 000000

C:\LAB1>GOOD_COM.COM
PCtype=AT
System version=5.0
OEM:255
Serial number= 000000
```

Рисунок 1 – Пример работы программ

Исходный код составленной программы представлен в приложении А.

## Ответы на контрольные вопросы.

1. Сколько сегментов должна содержать COM-программа?

Ровно один сегмент – сегмент кода.

2. EXE-программа?

Один и больше.

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

ORG – сдвигает адресацию в программе на 256 бай для расположения PSP,

ASSUME – ставит сегментным регистрам в соответствие требуемые сегменты.

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

Нельзя использовать команды с дальней адресацией, поскольку в COM-программе отсутствует таблица настроек, которая указывает, какие абсолютные адреса при загрузке должны быть изменены, так как до загрузки неизвестно, куда будет загружена программа.

## Отличия форматов файлов COM и EXE модулей.

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

COM-файл содержит данные и машинные команды. Код начинается с адреса 0h (см. рис. 4).



Рисунок 4 – Структура COM - модуля

2. Какова структура файла «плохого» EXE? С какого адреса располагается код?

Что располагается с адреса 0?

В «плохом» файле EXE данные и код содержатся в одном сегменте. Код располагается с адреса 300h(см. рис. 5). С адреса 0h располагается заголовок, таблица настроек, а также зарезервированные директивой ORG 100h байт.

0000000000:	4D	5A	C3	00	03	00	00	00	20	00	00	00	FF	FF	00	00
0000000010:	00	00	7D	D6	00	01	00	00	1E	00	00	00	01	00	00	00
0000000020:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000030:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000040:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000050:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000060:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000070:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Рисунок 5 – Заголовок EXE

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

В «хорошем» EXE код, стек и данные выделены в отдельные сегменты, тогда как в «плохом» всего один сегмент и для данных и для кода. В EXE программах нет необходимости в директиве ORG, поскольку загрузчик ставит программу после PSP. Код начинается с 400h.



Рисунок 6 – Структура «хорошего» EXE

### Загрузка COM модуля в основную память.

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

После загрузки COM-программы в память, сегментные регистры указывают на начало PSP. Начало кода определяется директивой ORG от начала выделенного фрагмента (100h).

2. Что располагается с адреса 0?

PSP.

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

Все сегментные регистры указывают на начало PSP.

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

Стек занимает весь фрагмент памяти, выделенный под программу и определяется регистрами SS и SP. Он занимает адреса 0000h-FFFFh.

### **Загрузка «хорошего» EXE модуля в основную память.**

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

DS и ES устанавливаются на начало сегмента PSP, SS – на начало сегмента стека, CS – на начало сегмента кода. В IP загружается смещение точки входа в программу, которая берётся из метки после директивы END.

2. На что указывают регистры DS и ES?

Начало сегмента PSP.

3. Как определяется стек?

Для стека в программе выделяется отдельный сегмент с параметром STACK. При запуске программы в SS заносится адрес сегмента стека, а в SP – адрес верхушки стека.

4. Как определяется точка входа?

С помощью директивы END, после которой указывается метка, куда переходит программа при запуске.

### **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 сравнены и проанализированы. 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 'PCtype=$'
PCtype_PC db 'PC',0DH,0AH,'$'
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,'$'
OEMdb'OEM: ',0DH,0AH,'$'
Serial_numberdb'Serialnumber= ',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
```

```
PC_type PROC near
    push ax
    lea dx,PCtype
    call Print
    mov ax,0F000h
    mov es,ax
    mov ax,es:0FFFEh

    cmp al,0FFh
    je case_PC

    cmp al,0FEh
    je case_PCXT

    cmp al,0FBh
    je case_PCXT

    cmp al,0FCh
    je case_AT

    cmp al,0FAh
    je case_PS2_30

    cmp al,0FCh
    je case_PS2_50_or_60

    cmp al,0F8h
    je case_PS2_80

    cmp al,0FDh
    je case_PCjr

    cmp al,0F9h
    je case_PC_Convertible

case_PC:
    lea dx,PCtype_PC
    jmp case_Print
case_PCXT:
```

```

        lea dx,PCTYPE_PCXT
        jmp case_Print
case_AT:
        lea dx,PCTYPE_AT
        jmp case_Print
case_PS2_30:
        lea dx,PCTYPE_PS2_30
        jmp case_Print
case_PS2_50_or_60:
        lea dx,PCTYPE_PS2_50_or_60
        jmp case_Print
case_PS2_80:
        lea dx,PCTYPE_PS2_80
        jmp case_Print
case_PCjr:
        lea dx,PCTYPE_PCjr
        jmp case_Print
case_PC_Convertible:
        lea dx,PCTYPE_PC_Convertible
        jmp case_Print

case_Print:
        call Print
        pop ax
        ret
PC_type 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,15
    call BYTE_TO_DEC
    add si,3
    mov al,ah
    call BYTE_TO_DEC
    call Print

```

; OEM (BH-серийный номер Original Equipment Manufacturer)

```

    lea dx,OEM

```

```
mov si,dx
add si,6
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 PC_type
call SystemVersion
xor AL,AL
mov AH,4Ch
int 21H
TESTPC ENDS
END START
```