

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МОЭВМ**

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

Студент гр. 8383

\_\_\_\_\_

Дейнега В.Е.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2020

### Цель работы.

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

### Ход работы.

Был написан текст исходного .COM модуля, который определяет тип ПК и версию системы. Код программы представлен в приложении А. Далее был построен .COM модуль (результат выполнения на рис. 1), а также «плохой» .EXE модуль (результат выполнения на рис. 2).

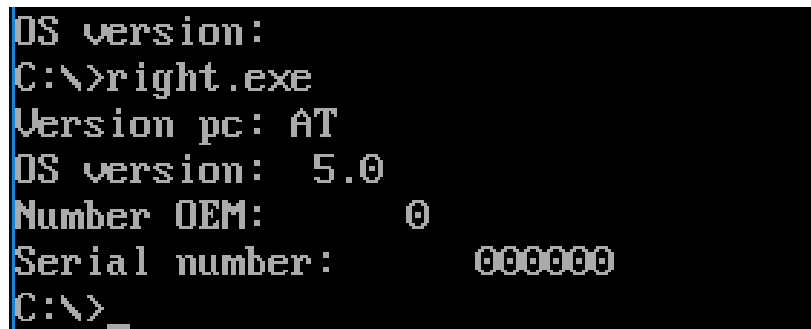
```
C:\>lab1_com.com
Version pc: AT
OS version: 5.0
Number OEM: 0
Serial number: 000000
C:\>
```

Рисунок 1 – Результат выполнения .COM модуля

```
OS version: 000
OS version: 5 0
OS version: 000
OS version: 0
OS version: 000
OS version: 000000
OS version: 000
C:\>_
```

Рисунок 2 – Результат выполнения «плохого» .EXE модуля

Далее был написан текст для «хорошего» .EXE модуля. Код программы представлен в приложении Б. Был построен .EXE модуль (результат выполнения на рис. 3).



```
OS version:
C:\>right.exe
Version pc: AT
OS version: 5.0
Number OEM: 0
Serial number: 000000
C:\>_
```

Рисунок 3 – Результат выполнения «хорошего» .EXE модуля

### Отличия исходных текстов COM и EXE программ

- 1) Сколько сегментов должна содержать COM-программа? COM-программа должна содержать ровно один сегмент.
- 2) EXE-программа? EXE программа может содержать несколько сегментов.
- 3) Какие директивы должны обязательно быть в тексте COM-программы? ORG 100h, которая устанавливает значение IP 100h, т.к. первые 256 байт занимает PSP.
- 4) Все ли форматы команд можно использовать в COM-программе? В COM-программах нельзя использовать команды вида mov <регистр>, <сегмент>.

При помощи приложения Far были открыты все созданные файлы загрузочных модулей в шестнадцатеричном виде. Результаты представлены на рис. 4 – 6.



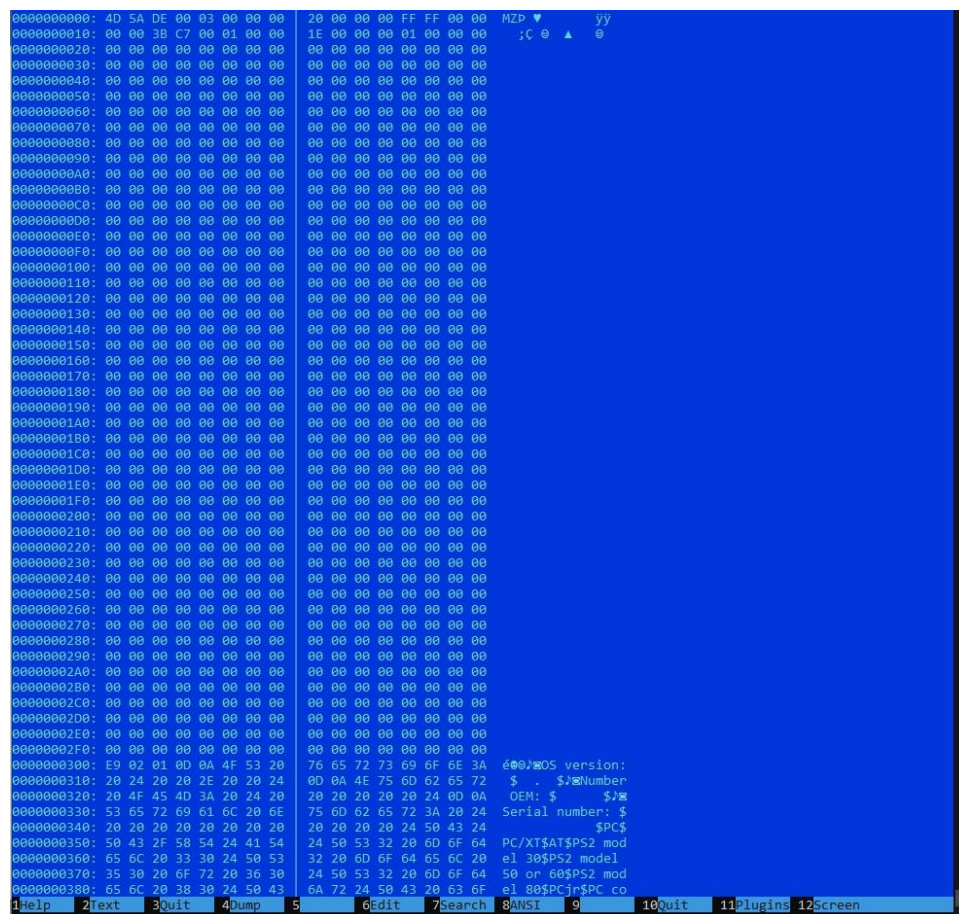


Рисунок 6 – Содержимое файла «плохого» EXE модуля

## Отличие форматов файлов COM и EXE модулей

- 1) Какова структура файла COM? С какого адреса располагается код? COM файл содержит только машинный код и данные программы. Код начинается с адреса 0h, но при загрузке происходит смещение на 100h.
- 2) Какова структура файла «плохого» EXE? В «плохом» EXE модуле машинный код и данные содержатся в одном сегменте. Код располагается со смещения 300h, до кода идет заголовок размером 512 байт.
- 3) Какова структура файла «хорошего» EXE, чем отличается от «плохого»? В «хорошем» EXE модуле данные, стек и машинный код находятся в разных сегментах. У «хорошего» EXE есть стек.

При помощи отладчика AFD COM файл был загружен в основную память. Результат продемонстрирован на рис. 7.



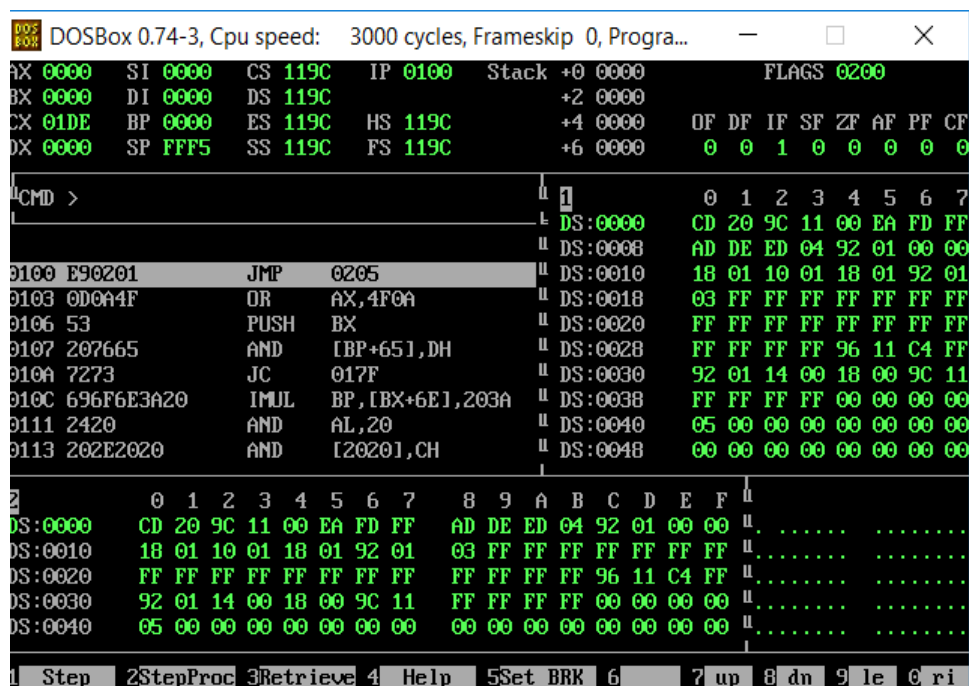


Рисунок 7 – Результат загрузки COM файла в память

### Загрузка COM модуля в основную память

- 1) Какой формат загрузки модуля COM? С какого адреса располагается код? Создается блок памяти. PSP, команды, данные и стек расположены в одном сегменте и идут по порядку. После загрузки в память IP становится равным 100h. Сегментные регистры устанавливаются на адрес сегмента PSP.
- 2) Что располагается с адреса 0? С адреса 0 начинается PSP.
- 3) Какие значения имеют сегментные регистры, на какие области памяти они указывают? Сегментные регистры указывают на начало PSP, имеют значения 119Ch.
- 4) Как определяется стек, какую область в памяти занимает, какие адреса? Стек занимает все доступное место после машинного кода. Регистр SP указывает на адрес FFF5h.

При помощи отладчика в основную память был записан так же «хороший» EXE файл. Результат продемонстрирован на рис. 8.



## ПРИЛОЖЕНИЕ А

### КОД ПРОГРАММЫ ДЛЯ СОМ ФАЙЛА

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

```
SYSTEM_VER db 13,10,"OS version: $"
FORM_SER db " . $"
OEM db 13, 10, "Number OEM: $"
OEM_FORM db "      $"
NUMBER db 13, 10, "Serial number: $"
NUM_FORM db "          $"
PC db "PC$"
PCXT db "PC/XT$"
A_T db "AT$"
PS2_30 db "PS2 model 30$"
PS2_50 db "PS2 model 50 or 60$"
PS2_80 db "PS2 model 80$"
PCJR db "PCjr$"
PCC db "PC convertible$"
PC_VER db "Version pc: $"
```

```
;-----
```

```
WRITE PROC near
```

```
    push ax
    mov ah, 9h
    int 21h
    pop ax
    ret
```

```
WRITE ENDP
```

```
;-----
```

```
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
```

```
push CX
mov AH,AL
call TETR_TO_HEX
xchg AL,AH
mov CL,4
shr AL,CL
call TETR_TO_HEX
pop CX
ret
```

```
BYTE_TO_HEX ENDP
```

```
;-----
```

```
WRD_TO_HEX PROC near
```

```
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
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_1
or AL,30h
mov [SI],AL
end_1: pop DX
pop CX
ret
BYTE_TO_DEC ENDP
;-----

```

```

BEGIN:
    mov ax, 0f000h
    mov es, ax
    mov al, es:[0ffffh]
    mov dx, offset PC_VER
    call WRITE
    cmp al, 0ffh
    je PC_TYPE
    cmp al, 0feh
    je XT_TYPE
    cmp al, 0fbh
    je XT_TYPE
    cmp al, 0fch
    je AT_TYPE
    cmp al, 0fah
    je PS2_30_TYPE
    cmp al, 0fch
    je PS2_5060_TYPE
    cmp al, 0f8h
    je PS2_80_TYPE
    cmp al, 0fdh
    je PCJR_TYPE
    cmp al, 0f9h
    je CON_TYPE

```

```

PC_TYPE:
    mov dx, offset PC
    call WRITE
    jmp FINISH
XT_TYPE:
    mov dx, offset PCXT
    call WRITE
    jmp FINISH
AT_TYPE:
    mov dx, offset A_T
    call WRITE
    jmp FINISH
PS2_30_TYPE:
    mov dx, offset PS2_30
    call WRITE
    jmp FINISH
PS2_5060_TYPE:
    mov dx, offset PS2_50
    call WRITE
    jmp FINISH
PS2_80_TYPE:
    mov dx, offset PS2_80
    call WRITE
    jmp FINISH
PCJR_TYPE:
    mov dx, offset PCJR
    call WRITE
    jmp FINISH
CON_TYPE:
    mov dx, offset PCC
    call WRITE
    jmp FINISH

FINISH:
;-----
    mov ah, 30h
    int 21h
    push ax
    mov dx, offset SYSTEM_VER
    call WRITE
    mov si, offset FORM_SER
    inc si
    call BYTE_TO_DEC
    pop ax
    mov al, ah
    add si, 3
    call BYTE_TO_DEC
    mov dx, offset FORM_SER
    call WRITE
    mov dx, offset OEM
    call WRITE
    mov si, offset OEM_FORM
    add si, 5
    mov al, bh
    call BYTE_TO_DEC
    mov dx, offset OEM_FORM
    call WRITE
    mov dx, offset NUMBER
    call WRITE
    mov di, offset NUM_FORM
    add di, 10

```

```

mov ax, cx
call WRD_TO_HEX
mov al, bl
call BYTE_TO_HEX
sub di, 2
mov [di], ax
mov dx, offset NUM_FORM
call WRITE
mov ax, 4c00h
int 21h

```

```

TESTPC ENDS
END START

```

## ПРИЛОЖЕНИЕ Б

### КОД ПРОГРАММЫ ДЛЯ EXE ФАЙЛА

```

ASTACK      SEGMENT      STACK
            DB 1024 DUP(?)
ASTACK      ENDS

```

```

DATA        SEGMENT
SYSTEM_VER db 13,10,"OS version: $"
FORM_SER db " . $"
OEM db 13, 10, "Number OEM: $"
OEM_FORM db "      $"
NUMBER db 13, 10, "Serial number: $"
NUM_FORM db "          $"
PC db "PC$"
PCXT db "PC/XT$"
A_T db "AT$"
PS2_30 db "PS2 model 30$"
PS2_50 db "PS2 model 50 or 60$"
PS2_80 db "PS2 model 80$"
PCJR db "PCjr$"
PCC db "PC convertible$"
PC_VER db "Version pc: $"

```

```
DATA      ENDS
```

```

CODE        SEGMENT
ASSUME CS:CODE, DS:DATA, SS:ASTACK

```

```
WRITE PROC near
```

```

    push ax
    mov ah, 9h
    int 21h
    pop ax
    ret

```

```
WRITE ENDP
```

```
;-----
```

```

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
push CX
mov AH,AL
call TETR_TO_HEX
xchg AL,AH
mov CL,4
shr AL,CL
call TETR_TO_HEX
pop CX
ret
BYTE_TO_HEX ENDP
;-----

WRD_TO_HEX PROC near
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
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_1
or AL,30h
mov [SI],AL
end_1: pop DX
pop CX
ret
BYTE_TO_DEC ENDP
;-----
Main      PROC FAR
push ax

```

```

    sub ax,ax
    mov ax,data
    mov ds,ax
    pop ax
    mov ax, 0f000h
    mov es, ax
    mov al, es:[0ffffh]
    mov dx, offset PC_VER
    call WRITE
    cmp al, 0ffh
    je PC_TYPE
    cmp al, 0feh
    je XT_TYPE
    cmp al, 0fbh
    je XT_TYPE
    cmp al, 0fch
    je AT_TYPE
    cmp al, 0fah
    je PS2_30_TYPE
    cmp al, 0fch
    je PS2_5060_TYPE
    cmp al, 0f8h
    je PS2_80_TYPE
    cmp al, 0fdh
    je PCJR_TYPE
    cmp al, 0f9h
    je CON_TYPE

PC_TYPE:
    mov dx, offset PC
    call WRITE
    jmp FINISH
XT_TYPE:
    mov dx, offset PCXT
    call WRITE
    jmp FINISH
AT_TYPE:
    mov dx, offset A_T
    call WRITE
    jmp FINISH
PS2_30_TYPE:
    mov dx,offset PS2_30
    call WRITE
    jmp FINISH
PS2_5060_TYPE:
    mov dx, offset PS2_50
    call WRITE
    jmp FINISH
PS2_80_TYPE:
    mov dx, offset PS2_80
    call WRITE
    jmp FINISH
PCJR_TYPE:
    mov dx, offset PCJR
    call WRITE
    jmp FINISH
CON_TYPE:
    mov dx, offset PCC
    call WRITE
    jmp FINISH

```

```

FINISH:
;-----
    mov ah, 30h
    int 21h
    push ax
    mov dx, offset SYSTEM_VER
    call WRITE
    mov si, offset FORM_SER
    inc si
    call BYTE_TO_DEC
    pop ax
    mov al, ah
    add si, 3
    call BYTE_TO_DEC
    mov dx, offset FORM_SER
    call WRITE
    mov dx, offset OEM
    call WRITE
    mov si, offset OEM_FORM
    add si, 5
    mov al, bh
    call BYTE_TO_DEC
    mov dx, offset OEM_FORM
    call WRITE
    mov dx, offset NUMBER
    call WRITE
    mov di, offset NUM_FORM
    add di, 10
    mov ax, cx
    call WRD_TO_HEX
    mov al, bl
    call BYTE_TO_HEX
    sub di, 2
    mov [di], ax
    mov dx, offset NUM_FORM
    call WRITE
    mov ax, 4c00h
    int 21h

Main      ENDP
CODE      ENDS
END Main

```