

**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.

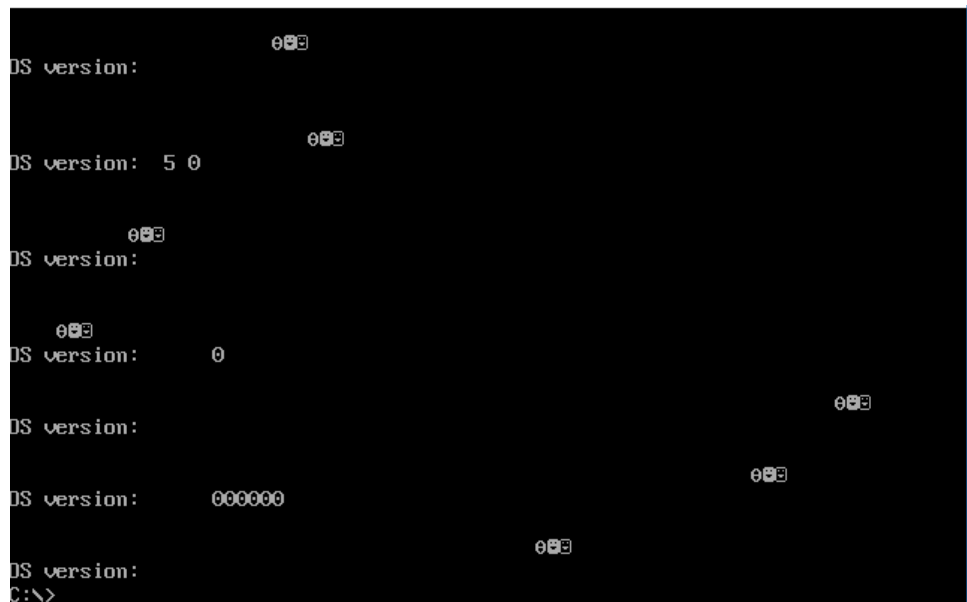
### Ход работы.

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: 0000000
C:\>
```

Рисунок 1 – Результат выполнения .COM модуля



```
OS version: 
OS version: 5 0
OS version: 
OS version: 
OS version: 0
OS version: 
OS version: 
OS version: 0000000
OS version: 
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

0) Без директивы ASSUME COM-программа работать не будет. Если закомментировать ASSUME произойдет ошибка (рис. 0).

```
lab1_com.asm(22): error A2062: Missing or unreachable CS
lab1_com.asm(30): error A2062: Missing or unreachable CS
lab1_com.asm(35): error A2062: Missing or unreachable CS
lab1_com.asm(40): error A2062: Missing or unreachable CS
lab1_com.asm(53): error A2062: Missing or unreachable CS
lab1_com.asm(71): error A2062: Missing or unreachable CS
lab1_com.asm(77): error A2062: Missing or unreachable CS
lab1_com.asm(88): error A2062: Missing or unreachable CS
lab1_com.asm(94): error A2062: Missing or unreachable CS
lab1_com.asm(119): error A2062: Missing or unreachable CS
lab1_com.asm(123): error A2062: Missing or unreachable CS
lab1_com.asm(127): error A2062: Missing or unreachable CS
lab1_com.asm(131): error A2062: Missing or unreachable CS
lab1_com.asm(135): error A2062: Missing or unreachable CS
lab1_com.asm(139): error A2062: Missing or unreachable CS
lab1_com.asm(143): error A2062: Missing or unreachable CS
lab1_com.asm(147): error A2062: Missing or unreachable CS
lab1_com.asm(152): error A2062: Missing or unreachable CS

49948 + 455265 Bytes symbol space free

0 Warning Errors
19 Severe Errors
C:\>
```

Рисунок 0 – Ошибка компиляции COM-программы без ASSUME

- 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> (т.к. в программе только 1 сегмент).

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

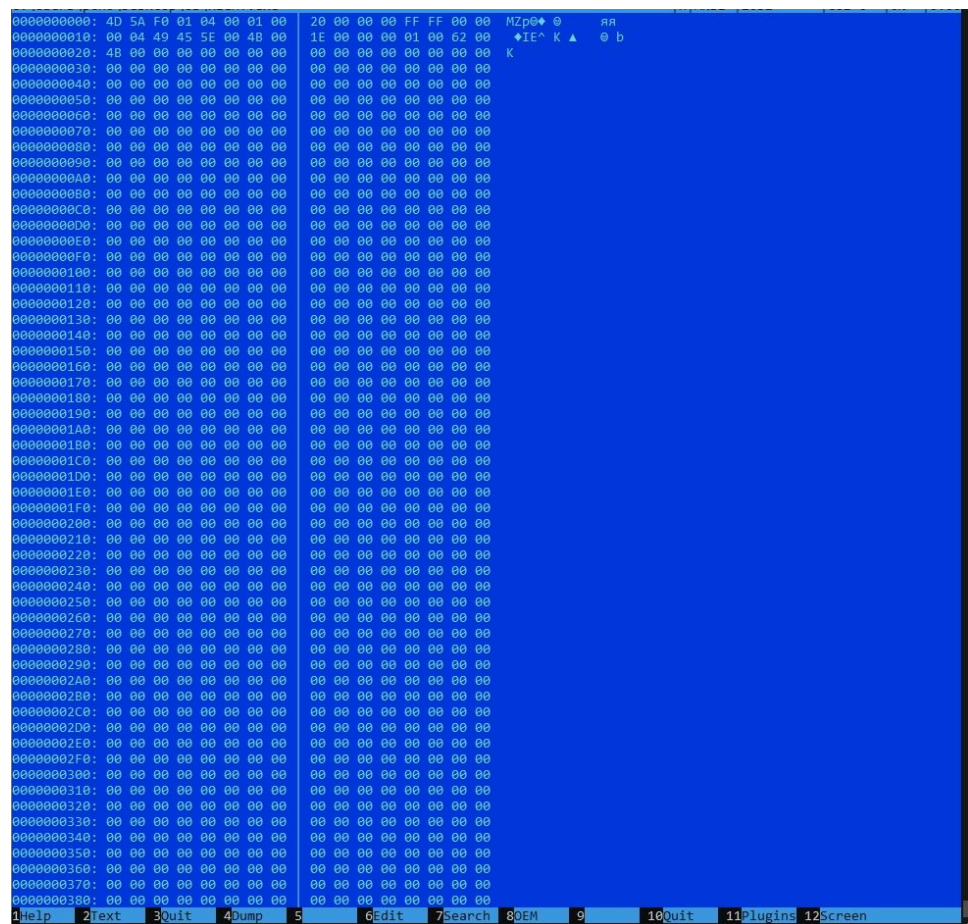


Рисунок 4.1 – Содержимое файла «хорошего» EXE модуля(очень большой стек)

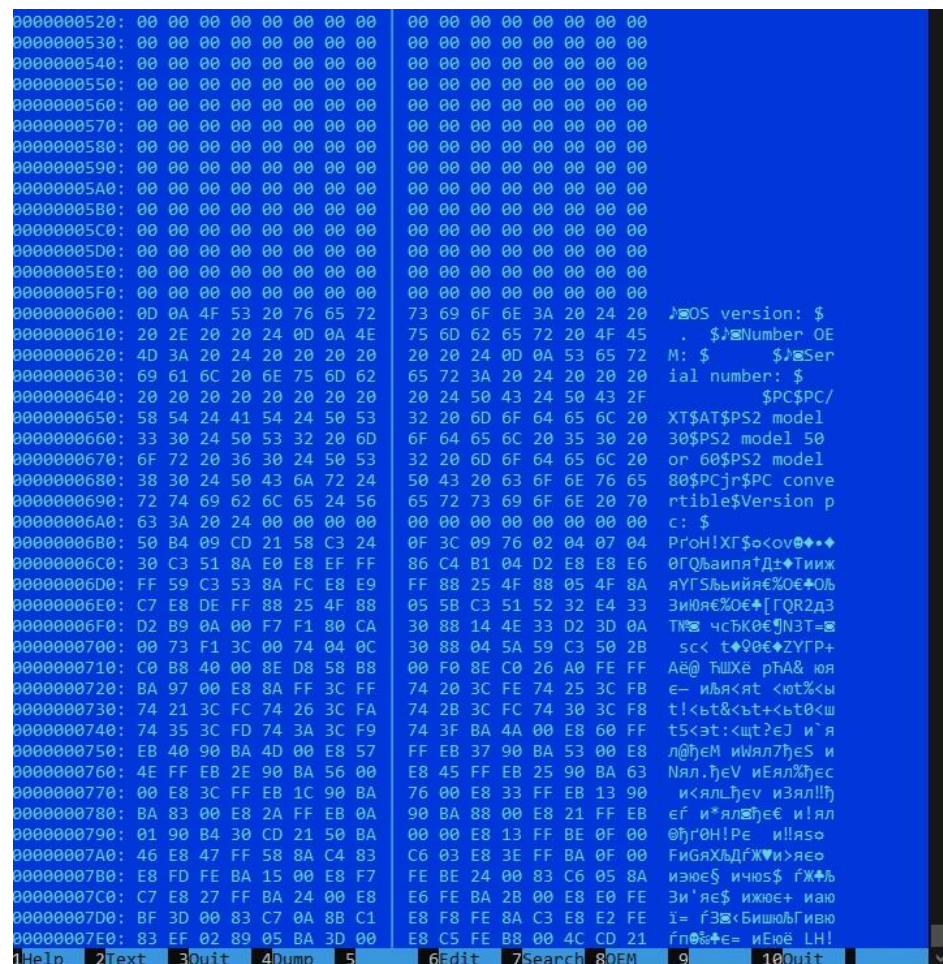


Рисунок 4.2 – Содержимое файла «хорошего» EXE модуля

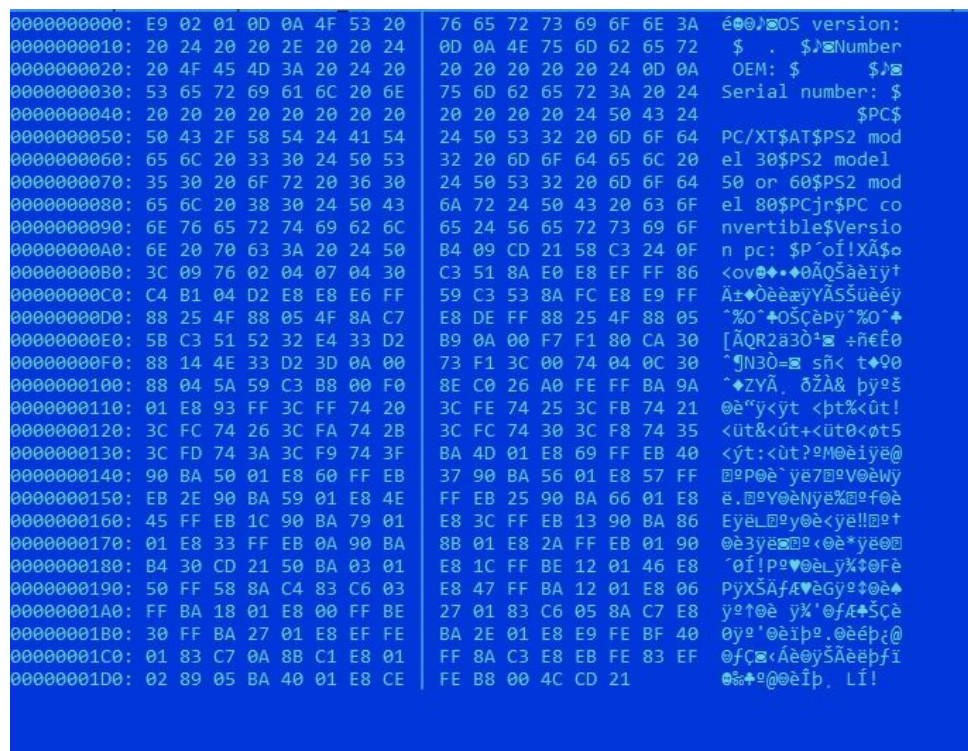


Рисунок 5 – Содержимое файла COM модуля



```

00000000: 4D 5A DE 00 03 00 00 00 20 00 00 00 FF FF 00 00 MZD  y y
00000010: 00 00 3B C7 00 01 00 00 1E 00 00 00 01 00 00 00 ;C  0  0
00000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000260: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000002A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000002B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000002C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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 02 01 00 0A 4F 53 20 76 65 72 73 69 6F 6E 3A é00/MOS version:
00000310: 20 24 20 20 2E 20 20 24 0D 0A 4E 75 6D 62 65 72 $ . $Number
00000320: 20 4F 45 4D 3A 20 24 20 20 20 20 20 24 0D 0A OEM: $ $
00000330: 53 65 72 69 61 6C 20 6E 75 6D 62 65 72 3A 20 24 Serial number: $
00000340: 20 20 20 20 20 20 20 20 20 20 20 24 50 43 24 $PC$
00000350: 50 43 2F 58 54 24 41 54 24 50 53 32 20 6D 6F 64 PC/XT$AT$PS2 mod
00000360: 65 6C 20 33 30 24 50 53 32 20 6D 6F 64 65 6C 20 e1 30$PS2 model
00000370: 35 30 20 6F 72 20 36 30 24 50 53 32 20 6D 6F 64 50 or 60$PS2 mod
00000380: 65 6C 20 38 30 24 50 43 6A 72 24 50 43 20 63 6F e1 80$PCjr$PC co

```

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. В начале файла располагается заголовок размера 512 байт, в нем содержится информация, необходимая системе для настройки регистров и программы для загрузки ее в память, далее со смещения 300h идет код.
- 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 выделенный в

программе стек, который находится сразу после заголовка в 512 байт, далее идет код, в «плохом» EXE код располагается со смещения в 300h.

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

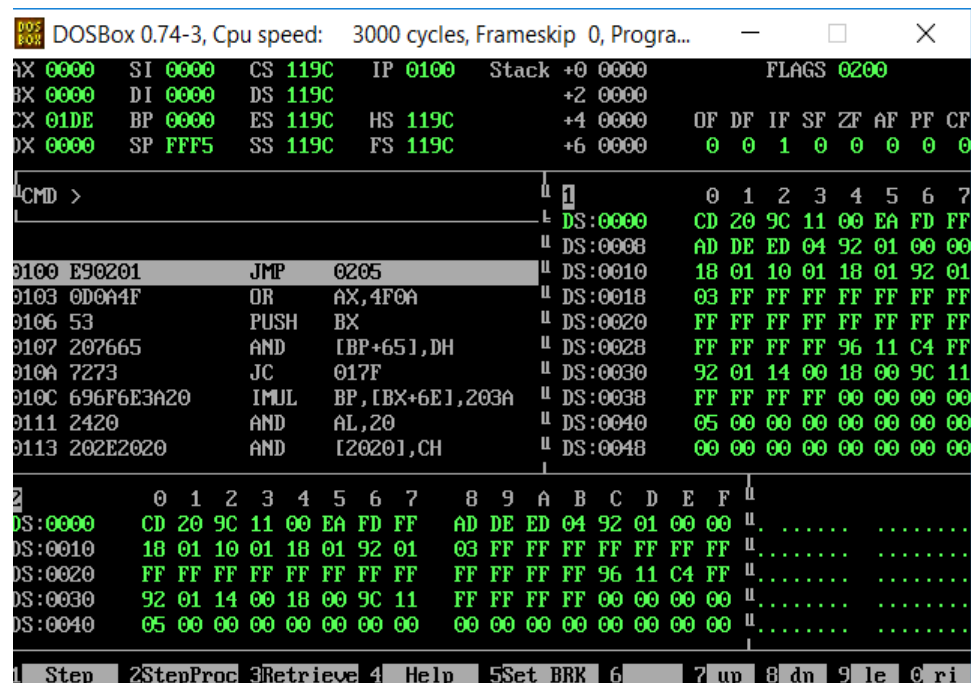


Рисунок 7 – Результат загрузки COM файла в память

### 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.

The screenshot shows the DOSBox 0.74-3 interface. At the top, it displays 'DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...'. Below this, a table of CPU registers is shown: AX: 0000, SI: 0000, CS: 11F7, IP: 005E, Stack: +0 0A0D, FLAGS: 0200; BX: 0000, DI: 0000, DS: 119C, +2 534F; CX: 05F0, BP: 0000, ES: 119C, HS: 119C, +4 7620, OF: 0, DF: 0, IF: 1, SF: 0, ZF: 0, AF: 0, PF: 0, CF: 0; DX: 0000, SP: 0400, SS: 11AC, FS: 119C, +6 7265. Below the registers, the command line shows 'CMD >'. The main window displays a list of instructions and their addresses: 005E 50 PUSH AX, 005F 2BC0 SUB AX,AX, 0061 B8EC11 MOV AX,11EC, 0064 8ED8 MOV DS,AX, 0066 58 POP AX, 0067 B800F0 MOV AX,F000, 006A 8EC0 MOV ES,AX, 006C 26A0FEFF MOV AL,ES:[FFFE]. To the right of the instructions, a memory dump shows the contents of memory segments: DS:0000 (CD 20 9C 11 00 EA FD FF), DS:0008 (AD DE ED 04 92 01 00 00), DS:0010 (18 01 10 01 18 01 92 01), DS:0018 (03 FF FF FF FF FF FF FF), DS:0020 (FF FF FF FF FF FF FF FF), DS:0028 (FF FF FF FF 96 11 C4 FF), DS:0030 (92 01 14 00 18 00 9C 11), DS:0038 (FF FF FF FF 00 00 00 00), DS:0040 (05 00 00 00 00 00 00 00), DS:0048 (00 00 00 00 00 00 00 00). At the bottom, a status bar shows '1 Step 2StepProc 3Retrieve 4 Help 5Set BRK 6 7 up 8 dn 9 le C ri'.

Рисунок 8 – Результат загрузки EXE файла в память

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

- 1) Как загружается «хороший» EXE? Какие значения имеют сегментные регистры? Во время загрузки модуля программы в память система создает блок памяти для PSP и программы. DS, ES указывают на начало PSP, CS на начало сегмента команд, SS на начало сегмента стека.
- 2) На что указывают регистры DS, ES? На начало PSP.
- 3) Как определяется стек? Стек определяется вручную при помощи директивы STACK, SS указывает на начало сегмента стека, в SP хранится смещение конца сегмента стека.
- 4) Как определяется точка входа? Смещение точки входа берется из операнда директивы END и загружается в IP.

### Выводы.

В ходе выполнения лабораторной работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.





## 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
```