

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

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

Студентка гр. 7381

Преподаватель

Кревчик А.Б.

Ефремов М.А.

Санкт-Петербург

2019

## Цель работы.

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

## Ход работы.

TETR\_TO\_HEX – вспомогательная функция, переводит из двоичной в шестнадцатеричную систему.

BYTE\_TO\_HEX – переводит число из регистра AL в шестнадцатеричную систему.

WRD\_TO\_HEX – переводит число из регистра AX в шестнадцатеричную систему.

BYTE\_TO\_DEC – переводит число из регистра AL в десятичную систему.

PRINT – печатает сообщение на экран.

TYPE\_PC – получает тип ПК.

PRINT\_TYPE\_PC – определяет и выводит тип ПК.

VERSION\_DOS – определяет версию системы.

OEM\_NUM – определяет серийный номер OEM.

USER\_NUM – определяет серийный модуль пользователя.

Программа выводит на экран тип IBM PC, версию ОС, серийный номер OEM и серийный номер пользователя.

Результаты работы программы представлены на рис. 1-3.

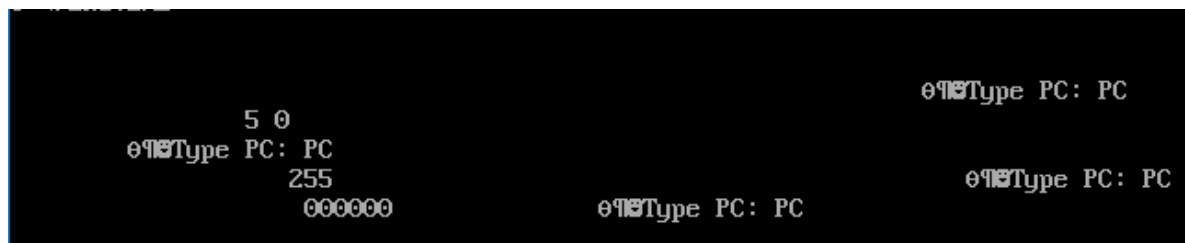
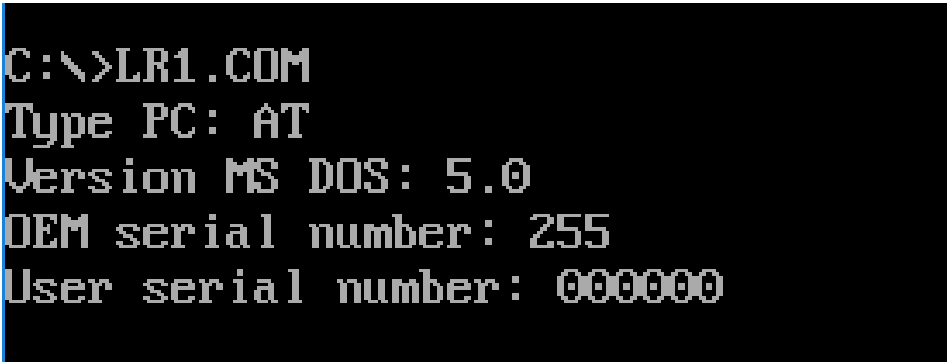
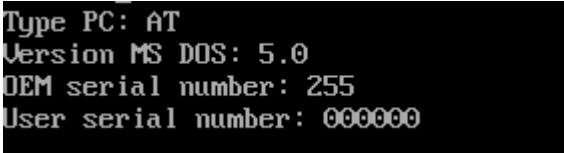


Рисунок 1 – Результат выполнения «плохого» .EXE модуля



```
C:\>LR1.COM
Type PC: AT
Version MS DOS: 5.0
OEM serial number: 255
User serial number: 000000
```

Рисунок 2 – Результат выполнения «хорошего» .COM модуля



```
Type PC: AT
Version MS DOS: 5.0
OEM serial number: 255
User serial number: 000000
```

Рисунок 3 – Результат выполнения «хорошего» .EXE модуля

### **Выводы.**

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

### **Ответы на контрольные вопросы.**

#### **Отличия исходных текстов COM и EXE программ.**

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.

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. С адреса 0h располагается заголовок, таблица настроек, а также зарезервированные директивой ORG 100h байт.

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 «плохо» всего один сегмент и для данных и для кода

Также, в «хорошем» EXE с адреса 200h идет сегмент стека, когда в «плохом» здесь располагаются зарезервированные ORG 100h байт и нет сегмента стека. В EXE программах нет необходимости в директиве ORG, поскольку загрузчик ставит программу после PSP.

### **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?

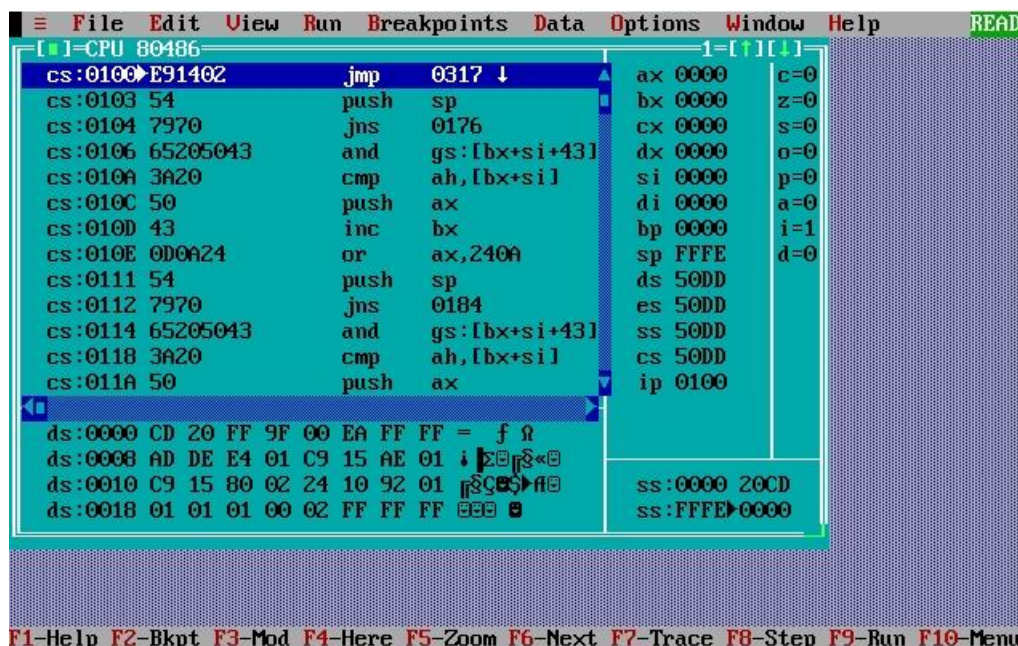
PSP.

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.

## ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД .COM МОДУЛЯ

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

; „ЂКЌ › ...

PC	DB 'TYPE PC: PC', 13, 10, '\$'
PC_XT	DB 'TYPE PC: PC/XT', 13, 10, '\$'
AT	DB 'TYPE PC: AT', 13, 10, '\$'
PS2_1	DB 'TYPE PC: PS2 MODEL 30', 10, 13, '\$'
PS2_2	DB 'TYPE PC: PS2 MODEL 50 OR 60', 10, 13, '\$'
PS2_3	DB 'TYPE PC: PS2 MODEL 80', 10, 13, '\$'
PCJR	DB 'TYPE PC: PCJR', 10, 13, '\$'
PC_CONVERTIBLE	DB 'TYPE PC: PC CONVERTIBLE', 10, 13, '\$'
VERS	DB 'VERSION MS DOS: . ', 10, 13, '\$'
OEM	DB 'OEM SERIAL NUMBER: ', 10, 13, '\$'
USER	DB 'USER SERIAL NUMBER: ', 10, 13, '\$'

; 𐎱𐎠𐎼𐎿—...,, “𐎠𐎹”

```
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                ;Ў @В Ў AL ĨГАГЎ®«ЁВБП Ў «Ў БЁ-Ў®«
ИГБВ-. ЗЁБ«   Ў АХ

```

```
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 ;İΓΑΓŸ®Ÿ Ÿ 16 Б/Б 16-БЁ А ŞАПŸ-®Ÿ® ЗЁБ» , Ÿ AX  
- ЗЁБ»®, DI - ŸАГБ İ®Б«ГŸ-ГŸ® БЁ-Ÿ®»

PUSH BX

```

        MOV     BH, AH
        CALL    BYTE_TO_HEX
        MOV     [DI], AH
        DEC     DI
        MOV     [DI], AL
        DEC     DI
        MOV     AL, BH
        XOR     AH, AH
        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    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
        PUSH    AX
        MOV     AH, 09H
        INT     21H
        POP     AX
        RET

PRINT ENDP

TYPE_PC PROC NEAR      ; ĭ®«ГЗГ-ĖГ БĖĭ  џЉ
        PUSH    DS

```



```

        MOV     BX,0F000H
        MOV     DS,BX
        SUB     AX,AX
        MOV     AH,DS:[0FFFEH]
        POP     DS
        RET
TYPE_PC ENDP

```

```

PRINT_TYPE_PC PROC NEAR ;®İAΓαΓ«Γ-ĖΓ Ė ŸŁŸ®α BĖİ ЦЉ

```

```

        PUSH    AX
        PUSH    BX
        PUSH    DI

```

```

        MOV     DX, OFFSET PC
        CMP     AH, 0FFH
        JE      PRINT_MSG

```

```

        MOV     DX, OFFSET PC_XT
        CMP     AH, 0FEH
        JE      PRINT_MSG

```

```

        MOV     DX, OFFSET PC_XT
        CMP     AH, 0FBH
        JE      PRINT_MSG

```

```

        MOV     DX, OFFSET AT
        CMP     AH, 0FCH
        JE      PRINT_MSG

```

```

        MOV     DX, OFFSET PS2_1
        CMP     AH, 0FAH
        JE      PRINT_MSG

```

```

        MOV     DX, OFFSET PS2_2
        CMP     AH, 0FCH
        JE      PRINT_MSG

```

```

        MOV     DX, OFFSET PS2_3
        CMP     AH, 0F8H
        JE      PRINT_MSG

```

```

        MOV     DX, OFFSET PCJR
        CMP     AH, 0FDH
        JE      PRINT_MSG

```

```

        MOV     DX, OFFSET PC_CONVERTIBLE
        CMP     AH, 0F9H
        JE      PRINT_MSG

```

```

        MOV     AL,AH

```

```

        CALL BYTE_TO_HEX
        MOV     DX, AX

PRINT_MSG:
        CALL PRINT
        POP     DI
        POP     BX
        POP     AX
        RET
PRINT_TYPE_PC ENDP

VERSION_DOS PROC NEAR      ;®İAΓᄡΓ«Γ-ËΓ ŸΓABËË BËBBΓ¬Л
        PUSH AX
        PUSH SI
        MOV     SI, OFFSET VERS
        ADD     SI, 10H
        CALL    BYTE_TO_DEC
        ADD     SI, 3H
        MOV     AL, AH
        CALL    BYTE_TO_DEC
        MOV     DX, OFFSET VERS
        CALL    PRINT
        POP     SI
        POP     AX
        RET
VERSION_DOS ENDP

OEM_NUM PROC NEAR      ;®İAΓᄡΓ«Γ-ËΓ БΓAË©-®Ј® -®¬ΓA OEM
        PUSH AX
        PUSH BX
        PUSH SI
        MOV     AL, BH
        MOV     SI, OFFSET OEM
        ADD     SI, 15H
        CALL    BYTE_TO_DEC
        MOV     DX, OFFSET OEM
        CALL    PRINT
        POP     SI
        POP     BX
        POP     AX
        RET
OEM_NUM ENDP

USER_NUM PROC NEAR      ;®İAΓᄡΓ«Γ-ËΓ БΓAË©-®Ј® -®¬ΓA İ®«M$®Ÿ БΓ«П
        PUSH CX
        PUSH DI
        PUSH AX
        MOV     DI, OFFSET USER
        ADD     DI, 19H
        MOV     AX, CX

```

```

CALL WRD_TO_HEX
MOV     AL, BL
MOV     DI, OFFSET USER
ADD     DI, 14H
CALL    BYTE_TO_HEX
MOV     [DI], AX
MOV     DX, OFFSET USER
CALL    PRINT
POP     AX
POP     DI
POP     CX
RET

```

```

USER_NUM ENDP

```

```

BEGIN:

```

```

CALL    TYPE_PC
CALL    PRINT_TYPE_PC
MOV     AH, 30H
INT     21H
CALL    VERSION_DOS
CALL    OEM_NUM
CALL    USER_NUM
XOR     AL, AL
MOV     AH, 4CH
INT     21H

```

```

TESTPC ENDS
END START

```

# ПРИЛОЖЕНИЕ Б

## ИСХОДНЫЙ КОД .EXE МОДУЛЯ

ASTACK SEGMENT STACK

DW 0100h DUP(?)

ASTACK ENDS

DATA SEGMENT

```

;,,Ѡќќ>...
    PC                                db 'Type PC: PC',
13, 10 , '$'
    PC_XT                            db 'Type PC: PC/XT',
13, 10, '$'
    AT                                db 'Type PC: AT',
13, 10, '$'
    PS2_1                            db 'Type PC: PS2
-®ѠГ«м 30', 10, 13, '$'
    PS2_2                            db 'Type PC: PS2
-®ѠГ«м 50 or 60', 10, 13, '$'
    PS2_3                            db 'Type PC: PS2
-®ѠГ«м 80', 10, 13, '$'
    PCjr                             db 'Type PC: PCjr',10,
13, '$'
    PC_CONVERTIBLE                  db 'Type PC: PC
Convertible',10, 13, '$'
    VERS                            db 'Version MS DOS:  .
', 10, 13, '$'
    OEM                             db 'OEM serial number:
', 10, 13, '$'
    USER                            db 'User serial
number:      ', 10, 13, '$'
DATA ENDS

```

CODE SEGMENT

ASSUME CS:CODE, DS:DATA,

ES:NOTHING, SS:ASTACK

;Ѡђђ-....,“ђ>

START: JMP BEGIN

TETR\_TO\_HEX PROC NEAR

```

and    AL,0Fh
cmp     AL,09
jbe     NEXT
add     AL,07
add     AL,30h
ret

```

NEXT:

TETR\_TO\_HEX ENDP

BYTE\_TO\_HEX PROC NEAR

;Ÿ Ѡв Ÿ AL ĩѓaѓŸ®ѠЂвѓп Ÿ ѠŸ

бѐ-ѣ®« иГбв-. зѐб« ѣ AX

бв аи п жѐда

BYTE\_TO\_HEX ENDP

WRD\_TO\_HEX PROC NEAR ;їГгГѣ®« ѣ 16 6/6 16-вѐ а ѣапѣ-®Ј® зѐб« , ѣ AX - зѐб«®, DI - ѣаГб ї®б«Гѣ-ГЈ® бѐ-ѣ®«

WRD\_TO\_HEX

BYTE\_TO\_DEC PROC NEAR  
- ѣаГб ї®«п -« ѣиГ® жѐдал

б®ѣГг!ѐв ѐбе®ѣ-л® ѣ ®в

loop\_bd: div

```
push CX
mov AH,AL
call TETR_TO_HEX
xchg AL,AH
mov CL,4
shr AL,CL
call TETR_TO_HEX ;ѣ AL
```

```
pop CX
;ѣ AH -« ѣи п
ret
```

```
push BX
mov BH,AH
call BYTE_TO_HEX
mov [DI],AH
dec DI
mov [DI],AL
dec DI
mov AL,BH
xor AH,AH
call BYTE_TO_HEX
mov [DI],AH
dec DI
mov [DI],AL
pop BX
ret
ENDP
```

; їГггГѣ®« ѣ ®в ѣ 106/6, SI

```
push AX ; AL
```

```
push CX
push DX
xor AH,AH
xor DX,DX
mov CX,10
CX
or DL,30h
mov [SI],DL
dec SI
xor DX,DX
cmp AX,10
jae loop_bd
cmp AL,00h
```

end\_1: pop

BYTE\_TO\_DEC ENDP

PRINT PROC NEAR  
нЕа -

PRINT ENDP

TYPE\_PC PROC NEAR

TYPE\_PC ENDP

PRINT\_TYPE\_PC PROC NEAR

```
je      end_1
or      AL, 30h
mov     [SI], AL
DX
pop     CX
pop     AX
ret
```

;İГз вМ 6®®ЎГ-Ёп -

```
push    AX
mov     AH, 09h
int     21h
pop     AX
ret
```

```
;İ®«ГзГ-ЁГ вЁİ ЦЉ
push    DS
mov     BX, 0F000H
mov     DS, BX
sub     AX, AX
mov     AH, DS:[0FFFEH]
pop     DS
ret
```

```
;ЎлЎ®я вЁİ ЦЉ
push    AX
push    BX
push    DI
```

```
mov     DX, OFFSET PC
cmp     AH, 0FFh
je      print_msg
```

```
mov     DX, OFFSET PC_XT
cmp     AH, 0FEh
je      print_msg
```

```
mov     DX, OFFSET PC_XT
cmp     AH, 0FBh
je      print_msg
```

```
mov     DX, OFFSET AT
cmp     AH, 0FCh
je      print_msg
```

```
mov     DX, OFFSET PS2_1
```

PC\_CONVERTIBLE

PRINT\_TYPE\_PC ENDP

VERSION\_DOS PROC NEAR

VERSION\_DOS ENDP

OEM\_NUM PROC NEAR

```
cmp    AH, 0FAh
je      print_msg
```

```
mov     DX, OFFSET PS2_2
cmp     AH, 0FCh
je      print_msg
```

```
mov     DX, OFFSET PS2_3
cmp     AH, 0F8h
je      print_msg
```

```
mov     DX, OFFSET PCjr
cmp     AH, 0FDh
je      print_msg
```

```
mov     DX, OFFSET
```

```
cmp     AH, 0F9h
je      print_msg
```

```
mov     AL, AH
call    BYTE_TO_HEX
mov     DX, AX
```

```
print_msg:
call    PRINT
pop     DI
pop     BX
pop     AX
ret
```

```
push    AX
push    SI
mov     SI, OFFSET VERS
add     SI, 10h
call    BYTE_TO_DEC
add     SI, 3h
mov     AL, AH
call    BYTE_TO_DEC
mov     DX, OFFSET VERS
call    PRINT
pop     SI
pop     AX
ret
```

```
push    AX
```

OEM\_NUM ENDP

USER\_NUM PROC NEAR

```
push BX
push SI
mov     AL, BH
mov     SI, OFFSET OEM
add     SI, 15h
call    BYTE_TO_DEC
mov     DX, OFFSET OEM
call    PRINT
pop     SI
pop     BX
pop     AX
ret
```

```
push CX
push DI
push AX

mov     DI, OFFSET USER
add     DI, 19h
mov     AX, CX
call    WRD_TO_HEX
mov     AL, BL
mov     DI, OFFSET USER
add     DI, 14h
call    BYTE_TO_HEX
mov     [DI], AX
mov     DX, OFFSET USER
call    PRINT
pop     AX
pop     DI
pop     CX
ret
```

USER\_NUM ENDP

BEGIN:

```
mov     AX, DATA
mov     DS, AX
mov     BX, DS
call    TYPE_PC
call    PRINT_TYPE_PC
mov     AH, 30h
int     21h
call    VERSION_DOS
call    OEM_NUM
call    USER_NUM
xor     AL, AL
mov     AH, 4Ch
```



CODE ENDS  
END START

int 21h