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.8382 _____ Синельников М.Р

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.

1)

рисунок 1 — результат работы файла .COM

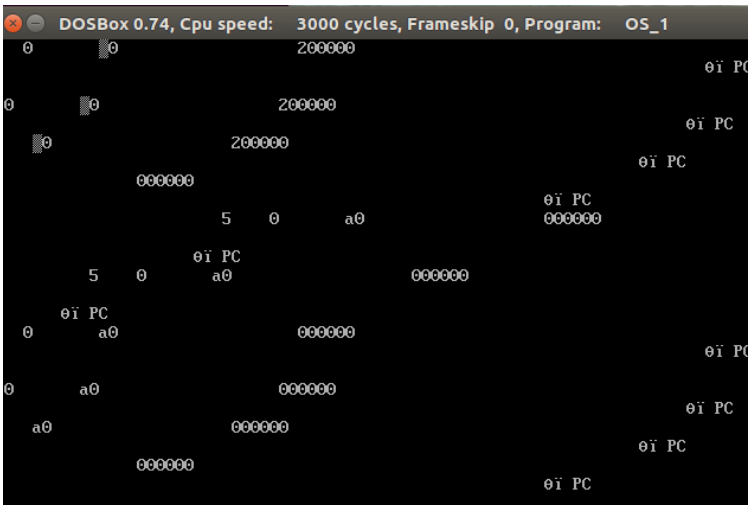

рисунок 2 — результат работы плохого .EXE

2)

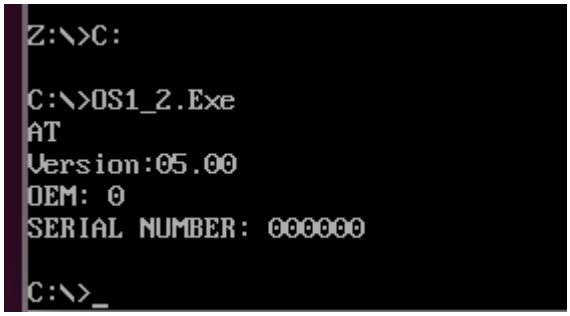

рисунок 3 — результат работы хорошего .EXE

3)

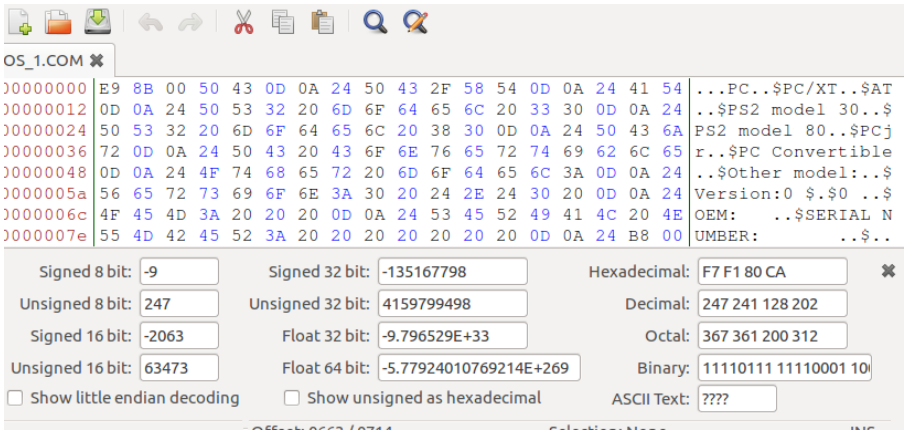

рисунок 4 — пример файла .COM в 16 ричном виде

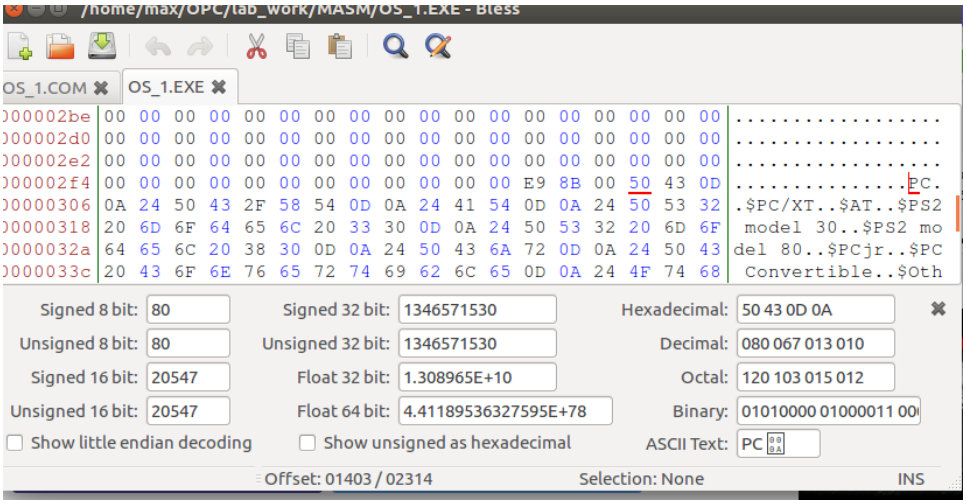

рисунок 5 — пример плохого .EXE в 16 ричном виде

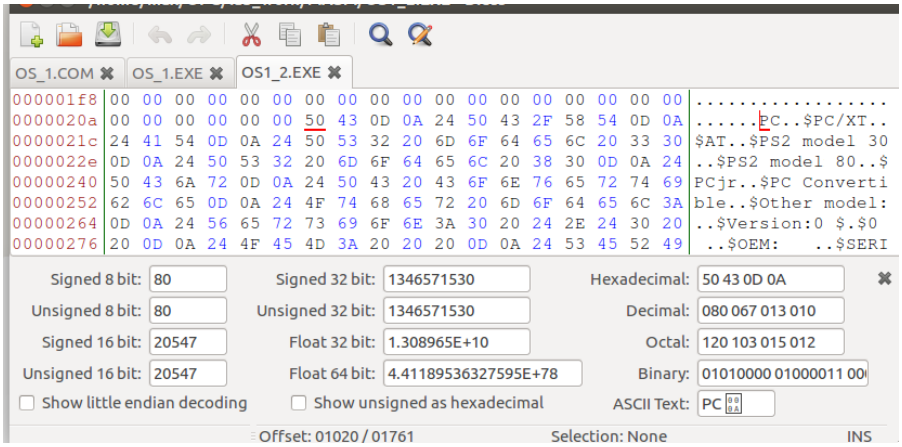

рисунок 6 — пример хорошего .EXE в 16 ричном виде

4)

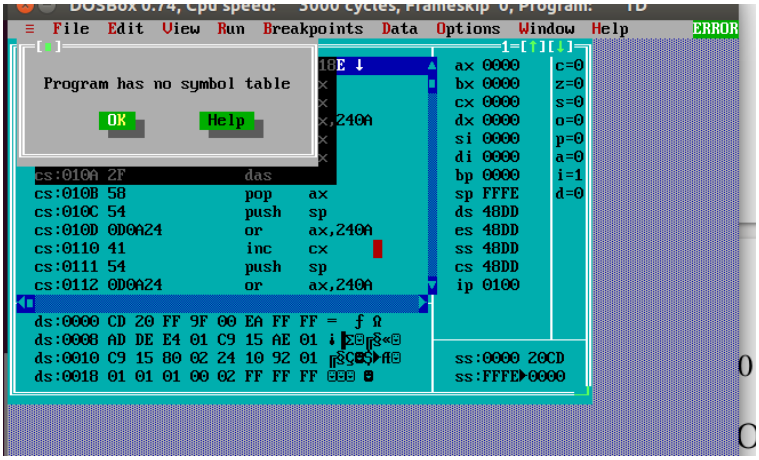

рисунок 7 — файл .COM в отладке

5)

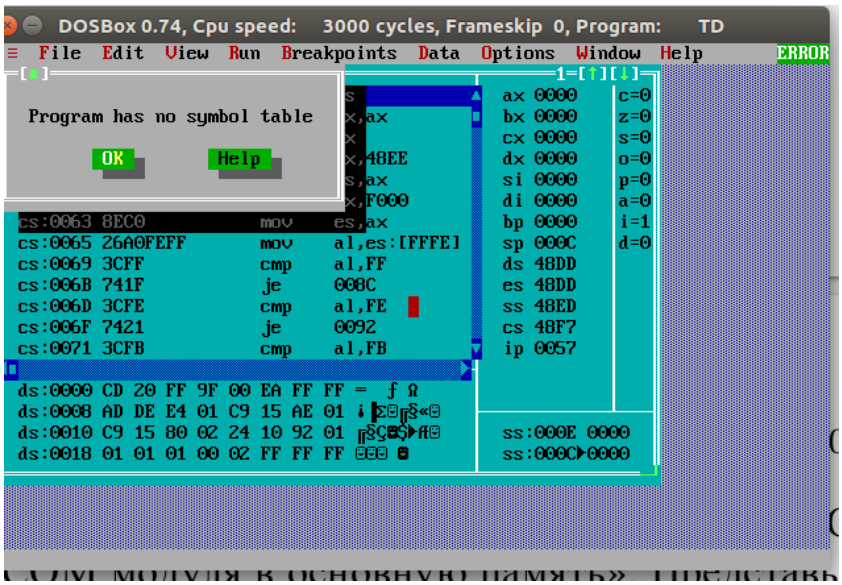

рисунок 8 — файл хорошего .EXE в отладке

Контрольные вопросы.

Отличия исходных текстов COM и EXE программ

1) Сколько сегментов должна содержать COM программа?

Com программа дожна содержать один сегмент.

2) EXE программа?

Exe программа три, поскольку присутствует разделение на сегменты стека, данных и кода.

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

Директива ASSUME, сообщающая компилятору о том, какой сегмент к какому сегментному регистру привязан и директива ORG, резервирующая место для PSP.

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

Есть ограничение только на размер кода + данных - не более 64к, т.е. размера сегмента.

Отличия форматов файлов COM и EXE модулей

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

Com файл содержит один сегмент, первые 100h байт отводятся для PSP, сам файл не превышает 64КБ. Код распологается с адреса 0h(рис. 4).

2) Какова структура плохого файала EXE? С какого адреса располагается код? Что располагается с адреса 0?

Плохой EXE имеет один сегмент. Код начинается с адреса 303h. С адреса 0h распологаются настройки(рис. 5)

3) Какова структура файла хорошего EXE? Чем он отличается от файла плохого EXE?

Хороший EXE имеет три сегмента - сегмент данных, кода и стека. Код начинается с адреса 210h(рис. 6). Отличие от плохого EXE состоит в количестве сегментов, а также в номере адреса, с которого начинается код.

Загрузка COM модуля в основную память

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

Код распологается с адреса 100h(рис. 7).

2) Что располагается с адреса 0?

С адреса 0 распологается PSP.

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

Сегментные регистры равняются 48DD(рис. 7). Они указывают на PSP.

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

Вершина стека указывает на FFFF. Под стек отводится свободная память, которая не занята программой. Адреса от FFFF до адреса, на котором кончается программа.

Загрузка хорошего EXE модуля в основную память

1) Как загружается хороший EXE? Какие значения имеют сегментные регистры?

Регистр SS указывает на начало сегмента стека, регистр DS на начало PSP, регистр CS на начало сегмента кода. DS равен 48DD, ES равен 48DD, SS равен 48ED, CS равен 48F7(рис. 8).

2) На что указывают регистры DS и ES?

DS и ES указывают на начало PSP.

3) Как определяется стек?

При помощи директивы ASSUME, которая устанавливает SS на начало сегмента стека.

4) Как определяется точка входа?

При помощи директивы END. Программа начинает выполняться с той метки, которая идёт после этой директивы.

Выводы.

Были получены навыки по работе с файлами COM и EXE. Также были реализованы программы, определяющие тип PC и версию системы для файлов .COM и .EXE.

Приложение А. Исходный код программы OS_1.asm

```
TESTPC          SEGMENT
                ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
                ORG 100H ; резервирование места для PSP
START: JMP begin

PC db 'PC',0DH,0AH,'$'
XT db 'PC/XT',0DH,0AH,'$'
AT db 'AT',0DH,0AH,'$'
PS2_30 db 'PS2 model 30',0DH,0AH,'$'
PS2_80 db  'PS2 model 80',0DH,0AH,'$'
PCjr db  'PCjr',0DH,0AH,'$'
PC_Covertible db  'PC Convertible',0DH,0AH,'$'

OTHER_MODEL db  'Other model:',0DH,0AH,'$'
VERSION db 'Version:0 $'
POINT db  '.$'
MODIFICATION db '0 ',0DH,0AH,'$'
OEM db  'OEM:   ',0DH,0AH,'$'
SERIAL_NUMBER db 'SERIAL NUMBER:       ', 0DH,0AH,'$'

begin:
        mov ax,0F000H
        mov es,ax
        mov al,es:[0FFFEH]
        cmp al,0FFH
        je PC_label
        cmp al,0FEH
        je XT_label
        cmp al,0FBH
        je XT_label
        cmp al,0FCH
        je AT_label
        cmp al,0FAH
        je PS2_30_label
        cmp al,0F8H
        je PS2_80_label
        cmp al,0FDH
        je PCjr_label
        cmp al, PC_Covertible_label
        jmp Other_version_label


PC_label:
        mov dx,offset PC
        jmp writeModel

XT_label:
        mov dx,offset XT
        jmp writeModel

AT_label:
        mov dx,offset AT
        jmp writeModel

PS2_30_label:
        mov dx,offset PS2_30
```

```
        jmp writeModel

PS2_80_label:
        mov dx,offset PS2_80
        jmp writeModel

PCjr_label:
        mov dx,offset PCjr
        jmp writeModel

PC_Covertible_label:
        mov dx,offset PC_Covertible
        jmp writeModel

Other_version_label:
        mov dx,offset OTHER_MODEL
        push ax
        mov ah,09h
        int 21h
        pop ax
        call BYTE_TO_HEX
        push ax
        mov dx, ax
        mov ah, 02h
        int 21h
        pop ax
        jmp OS_version_label

writeModel:
        push ax
        mov ah,09h
        int 21h
        pop ax

OS_version_label:
        mov ah,30h
        int 21h
        cmp al,0
        je MOD_label
        mov si, offset VERSION
        add si,8
        cmp al,9
        jg continue1
        add si,1
        continue1:
        push ax
        call BYTE_TO_DEC
        mov dx, offset VERSION
        mov ah, 09h
        int 21h
        pop ax
        jmp MOD_label

MOD_label:
        push ax
        mov dx,offset POINT
        mov ah, 09h
        int 21h
        pop ax
        push ax
        mov si, offset MODIFICATION
        mov al,ah
```

```
        cmp al,9
        jg continue2
        add si,1
        continue2:
        call BYTE_TO_DEC
        mov dx, offset MODIFICATION
        mov ah, 09h
        int 21h
        pop ax

OEM_label:
        push ax
        mov si, offset OEM
        add si,5
        mov al, bh
        call BYTE_TO_DEC
        mov dx, offset OEM
        mov ah, 09h
        int 21h
        pop ax

SERIAL_NUMBER_label:
        mov di, offset SERIAL_NUMBER
        add di,20
        mov ax, cx
        call WRD_TO_HEX
        mov al, bl
        call BYTE_TO_HEX
        sub di,2
        mov [di], ax
        mov dx, offset SERIAL_NUMBER
        mov ah, 09h
        int 21h


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 шест. 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 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
   ret
BYTE_TO_DEC ENDP

TESTPC          ENDS
          END START
```

Приложение Б. Исходный код программы OS1_2.asm

```
AStack  SEGMENT         STACK
                DB 12 DUP(0)
AStack  ENDS


DATA    SEGMENT
        PC db 'PC',0DH,0AH,'$'
        XT db 'PC/XT',0DH,0AH,'$'
        AT db 'AT',0DH,0AH,'$'
        PS2_30 db 'PS2 model 30',0DH,0AH,'$'
        PS2_80 db  'PS2 model 80',0DH,0AH,'$'
        PCjr db  'PCjr',0DH,0AH,'$'
        PC_Covertible db  'PC Convertible',0DH,0AH,'$'
        OTHER_MODEL db  'Other model:',0DH,0AH,'$'
        VERSION db 'Version:0 $'
        POINT db  '.$'
        MODIFICATION db '0  ',0DH,0AH,'$'
        OEM db  'OEM:   ',0DH,0AH,'$'
        SERIAL_NUMBER db 'SERIAL NUMBER:       ', 0DH,0AH,'$'
DATA ENDS

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


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 шест. 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 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
  ret
BYTE_TO_DEC ENDP

Main   PROC  far
           push ds
           sub  ax,ax
           push ax
           mov ax,DATA
           mov ds,ax
           mov ax,0F000H
           mov es,ax
           mov al,es:[0FFFEH]
           cmp al,0FFH
           je PC_label
           cmp al,0FEH
           je XT_label
           cmp al,0FBH
           je XT_label
           cmp al,0FCH
           je AT_label
           cmp al,0FAH
           je PS2_30_label
           cmp al,0F8H
           je PS2_80_label
           cmp al,0FDH
           je PCjr_label
           cmp al, PC_Covertible_label
```

```
        jmp Other_version_label


PC_label:
        mov dx,offset PC
        jmp writeModel

XT_label:
        mov dx,offset XT
        jmp writeModel

AT_label:
        mov dx,offset AT
        jmp writeModel

PS2_30_label:
        mov dx,offset PS2_30
        jmp writeModel

PS2_80_label:
        mov dx,offset PS2_80
        jmp writeModel

PCjr_label:
        mov dx,offset PCjr
        jmp writeModel

PC_Covertible_label:
        mov dx,offset PC_Covertible
        jmp writeModel

Other_version_label:
        mov dx,offset OTHER_MODEL
        push ax
        mov ah,09h
        int 21h
        pop ax
        call BYTE_TO_HEX
        push ax
        mov dx, ax
        mov ah, 02h
        int 21h
        pop ax
        jmp OS_version_label

writeModel:
        push ax
        mov ah,09h
        int 21h
        pop ax

OS_version_label:
        mov ah,30h
        int 21h
        cmp al,0
        je MOD_label
        mov si, offset VERSION
        add si,8
        cmp al,9
        jg continue1
        add si,1
        continue1:
```

```
        push ax
        call BYTE_TO_DEC
        add si, 1
        mov dx, offset VERSION
        mov ah, 09h
        int 21h
        pop ax
        jmp MOD_label

MOD_label:
        push ax
        mov dx,offset POINT
        mov ah, 09h
        int 21h
        pop ax
        push ax
        mov si, offset MODIFICATION
        mov al, ah
        cmp al,9
        jg continue2
        add si,1
        continue2:
        call BYTE_TO_DEC
        add si, 1
        mov dx, offset MODIFICATION
        mov ah, 09h
        int 21h
        pop ax

OEM_label:
        push ax
        mov si, offset OEM
        add si,5
        mov al, bh
        call BYTE_TO_DEC
        add si, 1
        mov dx, offset OEM
        mov ah, 09h
        int 21h
        pop ax

SERIAL_NUMBER_label:
        mov di, offset SERIAL_NUMBER
        add di,20
        mov ax, cx
        call WRD_TO_HEX
        mov al, bl
        call BYTE_TO_HEX
        sub di, 2
        mov [di], ax
        mov dx, offset SERIAL_NUMBER
        mov ah, 09h
        int 21h
        ret


Main  ENDP
CODE  ENDS
END Main
```