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

Студент гр. 9383 _____ Звега А.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.

Задание.

1. Напишите текст исходного .COM модуля, который определяет тип PC и версию системы.

2. Напишите текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1 и постройте и отладьте его. Таким образом, будет получен «хороший» .EXE.

3. Сравните исходные тексты для .COM и .EXE модулей. Ответьте на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

4. Запустите FAR и откройте (F3/F4) файл загрузочного модуля .COM и файл «плохого» .EXE в шестнадцатеричном виде. Затем откройте (F3/F4) файл загрузочного модуля «хорошего» .EXE и сравните его с предыдущими файлами. Ответьте на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

5. Откройте отладчик TD.EXE и загрузите .COM. Ответьте на контрольные вопросы «Загрузка COM модуля в основную память». Представьте в отчете план загрузки модуля .COM в основную память.

6. Откройте отладчик TD.EXE и загрузите «хороший» .EXE. Ответьте на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

7. Оформление отчета в соответствии с требованиями. В отчете необходимо привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей – в отладчике.

Выполнение работы.

Шаг 1. На языке ассемблера был написан исходный код модуля COM определяющий тип PC и версию системы(Рисунок 1). Кроме этого был получен «плохой» EXE модуль, полученный из исходного кода для COM модуля(Рисунок 2).

Рисунок 1: результат работы COM

Рисунок 2: результат работы „плохого“ EXE

Шаг 2. Был написан исходный код EXE модуля, который выполняет те же функции что и COM модуль(Рисунок 3).

Рисунок 3: результат работы „хорошего“ EXE

Вопросы к шагу 3. «Отличия исходных текстов COM и EXE программ».

1. Сколько сегментов должна содержать COM-программа?

COM-программа ограничена размером одного сегмента, так как данные и код находятся в одном сегменте, стек же устанавливается на последнюю ячейку сегмента.

2. Сколько сегментов должна содержать EXE-программа?

EXE-программа должна содержать один сегмент или больше.

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

Обязательно должна быть директива org100h, так как в DOS первые 256 байт занимает PSP. Директивой ASSUME нужно связать сегменты данных и кода в один сегмент.

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

Нельзя использовать команды вида mov <регистр>, seg <имя сегмента>. Так как в COM-программе нет таблицы настроек.

Шаг 4. Было выполнено сравнение файлов COM и EXE в шестнадцатеричном виде (см. Рисунки 4,5,6).

Рисунок 4: COM

Рисунок 5: "плохой" EXE

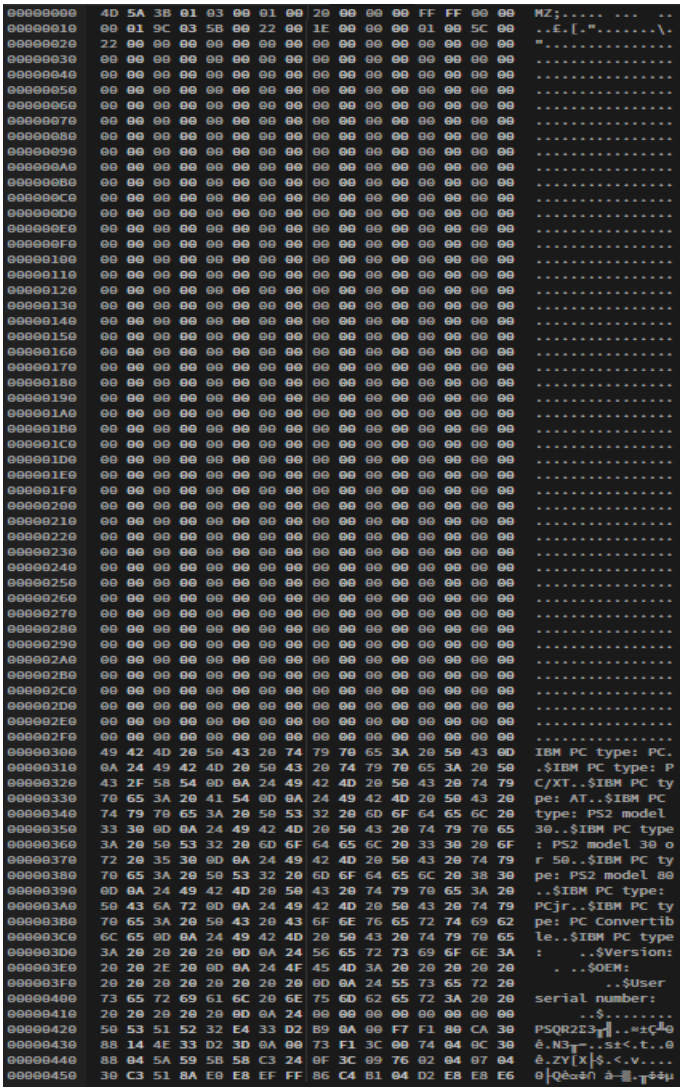

Рисунок 6: "хороший" EXE

Вопросы к шагу 4. «Отличия форматов файлов COM и EXE модулей».

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

Файл типа COM содержит команды и данные. Код располагается с адреса 0h.

2)Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с адреса 0?

Данные и код находятся в одном сегменте. Код расположен по адресу 300h. По адресу 0h располагается управляющая информация, заголовок и таблица настроек.

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

Файл состоит из трех сегментов: стек, данные и код. У файла неограниченный размер. С 0h адреса расположен заголовок, после него идет таблица настроек. У «хорошего» EXE выделяется память под стек между PSP и кодом. Различия: разделение сегментов и размер смещения (300h и 400h).

Вопросы к шагу 5. «Загрузка COM модуля в основную память»**.**

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

Сначала загружается PSP, затем код. Код располагается после PSP с адреса 100h.

2) Что располагается с адреса 0?

Располагается PSP(0h-100h).

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

При загрузке сегментные регистры указывают на начало PSP.

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

В COM модуле стек создается автоматически и занимает всю область сегмента. Адреса расположены в диапазоне 0h-ffffh.

Вопросы к шагу 6. «Загрузка «хорошего» EXE модуля в основную память».

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

Также как и COM. Сегментные регистры DS и ES устанавливаются на начало PSP, SS - на начало сегмента стека, CS — на начало сегмента команд.

2) На что указывают регистры DS и ES?

Сегментные регистры DS и ES указывают на начало PSP

3) Как определяется стек?

С помощью регистров SS и SP.

SS указывает на начало, а SS:SP на конец.

4) Как определяется точка входа?

Точка входа определяется директивой END + <метка или процедура>.

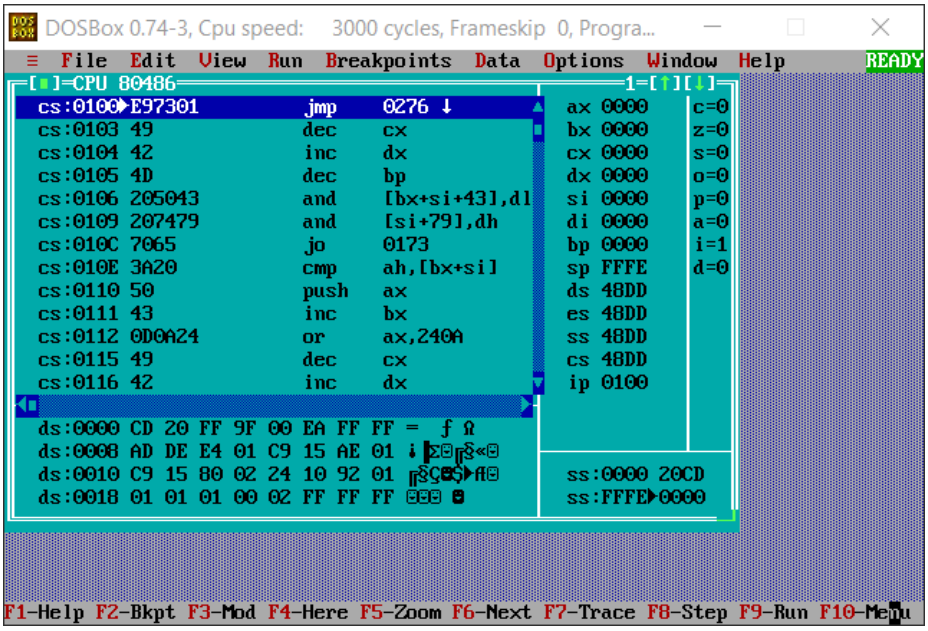

Рисунок 7: td lab1bin.com

Выводы.

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

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: lab1bin.asm

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

START: JMP BEGIN



pc db 'IBM PC type: PC',0dh,0ah,'$'
pc_xt db 'IBM PC type: PC/XT',0dh,0ah,'$'
at db 'IBM PC type: AT',0dh,0ah,'$'
ps2_30 db 'IBM PC type: PS2 model 30',0dh,0ah,'$'
ps2_50_60 db 'IBM PC type: PS2 model 30 or 50',0dh,0ah,'$'
ps2_80 db 'IBM PC type: PS2 model 80',0dh,0ah,'$'
pcjr db 'IBM PC type: PCjr',0dh,0ah,'$'
pc_convertible db 'IBM PC type: PC Convertible',0dh,0ah,'$'
unknown db 'IBM PC type:    ',0dh,0ah,'$'
version db 'Version:  . ',0dh,0ah,'$'
oem db 'OEM:            ',0dh,0ah,'$'
user_serial_number db 'User serial number:       ',0dh,0ah,'$'



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


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_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


BEGIN:
    XOR ax,ax
    MOV ax,0f000h
    MOV es,ax
    MOV al,es:[0fffeh]

    CMP al,0ffh
    JE Label_pc
    CMP al,0feh
    JE Label_pc_xt
```

```
    CMP al,0fbh
    JE Label_pc_xt
    CMP al,0fch
    JE Label_at
    CMP al,0fah
    JE Label_ps2_30
    CMP al,0fch
    JE Label_ps2_50_60
    CMP al,0f8h
    JE Label_ps2_80
    CMP al,0fdh
    JE Label_pcjr
    CMP al,0f9h
    JE Label_pc_convertible
    JNE label_unknown

label_pc:
    MOV dx,offset pc
    JMP print_version
label_pc_xt:
    MOV dx,offset pc_xt
    JMP print_version
label_at:
    MOV dx,offset at
    JMP print_version
label_ps2_30:
    MOV dx,offset ps2_30
    JMP print_version
label_ps2_50_60:
    MOV dx,offset ps2_50_60
    JMP print_version
label_ps2_80:
    MOV dx,offset ps2_80
    JMP print_version
```

```
label_pcjr:
    MOV dx,offset pcjr
    JMP print_version
labeL_PC_convertible:
    MOV dx,offset pc_convertible
    JMP print_version
label_unknown:
    CALL BYTE_TO_HEX
    MOV di, offset unknown+13
    MOV [di], ax
    MOV dx,offset unknown


print_version:
    MOV ah,09h
    INT 21h

    XOR bx, bx
    XOR ax, ax
    MOV ah, 30h
    INT 21h

    MOV si, offset version+9
    CALL BYTE_TO_DEC

    MOV al, ah
    MOV si, offset version+11
    CALL BYTE_TO_DEC

    MOV dx, offset version
    MOV ah,09h
    INT 21h

    MOV al, bh
```

```
    MOV si, offset oem+7
    CALL BYTE_TO_DEC

    MOV dx, offset oem
    MOV ah,09h
    INT 21h

    MOV al, bl
    CALL BYTE_TO_HEX

    MOV di, offset user_serial_number+20
    MOV [di], ax

    MOV ax, cx
    MOV di, offset user_serial_number+25
    CALL WRD_TO_HEX

    MOV dx, offset user_serial_number
    MOV ah,09h
    INT 21h

exit:
    XOR al,al
    MOV ah,4ch
    INT 21h
TESTPC ENDS
END START
```

Название файла: lab1.asm

```
AStack    SEGMENT  STACK
    DW 128 DUP(?)
AStack    ENDS
```

```
DATA SEGMENT
   pc db 'IBM PC type: PC',0dh,0ah,'$'
   pc_xt db 'IBM PC type: PC/XT',0dh,0ah,'$'
   at db 'IBM PC type: AT',0dh,0ah,'$'
   ps2_30 db 'IBM PC type: PS2 model 30',0dh,0ah,'$'
   ps2_50_60 db 'IBM PC type: PS2 model 30 or 50',0dh,0ah,'$'
   ps2_80 db 'IBM PC type: PS2 model 80',0dh,0ah,'$'
   pcjr db 'IBM PC type: PCjr',0dh,0ah,'$'
   pc_convertible db 'IBM PC type: PC Convertible',0dh,0ah,'$'
   unknown db 'IBM PC type:    ',0dh,0ah,'$'
   version db 'Version:  . ',0dh,0ah,'$'
   oem db 'OEM:            ',0dh,0ah,'$'
   user_serial_number db 'User serial number:       ',0dh,0ah,'$'
DATA ENDS


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


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


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


Main PROC FAR
   MOV ax, DATA
   MOV ds, ax

   XOR ax,ax
   MOV ax,0f000h
   MOV es,ax
   MOV al,es:[0fffeh]

   CMP al,0ffh
   JE label_pc
   CMP al,0feh
   JE label_pc_xt
   CMP al,0fbh
```

```
    JE label_pc_xt
    CMP al,0fch
    JE label_at
    CMP al,0fah
    JE label_ps2_30
    CMP al,0fch
    JE label_ps2_50_60
    CMP al,0f8h
    JE label_ps2_80
    CMP al,0fdh
    JE label_pcjr
    CMP al,0f9h
    JE label_pc_convertible
    JNE label_unknown

label_pc:
    MOV dx,offset pc
    JMP print_ibm_pc_version
label_pc_xt:
    MOV dx,offset pc_xt
    JMP print_ibm_pc_version
label_at:
    MOV dx,offset at
    JMP print_ibm_pc_version
label_ps2_30:
    MOV dx,offset ps2_30
    JMP print_ibm_pc_version
label_ps2_50_60:
    MOV dx,offset ps2_50_60
    JMP print_ibm_pc_version
label_ps2_80:
    MOV dx,offset ps2_80
    JMP print_ibm_pc_version
label_pcjr:
```

```
    MOV dx,offset pcjr
    JMP print_ibm_pc_version
label_pc_convertible:
    MOV dx,offset pc_convertible
    JMP print_ibm_pc_version
label_unknown:
    CALL BYTE_TO_HEX
    MOV di, offset unknown+13
    MOV [di], ax
    MOV dx,offset unknown


print_ibm_pc_version:
    MOV ah,09h
    INT 21h

    XOR bx, bx
    XOR ax, ax
    MOV ah, 30h
    INT 21h

    MOV si, offset version+9
    CALL BYTE_TO_DEC

    MOV al, ah
    MOV si, offset version+11
    CALL BYTE_TO_DEC

    MOV dx, offset version
    MOV ah,09h
    INT 21h

    MOV al, bh
    MOV si, offset oem+7
```

```
    CALL BYTE_TO_DEC

    MOV dx, offset oem
    MOV ah,09h
    INT 21h

    MOV al, bl
    CALL BYTE_TO_HEX

    MOV di, offset user_serial_number+20
    MOV [di], ax

    MOV ax, cx
    MOV di, offset user_serial_number+25
    CALL WRD_TO_HEX

    MOV dx, offset user_serial_number
    MOV ah,09h
    INT 21h

exit:
    XOR al,al
    MOV ah,4ch
    INT 21h

Main ENDP
CODE ENDS
END Main
```