# 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. 0382 |  | Морева Е.С. |
| Преподаватель |  | Ефремов М.А. |

Санкт-Петербург 2022

# 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.

**Шаг 1.** Написать текст исходно .COM модуля, который определяет тип PC и версию системы. Построить «плохой» .EXE модуль, полученный из исходного текста для .COM модуля.

**Шаг 2.** Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1, построить и отладить его. Таким образом будет получен «хороший» .EXE.

**Шаг 3.** Сравнить исходные тексты для .COM и .EXE модулей. Ответить на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

**Шаг 4.** Запустить FAR и открыть файлы загрузочного модуля .COM и файл

«плохого» .EXE в шестнадцатеричном виде. Затем открыть файл загрузочного модуля «хорошего» .EXE и сравнить его с предыдущими файлами. Ответить на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

**Шаг 5.** Открыть отладчик TD.EXE и загрузить .COM. Ответить на контрольные вопросы «Загрузка COM модуля в основную память». Представить в отчете план загрузки модуля .COM в основную память.

**Шаг 6.** Открыть отладчик TD.EXE и загрузить «хороший» .EXE. Ответить на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

# Выполнение работы.

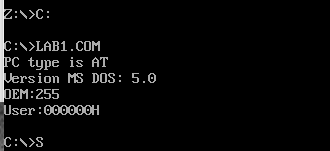
В ходе работы были написаны две процедуры: CHECK\_PC\_TYPE и CHECK\_OS\_VERS. Они реализованы с использованием шаблона из методических материалов.

C*HECK\_PC\_TYPE:* получаем информацию о типе PC из последнего байта ROM BIOS и выводим ее с помощью процедуры PRINT.

*CHECK\_OS\_VERS*: с помощью функции 30h прерывания 21h получаем информацию о версии MS DOS, затем с помощью процедур BYTE\_TO\_DEC, WRD\_TO\_HEX и BYTE\_TO\_HEX записываем полученную информацию в строки, далее выводим их.

В ходе работы было написано два файла lab1.asm (“Хороший” .com модуль и “плохой” .exe модуль) и prog\_exe.asm (“Хороший” .exe модуль).

Результаты их запусков:

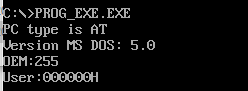


*Рисунок 1 – “Хороший” .com модуль*

**

*Рисунок 2 – “Плохой” .exe модуль*

Для того, чтобы получить “хороший” .exe модуль следует выделить сегменты стека, кода и данных, убрать ненужные директивы и установить сегмент данных.



*Рисунок 3 – “Хороший” .exe модуль*

Исходный программный код смотрите в приложении А.

# Контрольные вопросы.

1. Отличие исходных текстов COM и EXE программ:
   * Сколько сегментов должна содержать COM-программа? Ответ: Один сегмент. Код и данные находятся в одном сегменте, а стек генерируется автоматически.
   * ЕХЕ?

Ответ: один и больше

* + Какие директивы должны обязательно быть в тексте COM- программы?

Ответ: директива org 100h, которая смещает адресацию на размер PSP, а именно на 256 байтов, а также ASSUME, т.к. без этой директивы компилятор не будет знать какой сегмент относится к какому сегментному регистру.

* + Все ли форматы команд можно использовать в СОМ-программе? Ответ: нет, не все, в число запрещенных входят команды вида mov <регистр>, seg <имя сегмента> так как в COM-файле отсутствует таблица настроек.

2. Отличия форматов файлов COM и EXE модулей:

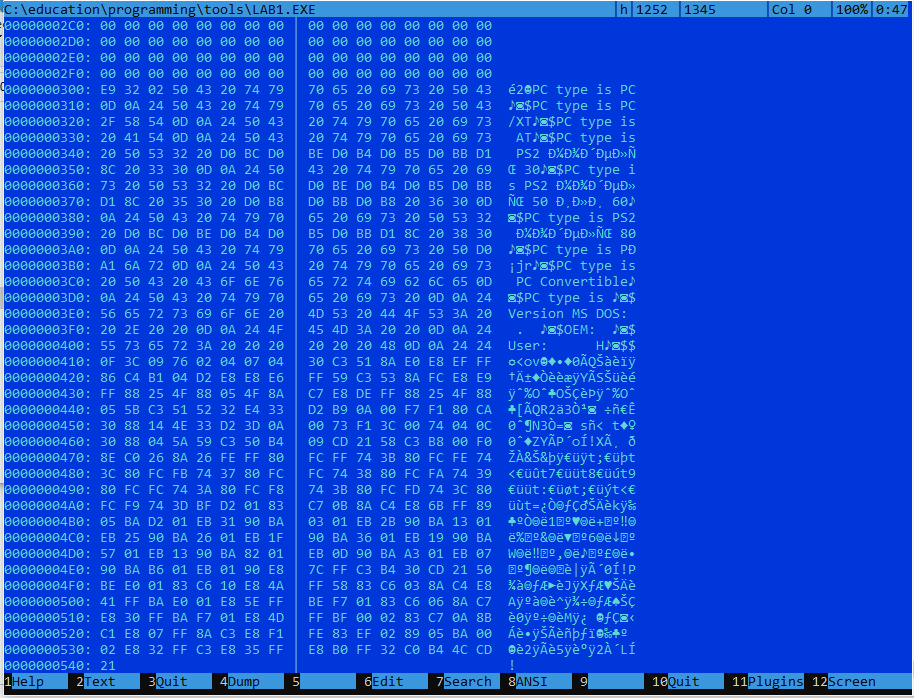
* + Какова структура файла COM? С какого адреса располагается код? Ответ: COM файлы состоят из одного сегмента, адресация которого начинается с 0h, но при загрузке программы в память, код сместиться на 100h, и будет начинаться сразу после PSP .



*Рисунок 4 – “Хороший” COM модуль в 16-ричном виде*

* + Какова структура файла “плохого” EXE? С какого адреса располагается код? Что располагается с адреса 0?

Ответ: В «плохом» EXE данные и код располагаются в одном сегменте, что для EXE файла некорректно, так как код и данные должны быть разделены на отдельные сегменты. Код располагается с адреса 300h, а с адреса 0h идёт таблица настроек.



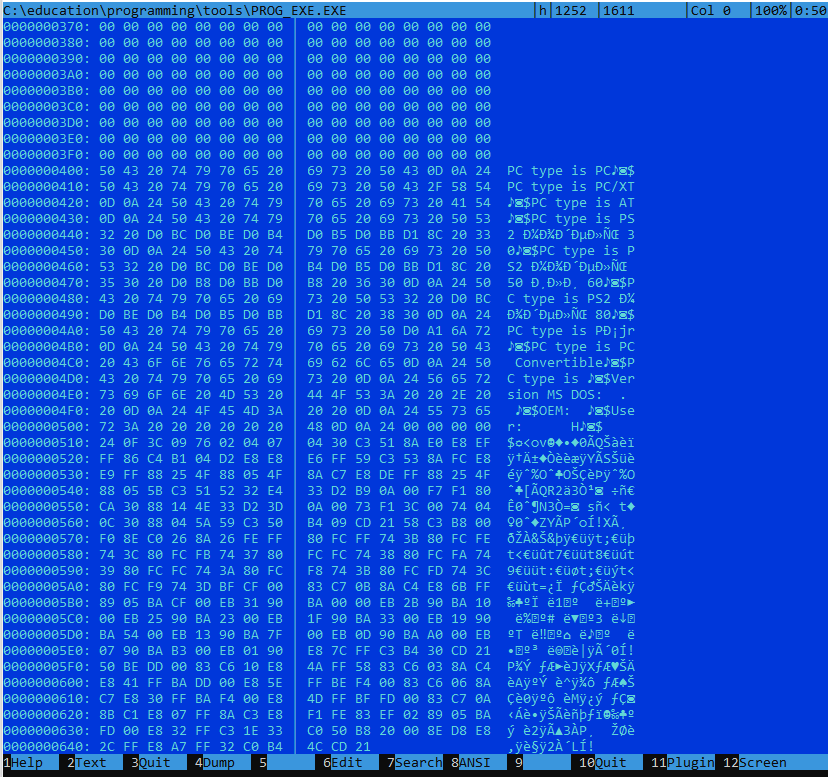
*Рисунок 5- “Плохой” ЕХЕ модуль в 16-ричном виде*

* + Какова структура файла “хорошего” ЕХЕ? Чем он отличается от файла “плохого” ЕХЕ?

Ответ: В “хорошем” EXE модуле файл начинается с заголовка и

relocation table, далее идёт программа, разбитая на сегменты, сначала

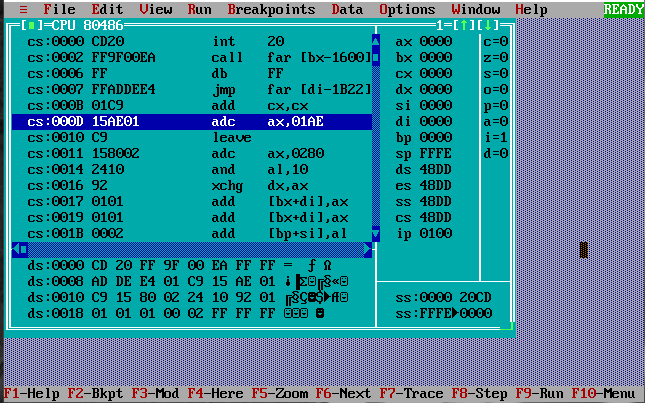
сегмент стека, сегмент данных и сегмент кода, в отличии от “плохого” ЕХЕ, где код и данные располагаются в одном сегменте. Кроме этого в отличии от “плохого” ЕХЕ отсутствует ненужное смещение на 256 байт.



*Рисунок 6 – “Хороший” ЕХЕ модуль в 16-ричном виде*

1. Загрузка СОМ модуля в основную память:
   * Какой формат загрузки модуля СОМ? С какого адреса располагается код?

Ответ: Определяется сегментный адрес участка ОП, у которого достаточно места для загрузки программы, образ COM-файла считывается с диска и помещается в память, начиная с PSP:0100h. После загрузки двоичного образа COM-программы сегментные регистры CS, DS, ES и SS указывают на PSP(в данном случае сегментные регистры указывают на 48DD), SP указывает на конец сегмента PSP(обычно FFFE), слово 00H помещено в стек, IP содержит 100H в результате команды JMP PSP:100H.



*Рисунок 7 – “Хорошая” СОМ программа в отладчике*

* + - Что располагается с адреса 0? Ответ: PSP (Program Segment Prefix).
    - Какие значения имеют сегментные регистры? На какие области памяти они указывают?

Ответ: Все сегментные регистры имеют значение 48DD и указывают на начало PSP.

* + - Как определяется стек? Какую область памяти он занимает? Какие адреса?

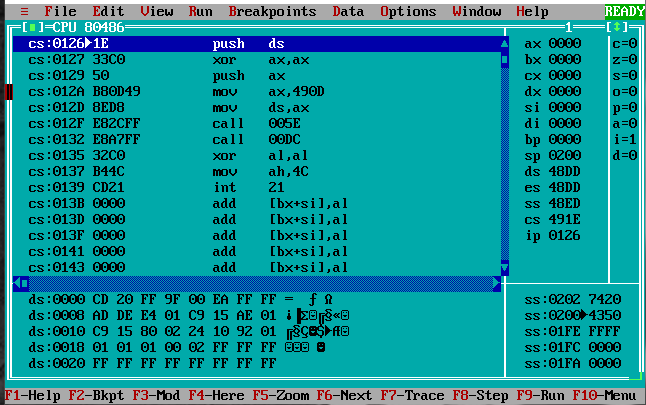
Ответ: Стек генерируется автоматически при создании COM-программы. SS – на начало (0h), регистр SP указывает на конец стека (FFFEh).Таким образом, стек занимает адреса SS:0000h-SS:FFFEh.

1. Загрузка “хорошего” ЕХЕ модуля в основную память
   * Как загружается “хороший” ЕХЕ? Какие значения имеют сегментные регистры?

Ответ: Аналогично COM модулю ЕХЕ загружается со смещением относительно PSP – 100h. Значения регистров DS=ES=48DD, CS=491Е, SS=48ED.

* + На что указывают регистры DS и ES? Ответ: На начало сегменты PSP.
  + Как определяется стек?

Ответ: он определяется вручную с помощью директивы SEGMENT STACK, в которой указывается размер стека.

* + Как определяется точка входа? Ответ: с помощью директивы END.

*Рисунок 8 – “Хороший” ЕХЕ модуль в отладчике*

# Выводы.

В ходе 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

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Название файла: lab1.asm

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:TESTPC, SS:TESTPC

ORG 100H

START: JMP BEGIN

; Данные

PC db 'PC type is PC',0DH,0AH,'$'

PC\_XT db 'PC type is PC/XT',0DH,0AH,'$'

AT db 'PC type is AT',0DH,0AH,'$'

PS2\_30 db 'PC type is PS2 модель 30',0DH,0AH,'$'

PS2\_50\_60 db 'PC type is PS2 модель 50 или 60',0DH,0AH,'$'

PS2\_80 db 'PC type is PS2 модель 80',0DH,0AH,'$'

PCJR db 'PC type is PСjr',0DH,0AH,'$'

PC\_CONVERTIBLE db 'PC type is PC Convertible',0DH,0AH,'$'

PC\_OTHER db 'PC type is ',0DH,0AH,'$'

DOS db 'Version MS DOS: . ',0DH,0AH,'$'

OEM db 'OEM: ',0DH,0AH,'$'

USER db 'User: H', 0DH, 0AH,'$'

; Процедуры

;-----------------------------------------------------

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

;-------------------------------

PRINT PROC near

push AX

mov AH,09h

int 21h

pop AX

ret

PRINT ENDP

CHECK\_PC\_TYPE PROC near

mov ax,0F000h

mov es,ax

mov ah,es:[0FFFEh]

;mov ah,0EBh

cmp ah,0FFh

je \_pc

cmp ah,0FEh

je \_xt

cmp ah, 0FBh

je \_xt

cmp ah, 0FCh

je \_at

cmp ah, 0FAh

je \_ps230

cmp ah, 0FCh

je \_ps25060

cmp ah, 0F8h

je \_ps280

cmp ah, 0FDh

je \_pcjr

cmp ah, 0F9h

je \_pcconv

;если нет в таблице

mov di,offset PC\_OTHER

add di,11

mov al,ah

call BYTE\_TO\_HEX

mov [di], ax

mov dx,offset PC\_OTHER

jmp \_out

\_pc:

mov dx,offset PC

jmp \_out

\_xt:

mov dx,offset PC\_XT

jmp \_out

\_at:

mov dx,offset AT

jmp \_out

\_ps230:

mov dx,offset PS2\_30

jmp \_out

\_ps25060:

mov dx,offset PS2\_50\_60

jmp \_out

\_ps280:

mov dx,offset PS2\_80

jmp \_out

\_pcjr:

mov dx,offset PCJR

jmp \_out

\_pcconv:

mov dx,offset PC\_CONVERTIBLE

jmp \_out

\_out:

call PRINT

ret

CHECK\_PC\_TYPE ENDP

CHECK\_OS\_VERS PROC near

mov ah,30h

int 21h

push ax

mov si,offset DOS

add si,16

call BYTE\_TO\_DEC

pop ax

add si,3

mov al,ah

call BYTE\_TO\_DEC

mov dx,offset DOS

call PRINT

mov si,offset OEM

add si,6

mov al,bh

call BYTE\_TO\_DEC

mov dx ,offset OEM

call PRINT

mov di, offset USER

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 USER

call PRINT

ret

CHECK\_OS\_VERS ENDP

; Код

BEGIN:

call CHECK\_PC\_TYPE

call CHECK\_OS\_VERS

xor AL,AL

mov AH,4Ch

int 21H

TESTPC ENDS

END START

Название файла: prog\_exe.asm

;стек

ASTACK SEGMENT stack

dw 256 dup(?)

ASTACK ENDS

; Данные

DATA SEGMENT

PC db 'PC type is PC',0DH,0AH,'$'

PC\_XT db 'PC type is PC/XT',0DH,0AH,'$'

AT db 'PC type is AT',0DH,0AH,'$'

PS2\_30 db 'PC type is PS2 модель 30',0DH,0AH,'$'

PS2\_50\_60 db 'PC type is PS2 модель 50 или 60',0DH,0AH,'$'

PS2\_80 db 'PC type is PS2 модель 80',0DH,0AH,'$'

PCJR db 'PC type is PСjr',0DH,0AH,'$'

PC\_CONVERTIBLE db 'PC type is PC Convertible',0DH,0AH,'$'

PC\_OTHER db 'PC type is ',0DH,0AH,'$'

DOS db 'Version MS DOS: . ',0DH,0AH,'$'

OEM db 'OEM: ',0DH,0AH,'$'

USER db 'User: H', 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

;-------------------------------

PRINT PROC near

push AX

mov AH,09h

int 21h

pop AX

ret

PRINT ENDP

CHECK\_PC\_TYPE PROC near

mov ax,0F000h

mov es,ax

mov ah,es:[0FFFEh]

cmp ah,0FFh

je \_pc

cmp ah,0FEh

je \_xt

cmp ah, 0FBh

je \_xt

cmp ah, 0FCh

je \_at

cmp ah, 0FAh

je \_ps230

cmp ah, 0FCh

je \_ps25060

cmp ah, 0F8h

je \_ps280

cmp ah, 0FDh

je \_pcjr

cmp ah, 0F9h

je \_pcconv

;если нет в таблице

mov di,offset PC\_OTHER

add di,11

mov al,ah

call BYTE\_TO\_HEX

mov [di], ax

mov dx,offset PC\_OTHER

jmp \_out

\_pc:

mov dx,offset PC

jmp \_out

\_xt:

mov dx,offset PC\_XT

jmp \_out

\_at:

mov dx,offset AT

jmp \_out

\_ps230:

mov dx,offset PS2\_30

jmp \_out

\_ps25060:

mov dx,offset PS2\_50\_60

jmp \_out

\_ps280:

mov dx,offset PS2\_80

jmp \_out

\_pcjr:

mov dx,offset PCJR

jmp \_out

\_pcconv:

mov dx,offset PC\_CONVERTIBLE

jmp \_out

\_out:

call PRINT

ret

CHECK\_PC\_TYPE ENDP

CHECK\_OS\_VERS PROC near

mov ah,30h

int 21h

push ax

mov si,offset DOS

add si,16

call BYTE\_TO\_DEC

pop ax

add si,3

mov al,ah

call BYTE\_TO\_DEC

mov dx,offset DOS

call PRINT

mov si,offset OEM

add si,6

mov al,bh

call BYTE\_TO\_DEC

mov dx ,offset OEM

call PRINT

mov di, offset USER

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 USER

call PRINT

ret

CHECK\_OS\_VERS ENDP

MAIN PROC far

push DS

xor AX,AX

push AX

mov AX,DATA

mov DS,AX

call CHECK\_PC\_TYPE

call CHECK\_OS\_VERS

xor AL,AL

mov AH,4Ch

int 21H

MAIN ENDP

CODE ENDS

END MAIN