**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Операционные системы»**

Тема: Исследование структур загрузочных модулей

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Быков И. В. |
| Преподаватель |  | Губкин А. Ф. |

Санкт-Петербург

2018

# Постановка задачи.

* 1. **Цель работы.**

Исследование различий в структурах исходных текстов модулей типов .СОМ и .ЕХЕ, структур файлов загрузочных модулей и способов их загрузки в основную память.

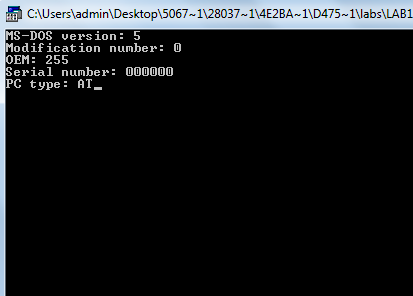
* 1. **Функции и структуры данных.**

|  |  |
| --- | --- |
| **Название процедуры** | **Описание процедуры** |
| WRITE\_MSG | Вывод сообщения на экран |
| TETR\_TO\_HEX | Десятичная цифра переводится в код символа |
| BYTE\_TO\_HEX | Байт в AL переводится в два символа шестн. числа в AX |
| WRD\_TO\_HEX | Перевод в 16 с/с 16-ти разрядного числа |
| BYTE\_TO\_DEC | Перевод байта в 16-ной с/с в символьный код в 10-ной с/с |
| OS\_VERSION | Определение версии ОС, ее модификации, ОЕМ номера и серийного номера |
| PC\_TYPE | определение типа ОС |

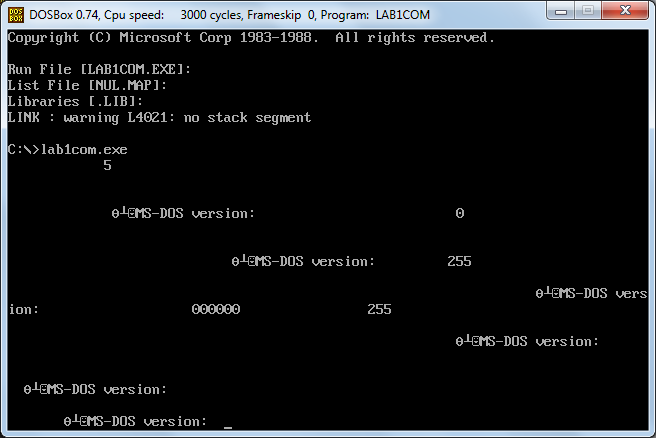
* 1. **Последовательность действий, выполняемых утилитой.**

1. Определение и вывод на экран версии IBM PC.
2. Определение и вывод на экран версии MS-DOS.
3. Определение и вывод на экран серийного номера OEM.
4. Определение и вывод на экран серийного номера пользователя.

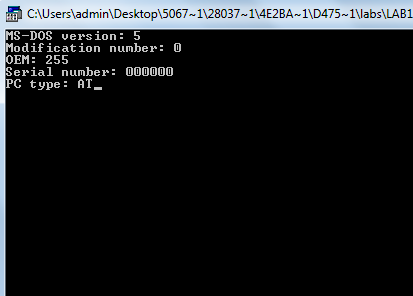
Результаты работы .СОМ файла lab1com.com, плохого .ЕХЕ файла lab1com.exe, хорошего .EXE файла представлены на рис. 1, рис. 2, рис. 3 соответственно.



*Рисунок 1. Результат выполнения программы laba1com.com*

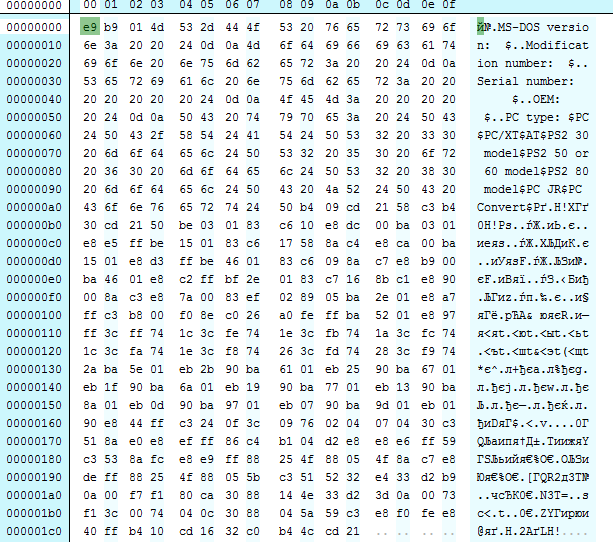


*Рисунок 2. Результат выполнения программы laba1com.exe*

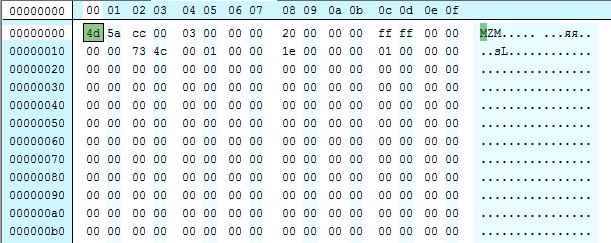
**

*Рисунок 3. Результат выполнения программы laba1exe.exe*

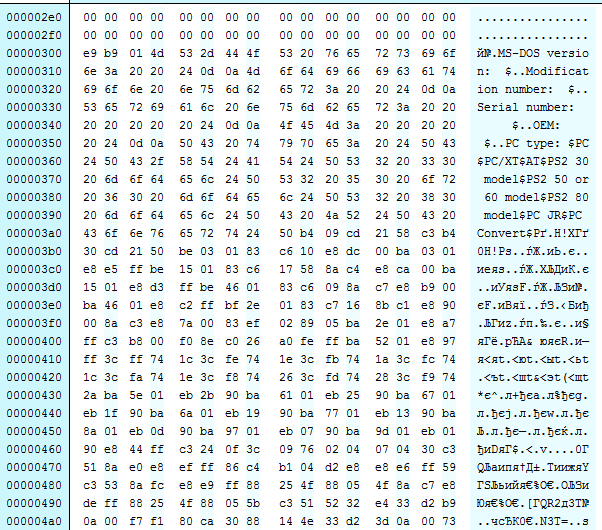
Шестнадцатеричные представления .COM файла, хорошего .ЕХЕ файла, плохого .ЕХЕ файла представлены на рис. 4, рис. 5, рис. 6 соответственно (см. след. страницу).



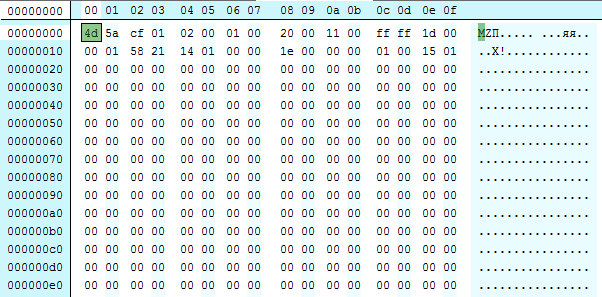
*Рисунок 4. HEX представление .СОМ файла.*

**

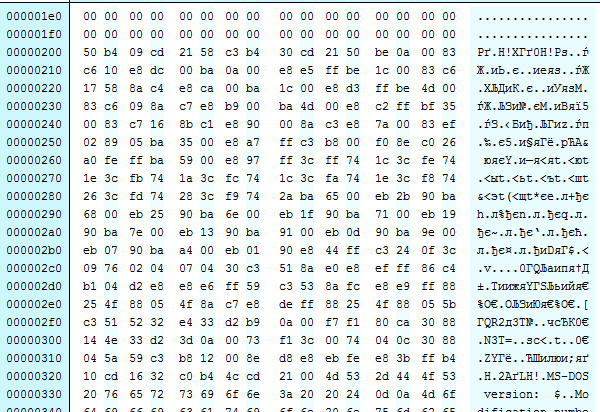
*…*

**

*Рисунок 5. HEX представление плохого .EXE файла.*

**

…

****

*Рисунок 6. HEX представление хорошего .EXE файла.*

**Вывод.**

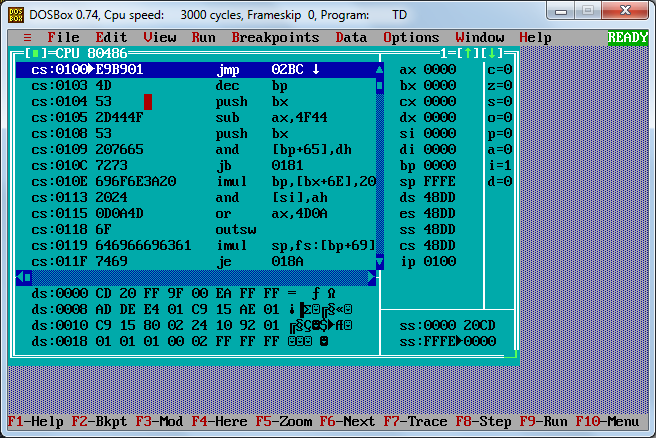
В результате выполнения лабораторной работы были исследованы различия в структурах исходных текстов модулей типов .СОМ и .ЕХЕ, структур файлов загрузочных модулей и способов их загрузки в основную память. Текст реализованной программы laba\_1.asm приложен в Приложении A.

# Ответы на контрольные вопросы.

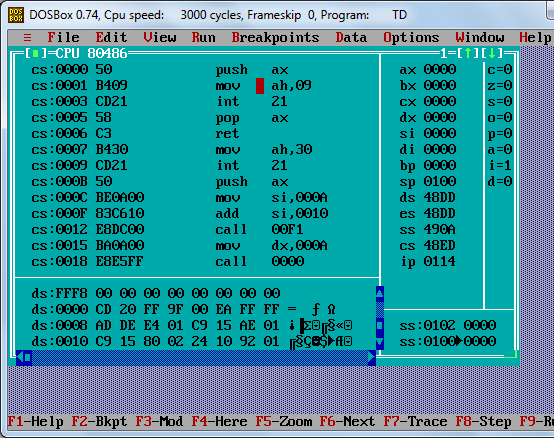
* 1. **Отличия исходных текстов COM и EXE программ**
     1. *Сколько сегментов должна содержать COM-программа?*
* Ровно один сегмент.
  + 1. *EXE-программа?*
* >= 1 сегмента
  + 1. *Какие директивы должны обязательно быть в тексте COM-программы?*
* ORG 100h. Этой директивой мы задаем смещение в 256 байт от нулевого адреса, так как при загрузке модуля в ОП в начале .COM – программы определяется 256 байтный префикс программного сегмента.
  + 1. *Все форматы команд можно использовать в COM-программе?*
* Нет. Нельзя использовать команды MOV, SEGMENT,CODE, REGISTER и команды, использующие дальнюю адресацию.

В данных командах используется таблица разметки (relocation table), в которых содержатся адреса сегментов. Данная таблица есть только в .EXE файлах.

* 1. **Отличия форматов файлов COM и EXE программ**
     1. *Какова структура файла COM? С какого адреса располагается код?*
* Структура .СОМ файла – данные и машинные команды. Код располагается с 0h, а при загрузке модуля в ОП устанавливается смещение в 100h
  + 1. *Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с адреса 0?*
* Данные и код содержатся в одном сегменте. Код располагается с 300h, 200h байт идет на заголовок, еще 100h за счет сдвига директивой ORG 100h. С 0 располагается заголовок, как сказано ранее.
  + 1. *Какова структура «хорошего» EXE? Чем он отличается от файла «плохого» EXE?*
* Данные, стек и код хорошего .EXE файла разделены по сегментам. Не требуется директива ORF 100h, так как загрузчик автоматически положит программу после PSP.
  1. **Загрузка COM модуля в основную память**



* + 1. *Какой формат загрузки модуля COM? С какого адреса располагается код?*
* После загрузки .COM модуля в память сегментные регистры указывают на PSP, сам код располагается с 100h.
  + 1. *Что располагается с адреса 0?*
* Сегмент PSP
  + 1. *Какие значения имеют сегментные регистры? На какие области памяти они указывают?*
* Все сегментные регистры равны 48DD и указывают на начало PSP
  + 1. *Как определяется стек? Какую область памяти он занимает? Какие адреса?*
* Cтек определяется автоматически, занимает весь сегмент программы. SP указывает на конец стека FFFEh, SS – на начало стека 0h. Диапазон адресов: 0h – FFFEh.
  1. **Загрузка «хорошего» EXE модуля в основную память**



* + 1. *Как загружается «хороший» EXE? Какие значения имеют сегментные регистры?*
* DS, ES устанавливаются на начало сегмента PSP; SS – на начало сегмента стека; СS – начало сегмента команд; в IP загружаем смещение точки входа
  + 1. *На что указывают регистры DS и ES?*
* на начало сегмента PSP
  + 1. *Как определяется стек?*
* директивой .STACK, которая определяет начало сегмента стека, выделяя «размер» байт. При исполнении в SS находится адрес начала сегмента стека, в SP - его вершины.
  + 1. *Как определяется точка входа?*
* директивой END, после нее ставится метка, на которую программа переходит при запуске

# Приложение А. lab1сom.asm

;BYKOV I V. 6383. .asm code to .com file

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

ORG 100h

START: JMP BEGIN

; DATA

VERSION db "MS-DOS version: $"

MODIFICATION db 13, 10, "Modification number: $"

SERIAL db 13, 10, "Serial number: $"

OEM db 13, 10, "OEM: $"

PCSTRING db 13, 10, "PC type: $"

PC db "PC$"

PCXT db "PC/XT$"

PCAT db "AT$"

PS2M\_30 db "PS2 30 model$"

PS2M\_50\_60 db "PS2 50 or 60 model$"

PS2M\_80 db "PS2 80 model$"

PCJR db "PC JR$"

PCCONV db "PC Convert$"

; PROCEDURES

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

WRITE\_MSG PROC near

push ax

mov ah, 09h

int 21h

pop ax

ret

WRITE\_MSG ENDP

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

OS\_VERSION PROC near

mov ah, 30h

int 21h

; al - version number

; ah - modification number

; bh - OEM serial number

; bl:cx - user serial number

push ax

mov si, offset VERSION

add si, 16

call BYTE\_TO\_DEC

mov dx, offset VERSION

call WRITE\_MSG

mov si, offset MODIFICATION

add si, 23

pop ax

mov al, ah

call BYTE\_TO\_DEC

mov dx, offset MODIFICATION

call WRITE\_MSG

mov si, offset OEM

add si, 9

mov al, bh

call BYTE\_TO\_DEC

mov dx, offset OEM

call WRITE\_MSG

mov di, offset SERIAL

add di, 22

mov ax, cx

call WRD\_TO\_HEX

mov al, bl

call BYTE\_TO\_HEX

sub di, 2

mov [di], ax

mov dx, offset SERIAL

call WRITE\_MSG

ret

OS\_VERSION ENDP

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

PC\_TYPE PROC near

mov ax, 0f000h

mov es, ax

mov al, es:[0fffeh]

mov dx, offset PCSTRING

call WRITE\_MSG

cmp al, 0ffh

jz pc\_

cmp al, 0feh

jz pcxt\_

cmp al, 0fbh

jz pcxt\_

cmp al, 0fch

jz pcat\_

cmp al, 0fah

jz pcps2m30\_

cmp al, 0f8h

jz pcps2m80\_

cmp al, 0fdh

jz pcjr\_

cmp al, 0f9h

jz pcconv\_

pc\_:

mov dx, offset PC

jmp write

pcxt\_:

mov dx, offset PCXT

jmp write

pcat\_:

mov dx, offset PCAT

jmp write

pcps2m30\_:

mov dx, offset PS2M\_30

jmp write

pcps2m5060\_:

mov dx, offset PS2M\_50\_60

jmp write

pcps2m80\_:

mov dx, offset PS2M\_80

jmp write

pcjr\_:

mov dx, offset PCJR

jmp write

pcconv\_:

mov dx, offset PCCONV

jmp write

write:

call WRITE\_MSG

ret

PC\_TYPE 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\_l

or AL,30h

mov [SI],AL

end\_l: pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

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

; CODE

BEGIN:

call OS\_VERSION

call PC\_TYPE

mov ah, 10h

int 16h

xor AL,AL

mov AH,4Ch

int 21H

TESTPC ENDS

END START

# Приложение Б. lab1exe.asm

;BYKOV I V. 6383. .asm code to good .exe file

.model small

; DATA

.data

VERSION db "MS-DOS version: $"

MODIFICATION db 13, 10, "Modification number: $"

SERIAL db 13, 10, "Serial number: $"

OEM db 13, 10, "OEM: $"

PCSTRING db 13, 10, "PC type: $"

PC db "PC$"

PCXT db "PC/XT$"

PCAT db "AT$"

PS2M\_30 db "PS2 30 model$"

PS2M\_50\_60 db "PS2 50 or 60 model$"

PS2M\_80 db "PS2 80 model$"

PCJR db "PC JR$"

PCCONV db "PC Convert$"

.stack 100h

.code

;PROCEDURES

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

WRITE\_MSG PROC near

push ax

mov ah, 09h

int 21h

pop ax

ret

WRITE\_MSG ENDP

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

OS\_VERSION PROC near

mov ah, 30h

int 21h

; al - version number

; ah - modification number

; bh - OEM serial number

; bl:cx - user serial number

push ax

mov si, offset VERSION

add si, 16

call BYTE\_TO\_DEC

mov dx, offset VERSION

call WRITE\_MSG

mov si, offset MODIFICATION

add si, 23

pop ax

mov al, ah

call BYTE\_TO\_DEC

mov dx, offset MODIFICATION

call WRITE\_MSG

mov si, offset OEM

add si, 9

mov al, bh

call BYTE\_TO\_DEC

mov dx, offset OEM

call WRITE\_MSG

mov di, offset SERIAL

add di, 22

mov ax, cx

call WRD\_TO\_HEX

mov al, bl

call BYTE\_TO\_HEX

sub di, 2

mov [di], ax

mov dx, offset SERIAL

call WRITE\_MSG

ret

OS\_VERSION ENDP

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

PC\_TYPE PROC near

mov ax, 0f000h

mov es, ax

mov al, es:[0fffeh]

mov dx, offset PCSTRING

call WRITE\_MSG

cmp al, 0ffh

jz pc\_

cmp al, 0feh

jz pcxt\_

cmp al, 0fbh

jz pcxt\_

cmp al, 0fch

jz pcat\_

cmp al, 0fah

jz pcps2m30\_

cmp al, 0f8h

jz pcps2m80\_

cmp al, 0fdh

jz pcjr\_

cmp al, 0f9h

jz pcconv\_

pc\_:

mov dx, offset PC

jmp write

pcxt\_:

mov dx, offset PCXT

jmp write

pcat\_:

mov dx, offset PCAT

jmp write

pcps2m30\_:

mov dx, offset PS2M\_30

jmp write

pcps2m5060\_:

mov dx, offset PS2M\_50\_60

jmp write

pcps2m80\_:

mov dx, offset PS2M\_80

jmp write

pcjr\_:

mov dx, offset PCJR

jmp write

pcconv\_:

mov dx, offset PCCONV

jmp write

write:

call WRITE\_MSG

ret

PC\_TYPE 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\_l

or AL,30h

mov [SI],AL

end\_l: pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

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

; CODE

BEGIN:

mov ax, @data

mov ds, ax

call OS\_VERSION

call PC\_TYPE

mov ah, 10h

int 16h

xor AL,AL

mov AH,4Ch

int 21H

END BEGIN