**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра MOЭBM**

отчет

**по лабораторной работе №1**

**по дисциплине «Операционные системы»**

**Тема: Исследование структур загрузочных модулей**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6381 |  | Вергейчик Г. Л. |
| Преподаватель |  | Губкин А. Ф. |

Санкт-Петербург

2018

**Цель работы**

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

**Дополнительные сведения**

Тип IBM PC хранится в байте по адресу 0F000:0FFFE, в предпоследнем байте ROM BIOS. Соответствие кода и типа в таблице:

PC FF

PC/XT FE,FB

AT FC

PS2 модель 30 FA

PS2 модель 50 или 60 FC

PS2 модель 80 F8

PCjr FD

PC Convertible F9

Для определения версии MS DOS следует воспользоваться функцией 30H прерывания 21H. Входным параметром является номер функции в AH:

MOV AH,30h

INT 21h

Выходными параметрами являются:

AL – номер основной версии. Если 0, то <2.0;

AH – номер модификации;

BH – серийный номер OEM (Original Equipment Manufacturer);

BL:CX – 24-битовый серийный номер пользователя;

**Ход работы**

1. Был написан файл исходного .COM модуля, определяющего тип PC и версию системы.
2. После отладки исходного модуля был получен «хороший» .COM модуль. А также, используя его код, был получен «плохой» .EXE модуль.
3. Был написан файл исходного .EXE модуля, определяющего тип PC и версию системы.
4. После отладки модуля был получен «хороший» .EXE модуль.

**Функции программы**

* Функция TETR\_TO\_HEX переводит половину байта в символ.
* Функция BYTE\_TO\_HEX переводит байт в AL в два символа шестн. числа в AX.
* Функция WRD\_TO\_HEX переводит в 16 с/с 16-ти разрядное число в AX - число, DI - адрес последнего символа.
* Функция BYTE\_TO\_DEC переводит в 10 с/с, SI - адрес поля младшей цифры.
* Функция PC\_INFO достает информацию о типе PC
* Функция SYSTEM\_INFO достает версию и модификацию системы
* Функция OEM\_INFO достает информацию о OEM
* Функция USER\_INFO достает информацию о пользователе

**Данные программы**

TYPE\_PC - Тип IBM PC   
 SYS\_VER - Версия системы   
 OEM\_NUM - Номер OEM   
 USER\_NUM - Серийный номер пользователя

**Результаты выполнения и отладки программ:**

1. **Хороший COM файл:**

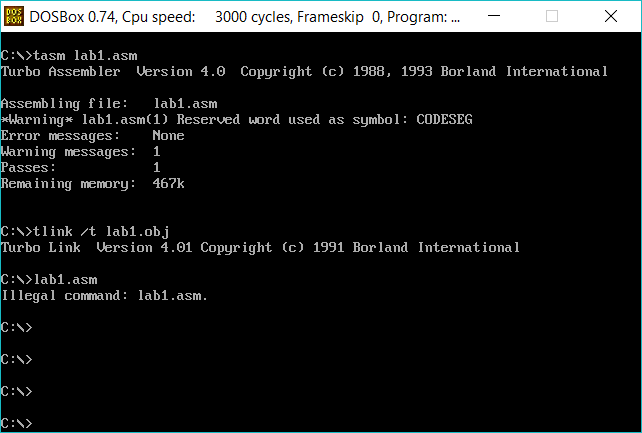


Рис1. Компиляция хорошего COM-файла

Результат работы программы(см Рис2.)

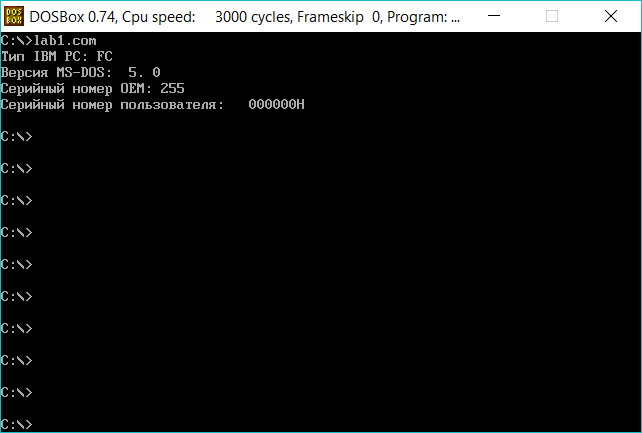


Рис2. Результат работы хорошего COM-файла

Файл в шестнадцатеричном виде(см Рис3.)

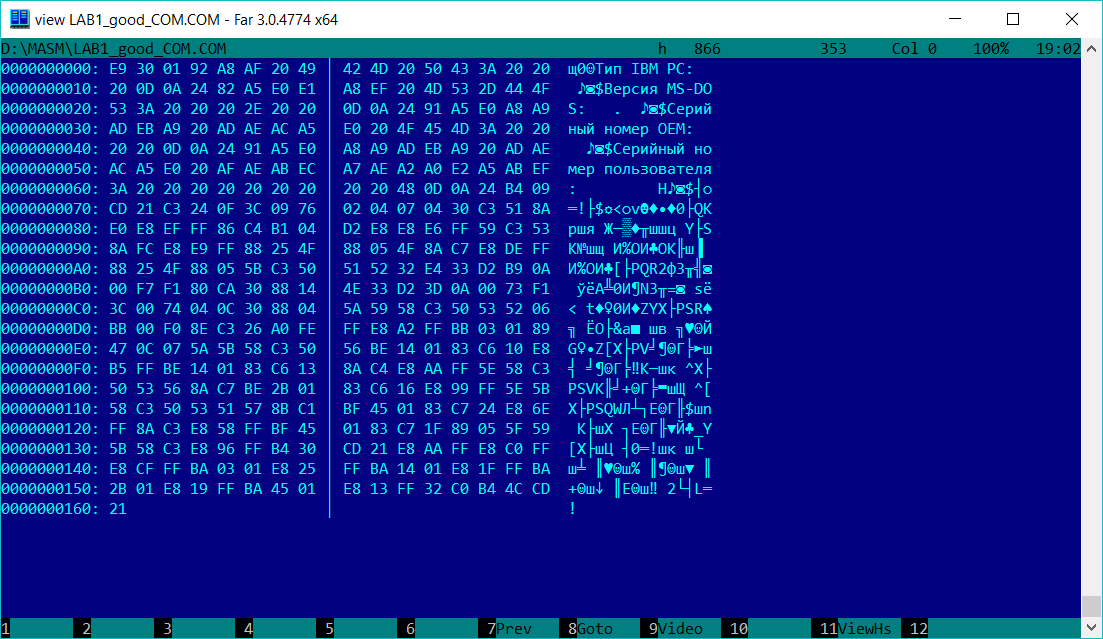


Рис3. Хороший COM-файл в шестнадцатеричном виде

1. **Плохой EXEфайл:**

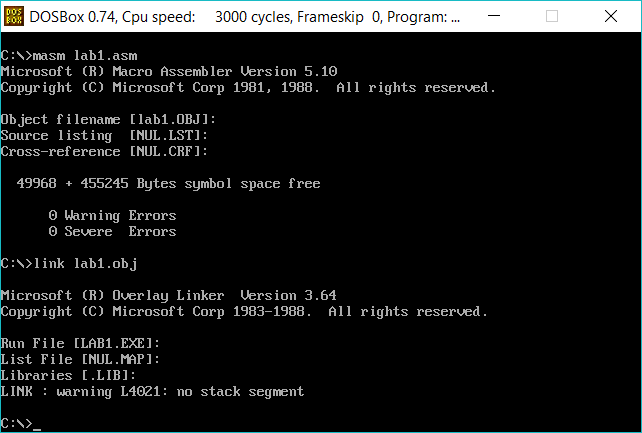


Рис4. Компиляция плохого EXE-файла

Результат работы программы(см Рис5.)

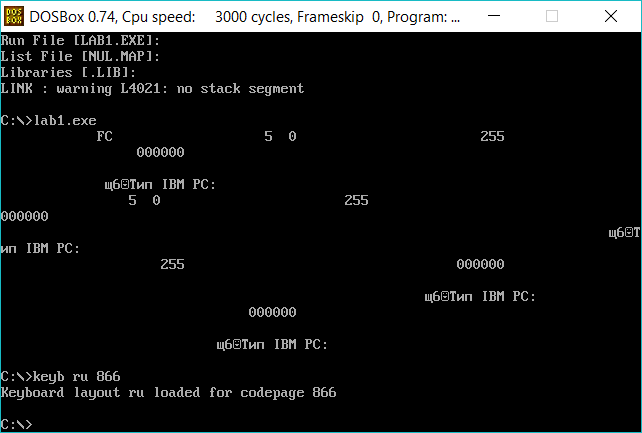


Рис5. Результат работы плохого EXE-файла

Файл в шестнадцатеричном виде(см Рис6-7.)

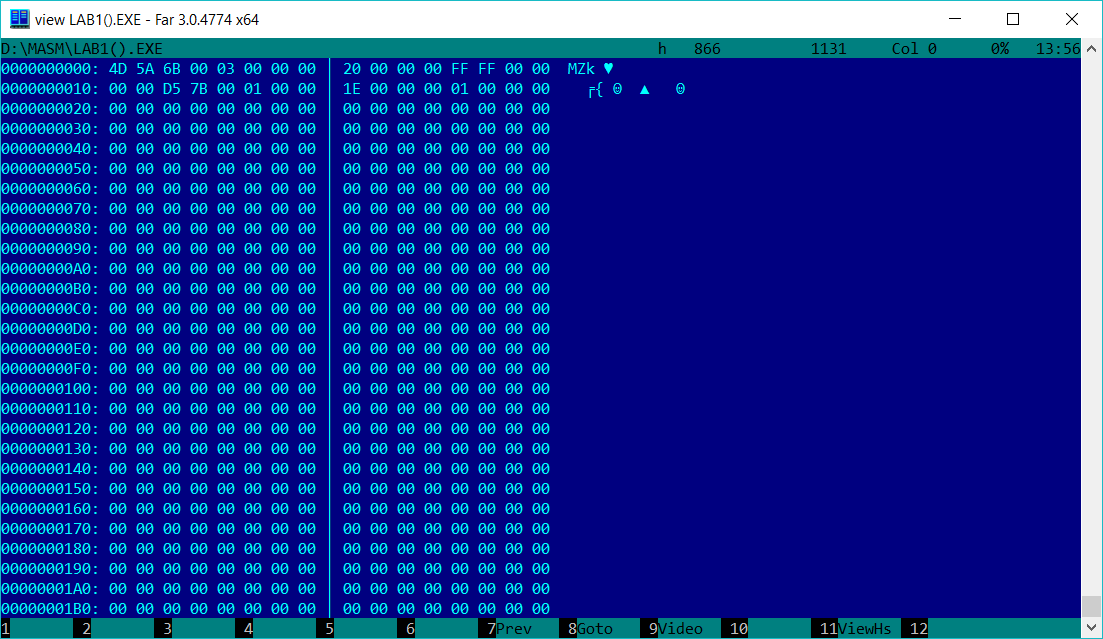


Рис6. Плохой EXE-файл в шестнадцатеричном виде

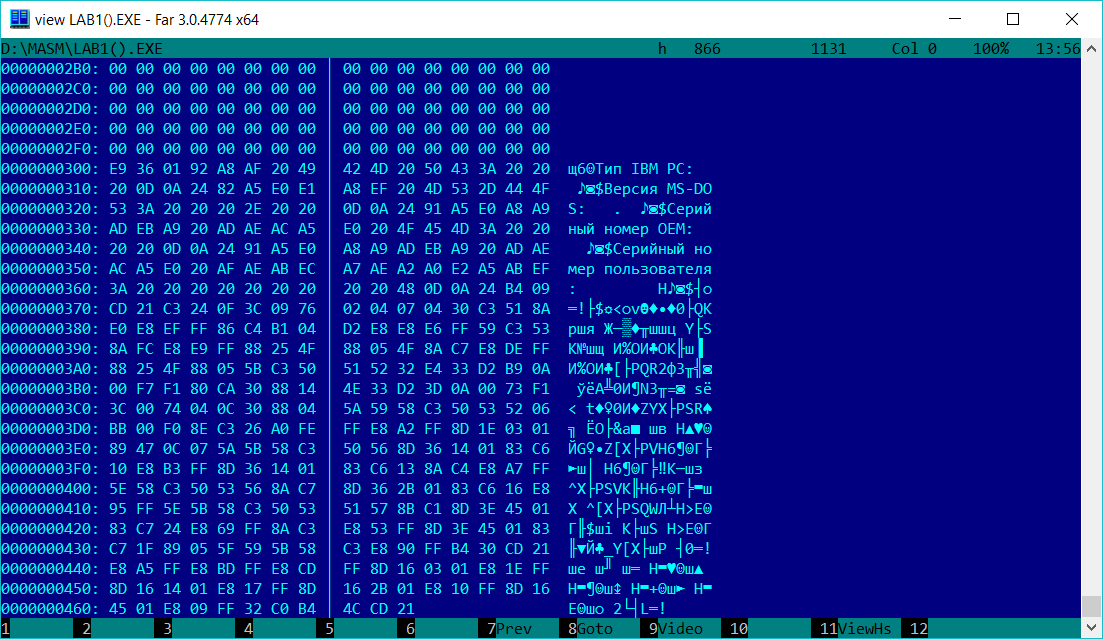


Рис7. Плохой EXE-файл в шестнадцатеричном виде

1. **Хороший EXEфайл:**

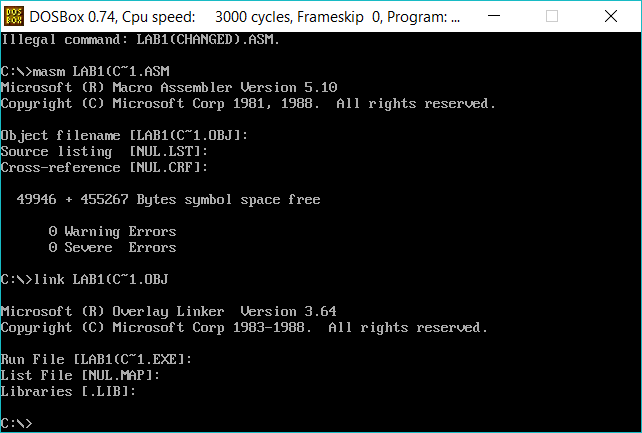


Рис8. Компиляция хорошего EXE-файла

Результат работы программы(см Рис9.)

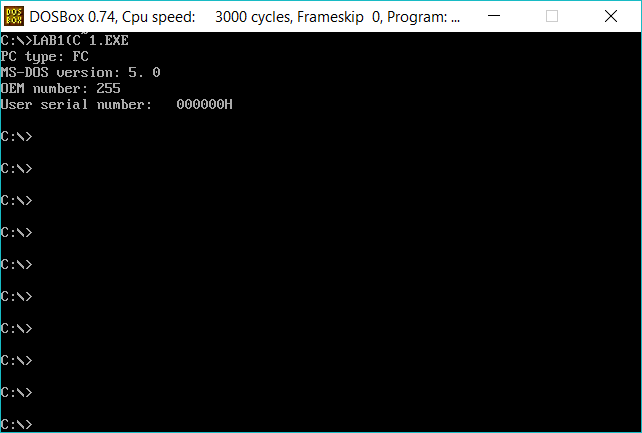


Рис9. Результат работы хорошего EXE-файла

Файл в шестнадцатеричном виде(см Рис10-11.)

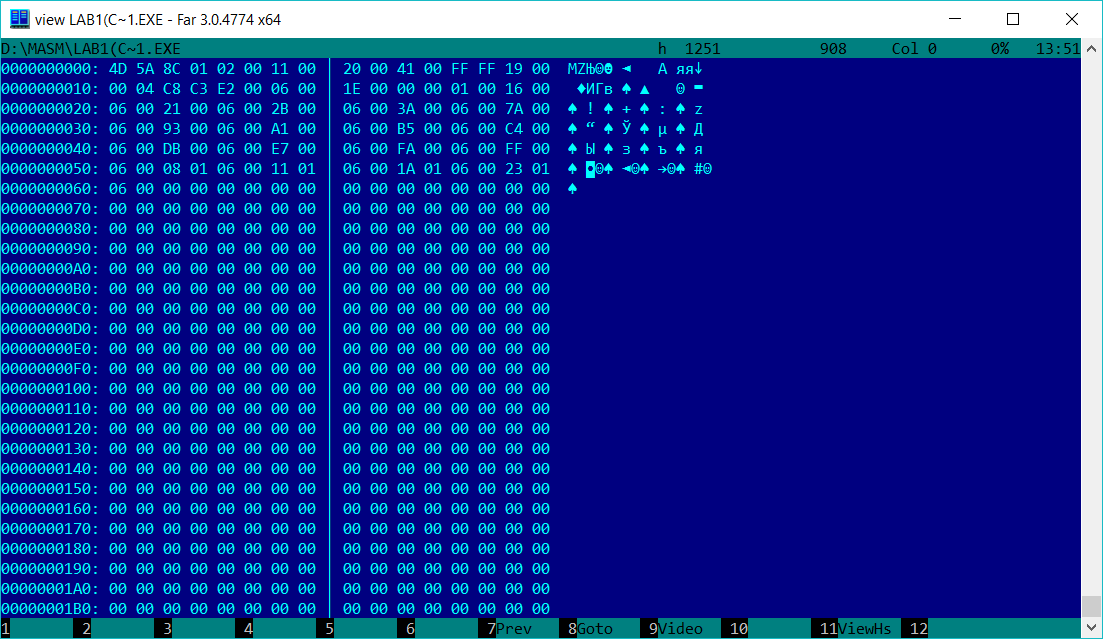


Рис10. Хороший EXE-файл в шестнадцатеричном виде

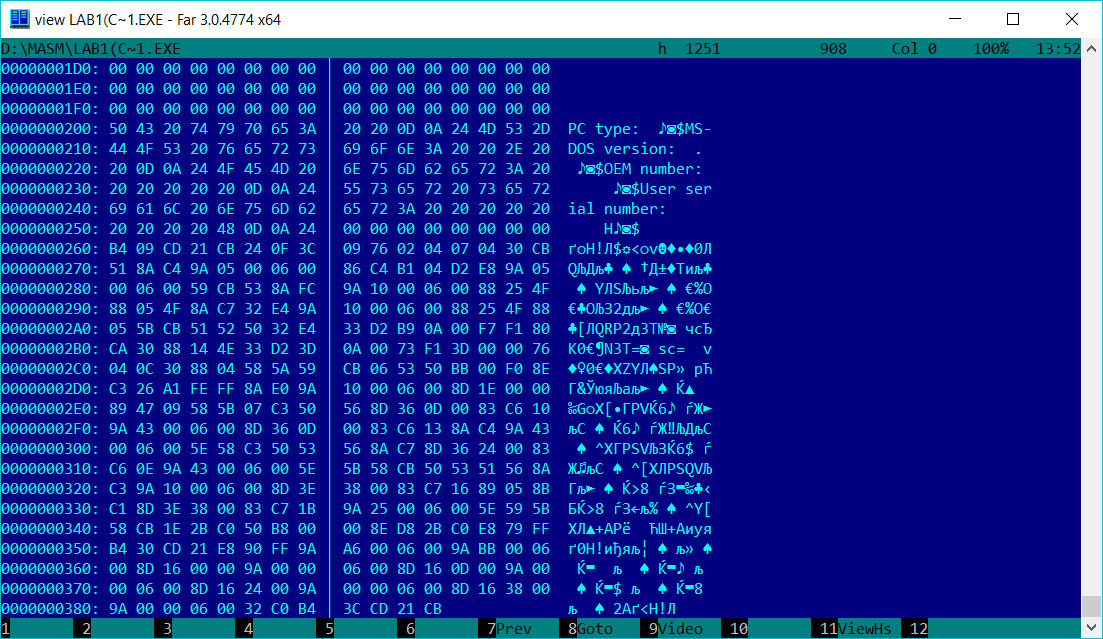


Рис11. Хороший EXE-файл в шестнадцатеричном виде

**Ответы на вопросы**

**Отличия исходных текстов COM- и EXE- программ**

1. Сколько сегментов должна содержать COM-программа?

Один сегмент кода.

1. EXE-программа?

Программы в формате EXE могут иметь более одного сегмента. Количество допустимых сегментов определяется используемой моделью памяти:

*small* - один сегмент кода, один сегмент данных

*compact*- один сегмент кода, несколько сегментов данных

*medium* - несколько сегментов кода, один сегмент данных

*large* - несколько сегментов кода, несколько сегментов данных

*huge* - много сегментов кода, много сегментов данных

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

Директива ASSUME, которая указывает ассемблеру, с каким сегментом связан тот или иной сегментный регистр, без данной директивы программа не скомпилируется, т,к не будет знать, где начинается сегмент кода. Директива ORG 100h. Требуется сразу после предложения SEGMENT в сегменте кода. Эта директива устанавливает адрес смещения на ячейку, следующую сразу за PSP, и выполнение программы начинается с этой ячейки. Если в начале программы были расположены данные, перед ними следует поместить команду перехода на первую команду программы.

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

Не все. Команды, использующие адреса сегментов запрещены, т.к. отсутствует таблица настроек (Relocation Table).

При загрузке модуля в память адреса сегментов определяются с помощью данной таблицы (происходит привязка exe-программы к сегментным адресам памяти).

В com-программе такая таблица отсутствует. Поэтому команды, использующие адреса сегментов, запрещены.

**Отличие форматов файлов COM и EXE модулей**

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

Программа содержит единственный сегмент, содержащий и данные, и команды. Код располагается с 0h адреса.

1. Какова структура «плохого» ЕХЕ? С какого адреса располагается код? Что располагается с адреса 0?

«Плохой» EXE-файл содержит таблицу настроек (Relocation table) и исполняемый код, располагающийся с адреса 300h, так как присутствует директива org 100h, выделяющая место под PSP. С адреса 0h располагается таблица настроек.

1. Какова структура файла «хорошего» EXE? Чем он отличается от файла «плохого» EXE?

У «хорошего» EXE-файла структура такая же, как и у «плохого» - сначала идет таблица настроек, после – код. Но код начинается с адреса 200h, так как отсутствует директива org 100h, выделяющая место под PSP.

**Загрузка COM модуля в основную память**

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

После загрузки COM-программы все 4 сегментных регистра указывают на начало единственного сегмента, т.е. фактически на начало PSP. Поскольку PSP имеет размер 100h байтов, адресация сегмента кода начинается со смещения 100h байт.

1. Что располагается с адреса 0?

Адрес начала PSP(префикс программного сегмента).

1. Какие значения имеют сегментные регистры? На какие области памяти они указывают?

Когда загрузчик программ загружает для выполнения программу .COM, он автоматически инициализирует CS, DS, SS, ES адресом PSP. Теперь CS и DS содержат правильные адреса начала сегмента в период выполнения, и программа типа .COM не должна их инициализировать.

1. Как определяется стек? Какую область памяти он занимает? Какие адреса?

Стек определяется автоматически. Указатель стека устанавливается на последнее слово сегмента (SP=FFFEh), таким образом, независимо от фактического размера программы, ей выделяется 64 Кбайт адресного пространства, всю нижнюю часть которого занимает стек.

**Загрузка «хорошего» ЕХЕ модуля в основную память**

1. Как загружается «хороший» ЕХЕ? Какие значения имеют сегментные регистры?   
   Система, загрузив программу в память, инициализирует сегментные регистры, так что регистры DS и ES указывают на начало PSP, CS – на начало сегмента команд, а SS – на начало сегмента стека. В указатель команд IP загружается смещение точки входа в программу (которое берется из операнда директивы END), а в указатель стека SP – смещение конца сегмента стека. Таким образом, после загрузки программы в память адресуемыми оказываются все сегменты, кроме сегмента (или сегментов) данных.
2. На что указывают регистры DS и ES?

На начало сегмента PSP.

1. Как определяется стек?  
   Стек определяется при объявлении сегмента стека при помощи директивы «DW 512 DUP(?)».
2. Как определяется точка входа?  
   В указатель команд IP загружается смещение точки входа в программу, которое берется из операнда директивы END.В главном модуле операндом этой директивы является имя точки, на которую передается управление при начале выполнения программы. Во всех других модулях эта директива употребляется без операндов.

**Вывод**

В ходе выполнения данной лабораторной работы были исследованы различия в структурах исходных текстов загрузочный модулей типов .COM и .EXE, структур файлов этих модулей и способах их загрузки в основную память.

**Приложение**

1. Файл lab1\_good\_COM.asm

CODESEG SEGMENT

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

ORG 100H

START: JMP BEGIN

; ДАННЫЕ

TYPE\_PC db 'Тип IBM PC: ',0DH,0AH,'$'

MSDOS\_VER db 'Версия MS-DOS: . ',0DH,0AH,'$'

OEM\_NUM db 'Серийный номер OEM: ',0DH,0AH,'$'

USER\_NUM db 'Серийный номер пользователя: H',0DH,0AH,'$'

;ПРОЦЕДУРЫ

; Процедура печати строки

WriteMsg PROC NEAR

mov AH,09h

int 21h ; Вызов функции DOS по прерыванию

ret

WriteMsg 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

; байт в AL переводится в два символа шестн. 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 байта в 10с/с, SI - адрес поля младшей цифры

; AL содержит исходный байт

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

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

; Процедура определения типа PC

PC\_INFO PROC NEAR

push AX

push BX

push DX

push ES

mov BX,0F000H

mov ES,BX

mov AL,ES:[0FFFEH]

call BYTE\_TO\_HEX

lea BX,TYPE\_PC

mov [BX+12],AX

pop ES

pop DX

pop BX

pop AX

ret

PC\_INFO ENDP

; Процедура определения версии MS-DOS

SYSTEM\_INFO PROC NEAR

push AX

push SI

lea SI,MSDOS\_VER

add SI,16

call BYTE\_TO\_DEC

lea SI,MSDOS\_VER

add SI,19

mov AL,AH

call BYTE\_TO\_DEC

pop SI

pop AX

ret

SYSTEM\_INFO ENDP

; Процедура определения номера OEM

OEM\_INFO PROC NEAR

push AX

push BX

push SI

mov AL,BH

lea SI,OEM\_NUM

add SI,22

call BYTE\_TO\_DEC

pop SI

pop BX

pop AX

ret

OEM\_INFO ENDP

; Процедура определения номера пользователя

USER\_INFO PROC NEAR

push AX

push BX

push CX

push DI

mov AX,CX

lea DI,USER\_NUM

add DI,36

call WRD\_TO\_HEX

mov AL,BL

call BYTE\_TO\_HEX

lea DI,USER\_NUM

add DI,31

mov [DI],AX

pop DI

pop CX

pop BX

pop AX

ret

USER\_INFO ENDP

; КОД

BEGIN:

call PC\_INFO

mov AH,30H

INT 21H

call SYSTEM\_INFO

call OEM\_INFO

call USER\_INFO

; Вывод версии PC

lea DX,TYPE\_PC

call WriteMsg

; Вывод версии MS-DOS

lea DX,MSDOS\_VER

call WriteMsg

; Вывод номена OEM

lea DX,OEM\_NUM

call WriteMsg

; Вывод номера пользователя

lea DX,USER\_NUM

call WriteMsg

; Выход в DOS

xor AL,AL

mov AH,4Ch

int 21H

CODESEG ENDS

END START ;конец модуля, START - точка входа

2. Файл lab1(goodExe).asm

EOL EQU '$'

DATA SEGMENT

TYPE\_PC db 'PC type: ',0dh,0ah,'$'

SYS\_VER db 'MS-DOS version: . ',0dh,0ah,'$'

OEM\_NUM db 'OEM number: ',0dh,0ah,'$'

USER\_NUM db 'User serial number: H',0dh,0ah,'$'

DATA ENDS

CODE SEGMENT

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

Write\_msg PROC FAR

mov ah,09h

int 21h

ret

Write\_msg ENDP

AStack SEGMENT STACK

DW 512 DUP(?)

AStack ENDS

TETR\_TO\_HEX PROC FAR

and al,0fh

cmp al,09

jbe NEXT

add al,07

NEXT: add al,30h

ret

TETR\_TO\_HEX ENDP

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

BYTE\_TO\_HEX PROC FAR

; байт в AL переводится в два символа шестн. числа в AX

push cx

mov al,ah

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 FAR

; первод 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

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 FAR

; перевод в 10 с/с, SI - адрес поля младшей цифры

; AL содержит исходный байт

push cx

push dx

push ax

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 ax,00h

jbe end\_l

or al,30h

mov [si],al

end\_l: pop ax

pop dx

pop cx

ret

BYTE\_TO\_DEC ENDP

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

; Процедура определения типа PC

PC\_INFO PROC NEAR

push es

push bx

push ax

mov bx,0f000h

mov es,bx

mov ax,es:[0fffeh]

mov ah,al

call BYTE\_TO\_HEX

lea bx,TYPE\_PC

mov [bx+9],ax

pop ax

pop bx

pop es

ret

PC\_INFO ENDP

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

; Процедура определения версии MS-DOS

SYSTEM\_INFO PROC NEAR

push AX

push SI

lea SI,SYS\_VER

add SI,16

call BYTE\_TO\_DEC

lea SI,SYS\_VER

add SI,19

mov AL,AH

call BYTE\_TO\_DEC

pop SI

pop AX

ret

SYSTEM\_INFO ENDP

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

OEM\_INFO PROC FAR

; функция определяющая OEM

push ax

push bx

push si

mov al,bh

lea si,OEM\_NUM

add si,14

call BYTE\_TO\_DEC

pop si

pop bx

pop ax

ret

OEM\_INFO ENDP

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

USER\_INFO PROC FAR

push ax

push bx

push cx

push si

mov al,bl

call BYTE\_TO\_HEX

lea di,USER\_NUM

add di,22

mov [di],AX

mov ax,cx

lea di,USER\_NUM

add di,27

call WRD\_TO\_HEX

pop si

pop cx

pop bx

pop ax

ret

USER\_INFO ENDP

Main PROC FAR

push DS

sub AX,AX

push AX

mov AX,DATA

mov DS,AX

sub AX,AX

call PC\_INFO

mov ah,30h

int 21h

call SYSTEM\_INFO

call OEM\_INFO

call USER\_INFO

; выводим информацию

lea dx,TYPE\_PC

call Write\_msg

lea dx,SYS\_VER

call Write\_msg

lea dx,OEM\_NUM

call Write\_msg

lea dx,USER\_NUM

call Write\_msg

; выход в DOS

xor al,al

mov ah,3Ch

int 21h

ret

Main ENDP

CODE ENDS

END Main