**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 моделей**

|  |  |  |
| --- | --- | --- |
| Студент гр.0382 |  | Литягин С.М. |
| Преподаватель |  | Ефремов М.А. |

Санкт-Петербург

2022

## Цель работы.

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

## Задание.

1. Написать текст исходного .COM модуля, который определяет тип PC и версию системы. Результатом будет “хороший” .COM модуль, а также “плохой” .EXE, полученный из исходного текста для .COM модуля.

2. Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1. Результатом будет “хороший” .EXE.

3. Сравнить исходные тексты для .COM и .EXE модулей. Ответить на контрольные вопросы “Отличия исходных текстов COM и EXE программ”.

4. Запустить FAR и открыть файл 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”.

5. Открыть отладчик TD.EXE и загрузить .COM. Ответить на контрольные вопросы “Загрузка COM модуля в основную память”. Представить в отчете план загрузки .COM модуля в основную память.

6. Открыть отладчик TD.EXE и загрузить “хороший” .EXE. Ответить на контрольные вопросы “Загрузка “хорошего” EXE модуля в основную память”.

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

**Выполнение работы:**

1. За основу был взят шаблон .COM модуля из методического пособия, в котором реализованы процедуры преобразования двоичных кодов в символы шестнадцатеричных и десятичных чисел. Для определения типа PC и версии системы были написаны процедуры: IBM\_TYPE, DOS\_VER, OEMN, USERN. Тип IBM PC был получен согласно байту по адресу 0F000:0FFFFh. Для определения версии системы же использовалась функция 30H прерывания 21H. Ее выходными параметрами являются: AL – номер основной версии, AH – номер модификации, BH – серийный номер OEM, BL:CX – 24-битовый серийный номер пользователя.

В результате шага имеем “хороший” .COM модуль и “плохой” .EXE модуль. Выводы, полученные при их запуске, представлены на рисунке 1 и рисунке 2 соответственно.



Рисунок 1 – результат запуска модуля lb1\_com.com

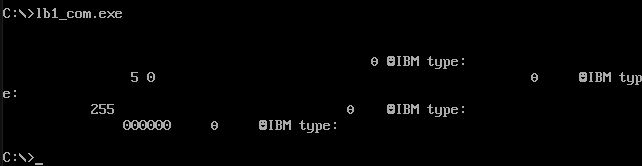


Рисунок 2 – результат запуска модуля lb1\_com.exe

2. Для написания “хорошего” .EXE модуля разобьем программу на сегменты кода, данных и стека, также добавим главную процедуру. В .COM модуле имелась директива org 100h, что нужна, поскольку при загрузке COM модуля в память DOS первые 256 байт блоком данных занимает PSP, код программы располагается лишь после этого блока. В .EXE модуле же мы в этом не нуждаемся, поскольку блок PSP расположен вне сегмента кода. На рисунке 3 представлены результаты запуска данного модуля.

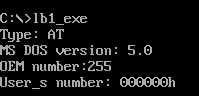


Рисунок 3 – результат запуска модуля lb1\_exe.exe

3. Сравнивая исходные тексты для .COM и .EXE модулей, ответим на контрольные вопросы “Отличия исходных текстов COM и EXE программ”:

- Сколько сегментов должна содержать COM-программа?

Один сегмент (содержит и код, и данные; стек же генерируется сам).

- EXE-программа?

Обязательно один – сегмент кода. Также можно добавить сегмент данных и стека, они описываются отдельно друг от друга.

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

Обязательно нужна директива ORG 100h, поскольку первые 256 байт занимает блок данных PSP. Поэтому нужно, чтобы адресация имела смещение в 256 байт от нулевого адреса. Также директива ASSUME указывает, с каким сегментом нужно связать регистры сегмента кода и сегмента данных.

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

Поскольку сегментные регистры в .COM-программе определяются в момент запуска программы, а не при линковке (не имеет relocation table), то мы не можем использовать команды с указанием сегментов.

4. Рассматривая шестнадцатеричные виды модулей, ответим на контрольные вопросы “Отличия форматов файлов COM и EXE модулей”:

- Какова структура файла COM? С какого адреса располагается код?

.COM модуль состоит из одного сегмента, содержащего и код, и данные. Код располагается с нулевого адреса. Как уже упоминалось ранее, PSP занимает 100h байт, поэтому устанавливается смещение 100h. Модуль в шестнадцатеричном виде представлен на рисунке 4.



Рисунок 4 – “хороший” .COM в 16-м виде

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

Также как и .COM модуль, состоит из одного сегмента. Код располагается с адреса 300h (200h занимает заголовок и relocation table, 100h - смещение). Модуль в шестнадцатеричном виде представлен на рисунке 5.



Рисунок 5 – “плохой” .EXE в 16-м виде

- Какова структура файла “хорошего” EXE? Чем он отличается от файла “плохого” EXE?

Имеет несколько сегментов, в начале модуля располагается заголовок и relocation table (200h байт), затем сегменты в порядке их определения в коде, т.е. сначала сегмент стека, потом сегмент данных, а затем – сегмент кода. Модуль в шестнадцатеричном виде представлен на рисунке 6.

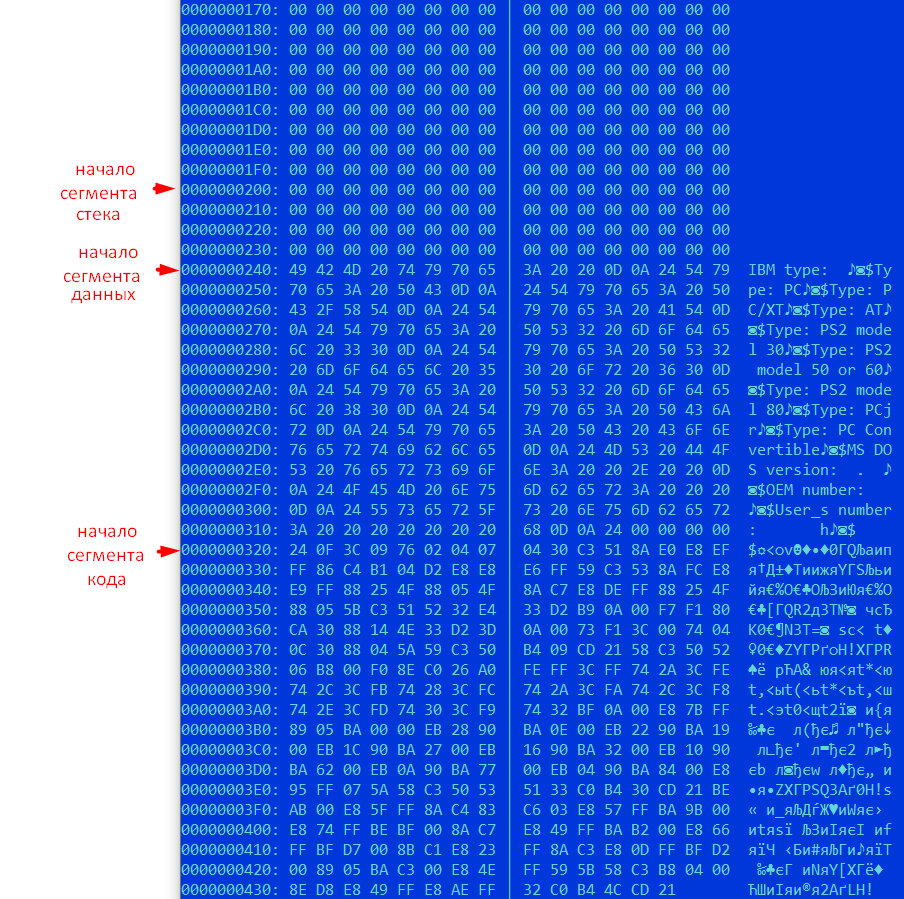


Рисунок 6 – “хороший” .EXE в 16-м виде

5. Загружая .COM модуль в отладчике TD.EXE, ответим на контрольные вопросы “Загрузка COM модуля в основную память”. Результат загрузки программы в отладчик представлены на рисунке 7.

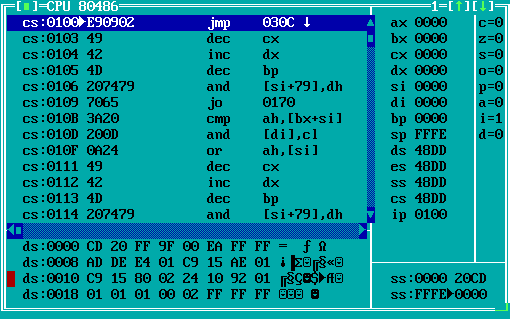


Рисунок 7 – загруженный в отладчик “хороший” .COM модуль

- Какой формат загрузки модуля COM? С какого адреса располагается код?

Сначала определяется сегментный адрес участка ОП, у которого достаточно места для загрузки программы. Затем создается блок памяти для PSP и программы. Начиная с PSP:0100h помещается считанный образ COM-файла. Код же располагается с адреса CS:0100h.

- Что располагается с адреса 0?

Сегмент PSP.

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

Сегментные регистры CS, DS, ES, SS указывают на PSP (в данном случае имеет значение 48DDh)

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

Для .COM файлов стек определяется от FFFEh до 0000h, т.е. весь сегмент. SS указывает на начало сегмента (т.е. на 48DDh); SP – на последний адрес, что кратен двум (т.е. на FFFEh). Соответственно, адреса 48DD:0000h – 48DD:FFFEh.

6. Загружая “хороший” .EXE модуль в отладчике TD.EXE, ответим на контрольные вопросы “Загрузка “хорошего” EXE модуля в основную память”. Результат загрузки программы в отладчик представлены на рисунке 8.

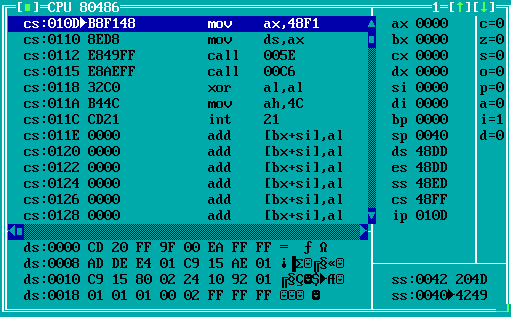


Рисунок 8 – загруженный в отладчик “хороший” .COM модуль

- Как загружается “хороший” EXE? Какие значения имеют сегментные регистры?

Загружается в память PSP. Затем, по информации в заголовке, загружается .EXE модуль. Значения регистров: DS = ES = 48DD, CS = 48FF, SS = 48ED.

- На что указывают регистры DS и ES?

Эти регистры указывают на начало PSP.

- Как определяется стек?

В коде программы описывается сегмент стека. При помощи директивы ASSUME устанавливает сегментный регистр на описанный сегмент стека. Соответственно, SS будет указывать на начало сегмента стека, а SP – на его конец.

- Как определяется точка входа?

Точка входа определяется директивой END \*метка\*. Данная метка определяет адрес, с которого начинается выполнение программы, т.е. – точка входа в программу.

Исходный код программы см. в приложении А.

## Выводы.

В ходе работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, в структурах файлов загрузочных моделей и способов их загрузки в основную память; была написана программа, выводящую информацию о типе PC и версии системы.

# Приложение А Исходный код программы

Название файла: lb1\_com.asm

TESTPC SEGMENT

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

ORG 100H

START: jmp BEGIN

; ДАННЫЕ

NOTYPE db 'IBM type: ', 0DH, 0AH, '$'

PC db 'IBM type: PC', 0DH, 0AH, '$'

XT db 'IBM type: PC/XT', 0DH, 0AH, '$'

AT db 'IBM type: AT', 0DH, 0AH, '$'

PS230 db 'IBM type: PS2 model 30', 0DH, 0AH, '$'

PS25060 db 'IBM type: PS2 model 50 or 60', 0DH, 0AH, '$'

PS280 db 'IBM type: PS2 model 80', 0DH, 0AH, '$'

PCjr db 'IBM type: PCjr', 0DH, 0AH, '$'

PCConvertible db 'IBM type: PC Convertible', 0DH, 0AH, '$'

DOSV db 'MS DOS version: . ', 0DH, 0AH, '$'

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

USR db 'User\_s number: 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 16-го 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, в 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 10 с/с, в 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\_1

or AL, 30h

mov [SI], AL

end\_1: pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

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

PRINT PROC near

push AX

mov AH, 09h

int 21h

pop AX

ret

PRINT ENDP

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

IBM\_TYPE PROC near

push BX

push DX

push ES

mov BX, 0F000h

mov ES, BX

mov AL, ES:[0FFFEh]

cmp AL, 0FFh

je \_pc

cmp AL, 0FEh

je \_xt

cmp AL, 0FBh

je \_xt

cmp AL, 0FCh

je \_at

cmp AL, 0FAh

je \_ps230

cmp AL, 0F8h

je \_ps280

cmp AL, 0FDh

je \_pcjr

cmp AL, 0F9h

je \_pcc

mov DI, offset NOTYPE + 10

call BYTE\_TO\_HEX

mov [DI], AX

mov DX, offset NOTYPE

jmp p\_ibm

\_pc:

mov DX, offset PC

jmp p\_ibm

\_xt:

mov DX, offset XT

jmp p\_ibm

\_at:

mov DX, offset AT

jmp p\_ibm

\_ps230:

mov DX, offset PS230

jmp p\_ibm

\_ps280:

mov DX, offset PS280

jmp p\_ibm

\_pcjr:

mov DX, offset PCjr

jmp p\_ibm

\_pcc:

mov DX, offset PCConvertible

p\_ibm:

call PRINT

pop ES

pop DX

pop AX

ret

IBM\_TYPE ENDP

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

DOS\_VER PROC near

push AX

push BX

push CX

xor AX, AX

mov AH, 30h

int 21h

mov SI, offset DOSV + 16

call BYTE\_TO\_DEC

mov AL, AH

add SI, 3

call BYTE\_TO\_DEC

mov DX, offset DOSV

call PRINT

mov SI, offset OEM + 13

mov AL, BH

call BYTE\_TO\_DEC

mov DX, offset OEM

call PRINT

mov DI, offset USR + 20

mov AX, CX

call WRD\_TO\_HEX

mov AL, BL

call BYTE\_TO\_HEX

mov di, offset USR + 15

mov [DI], AX

mov DX, offset USR

call PRINT

pop CX

pop BX

pop AX

ret

DOS\_VER ENDP

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

; КОД

BEGIN:

mov AX, 0F000h

mov ES, AX

mov AL, ES:[0FFFFh]

call IBM\_TYPE

call DOS\_VER

xor AL, AL

mov AH, 4Ch

int 21h

TESTPC ENDS

END START

Название файла: lb1\_exe.asm

AStack SEGMENT STACK

DW 32 DUP(?)

AStack ENDS

DATA SEGMENT

NOTYPE db 'IBM type: ', 0DH, 0AH, '$'

PC db 'Type: PC', 0DH, 0AH, '$'

XT db 'Type: PC/XT', 0DH, 0AH, '$'

AT db 'Type: AT', 0DH, 0AH, '$'

PS230 db 'Type: PS2 model 30', 0DH, 0AH, '$'

PS25060 db 'Type: PS2 model 50 or 60', 0DH, 0AH, '$'

PS280 db 'Type: PS2 model 80', 0DH, 0AH, '$'

PCjr db 'Type: PCjr', 0DH, 0AH, '$'

PCConvertible db 'Type: PC Convertible', 0DH, 0AH, '$'

DOSV db 'MS DOS version: . ', 0DH, 0AH, '$'

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

USR db 'User\_s number: h', 0DH, 0AH, '$'

DATA ENDS

TESTPC SEGMENT

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

; 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 16-го 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, в 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 10 с/с, в 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\_1

or AL, 30h

mov [SI], AL

end\_1: pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

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

PRINT PROC near

push AX

mov AH, 09h

int 21h

pop AX

ret

PRINT ENDP

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

IBM\_TYPE PROC near

push AX

push DX

push ES

mov AX, 0F000h

mov ES, AX

mov AL, ES:[0FFFEh]

cmp AL, 0FFh

je \_pc

cmp AL, 0FEh

je \_xt

cmp AL, 0FBh

je \_xt

cmp AL, 0FCh

je \_at

cmp AL, 0FAh

je \_ps230

cmp AL, 0F8h

je \_ps280

cmp AL, 0FDh

je \_pcjr

cmp AL, 0F9h

je \_pcc

mov DI, offset NOTYPE + 10

call BYTE\_TO\_HEX

mov [DI], AX

mov DX, offset NOTYPE

jmp p\_ibm

\_pc:

mov DX, offset PC

jmp p\_ibm

\_xt:

mov DX, offset XT

jmp p\_ibm

\_at:

mov DX, offset AT

jmp p\_ibm

\_ps230:

mov DX, offset PS230

jmp p\_ibm

\_ps280:

mov DX, offset PS280

jmp p\_ibm

\_pcjr:

mov DX, offset PCjr

jmp p\_ibm

\_pcc:

mov DX, offset PCConvertible

p\_ibm:

call PRINT

pop ES

pop DX

pop AX

ret

IBM\_TYPE ENDP

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

DOS\_VER PROC near

push AX

push BX

push CX

xor AX, AX

mov AH, 30h

int 21h

mov SI, offset DOSV + 16

call BYTE\_TO\_DEC

mov AL, AH

add SI, 3

call BYTE\_TO\_DEC

mov DX, offset DOSV

call PRINT

mov SI, offset OEM + 13

mov AL, BH

call BYTE\_TO\_DEC

mov DX, offset OEM

call PRINT

mov DI, offset USR + 20

mov AX, CX

call WRD\_TO\_HEX

mov AL, BL

call BYTE\_TO\_HEX

mov di, offset USR + 15

mov [DI], AX

mov DX, offset USR

call PRINT

pop CX

pop BX

pop AX

ret

DOS\_VER ENDP

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

; КОД

MAIN PROC far

mov ax, data

mov ds, ax

call IBM\_TYPE

call DOS\_VER

; Выход в DOS

xor AL, AL

mov AH, 4Ch

int 21h

MAIN ENDP

TESTPC ENDS

END MAIN