**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Операционные системы»**

Тема: Исследование структур загрузочных модулей

|  |  |  |
| --- | --- | --- |
| Студент гр. 9383 |  | Мосин К.К. |
| Преподаватель |  | Ефремов М.А. |

Санкт-Петербург

2021

**Цель работы.**

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

**Основные теоретические положения.**

Тип IBM PC хранится в байте по адресу 0F000:0FFFEh, в предпоследнем байте ROM BIOS. Соответствие кода и типа приведены в табл. 1.

Табл. 1 - Соответствие кода и типа PC

|  |  |
| --- | --- |
| 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

**Функции и структуры данных.**

Функции, используемые в программе, приведены в табл. 2.

Табл. 2 - Функции и структуры данных

|  |  |
| --- | --- |
| Название | Описание |
| tetr\_to\_hex | Перевод из 4-ной с/с в 16-ную с/с |
| byte\_to\_hex | Перевод из 2-ной с/с в 16-ную с/с |
| wrd\_to\_hex | Перевод слова (2 байта) в 16-ную с/с |
| byte\_to\_dec | Перевод из 2-ной с/с в 10-ную с/с |
| print | Вывод сообщения на экран |

**Задание.**

Шаг 1. Напишите текст исходного .COM модуля, определяющий тип PC и версию системы.

Шаг 2. Напишите текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1 и постройте и отладьте его.

Шаг 3. Сравните исходные тексты для .COM и .EXE модулей.

Шаг 4. Запустите FAR и откройте (F3/F4) файл загрузочного модуля .COM и файл “плохого” .EXE в 16-ном виде. Затем откройте (F3/F4) файл загрузочного модуля “хорошего” .EXE и сравните его с предыдущими файлами.

Шаг 5. Откройте отладчик TD.EXE и запустите .COM.

Шаг 6. Откройте отладчик TD.EXE и загрузите “хороший” .EXE.

Шаг 7. Оформление отчета.

**Выполнение работы.**

Пример .COM модуля, а также “хорошего” и “плохого” .EXE модулей проиллюстрированы на изображениях 1, 2 и 3.

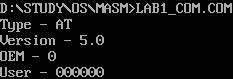
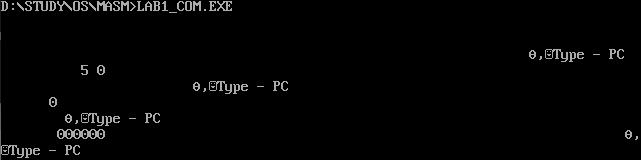


Рис 1 - пример .COM модуля

Рис 2 - пример “плохого” .EXE модуля

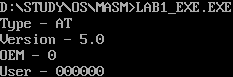


Рис 3 - пример “хорошего” .EXE модуля

**Ответы на вопросы.**

1. Отличие исходных текстов COM и EXE программ.

1) Сколько сегментов должна содержать COM-программа?

Для COM-программы достаточно одного сегмента.

2) EXE-программа?

Для EXE-программы количество сегментов начинается с единицы, но сегменты кода, данных и стека всегда описываются отдельно друг от друга.

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

Так как первые 256 байт занимает префикс программного сегмента, необходима директива *org 100h*, обеспечивающая смещение.

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

Команды вида mov <регистр> или seg <сегмент> не подлежат исполнению.

1. Отличие форматов файлов COM и EXE модулей.

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

В COM файле находится один сегмент, включающий в себя сегмент кода и данных. При этом стек генерируется автоматически. На проиллюстрированном ниже изображении (рис. 4) наблюдается, что код начинается в адреса 0.

Рис 4 – 16-ное представление .COM файла

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

Сегмент кода и данных не разделены на разные сегменты, что приводит к некорректной работе EXE модуля. Код располагается с адреса 300, когда с 0 располагаются настройки.

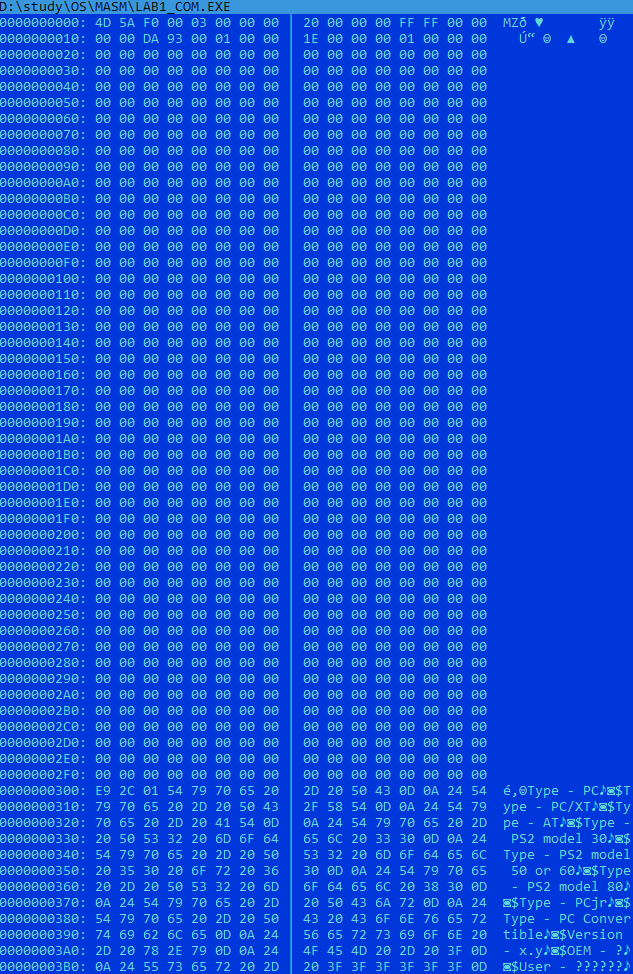


Рис 5 – 16-ное представление “плохого” .EXE файла

3) Какова структура “хорошего" EXE? Чем он отличается от файла "плохого" EXE?

Теперь же присутствует ручное определение стека, а сегмент данных и сегмент кода определены отдельно друг от друга. В “хорошем” EXE модуле отсутствует смещение, определенное в COM модуле директивой *org 100h*.

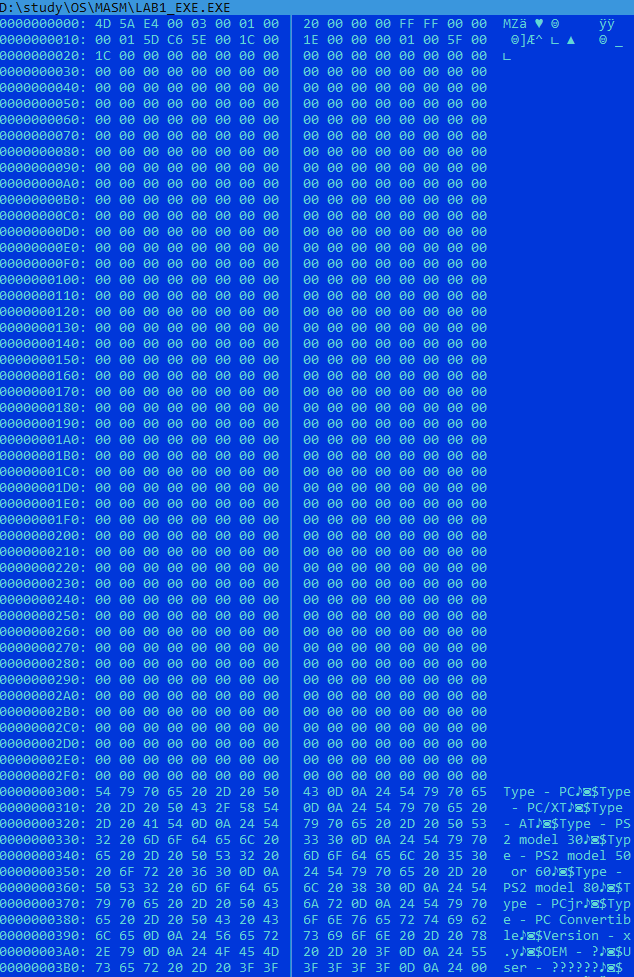
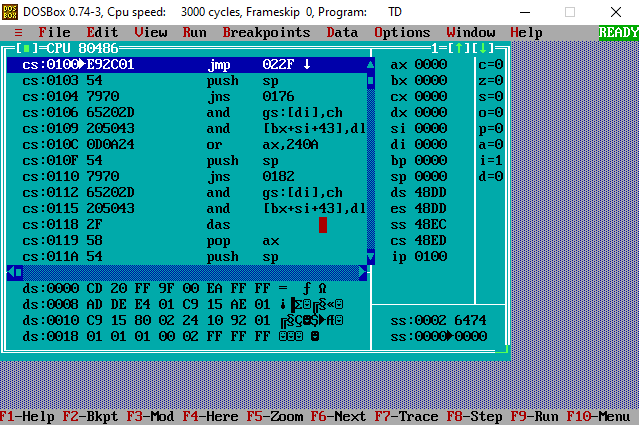


Рис 6 – 16-ное представление “хорошего” .EXE файла

1. Загрузка COM модуля в основную память.

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

Загрузка в память происходит считыванием COM файла с диска. Код располагается с адреса IP = 100.



2) Что располагается с адреса 0?

Префикс программного сегмента.

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

ds = es = 48DD, ss = 48EC, cs = 48ED. Данные регистры указывают на префикс программного сегмента.

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

Стек определяется автоматически. SS указывает на начало, SP на конец стека. Диапазон равен 0 – FFFEh.

1. Загрузка “хорошего” EXE модуля в основную память

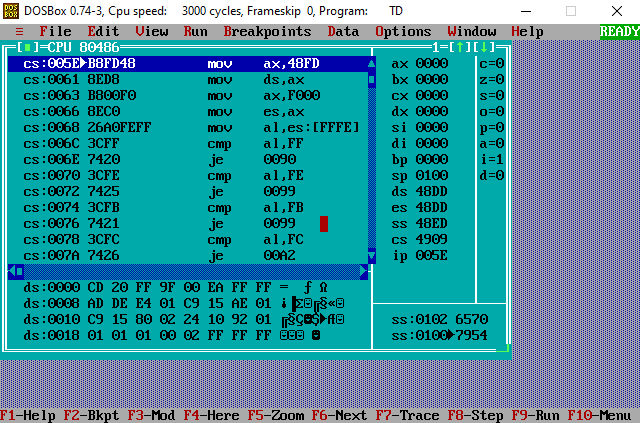
1) Как загружается "хороший" .EXE? Какие значения имеют сегментные регистры?

При загрузки происходит перемещение адресов сегментов:

ds=es=48DD,

ss=48ED,

cs=4909



2) На что указывают регистры DS и ES?

На префикс программного сегмента.

3) Как определяется стек?

Стек определяется директивой stack.

4) Как определяется точка входа?

Директивой END.

**Выводы.**

В ходе выполнения лабораторной работы были исследованы различия .COM и .EXE модулей.

Приложение а

Исходный код программы

Название файла: lab1\_com.asm

testpc segment

assume cs:testpc,ds:testpc,es:nothing,ss:nothing

org 100h

start:

jmp begin

pc db 'Type - PC',0dh,0ah,'$'

pcxt db 'Type - PC/XT',0dh,0ah,'$'

pcat db 'Type - AT',0dh,0ah,'$'

ps2\_30 db 'Type - PS2 model 30',0dh,0ah,'$'

ps2\_50\_60 db 'Type - PS2 model 50 or 60',0dh,0ah,'$'

ps2\_80 db 'Type - PS2 model 80',0dh,0ah,'$'

pcjr db 'Type - PCjr',0dh,0ah,'$'

pc\_convertible db 'Type - PC Convertible',0dh,0ah,'$'

version db 'Version - x.y',0dh,0ah,'$'

oem db 'OEM - ?',0dh,0ah,'$'

user db 'User - ??????',0dh,0ah,'$'

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],ah

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

print proc near

push ax

mov ah,09h

int 21h

pop ax

ret

print endp

begin:

mov ax,0f000h

mov es,ax

mov al,es:[0fffeh]

cmp al,0ffh

je type\_pc

cmp al,0feh

je type\_pcxt

cmp al,0fbh

je type\_pcxt

cmp al,0fch

je type\_at

cmp al,0fah

je type\_ps2\_30

cmp al,0fch

je type\_ps2\_50\_or\_60

cmp al,0f8h

je type\_ps2\_80

cmp al,0fdh

je type\_pcjr

cmp al,0f9h

je type\_pc\_convertible

type\_pc:

mov dx,offset pc

call print

jmp exit

type\_pcxt:

mov dx,offset pcxt

call print

jmp exit

type\_at:

mov dx,offset pcat

call print

jmp exit

type\_ps2\_30:

mov dx,offset ps2\_30

call print

jmp exit

type\_ps2\_50\_or\_60:

mov dx,offset ps2\_50\_60

call print

jmp exit

type\_ps2\_80:

mov dx,offset ps2\_80

call print

jmp exit

type\_pcjr:

mov dx,offset pcjr

call print

jmp exit

type\_pc\_convertible:

mov dx,offset pc\_convertible

call print

exit:

mov ah,30h

int 21h

mov si,offset version

add si,10

push ax

call byte\_to\_dec

pop ax

mov al,ah

add si,3

call byte\_to\_dec

mov dx,offset version

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,12

call wrd\_to\_hex

mov ax,cx

mov al,bl

call byte\_to\_hex

sub di,2

mov [di],ax

mov dx,offset user

call print

xor al,al

mov ah,4ch

int 21h

testpc ends

end start

Название файла: lab1\_exe.asm

stacksg segment stack

dw 128 dup(?)

stacksg ends

datasg segment

pc db 'Type - PC',0dh,0ah,'$'

pcxt db 'Type - PC/XT',0dh,0ah,'$'

pcat db 'Type - AT',0dh,0ah,'$'

ps2\_30 db 'Type - PS2 model 30',0dh,0ah,'$'

ps2\_50\_60 db 'Type - PS2 model 50 or 60',0dh,0ah,'$'

ps2\_80 db 'Type - PS2 model 80',0dh,0ah,'$'

pcjr db 'Type - PCjr',0dh,0ah,'$'

pc\_convertible db 'Type - PC Convertible',0dh,0ah,'$'

version db 'Version - x.y',0dh,0ah,'$'

oem db 'OEM - ?',0dh,0ah,'$'

user db 'User - ??????',0dh,0ah,'$'

datasg ends

testpc segment

assume cs:testpc,ds:datasg,ss:stacksg

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],ah

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

print proc near

push ax

mov ah,09h

int 21h

pop ax

ret

print endp

main proc far

mov ax,datasg

mov ds,ax

mov ax,0f000h

mov es,ax

mov al,es:[0fffeh]

cmp al,0ffh

je type\_pc

cmp al,0feh

je type\_pcxt

cmp al,0fbh

je type\_pcxt

cmp al,0fch

je type\_at

cmp al,0fah

je type\_ps2\_30

cmp al,0fch

je type\_ps2\_50\_or\_60

cmp al,0f8h

je type\_ps2\_80

cmp al,0fdh

je type\_pcjr

cmp al,0f9h

je type\_pc\_convertible

type\_pc:

mov dx,offset pc

call print

jmp exit

type\_pcxt:

mov dx,offset pcxt

call print

jmp exit

type\_at:

mov dx,offset pcat

call print

jmp exit

type\_ps2\_30:

mov dx,offset ps2\_30

call print

jmp exit

type\_ps2\_50\_or\_60:

mov dx,offset ps2\_50\_60

call print

jmp exit

type\_ps2\_80:

mov dx,offset ps2\_80

call print

jmp exit

type\_pcjr:

mov dx,offset pcjr

call print

jmp exit

type\_pc\_convertible:

mov dx,offset pc\_convertible

call print

exit:

mov ah,30h

int 21h

mov si,offset version

add si,10

push ax

call byte\_to\_dec

pop ax

mov al,ah

add si,3

call byte\_to\_dec

mov dx,offset version

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,12

call wrd\_to\_hex

mov ax,cx

mov al,bl

call byte\_to\_hex

sub di,2

mov [di],ax

mov dx,offset user

call print

xor al,al

mov ah,4ch

int 21h

main endp

testpc ends

end main