**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. 6381 |  | Кухарев М.А. |
| Преподаватель |  | Губкин А.Ф. |

Санкт-Петербург

2018

**Цель работы:**

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

**Постановка задачи:**

Требуется написать текст исходного .COM модуля, который определяет тип PC и версию системы. Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип РС и выводить строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код переводится в символьную строку, содержащую запись шестнадцатеричного числа и выводится на экран в виде соответствующего сообщения. Затем определяется версия системы. Ассемблерная программа должна по значениям регистров AL и AH формировать текстовую строку в формате xx.yy, где xx - номер основной версии, а yy - номер модификации в десятичной системе счисления, формировать строки с серийным номером OEM (Original Equipment Manufacturer) и серийным номером пользователя. Полученные строки выводятся на экран.

Далее необходимо отладить полученный исходный модуль и получить «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля.

Затем нужно написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль .COM, далее его построить, отладить и сравнить исходные тексты для .COM и .EXE модулей.

Ответить на контрольные вопросы.

**Общие 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;

**Описание данных**

|  |  |
| --- | --- |
| **Название** | **Назначение** |
| \_type | строка, содержащая «IBM PC: » |
| \_ver | строка, содержащая «System version: » |
| \_user | строка, содержащая «Serial number of user: » |
| \_oem | строка, содержащая «OEM serial number: » |
| \_PC | строка, содержащая «PC » |
| \_PC\_XT | строка, содержащая «PC/XT » |
| \_AT | строка, содержащая «AT » |
| \_PC2\_30 | строка, содержащая «PS2 model 30 » |
| \_PC2\_80 | строка, содержащая «PS2 model 80 » |
| \_PCjr | строка, содержащая «PCjr » |
| \_PC\_Conv | строка, содержащая «PC Convertible » |

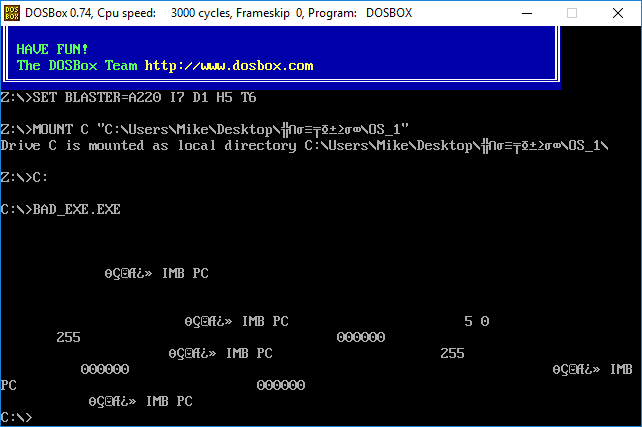
**Описание реализованных функций и структур данных:**

|  |  |
| --- | --- |
| **Название процедуры** | **Назначение** |
| BYTE\_TO\_HEX | Перевод байта регистра AL в шестнадцатеричную систему счисления, помещая результат в AX |
| BYTE\_TO\_DEC | Перевод байта регистра AL в десятичную систему счисления, помещая результат в SI |
| WRD\_TO\_HEX | Перевод двух байт регистра AX в шестнадцатеричную систему счисления, помещая результат в регистр DI |
| TETR\_TO\_HEX | Перевод половины байта в шестнадцатеричную систему счисления |
| PRINT | Выводит на экран строку |
| TYPE\_IBM\_PC | Определение тип IBM PC |
| VERS\_DOS | Определение версии MS DOS |
| USER\_DOS | Определение серийного номера пользователя |
| OEM\_DOS | Определение серийного номера OEM |

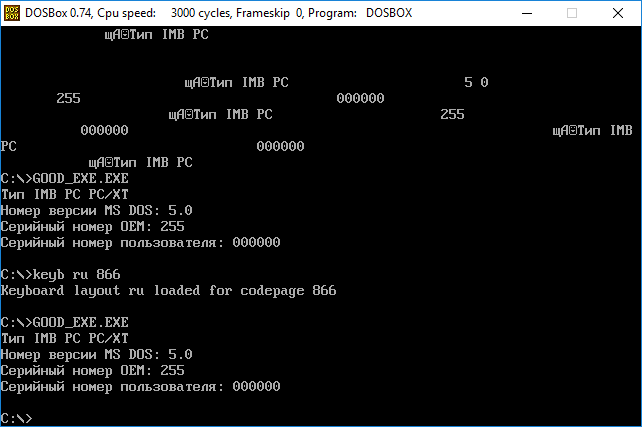
**Ход работы**

В результате выполнения лабораторной работы была написана программы, результаты работы которых показаны на рисунках ниже.

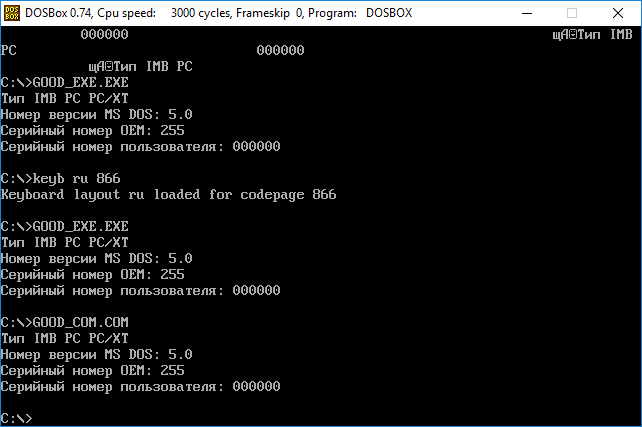
1. Запуск плохого .EXE файла(BAD\_EXE.exe): Рисунок 1



1. Хороший .EXE модуль(GOOD\_EXE.exe): Рисунок 2



1. Хороший .COM модуль(GOOD\_COM.com): Рисунок 3



**Контрольные вопросы:**

**1. Отличия исходных текстов COM и EXE программ.**

a) Сколько сегментов должна содержать COM программа?

.COM - программы содержат только один сегмент.

б) EXE программа?

. COM программа может содержать только один сегмент, а .EXE программа может содержать больше одного сегмента. В программах этого типа предусматривают отдельные сегменты для кода, данных и стека.

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

Во-первых, директива ORG 100h, потому что при загрузке СОМ-файла в память DOS занимает первые 256 байт (100h) сегментом данных PSP, а после него располагает код программы.

г) Все ли форматы команд можно использовать в COM программе?

Нельзя использовать команды, связанные с адресом сегмента, потому что адрес сегмента до загрузки неизвестен. В итоге загрузчик не сможет его определить. Также нельзя использовать оператор FAR - переход на метку возможен только в результате межсегментной передачи управления, а так как в .com-файле только один сегмент, то никаких межсегментных переходов и быть не может.

Доп: Это связано с тем, что для этих команд(например, mov ax, seg) используются данные о сегментах, находящиеся в таблице *relocation table*. Такая таблица содержится в EXE-модулях, COM-модули ее не содержат. Таблица релокаций содержит список перемещений, т.е. тех мест в образе файла в памяти, в которых необходимо учесть различие между указанным в заголовке базовым адресом загрузки и реальным адресом загрузки.

Доп.Доп:  
mov ax, seg  
Таблица состоит из элементов, число которых записано в байтах 06-07. Элемент таблицы настройки состоит из двух полей: 2-байтного смещения и 2-байтного сегмента, и указывает слова в загрузочном модуле, содержащее адрес, который должен быть настроен на место памяти, в которое загружается задача.

**2. Отличия форматов файлов COM и EXE модулей.**

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

.COM-файл состоит из команд, процедур и данных, используемых в программе. Код начинается с нулевого адреса.

б) Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с 0 адреса?

В файле EXE содержится информация для загрузчика, данные и код. С 0 адреса располагается управляющая информация для загрузчика. Код располагается с адреса 300h.

Код располагается с адреса 300h, т.к. заголовок занимает 200h байт(байты 8 и 9 указывают, сколько параграфов занимает заголовок) и команда ORG 100h «сдвигает» код на дополнительные 100h. С нулевого адреса располагается заголовок. В первых двух байтах можно увидеть символы MZ, означающие, что формат файла – 16-битный и его следует запускать в соответствии со структурой EXE-файлов. За заголовком следует relocation table.

в) Какова структура файла «хорошего» EXE? Чем он отличается от «плохого» EXE файла?

В «хорошем» файле EXE содержится информация для загрузчика, сегмент стека, сегмент данных и сегмент кода (3 сегмента вместо одного в «плохом» .EXE). Код располагается с адреса 200h в отличие от 300h в «плохом» .EXE файле. В EXE-файле присутствует специальный начальный блок (заголовок) размером не менее 200h (512 байт), кроме того в «плохом» .EXE есть смещение в 100h (256 байт), полученное после преобразования COM-файла в EXE-файл, откуда в сумме получается 300h.

При прошлом ответе ошибся: имел в виду, что в хорошем больше одного сегмента.  
Нельзя использовать команды формата mov <регистр>, SEG <имя сегмента>

**3. Загрузка COM модуля в основную память.**

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

После загрузки COM-программы в память сегментные регистры указывают на начало PSP. Код располагается с адреса 100h.

б) Что располагается с 0 адреса?

С нулевого адреса располагается заголовок PSP.

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

Все сегментные регистры имеют значения 119С. Они указывают на PSP.



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

Cтек создается автоматически, указатель стека в конце сегмента. Из этого следует, что он занимает оставшуюся память и адреса изменяются от больших к меньшим, то есть от FFFEh к 0000h.

**4. Загрузка «хорошего» EXE модуля в память.**

а) Как загружается «хороший» EXE? Какие значения имеют сегментные регистры?

Сначала создается PSP. Затем определяется длина тела загрузочного модуля, определяется начальный сегмент. Загрузочный модуль считывается в начальный сегмент, таблица настройки считывается в рабочую память, к полю каждого сегмента прибавляется сегментный адрес начального сегмента, определяются значения сегментных регистров. DS и ES указывают на начало PSP (119С), CS – на начало сегмента команд (11F2h), а SS – на начало сегмента стека (11AC).



б) На что указывают регистры DS и ES?

Изначально регистры DS и ES указывают на начало сегмента PSP.

в) Как определяется стек?

Регистры SS и SP принимают значения, указанные в заголовке, потом к SS прибавляется сегментный адрес начального сегмента.

г) Как определяется точка входа?

Смещение точки входа в программу загружается в указатель команд IP . IP, а именно адрес, с которого начинается выполнение программы, определяется операндом директивы END, который называется точкой входа.

**Приложение. Код программ**

BAD\_EXE.asm и GOOD\_COM.asm:

TESTPC SEGMENT

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

ORG 100H

START: JMP BEGIN

; ДАННЫЕ:

; дополнительные данные

EOF EQU '$' ; определение символьной константы

; данные IBM PC

\_type db 'Тип IMB PC ',EOF

\_PC db 'PC',0DH,0AH,EOF

\_PC\_XT db 'PC/XT',0DH,0AH,EOF

\_AT db 'AT',0DH,0AH,EOF

\_PS2\_30 db 'PS2 модель 30',0DH,0AH,EOF

\_PS2\_50\_60 db 'PS2 модель 50 или 60',0DH,0AH,EOF

\_PS2\_80 db 'PS2 модель 80',0DH,0AH,EOF

\_PCjr db 'PCjr',0DH,0AH,EOF

\_PC\_Conv db 'PC Convertible',0DH,0AH,EOF

; данные MS DOS

\_ver db 'Номер версии MS DOS: . ',0DH,0AH,EOF

\_oem db 'Серийный номер OEM: ',0DH,0AH,EOF

\_user db 'Серийный номер пользователя: ',0DH,0AH,EOF

; 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 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, в AH - младшая

pop CX

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

; функция определения типа IMB PC

TYPE\_IMB\_PC PROC NEAR

push ds

mov ax, 0F000h

mov ds, ax

sub bx, bx

mov bh, [0FFFEh]

pop ds

ret

TYPE\_IMB\_PC ENDp

; функция определения версии MS DOS (al)

VERS\_DOS PROC NEAR

push ax

push si

mov si, offset \_ver

add si, 15h

call BYTE\_TO\_DEC

mov si, offset \_ver

add si, 17h

mov al, ah

call BYTE\_TO\_DEC

pop si

pop ax

ret

VERS\_DOS ENDP

; функция опеделения серийного номера OEM (bh)

OEM\_DOS PROC NEAR

push ax

push bx

push si

mov si, offset \_oem

add si, 16h

mov al, bh

call BYTE\_TO\_DEC

pop si

pop bx

pop ax

ret

OEM\_DOS ENDP

; функция определения серийного номера пользователя (bl:cx)

USER\_DOS PROC NEAR

push bx

push cx

push di

push ax

mov di, offset \_user

add di, 22h

mov ax, cx

call WRD\_TO\_HEX

mov al, bl

call BYTE\_TO\_HEX

mov di, offset \_user

add di, 1Dh

mov [di], ax

pop ax

pop di

pop cx

pop bx

ret

USER\_DOS ENDP

; функция вывода на экран

PRINT PROC NEAR

push ax

mov ah, 09h

int 21h

pop ax

ret

PRINT ENDP

; КОД

BEGIN:

; вызываем функцию определения типа IBM PC

call TYPE\_IMB\_PC

; вывод поясняющей строки

mov dx, offset \_type

call PRINT

; определяем по предпоследнему биту ROM BIOS тип IBM PC

; предварительно загружаем смещения определённого типа

; и в случае успеха - выводим тип на экран

mov dx, offset \_PC

cmp bh, 0FFh

je to\_print

mov dx, offset \_PC\_XT

cmp bh, 0FEh

je to\_print

mov dx, offset \_PC\_XT

cmp bh, 0FBh

je to\_print

mov dx, offset \_AT

cmp bh, 0FCh

je to\_print

mov dx, offset \_PS2\_30

cmp bh, 0FAh

je to\_print

mov dx, offset \_PS2\_50\_60

cmp bh, 0FCh

je to\_print

mov dx, offset \_PS2\_80

cmp bh, 0F8h

je to\_print

mov dx, offset \_PCjr

cmp bh, 0FDh

je to\_print

mov dx, offset \_PC\_Conv

cmp bh, 0F9h

je to\_print

mov al, bh

call BYTE\_TO\_HEX

mov dx, ax

; вызываем функцию вывода на экран

to\_print:

call PRINT

; вызываем функцию определения версии MS DOS

mov ah, 30h

int 21h

; сохраняем полученные значения

call VERS\_DOS

call OEM\_DOS

call USER\_DOS

; выводим полученные значения

mov dx, offset \_ver

call PRINT

mov dx, offset \_oem

call PRINT

mov dx, offset \_user

call PRINT

; выход в DOS

xor al, al

mov ah, 4ch

int 21h

TESTPC ENDS

END START ; конец модуля

GOOD\_EXE.asm:

.MODEL SMALL

.STACK 200h

.DATA

; ДАННЫЕ:

; дополнительные данные

EOF EQU '$' ; определение символьной константы

; данные IBM PC

\_type db 'Тип IMB PC ',EOF

\_PC db 'PC',0DH,0AH,EOF

\_PC\_XT db 'PC/XT',0DH,0AH,EOF

\_AT db 'AT',0DH,0AH,EOF

\_PC2\_30 db 'PC2 модель 30',0DH,0AH,EOF

\_PC2\_50\_60 db 'PC2 модель 50 или 60',0DH,0AH,EOF

\_PC2\_80 db 'PC2 модель 80',0DH,0AH,EOF

\_PCjr db 'PCjr',0DH,0AH,EOF

\_PC\_Conv db 'PC Convertible',0DH,0AH,EOF

; данные MS DOS

\_ver db 'Номер версии MS DOS: . ',0DH,0AH,EOF

\_oem db 'Серийный номер OEM: ',0DH,0AH,EOF

\_user db 'Серийный номер пользователя: ',0DH,0AH,EOF

.CODE

START:

JMP BEGIN

; 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 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, в AH - младшая

pop CX

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

; функция определения типа IMB PC

TYPE\_IMB\_PC PROC NEAR

push ds

mov ax, 0F000h

mov ds, ax

sub bx, bx

mov bh, [0FFFEh]

pop ds

ret

TYPE\_IMB\_PC ENDp

; функция определения версии MS DOS (al)

VERS\_DOS PROC NEAR

push ax

push si

mov si, offset \_ver

add si, 15h

call BYTE\_TO\_DEC

mov si, offset \_ver

add si, 17h

mov al, ah

call BYTE\_TO\_DEC

pop si

pop ax

ret

VERS\_DOS ENDP

; функция опеделения серийного номера OEM (bh)

OEM\_DOS PROC NEAR

push ax

push bx

push si

mov si, offset \_oem

add si, 16h

mov al, bh

call BYTE\_TO\_DEC

pop si

pop bx

pop ax

ret

OEM\_DOS ENDP

; функция определения серийного номера пользователя (bl:cx)

USER\_DOS PROC NEAR

push bx

push cx

push di

push ax

mov di, offset \_user

add di, 22h

mov ax, cx

call WRD\_TO\_HEX

mov al, bl

call BYTE\_TO\_HEX

mov di, offset \_user

add di, 1Dh

mov [di], ax

pop ax

pop di

pop cx

pop bx

ret

USER\_DOS ENDP

; функция вывода на экран

PRINT PROC NEAR

push ax

mov ah, 09h

int 21h

pop ax

ret

PRINT ENDP

; КОД

BEGIN:

mov ax, @data

mov ds, ax

mov bx, ds

; вызываем функцию определения типа IBM PC

call TYPE\_IMB\_PC

; вывод поясняющей строки

mov dx, offset \_type

call PRINT

; определяем по предпоследнему биту ROM BIOS тип IBM PC

; предварительно загружаем смещения определённого типа

; и в случае успеха - выводим тип на экран

mov dx, offset \_PC

cmp bh, 0FFh

je to\_print

mov dx, offset \_PC\_XT

cmp bh, 0FEh

je to\_print

mov dx, offset \_PC\_XT

cmp bh, 0FBh

je to\_print

mov dx, offset \_AT

cmp bh, 0FCh

je to\_print

mov dx, offset \_PC2\_30

cmp bh, 0FAh

je to\_print

mov dx, offset \_PC2\_50\_60

cmp bh, 0FCh

je to\_print

mov dx, offset \_PC2\_80

cmp bh, 0F8h

je to\_print

mov dx, offset \_PCjr

cmp bh, 0FDh

je to\_print

mov dx, offset \_PC\_Conv

cmp bh, 0F9h

je to\_print

mov al, bh

call BYTE\_TO\_HEX

mov dx, ax

; вызываем функцию вывода на экран

to\_print:

call PRINT

; вызываем функцию определения версии MS DOS

mov ah, 30h

int 21h

; сохраняем полученные значения

call VERS\_DOS

call OEM\_DOS

call USER\_DOS

; выводим полученные значения

mov dx, offset \_ver

call PRINT

mov dx, offset \_oem

call PRINT

mov dx, offset \_user

call PRINT

; выход в DOS

xor al, al

mov ah, 4ch

int 21h

END START ; конец модуля