

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

Студентка гр. 7381

Процветкина А.В.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2019

Цель работы.

Исследование различий в структурах исходных текстов модулей .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 байта типу IBM PC представлено в табл. 1.

Таблица 1 – Соответствие байта и типа IBM PC

Значение байта	Тип IBM PC
FF	PC
FE, FB	PC/XT
FC	AT
FA	PS2 модель 30
FC	PS2 модель 50/60
F8	PS2 модель 80
FD	PCjr
F9	PC Convertible

План загрузки в память модулей .COM:

При загрузке программы типа .COM регистр IP всегда инициализируется числом 100h, поэтому сразу за директивой org 100h должно стоять первое выполнимое предложение программы. После загрузки программы все 4 сегментных регистра указывают на начало единственного сегмента, т. е. фактически на начало PSP. Указатель стека автоматически инициализируется числом FFFEh. Таким образом, независимо от фактического размера программы, ей выделяется 64 Кбайт адресного пространства, всю нижнюю часть которого занимает стек.

Постановка задачи.

Составить исходный .COM модуль, определяющий тип PC и версию системы. Получить "плохой".EXE модуль из программы, предназначенной для COM модуля, после чего построить "хороший" .EXE модуль выполняющий те же функции, что и отлаженный .COM модуль. Сравнить тексты полученных программ и модулей. Ответить на контрольные вопросы.

Выполнение работы.

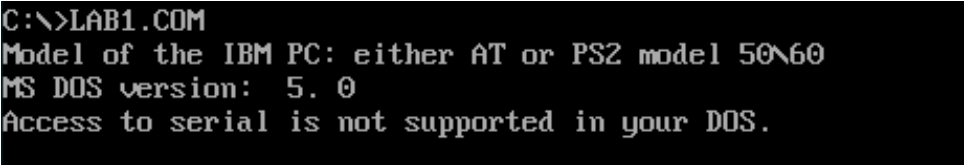
Был написан текст для .COM модуля, определяющий тип PC и версию системы. Ассемблерная программа считывает предпоследний байт ROM BIOS и после сравнения его с имеющимися данными выводит на экран либо идентифицированный тип PC, либо этот самый байт в шестнадцатеричном представлении.

Затем определяется версия системы с помощью вызова функции 30h прерывания 21h, которая имеет результатом следующий набор данных:

1. AL – номер основной версии
2. AH – номер модификации
3. BH – серийный номер OEM (original equipment manufacturer)
4. BL:CH – 24-х битовый серийный номер пользователя.

Примечание: для большинства версий DOS значения регистров BH и CH после вызова данной функции равны 0.

Содержимое регистров преобразуется в строковый формат, после чего полученная о системе информация выводится на экран, как показано на рис. 1.



```
C:\>LAB1.COM
Model of the IBM PC: either AT or PS2 model 50\60
MS DOS version: 5. 0
Access to serial is not supported in your DOS.
```

Рисунок 1 – Пример работы программы

Поскольку серийные номера OEM и пользователя недоступны в

эмулированной в работе системе DOS (значения регистров BX и CX равны 0), проверить полную работоспособность программы напрямую не представляется возможным, однако после изменения значений регистров в отладчике TD, можно убедиться в корректности выдаваемого результата, представленного на рис.3, для случайных значений из рис.2.

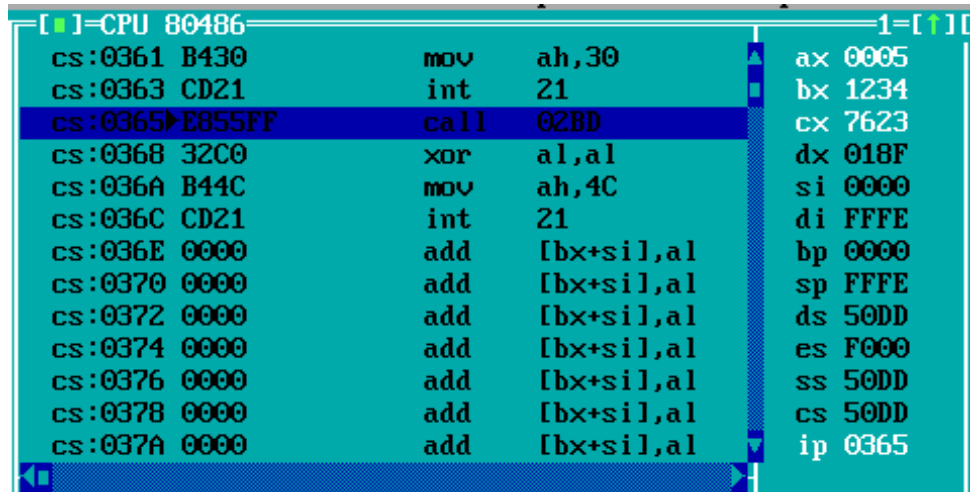


Рисунок 2 – Изменение значений регистров вручную

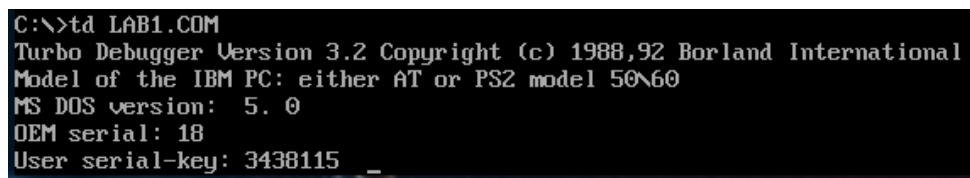


Рисунок 3 – Результат работы программы

Исходный код составленной программы представлен в приложении А.

Ответы на контрольные вопросы.

Отличия исходных текстов .COM и .EXE программ

1. Сколько сегментов должна содержать .COM-программа?

Ответ: 1.

2. EXE-программа?

Ответ: Обязательно как минимум 1 – сегмент кода, логически их обычно 3: сегмент кода, данных и стека, однако и без двух последних .EXE-программы работают корректно.

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

Ответ: В программе обязательно должны присутствовать директивы ORG, END.

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

Ответ: Все.

Отличия форматов файлов .COM и .EXE модулей

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

Ответ: см. рис.4.



Рисунок 4 – Организация .COM-модуля

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

Ответ: "Плохой" .EXE отличается от файла .COM при просмотре через FAR добавленным заголовком, который располагается с адреса 0. Код начинается с адреса 300h (см. рис.5).

000000002E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000002F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000300:	E9 40 02 4D 6F 64 65 6C	20 6F 66 20 74 68 65 20	й@Model of the
00000000310:	49 42 4D 20 50 43 3A 20	24 50 43 0D 0A 24 50 43	IBM PC: \$PC;\$PC
00000000320:	2F 58 54 0D 0A 24 65 69	74 68 65 72 20 41 54 20	/XT;\$either AT
00000000330:	6F 72 20 50 53 32 20 6D	6F 64 65 6C 20 35 30 5C	or PS2 model 50\
00000000340:	36 30 0D 0A 24 50 53 32	20 6D 6F 64 65 6C 20 33	60;\$PS2 model 3

Рисунок 5 – Вид "плохого" .EXE-модуля

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

Ответ: В "хорошем" .EXE-файле присутствует разбиение на сегменты. Есть стек. Структура "хорошего" .EXE приведена на рис.6.



Рисунок 6 – Структура "хорошего" .EXE-модуля

Загрузка .COM модуля в основную память

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

Ответ: Для .COM-файла DOS автоматически определяет стек и устанавливает одинаковый общий сегментный адрес во всех четырех сегментных регистрах (начало PSP). Если для программы размер сегмента в 64К является достаточным, то DOS устанавливает в регистре SP адрес конца сегмента – FFFE. PSP заполняет по-прежнему система, но место под него в начале сегмента должен отвести программист. Код располагается с адреса 100h.

2. Что располагается с адреса 0?

Ответ: PSP.

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

Ответ: Все сегментные регистры указывают на PSP.

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

Ответ: Значение регистра SP устанавливается так, чтобы он указывал на последнюю доступную в сегменте ячейку памяти. Таким образом программа занимает начало, а стек – конец сегмента.

Загрузка "хорошего".EXE модуля в основную память

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

Ответ: Сначала формируется PSP, затем стандартная часть заголовка считывается в память, после чего загрузочный модуль считывается в начальный сегмент. DS и ES указывают на начало префикса программного сегмента. Регистры CS и SS получают значения, указанные компоновщиком.

2. На что указывают регистры ES и DS?

Ответ: на начало PSP.

3. Как определяется стек?

Ответ: В регистры SS и SP записываются значения, указанные в заголовке, а к SS прибавляется сегментный адрес начального сегмента.

4. Как определяется точка входа?

Ответ: Оператором `END start_procedure_name`. Эта информация хранится в заголовке модуля.

Выводы.

В ходе лабораторной работы был написан .COM модуль, определяющий тип РС и версию системы. Из него получен "плохой" .EXE модуль, после чего построен "хороший". Файлы были сопоставлены и изучены. Были исследованы особенности загрузки каждого из модулей в память.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ ТЕКСТ .COM МОДУЛЯ

```

TESTPC  SEGMENT
ASSUME  CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG     100H
START:  JMP     BEGIN

string db 'Model of the IBM PC: ', '$'
PC      db      'PC', 0DH, 0AH, '$'
PC_XT   db 'PC/XT', 0DH, 0AH, '$'
AT_or_PS2_50 db 'either AT or PS2 model 50\60', 0DH, 0AH, '$'
PS2_30  db 'PS2 model 30', 0DH, 0AH, '$'
PS2_80  db 'PS2 model 80', 0DH, 0AH, '$'
PCjr    db 'PCjr', 0DH, 0AH, '$'
PC_conv db 'PC convertible', 0DH, 0AH, '$'
default db ' (undefined type)', 0DH, 0AH, '$'
DOS_str db 'MS DOS version: ', '$'
OEM_str db 'OEM serial: ', '$'
ERR_MSG db 'Access to serial is not supported in your DOS.', 0DH, 0AH, '$'

OS_str1 db 6 dup(?)
OEM db 4 dup(?)
serial_info db 'User serial-key: ', '$'
serial db 10 dup(?)

;-----
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                                ;num stored in AL into
    ASCII in AX in hex
    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
;-----
BYTE_TO_DEC      PROC      near      ;num in AL into ASCII
        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
;-----

DET_TYPE PROC near
        cmp al, 0FFh
        jne cmp_1
        mov dx, offset PC
        ret
cmp_1:
        cmp al, 0FEh
        jne cmp_2
        mov dx, offset PC_XT
        ret
cmp_2:
        cmp al, 0FBh
        jne cmp_3
        mov dx, offset PC_XT
        ret
cmp_3:

```

```

        cmp al, 0FCh
        jne cmp_4
        mov dx, offset AT_or_PS2_50
        ret
cmp_4:
        cmp al, 0F9h
        jne cmp_5
        mov dx, offset PC_conv
        ret
cmp_5:
        cmp al, 0FAh
        jne cmp_6
        mov dx, offset PS2_30
        ret
cmp_6:
        cmp al, 0F8h
        jne cmp_7
        mov dx, offset PS2_80
        ret
cmp_7:
        cmp al, 0FDh
        jne undefined
        mov dx, offset PCjr
        ret
undefined:
        call BYTE_TO_HEX
        mov bh, al
        mov bl, ah
        mov dl, bh
        mov ah, 02h                ;char output
        int 21h
        mov dl, bl
        int 21h
        mov dx, offset default
        ret
DET_TYPE ENDP

DIV10 proc near
        mov cx, 10
        mov bx, ax
        xchg ax, dx
        xor dx, dx
        div cx
        xchg bx, ax
        div cx

```

```

        xchg dx, bx
        ret
DIV10 endp

DW_TO_ASCII proc near
        call DIV10
        mov si, dx
        or si, ax
        jz Done
        push bx
        call DW_TO_ASCII
        pop bx
Done:
        add bl, '0'
        mov [di], bl
        inc di
        ret
DW_TO_ASCII endp

WRITE_OS proc near
        mov si, offset OS_str1
        inc si
        call BYTE_TO_DEC
        add si, 4
        mov al, ah
        push bx
        call BYTE_TO_DEC
        pop bx
        dec si
        mov byte ptr [si], 46                ;dot
        mov byte ptr [si+3], '$'
        mov dx, offset OS_str1
        mov ah, 09h
        int 21h
        mov dl, 0Dh
        mov ah, 02h
        int 21h
        mov dl, 0Ah
        int 21h

check_a:                                ;in most DOS (30h, int 21h)
        returns 0 in bx & cx
        cmp bx, 0
        je check_b
        mov si, offset OEM

```

```

    inc si
    mov al, bh
    push bx
    call BYTE_TO_DEC
    pop bx
    mov dx, offset OEM_str
    mov ah, 09h
    int 21h
    mov si, offset OEM
    mov byte ptr [si+3], '$'
    mov dx, si
    int 21h
    mov dl, 0Dh
    mov ah, 02h
    int 21h
    mov dl, 0Ah
    int 21h

check_b:
    cmp cx, 0
    je print_err
    mov di, offset serial
    mov dl, bl
    mov ah, ch
    mov al, cl
    mov dh, 0
    call DW_TO_ASCII
    mov dx, offset serial_info
    mov ah, 09h
    int 21h
    mov di, offset serial
    mov byte ptr [di+9], '$'
    mov dx, di
    mov ah, 09h
    int 21h
    ret

print_err:
    mov dx, offset ERR_MSG
    mov ah, 09h
    int 21h
    ret
WRITE_OS endp

BEGIN:

```

```

    mov dx, offset string
    mov ah, 09h
    int 21h

    mov ax, 0F000h
    mov es, ax
    mov di, 0FFFFh
    mov al, byte ptr es:di
    call DET_TYPE
    mov ah, 09h
    int 21h

    mov dx, offset DOS_str
    int 21h
    mov ah, 30h
    int 21h
    call WRITE_OS

    xor     al, al
    mov     ah, 4Ch
    int     21h
TESTPC    ENDS

END       START

```