МИНОБРНАУТhИ РОССИИ

САНТhТ-ПЕТЕРБУРГСТhИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕТhТРОТЕХНИЧЕСТhИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Thафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Операционные системы»

Тема: Исследование структур загрузочных модулей

Студентка гр. 7381 _____ Кортев Ю.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.

Затем исходный код COM модуля переделан для корректной работы EXE модуля.

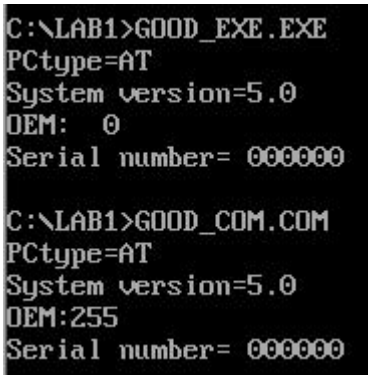

Рисунок 1 – Пример работы программ

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 3: сегмент кода, данных и стека, однако и без двух последних .EXE- програмы работают корректно.

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

Ответ: В программе обязательно должны присутствовать директивы ORG, END, ASSUME. При их удалении возникают ошибки компиляции.

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

Ответ: Нет, не все. COM-программа подразумевает наличие только одного сегмента, а значит, можно использовать только near-переходы, так как в far переходах подразумевается использование нескольких сегментов. Также нельзя использовать команды, связанные с адресом сегмента, потому что адрес сегмента до загрузки неизвестен, так как в COM-программах в DOS не содержится таблицы настройки, которая содержит описание адресов, зависящих от размещения загрузочного модуля в ОП, потому что подобные адреса в нем запрещены.

Отличия форматов файлов .COM и .EXE модулей

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

Ответ: см. рис.2.

Рисунок 2 – Структура .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(см.рис.3).

```
0000000000:  4D 5A C3 00 03 00 00 00  20 00 00 00 FF FF 00 00
0000000010:  00 00 7D D6 00 01 00 00  1E 00 00 00 01 00 00 00
0000000020:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000030:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000040:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000050:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000060:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000070:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

Рисунок 3 - Заголовок .EXE-модуля

Заголовок содержит необходимую информацию для загрузки программы в память и специальную таблицу, необходимую для настройки ссылок на дальние

сегменты программы (relocation table - таблица перемещения). Дело в том, что при создании COM программы весь код программы находится в одном сегменте. Чтобы такая программа корректно работала, ей не нужно знать, в каком сегменте располагается её код, имеет значение лишь адрес

внутри сегмента (смещение). В EXE программе кодовых сегментов может быть несколько и для обращения к коду другого сегмента (например, дальний вызов процедуры) нужно знать не только смещение внутри этого сегмента, но и его сегментный адрес. Но программа, не загруженная в память, не может знать этот сегментный адрес, так как заранее не известно, в какое место памяти операционная система поместит код программы, прочитанный из EXE файла. Для решения этой проблемы и служит таблица перемещения. Код начинается с адреса 300h (см. рис.4).

Рисунок 4 – Вид "плохого" .EXE-модуля

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

*Ответ:*В отличие от плохого, хороший EXE-файл не содержит директивы ORG 100h (которая выделяет память под PSP), поэтому код начинается с меньшего адреса. В "хорошем" .EXE-файле присутствует разбиение на сегменты. Есть стек. Структура "хорошего" .EXE приведена на рис.8.

Рисунок 5 – 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. 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. 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 PC и версию системы. Из него получен "плохой" .EXE модуль, после чего построен "хороший". Файлы были сравнены и проанализированы. Были исследованы особенности загрузки каждого из модулей в память.

ПРИЛОЖЕНИЕ А

КОД ИСХОДНОЙ ПРОГРАММЫ

```
TESTPC    SEGMENT
          ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING,
SS:NOTHING
          ORG  100H

START: JMP BEGIN

PCtype db 'PCtype=$'
PCtype_PC db 'PC',0DH,0AH,'$'
PCtype_PCXT db 'PC/XT',0DH,0AH,'$'
PCtype_AT db 'AT',0DH,0AH,'$'
PCtype_PS2_30 db 'PS2 model 30',0DH,0AH,'$'
PCtype_PS2_50_or_60 db 'PS2 model 50 or 60',0DH,0AH,'$'
PCtype_PS2_80 db 'PS2 model 80',0DH,0AH,'$'
PCtype_PCjr db 'PCjr',0DH,0AH,'$'
PCtype_PC_Convertible db 'PC Convertible',0DH,0AH,'$'

System_version db 'System version= . ',0DH,0AH,'$'
OEM db 'OEM:   ',0DH,0AH,'$'
Serial_number db 'Serial number=      ',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
;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 в 10с/с, SI - адрес поля младшей цифры
    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
;------------------------------
```

```
Print PROC near
        mov AH,09h
        int 21h
        ret
Print ENDP


PC_type PROC near
   push ax
        lea dx,PCtype
        call Print
   mov ax,0F000h
        mov es,ax
        mov ax,es:0FFFEh

        cmp al,0FFh
        je case_PC

        cmp al,0FEh
        je case_PCXT

        cmp al,0FBh
        je case_PCXT

        cmp al,0FCh
        je case_AT

        cmp al,0FAh
        je case_PS2_30

        cmp al,0FCh
        je case_PS2_50_or_60

        cmp al,0F8h
        je case_PS2_80

        cmp al,0FDh
        je case_PCjr

        cmp al,0F9h
        je case_PC_Convertible

        case_PC:
                lea dx,PCtype_PC
                jmp case_Print
        case_PCXT:
```

```
            lea dx,PCtype_PCXT
            jmp case_Print
        case_AT:
            lea dx,PCtype_AT
            jmp case_Print
        case_PS2_30:
            lea dx,PCtype_PS2_30
            jmp case_Print
        case_PS2_50_or_60:
            lea dx,PCtype_PS2_50_or_60
            jmp case_Print
        case_PS2_80:
            lea dx,PCtype_PS2_80
            jmp case_Print
        case_PCjr:
            lea dx,PCtype_PCjr
            jmp case_Print
        case_PC_Convertible:
            lea dx,PCtype_PC_Convertible
            jmp case_Print

        case_Print:
            call Print
            pop ax
            ret
PC_type ENDP


SystemVersion PROC near
        mov ah,30h
        int 21h

    ; System version (AL- номер основной версии, AH - номер модификации)

        lea dx,System_version
        mov si,dx
        add si,15
        call BYTE_TO_DEC
        add si,3
        mov al,ah
        call BYTE_TO_DEC
        call Print

    ; OEM (BH-сериный номер Original Equipment Manufacturer)

        lea dx,OEM
```

```
        mov si,dx
        add si,6
        mov al,bh
        call BYTE_TO_DEC
        call Print

    ; Serial number (BL:CX - 24-битовый серийный номер пользователя)

        lea dx,Serial_number
        mov di,dx
        mov al,bl
        call BYTE_TO_HEX
        add di,15
        mov [di],ax
        mov ax,cx
        mov di,dx
        add di,20
        call WRD_TO_HEX
        call Print

        ret
SystemVersion ENDP

; Код

BEGIN:

    call PC_type
        call SystemVersion
        xor AL,AL
        mov AH,4Ch
        int 21H
TESTPC ENDS
    END START
```