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. 7381 _____ Адамов Я.В.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2019

Цель работы.

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

Описание функций.

Название функции	Описание
TETR_TO_HEX	Перевод 4-битного числа в код символа в 16 с/с. Вспомогательная функция для BYTE_TO_HEX.
BYTE_TO_HEX	Байт в AL переводится в два символа шестн. числа в AX.
WRD_TO_HEX	Перевод в 16 с/с 16-ти разрядного числа, в AX – число, DI – адрес последнего символа.
BYTE_TO_DEC	Перевод в 10 с/с, SI – адрес поля младшей цифры.
PrintMsg	Печать строки, адрес которой помещен в DX.
PRINT_PC_TYPE	Печать типа PC.
PRINT_SYSTEM_VERSION	Печать версии системы, OEM и серийного номера.

Описание структур данных.

Название	Тип	Описание
PC_type	db	Строка для вывода: «PC type: ».
PCtype_PC	db	Тип PC: PC

PCtype_PCXT	db	Тип PC: PC/XT
PCtype_AT	db	Тип PC: AT
PCtype_PS2_30	db	Тип PC: PS2 модель 30
PCtype_PS2_50_or_60	db	Тип PC: PS2 модель 50 или 60
PCtype_PS2_80	db	Тип PC: PS2 модель 80
PCtype_PCjr	db	Тип PC: PCjr
PCtype_PC_Convertible	db	Тип PC: Convertible
System_version	db	Строка для вывода: «System version: . »
OEM	db	Строка для вывода: «OEM: »
Serial_number	db	Строка для вывода: «Serial number: »

Описание работы утилиты.

Программа выводит на экран сведения о типе PC, версии системы, серийном номере OEM и серийном номере пользователя. Результат работы «хорошего» .COM модуля представлен на рис. 1, «плохого» .EXE модуля – на рис. 2, «хорошего» .EXE модуля – на рис. 3.

Рисунок 1 – результат работы «хорошего» .COM модуля.

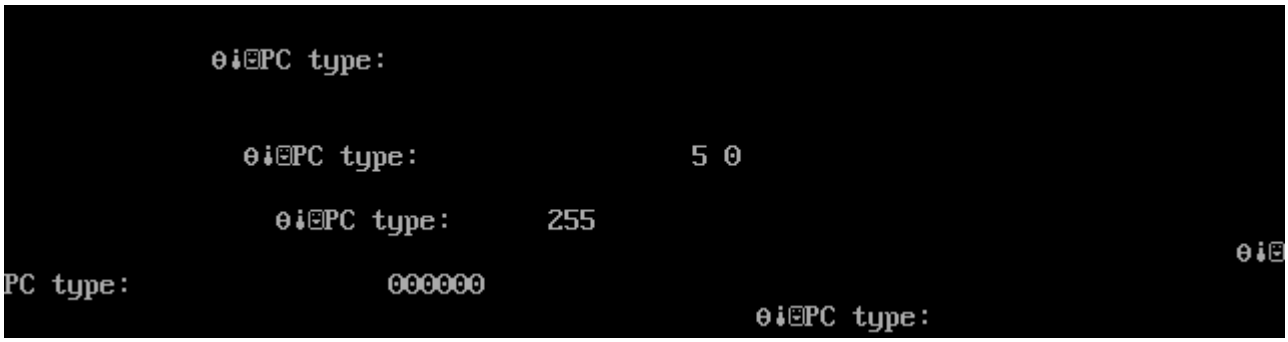

Рисунок 2 – результат работы «плохого» .EXE модуля.

Рисунок 3 – результат работы «хорошего» .EXE модуля.

Вывод.

В ходе выполнения лабораторной работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

Ответы на контрольные вопросы.

Отличия исходных текстов COM и EXE программ:

Сколько сегментов должна содержать COM-программа?

Один.

Сколько сегментов должна содержать EXE-программа?

Больше нуля.

Какие директивы должны обязательно быть в тексте COM-программы?

ORG 100h для резервирования 100h байт памяти в начале программы, в которые будет помещен PSP, и ASSUME для инициализации сегментных регистров.

Все ли форматы команд можно использовать в COM-программе?

Не все: в .COM программе нельзя получить адрес сегмента, так как он не известен до загрузки модуля в память ввиду отсутствия таблицы настройки, содержащей адреса сегментов, из-за чего также невозможны far-переходы.

Отличия форматов файлов COM и EXE модулей:

Какова структура файла COM? С какого адреса располагается код?

COM файл состоит из кода и данных, код располагается с нулевого адреса.

Какова структура файла «плохо» EXE? С какого адреса располагается код? Что располагается с адреса 0?

В «плохо» EXE-файле код и данные располагаются в одном сегменте, который начинается с адреса 300h, что представлено на рис. 4. Первые 200h байт занимают заголовок и таблица настроек, следующие 100h байт памяти зарезервированы директивой ORG 100h.

Рисунок 4 – структура «плохого» EXE файла.

Какова структура файла «хорошего» EXE? Чем он отличается от файла «плохо» EXE?

В отличии от «плохого» EXE-файла, «хороший» разбит на сегменты (в данном случае 3, но их могло быть произвольное количество). Первый сегмент (в данном случае стек, который занимает 200h байт) начинается

с адреса 200h, так как перед ним следуют заголовок и таблица настроек. Структура файла представлена на рис. 5.

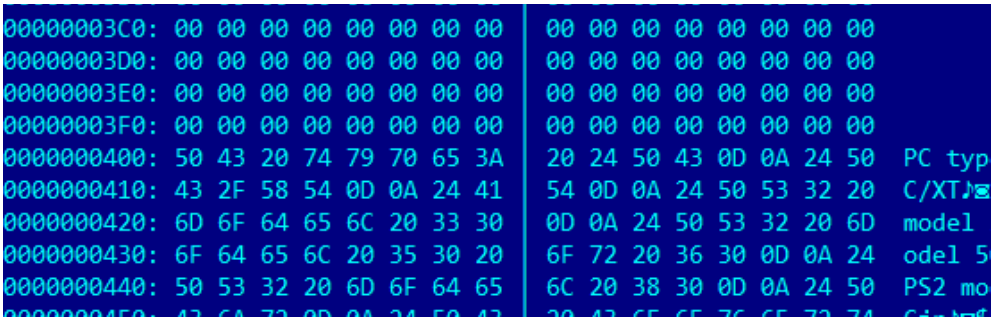

Рисунок 5 – структура «хорошего» EXE файла.

Загрузка COM модуля в основную память:

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

Система выделяет необходимое количество памяти для единственного сегмента, в первых 100h байтах (зарезервированных директивой ORG 100h) которого строится PSP. Код и данные начинаются сразу за PSP.

Что располагается с адреса 0?

PSP (Program Segment Prefix).

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

Все регистры указывают на начало единственного сегмента, то есть на начало PSP. Значения регистров представлены на рис. 6.

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

Как уже было сказано, SS указывает на начало PSP, SP же указывает на конец сегмента (FFFEh), то есть этот сегмент является также и стеком.

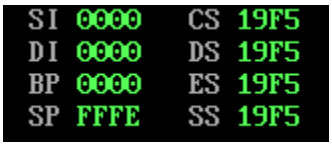

Рисунок 6 – состояние регистров .COM программы.

Загрузка «хорошего» EXE модуля в основную память:

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

Система выделяет 100h байт памяти под PSP, затем в память загружается сама программа. CS и SS указывают на соответствующие сегменты, которые были указаны с помощью директивы ASSUME. Значения регистров представлены на рис. 7.

На что указывают регистры DS и ES?

На начало PSP.

Как определяется стек?

В исходном коде стек определяется с помощью директивы STACK. Стеков может быть несколько, поэтому для начальной инициализации нужный указывается с помощью директивы ASSUME.

Как определяется точка входа?

За директивой END следует название метки, указывающее на точку входа.

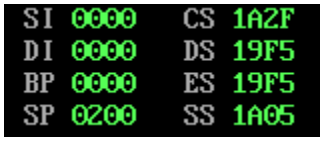

Рисунок 7 – состояние регистров .EXE программы.

Приложение А. Файл com.asm.

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

START: JMP BEGIN

; ---------------------------------------------
; Данные

PC_type db 'PC type: $'
PCtype_PC db 'PC',0DH,0AH,'$'
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,'$'


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

PrintMsg PROC near
        mov AH,09h
        int 21h
        ret
PrintMsg ENDP


PRINT_PC_TYPE PROC near
   push ax
        lea dx,PC_type
        call PrintMsg
   mov ax,0F000h
        mov es,ax
        mov ax,es:0FFFEh

        cmp al,0FFh
        je PC_metka
```

```
cmp al,0FEh
je PCXT_metka
cmp al,0FBh
je PCXT_metka
cmp al,0FCh
je AT_metka
cmp al,0FAh
je PS2_30_metka
cmp al,0FCh
je PS2_50_or_60_metka
cmp al,0F8h
je PS2_80_metka
cmp al,0FDh
je PCjr_metka
cmp al,0F9h
je PC_Convertible_metka

PC_metka:
        lea dx,PCtype_PC
        jmp end_of_print_PC_type
PCXT_metka:
        lea dx,PCtype_PCXT
        jmp end_of_print_PC_type
AT_metka:
        lea dx,PCtype_AT
        jmp end_of_print_PC_type
PS2_30_metka:
        lea dx,PCtype_PS2_30
```

```
            jmp end_of_print_PC_type
        PS2_50_or_60_metka:
            lea dx,PCtype_PS2_50_or_60
            jmp end_of_print_PC_type
        PS2_80_metka:
            lea dx,PCtype_PS2_80
            jmp end_of_print_PC_type
        PCjr_metka:
            lea dx,PCtype_PCjr
            jmp end_of_print_PC_type
        PC_Convertible_metka:
            lea dx,PCtype_PC_Convertible
            jmp end_of_print_PC_type

        end_of_print_PC_type:
        call PrintMsg
        pop ax
        ret
PRINT_PC_TYPE ENDP


PRINT_SYSTEM_VERSION PROC near
        mov ah,30h
        int 21h

    ; System version

        lea dx,System_version
```

```
    mov si,dx
    add si,16
    call BYTE_TO_DEC
    add si,3
    mov al,ah
    call BYTE_TO_DEC
    call PrintMsg


; OEM

    lea dx,OEM
    mov si,dx
    add si,7
    mov al,bh
    call BYTE_TO_DEC
    call PrintMsg


; Serial number

    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 PrintMsg

        ret
PRINT_SYSTEM_VERSION ENDP

; ____________________________________________
; Код

BEGIN:
   call PRINT_PC_TYPE
        call PRINT_SYSTEM_VERSION
        xor AL,AL
        mov AH,4Ch
        int 21H
TESTPC ENDS
   END START
```

Приложение Б. Файл exe.asm.

```
AStack SEGMENT STACK
        DW 100h DUP(?)
AStack ENDS

;_____________________________________________
DATA SEGMENT

PC_type db 'PC type: $'
PCtype_PC db 'PC',0DH,0AH,'$'
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,'$'

DATA ENDS


;_____________________________________________
CODE SEGMENT
```

```
   ASSUME CS:CODE, DS:DATA, SS:AStack

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
        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 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

PrintMsg PROC near
        mov AH,09h
        int 21h
        ret
PrintMsg ENDP


PRINT_PC_TYPE PROC near
   push ax
        lea dx,PC_type
        call PrintMsg
   mov ax,0F000h
        mov es,ax
        mov ax,es:0FFFEh

        cmp al,0FFh
```

```
je PC_metka
cmp al,0FEh
je PCXT_metka
cmp al,0FBh
je PCXT_metka
cmp al,0FCh
je AT_metka
cmp al,0FAh
je PS2_30_metka
cmp al,0FCh
je PS2_50_or_60_metka
cmp al,0F8h
je PS2_80_metka
cmp al,0FDh
je PCjr_metka
cmp al,0F9h
je PC_Convertible_metka

PC_metka:
        lea dx,PCtype_PC
        jmp end_of_print_PC_type
PCXT_metka:
        lea dx,PCtype_PCXT
        jmp end_of_print_PC_type
AT_metka:
        lea dx,PCtype_AT
        jmp end_of_print_PC_type
PS2_30_metka:
```

```
        lea dx,PCtype_PS2_30
        jmp end_of_print_PC_type
    PS2_50_or_60_metka:
        lea dx,PCtype_PS2_50_or_60
        jmp end_of_print_PC_type
    PS2_80_metka:
        lea dx,PCtype_PS2_80
        jmp end_of_print_PC_type
    PCjr_metka:
        lea dx,PCtype_PCjr
        jmp end_of_print_PC_type
    PC_Convertible_metka:
        lea dx,PCtype_PC_Convertible
        jmp end_of_print_PC_type

    end_of_print_PC_type:
    call PrintMsg
    pop ax
    ret
PRINT_PC_TYPE ENDP


PRINT_SYSTEM_VERSION PROC near
    mov ah,30h
    int 21h

  ; System version
```

```
    lea dx,System_version
    mov si,dx
    add si,16
    call BYTE_TO_DEC
    add si,3
    mov al,ah
    call BYTE_TO_DEC
    call PrintMsg


; OEM

    lea dx,OEM
    mov si,dx
    add si,7
    mov al,bh
    call BYTE_TO_DEC
    call PrintMsg


; Serial number

    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 PrintMsg

        ret
PRINT_SYSTEM_VERSION ENDP

; _____________________________________________
; Код

BEGIN:
   mov ax,DATA
        mov ds,ax
   call PRINT_PC_TYPE
        call PRINT_SYSTEM_VERSION
        xor AL,AL
        mov AH,4Ch
        int 21H
CODE ENDS
   END BEGIN
```