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. 7383 _____ Корякин М.П.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2019

Постановка задачи.

Исследование структур модулей типов .COM и .EXE, различие типов. Изучение способов их загрузки в основную память.

Описание функций:

- Write – вызывов функции печати строки.
- TYPE_OS – запись типа ОС.
- VERSION – запись версии ОС, номер OEM, а также номер пользователя.
- HELP_BYTE_TO_HEX – выводит на экран номер пользователя.
- TETR_TO_HEX – вспомогательная функция для работы BYTE_TO_HEX.
- BYTE_TO_HEX - Байт AL переводится в два символа шестн. числа AX.
- BYTE_TO_DEC – Перевод в 10сс, SI - адрес поля младшей цифры.

Описание данных:

- TypeOS – для вывода типа ОС.
- VERS – для вывода версии ОС.
- OEM – для вывода нмера OEM.
- USR_NUM –для вывода номера пользователя.
- ENDSTR – для переноса строки.
- PC, PC_XT, AT, PS2_30, PS2_80, PSjr, PC_Conv – для определения типа ОС, и в последствии вывода одного из них.

Программа определяет и выводит на экран: тип ОС, версию ОС, номер OEM, серийный номер пользователя.

Результаты работы программы.

Рисунок 1 – Результат работы программы l1com.com

Рисунок 2 – Результат работы программы l1com.exe

Рисунок 3 – Результат работы программы l1.exe

Вывод.

В результате исследование структур модулей .COM и .EXE были изучены различия этих структур, а также способы их загрузки в основную память.

Ответы на контрольные вопросы лабораторной работы №1:

Отличия исходных текстов COM и EXE программ

1. Сколько сегментов должна содержать COM-программа?

   Один сегмент.

2. EXE-программа?

   EXE-программа должна содержать минимум один сегмент.

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

   Обязательно должна быть директива ORG 100h, которая резервирует 256 байт для PSP. Также обязательной директивой является ASSUME, которая ставит в соответствие начало программы сегментам кода и данных.

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

   Нет, не все. Нельзя использовать команды вида mov регистр, сегмент(DATA, CODE). Нельзя использовать far – адресацию (дальняя адресация), т.к. в этом случае будет использоваться таблица настройки, которой нет в COM файле(но есть в EXE файле).

Отличия форматов файлов COM и EXE модулей

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

Рисунок 4 – Представление COM файла в шестнадцатеричном виде

COM файл не содержит таблицы настроек. (Таблица настроек необходима для EXE файла, она содержит ссылки на сегменты программы. При создании COM программы код находится в одном

сегменте программы, и следовательно не нужно знать ссылки на сегменты. В EXE программе сегментов может быть несколько, и для этого нужно знать их адреса). Он содержит только код и данные. Поэтому код располагается с нулевого адреса, как показано на рис.4.

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

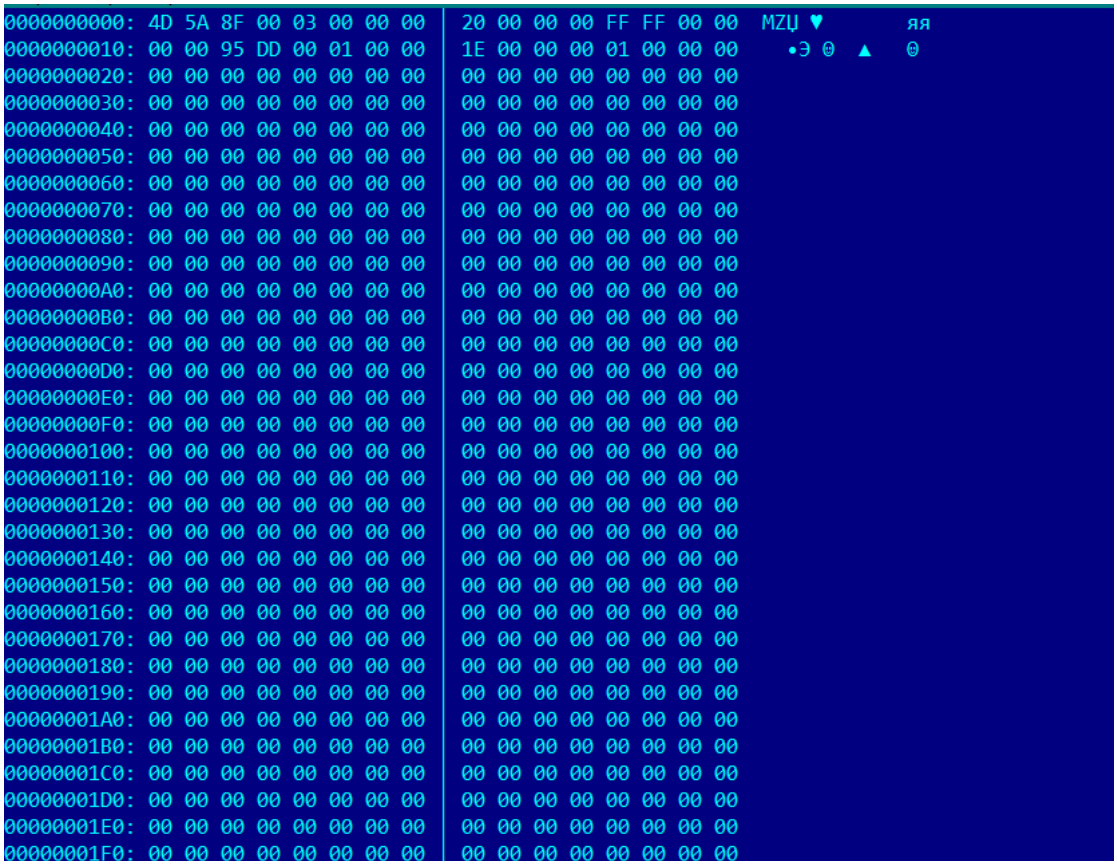

Рисунок 5 – Представление плохого EXE файла

Рисунок 6 – Представление плохого EXE файла

В структуре данного файла нет распределения по сегментам. С нулевого адреса располагается заголовок, после него следует таблица настройки. Сам код располагается с 300h, потому как заголовок занимает 200h, а также 100h сдвига дает директива ORG 100h.

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

   0-200h занимает заголовок и таблица настроек, 200-400h стек, 400-590h сегменты данных и кода. Хороший EXE файл отличается от плохого отдельно выделенными сегментами для кода и данных, дополнительным выделением под стек 100h. Также в хорошем EXE файле не используется директива ORG 100h, потому как загрузчик сам располагает программу после PSP.

Загрузка COM модуля в основную память

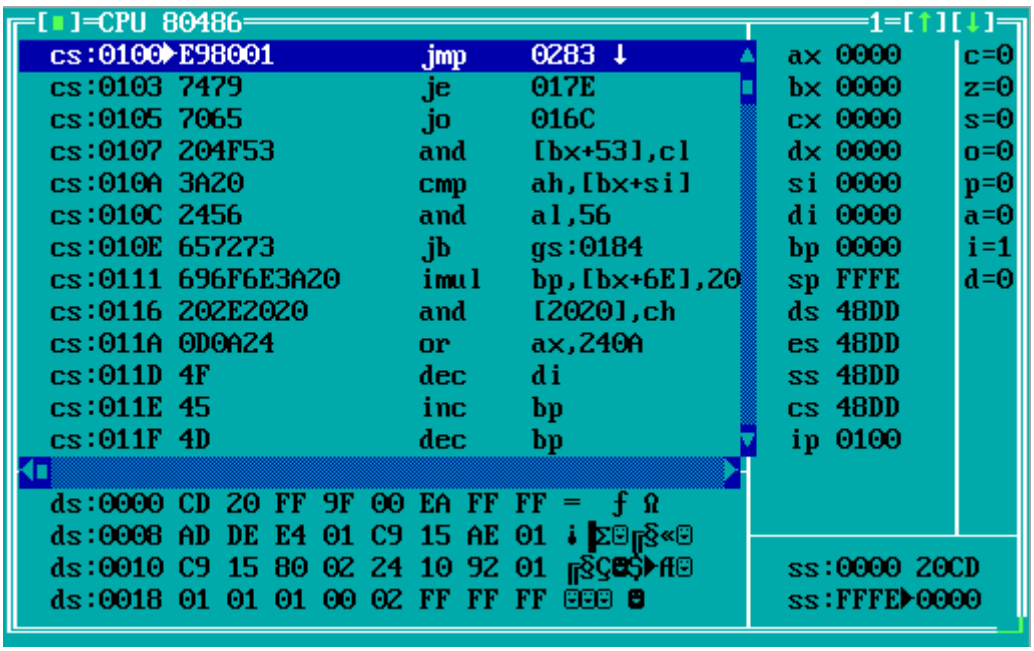

Рисунок 7 – Открытый COM файл в отладчике TD.EXE

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

   Сперва выделяется сегмент памяти для загрузки модуля, далее устанавливаются все сегментные регистры на начло памяти. В первых 100h выстраивается PSP. После этого загружается содержимое COM файла и регистру IP приставевается значение 100h. Код располагается со значения CS, в нашем случае это 48DD.

2. Что располагается с адреса 0?

   PSP

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

   Все сегментные регистры имеют значение 48DD и указывают на PSP.

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

   Адреса стека расположены в диапазоне 0000h-FFFEh и он занимает весь сегмент COM программы. Начало стека находится в регистрие SS, а его конец в SP.

Загрузка «хорошего» EXE модуля в основную память

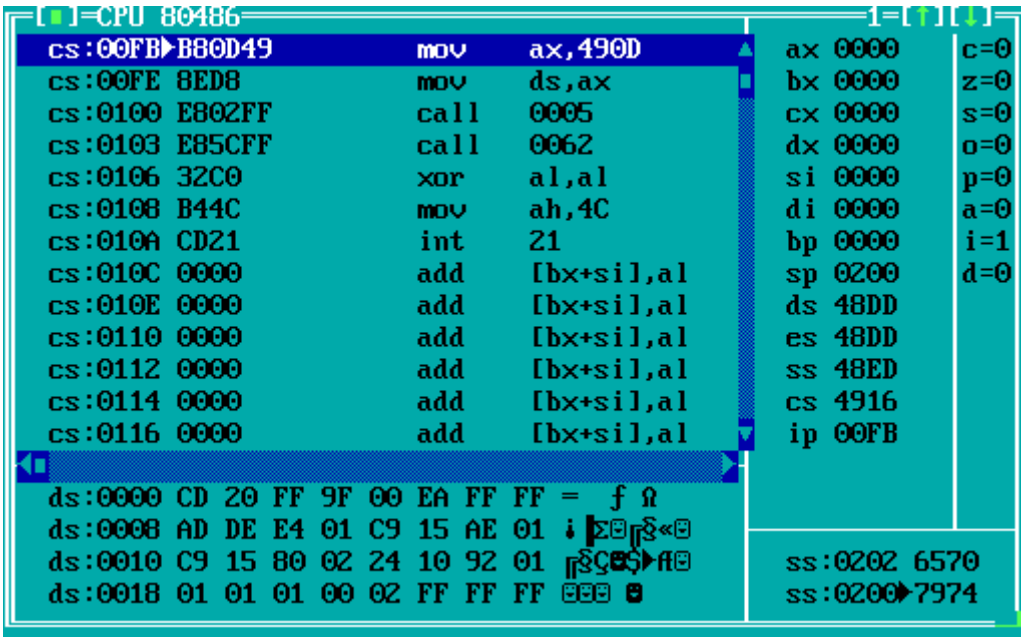

Рисунок 8 – Открытый EXE файл в отладчике TD.EXE

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

   При загрузке модуля программы в память в начало программы ставится PSP. После этого этого инициализируется сегментные регистры. DS и ES равны 48DD, SS равен 48ED и указывает на начало сегмента стека, CS равен 4916 и указывает на начало сегмента команд.

2. На что указывают регистры DS и ES?

   На начало PSP.

3. Как определяется стек?

   Стек определяется директивой STACK. На начало стека указыает регистр SS, а на его конец регистр SP.

4. Как определяется точка входа?

   Вход определяется директивой END. После этой директивы указывается метка, в которую переходит программа.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл l1.asm:

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

DATA SEGMENT
TypeOS db 'type OS: $'
VERS db 'Version:  .  ',0DH,0AH,'$'
OEM db 'OEM:    ',0DH,0AH,'$'
USR_NUM db 'Serial number user: ','$'
ENDSTR db 0DH,0AH,'$'

PC db 'PC',0DH,0AH,'$'
PC_XT db 'PC/XT',0DH,0AH,'$'
AT db 'AT',0DH,0AH,'$'
PS2_30 db 'PS2 model 30',0DH,0AH,'$'
PS2_80 db 'PS2 model 80',0DH,0AH,'$'
```

```
PCjr db 'PCjr',0DH,0AH,'$'
PC_Conv db 'PC Convertible',0DH,0AH,'$'
DATA ENDS

CODE SEGMENT
 ASSUME CS:CODE, DS:DATA, ES:NOTHING, SS:STACK

; Вывод на консоль
Write PROC near
     mov AH,09h
     int 21h
     ret
Write ENDP

; Запись и вывод типа системы
TYPE_OS PROC near
     mov dx, OFFSET TypeOS
     call Write
     mov ax,0F000h
     mov es,ax
     mov ax,es:0FFFEh

     ; Сравниваем ключи для определения типа системы
     cmp al,0FFh
          je PCpt
     cmp al,0FEh
          je PC_XTpt
     cmp al,0FBh
          je PC_XTpt
     cmp al,0FCh
          je ATpt
     cmp al,0FAh
          je PS2_30pt
     cmp al,0F8h
          je PS2_80pt
     cmp al,0FDh
          je PCjrpt
     cmp al,0F9h
          je PC_Convpt

     PCpt:
          mov dx, OFFSET PC
               jmp close
     PC_XTpt:
```

```
          mov dx, OFFSET PC_XT
               jmp close
     ATpt:
          mov dx, OFFSET AT
               jmp close
     PS2_30pt:
          mov dx, OFFSET PS2_30
               jmp close
     PS2_80pt:
          mov dx, OFFSET PS2_80
               jmp close
     PCjrpt:
          mov dx, OFFSET PCjr
               jmp close
     PC_Convpt:
          mov dx, OFFSET PC_Conv
               jmp close

     close:
          call Write
          ret
TYPE_OS ENDP

; Запись Version и OEM
VERSION PROC near
     ; Получаем номер версии и модификации версии
системы
     mov ax,0
     mov ah,30h
     int 21h

     ; Запись версии системы в строку VERS
     mov si,offset VERS
     add si,9
     call BYTE_TO_DEC

     ; Запись модификации в строку VERS
     mov al,ah
     add si,3
     call BYTE_TO_DEC

     ; Вывод полной версии в консоль
     mov dx,offset VERS
     call Write
```

```
     ; Вывод OEM
     mov si,offset OEM
     add si,7
     mov al,bh
     call BYTE_TO_DEC
     mov dx,offset OEM
     call Write

     ; Вывод серийного номера юзера
     mov dx,offset USR_NUM
     call Write
     mov al,bl
     call HELP_BYTE_TO_HEX
     mov al,ch
     call HELP_BYTE_TO_HEX
     mov al,cl
     call HELP_BYTE_TO_HEX

     mov dx,offset ENDSTR
     call Write

     ret
VERSION ENDP

HELP_BYTE_TO_HEX PROC near
     call BYTE_TO_HEX
     mov bx,ax
     mov dl,bl
     mov ah,02h
     int 21h
     mov dl, bh
     int 21h
HELP_BYTE_TO_HEX ENDP

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
```

```
; Байт AL переводится в два символа шестн. числа AX
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



; Перевод в 10сс, SI - адрес поля младшей цифры
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

BEGIN:
     mov ax,DATA
     mov ds,ax
     call TYPE_OS
     call VERSION
     xor AL,AL
```

```
        mov AH,4Ch
        int 21H
    CODE ENDS
END BEGIN
```

Файл l1com.asm:

```
    TESTPC SEGMENT
     ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
     ORG 100H
    START: JMP BEGIN
    TypeOS db 'type OS: $'
    VERS db 'Version:  .  ',0DH,0AH,'$'
    OEM db 'OEM:    ',0DH,0AH,'$'
    USR_NUM db 'Serial number user: ','$'
    ENDSTR db 0DH,0AH,'$'

    PC db 'PC',0DH,0AH,'$'
    PC_XT db 'PC/XT',0DH,0AH,'$'
    AT db 'AT',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,'$'

    ; Вывод на консоль
    Write PROC near
        mov AH,09h
        int 21h
        ret
    Write ENDP

    ; Запись и вывод типа системы
    TYPE_OS PROC near
        mov dx, OFFSET TypeOS
        call Write
        mov ax,0F000h
        mov es,ax
        mov ax,es:0FFFEh

        ; Сравниваем ключи для определения типа системы
        cmp al,0FFh
            je PCpt
```

```
    cmp al,0FEh
         je PC_XTpt
    cmp al,0FBh
         je PC_XTpt
    cmp al,0FCh
         je ATpt
    cmp al,0FAh
         je PS2_30pt
    cmp al,0F8h
         je PS2_80pt
    cmp al,0FDh
         je PCjrpt
    cmp al,0F9h
         je PC_Convpt

    PCpt:
         mov dx, OFFSET PC
              jmp close
    PC_XTpt:
         mov dx, OFFSET PC_XT
              jmp close
    ATpt:
         mov dx, OFFSET AT
              jmp close
    PS2_30pt:
         mov dx, OFFSET PS2_30
              jmp close
    PS2_80pt:
         mov dx, OFFSET PS2_80
              jmp close
    PCjrpt:
         mov dx, OFFSET PCjr
              jmp close
    PC_Convpt:
         mov dx, OFFSET PC_Conv
              jmp close

    close:
         call Write
         ret
TYPE_OS ENDP

; Запись Version и OEM
VERSION PROC near
```

```
    ; Получаем номер версии и модификации версии
системы
    mov ax,0
    mov ah,30h
    int 21h

    ; Запись версии системы в строку VERS
    mov si,offset VERS
    add si,9
    call BYTE_TO_DEC

    ; Запись модификации в строку VERS
    mov al,ah
    add si,3
    call BYTE_TO_DEC

    ; Вывод полной версии в консоль
    mov dx,offset VERS
    call Write

    ; Вывод OEM
    mov si,offset OEM
    add si,7
    mov al,bh
    call BYTE_TO_DEC
    mov dx,offset OEM
    call Write

    ; Вывод серийного номера юзера
    mov dx,offset USR_NUM
    call Write
    mov al,bl
    call HELP_BYTE_TO_HEX
    mov al,ch
    call HELP_BYTE_TO_HEX
    mov al,cl
    call HELP_BYTE_TO_HEX

    mov dx,offset ENDSTR
    call Write

    ret
VERSION ENDP
```

```
HELP_BYTE_TO_HEX PROC near
     call BYTE_TO_HEX
     mov bx,ax
     mov dl,bl
     mov ah,02h
     int 21h
     mov dl, bh
     int 21h
HELP_BYTE_TO_HEX ENDP

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

; Байт AL переводится в два символа шестн. числа AX
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



; Перевод в 10сс, SI - адрес поля младшей цифры
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

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