

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. 9383

Нистратов Д.Г.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Постановка задачи.

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

Функции и структуры данных

Таблица 1 – Функции и структуры данных

Название функций	Описание функций
TETR_TO_HEX	Перевод из четверичной с/с в шестнадцатеричную с/с
BYTE_TO_HEX	Перевод из двоичной с/с в шестнадцатеричную с/с
WRD_TO_HEX	Перевод 2 байтов в шестнадцатеричную с/с
BYTE_TO_DEC	Перевод из двоичной с/с в десятичную с/с
WRITE	Вывод строки на экран
OS_TYPE	Определение версии ОС, серийного номера и OEM номера
PC_TYPE	Определение типа PC

Последовательность действий

Шаг 1. Напишите текст исходного .COM модуля, который определяет тип PC и версию системы.

Шаг 2. Напишите текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1 и постройте, и отладьте его. Таким образом, будет получен “хороший” .EXE.

Шаг 3. Сравните исходные тексты для .COM и .EXE модулей. Ответьте на контрольные вопросы “Отличие исходных текстов COM и EXE программ”

Шаг 4. Запустите FAR и откройте (F3/F4) файл загрузочного модуля. COM и файл «плохого». EXE в шестнадцатеричном виде. Затем откройте (F3/F4) файл загрузочного модуля «хорошего». EXE и сравните его с предыдущими файлами. Ответьте на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

Шаг 5. Откройте отладчик TD.EXE и загрузите. COM. Ответьте на контрольные вопросы «Загрузка COM модуля в основную память». Представьте в отчете план загрузки модуля. COM в основную память.

Шаг 6. Откройте отладчик TD.EXE и загрузите «хороший». EXE. Ответьте на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

Шаг 7. Оформление отчета в соответствии с требованиями. В отчете необходимо привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей – в отладчике

Выполнение работы.

Шаг 1. При выполнении лабораторной работы был написан .COM модуль, определяющий тип PC и версию системы см. *Изображение 1 – пример работы .COM модуля* *Изображение 1*, а также был построен “плохой” .EXE модуль из исходного текста для .COM модуля см. *Изображение 2*.

```
D:\LETI\OS\MASM>LAB1.COM
PC TYPE: AT
OS version: 5.0
OEM: 255
Serial number: 000000
```

Изображение 1 – пример работы .COM модуля

```
D:\LETI\OS\MASM>lab1.exe

                                0п00S version:
on:                               5                                0п00S versi
                                0п00S version: 0
                                0п00S version: 255
                                0п00S version: 000000 255
                                0п00S version:
```

Изображение 2 – “плохой” .EXE модуль

Шаг 2. Был описан текст “хорошего” .EXE модуля, выполняющий те же функции что и модуль в Шаге 1, см. *Изображение 3*.

```
D:\LETI\OS\MASM>lab1_exe.exe
PC TYPE: AT
OS version: 5.0
OEM: 255
Serial number: 000000
```

Изображение 3 - "хороший" .EXE модуль

Шаг 3. Отличия исходных текстов COM и EXE программ

1) Сколько сегментов должна содержать COM-программа?

Ответ: COM-программа должна содержать только один сегмент.

2) EXE программа?

Ответ: EXE программа должна содержать один или более сегментов.

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

Ответ: обязательно должна быть директива *org 100h* для смещения программного сегмента, т. к. первые 256 байтов занимает PSP

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

Ответ: в COM-программе нельзя использовать команды вида *mov <регистр>, <сегмент>*, а также команды работающие с 64-х битными регистрами.

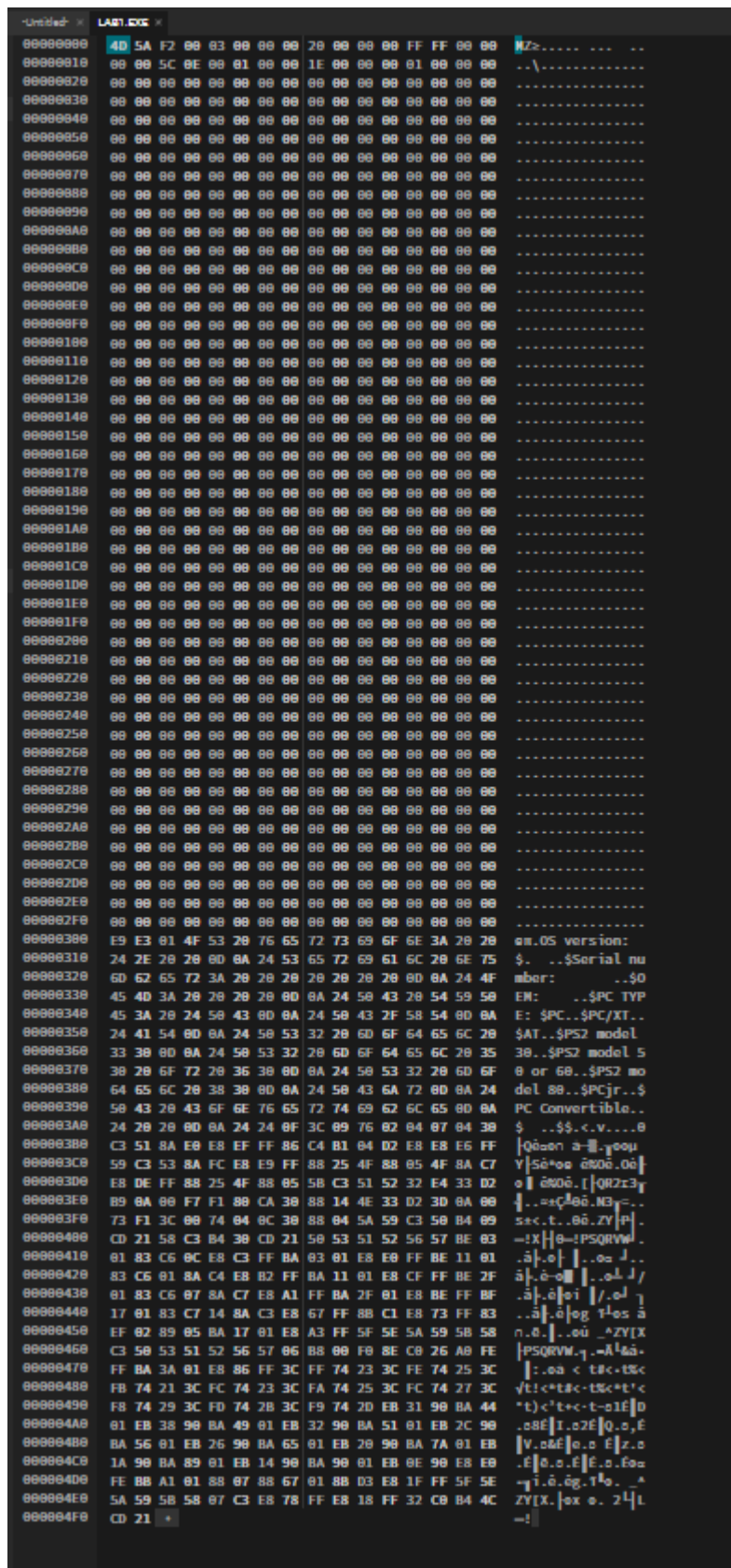
Шаг 4. Рассмотрим шестнадцатеричный вид загрузочного файла модуля .COM и “плохого” .EXE, см. Изображение 4 и Изображение 5. А также загрузочный файл “хорошего” модуля .EXE, см. Изображение 6.

```

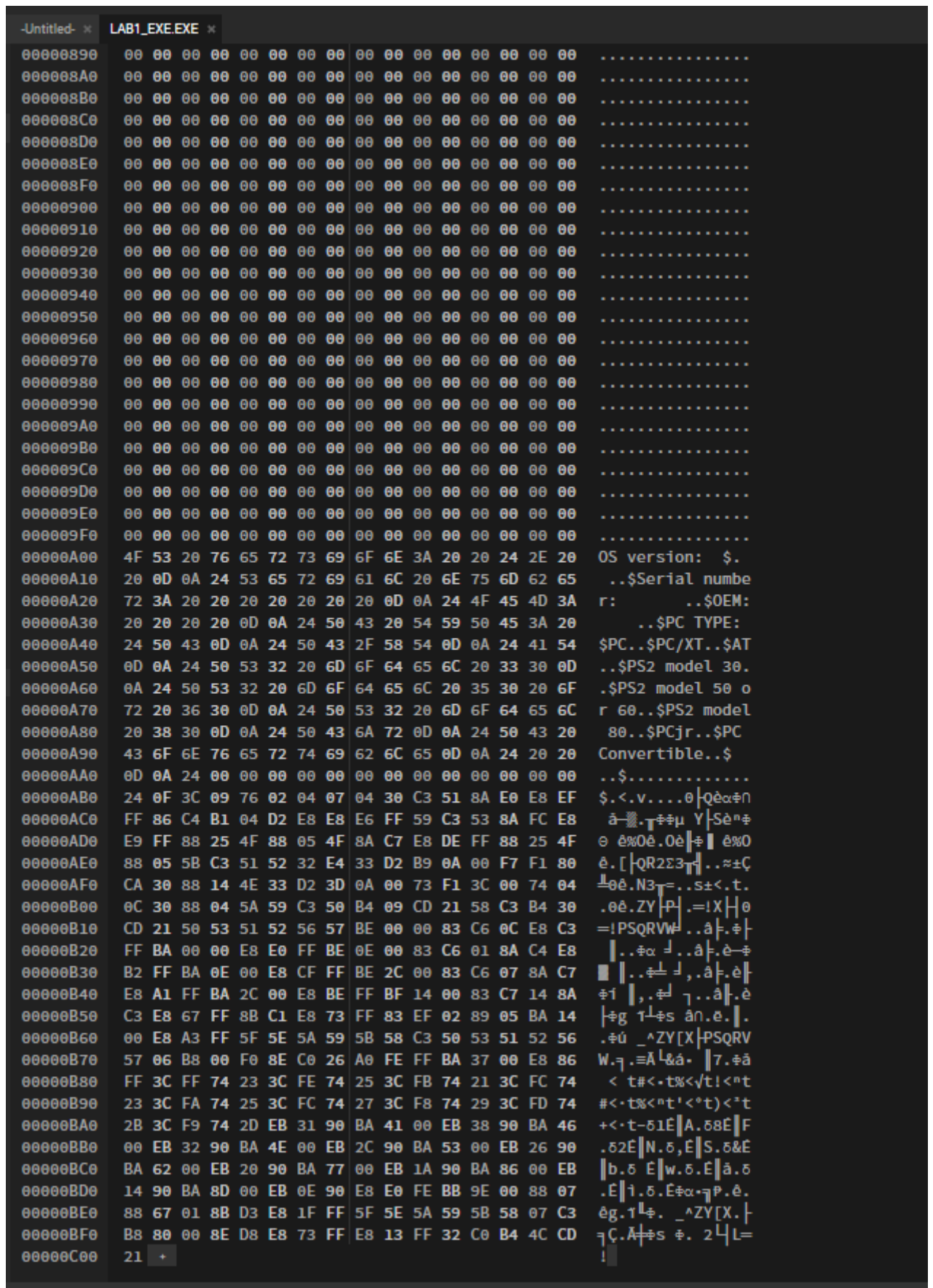
00000000 E9 E3 01 4F 53 20 76 65 72 73 69 6F 6E 3A 20 20 0π.OS version:
00000010 24 2E 20 20 0D 0A 24 53 65 72 69 61 6C 20 6E 75 $. ..$Serial nu
00000020 6D 62 65 72 3A 20 20 20 20 20 20 0D 0A 24 4F mber: ..$0
00000030 45 4D 3A 20 20 20 20 0D 0A 24 50 43 20 54 59 50 EM: ..$PC TYP
00000040 45 3A 20 24 50 43 0D 0A 24 50 43 2F 58 54 0D 0A E: $PC..$PC/XT..
00000050 24 41 54 0D 0A 24 50 53 32 20 6D 6F 64 65 6C 20 $AT..$PS2 model
00000060 33 30 0D 0A 24 50 53 32 20 6D 6F 64 65 6C 20 35 30..$PS2 model 5
00000070 30 20 6F 72 20 36 30 0D 0A 24 50 53 32 20 6D 6F 0 or 60..$PS2 mo
00000080 64 65 6C 20 38 30 0D 0A 24 50 43 6A 72 0D 0A 24 del 80..$PCjr..$
00000090 50 43 20 43 6F 6E 76 65 72 74 69 62 6C 65 0D 0A PC Convertible..
000000A0 24 20 20 0D 0A 24 24 0F 3C 09 76 02 04 07 04 30 $ ..$$.<.v....0
000000B0 C3 51 8A E0 E8 EF FF 86 C4 B1 04 D2 E8 E8 E6 FF |Qèαφn â—Tφφμ
000000C0 59 C3 53 8A FC E8 E9 FF 88 25 4F 88 05 4F 8A C7 Y|Sè^nφê%0ê.Oê|
000000D0 E8 DE FF 88 25 4F 88 05 5B C3 51 52 32 E4 33 D2 φ|ê%0ê.[|QR2Σ3T
000000E0 B9 0A 00 F7 F1 80 CA 30 88 14 4E 33 D2 3D 0A 00 ¶|..±çl0ê.N3T=..
000000F0 73 F1 3C 00 74 04 0C 30 88 04 5A 59 C3 50 B4 09 s±<.t..0ê.ZY|P|.
00000100 CD 21 58 C3 B4 30 CD 21 50 53 51 52 56 57 BE 03 =!X|0=|PSQRVW|.
00000110 01 83 C6 0C E8 C3 FF BA 03 01 E8 E0 FF BE 11 01 .â|.φ|..φα ∫..
00000120 83 C6 01 8A C4 E8 B2 FF BA 11 01 E8 CF FF BE 2F â|.è—φ|..φ± ∫/
00000130 01 83 C6 07 8A C7 E8 A1 FF BA 2F 01 E8 BE FF BF .â|.è|φí |/.φ ∫
00000140 17 01 83 C7 14 8A C3 E8 67 FF 8B C1 E8 73 FF 83 ..â|.è|φg ïlφs â
00000150 EF 02 89 05 BA 17 01 E8 A3 FF 5F 5E 5A 59 5B 58 n.ë.|..φú _^ZY[X
00000160 C3 50 53 51 52 56 57 06 B8 00 F0 8E C0 26 A0 FE |PSQRVW. ∫.≡ÃL&â.
00000170 FF BA 3A 01 E8 86 FF 3C FF 74 23 3C FE 74 25 3C |: .φâ < t#<.t%<
00000180 FB 74 21 3C FC 74 23 3C FA 74 25 3C FC 74 27 3C √t!<^t#<.t%<^t!<
00000190 F8 74 29 3C FD 74 2B 3C F9 74 2D EB 31 90 BA 44 °t)<^t+<.t-δ1É|D
000001A0 01 EB 38 90 BA 49 01 EB 32 90 BA 51 01 EB 2C 90 .δ8É|I.δ2É|Q.δ.É
000001B0 BA 56 01 EB 26 90 BA 65 01 EB 20 90 BA 7A 01 EB |V.δ&É|e.δ É|z.δ
000001C0 1A 90 BA 89 01 EB 14 90 BA 90 01 EB 0E 90 E8 E0 .É|ë.δ.É|É.δ.Éφα
000001D0 FE BB A1 01 88 07 88 67 01 8B D3 E8 1F FF 5F 5E ∫í.è.ég.ïlφ. _^
000001E0 5A 59 5B 58 07 C3 E8 78 FF E8 18 FF 32 C0 B4 4C ZY[X. |φx φ. 2 ∫L
000001F0 CD 21 + =!|

```

Изображение 4 - шестнадцатеричное представление модуля .COM



Изображение 5 - шестнадцатеричное представление "плохого" модуля .EXE



Изображение 6 - шестнадцатеричное представление "хорошего" модуля .EXE

Шаг 5. Отличие форматов файлов COM и EXE модулей

- 1) Какова структура файла COM? С какого адреса располагается код?

Ответ: COM содержит данные и машинные команды. Код располагается в начале 0h, но при загрузке модуля смещается на 100h.

- 2) Какова структура “плохого” EXE? С какого адреса располагается код? Что располагается с адреса 0?

Ответ: В плохом EXE данные и код содержатся в одном сегменте. Код располагается с 300h. С адреса 0 идет таблица настроек.

- 3) Какова структура “хорошего” EXE? Чем он отличается от файла “плохого” EXE?

Ответ: В хорошем EXE стек и данные разделены по сегментам. В отличие от плохого EXE таблица настроек расположена с 0 адреса, а код начинается с 400h.

Шаг5. Загрузка модуля .COM в основную память

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

Ответ: определяется сегментный участок, с достаточным местом для загрузки программы. Создается блок памяти для PSP и программы, затем подгружается COM-файл. Код располагается с 100h.

- 2) Что располагается с адреса 0?

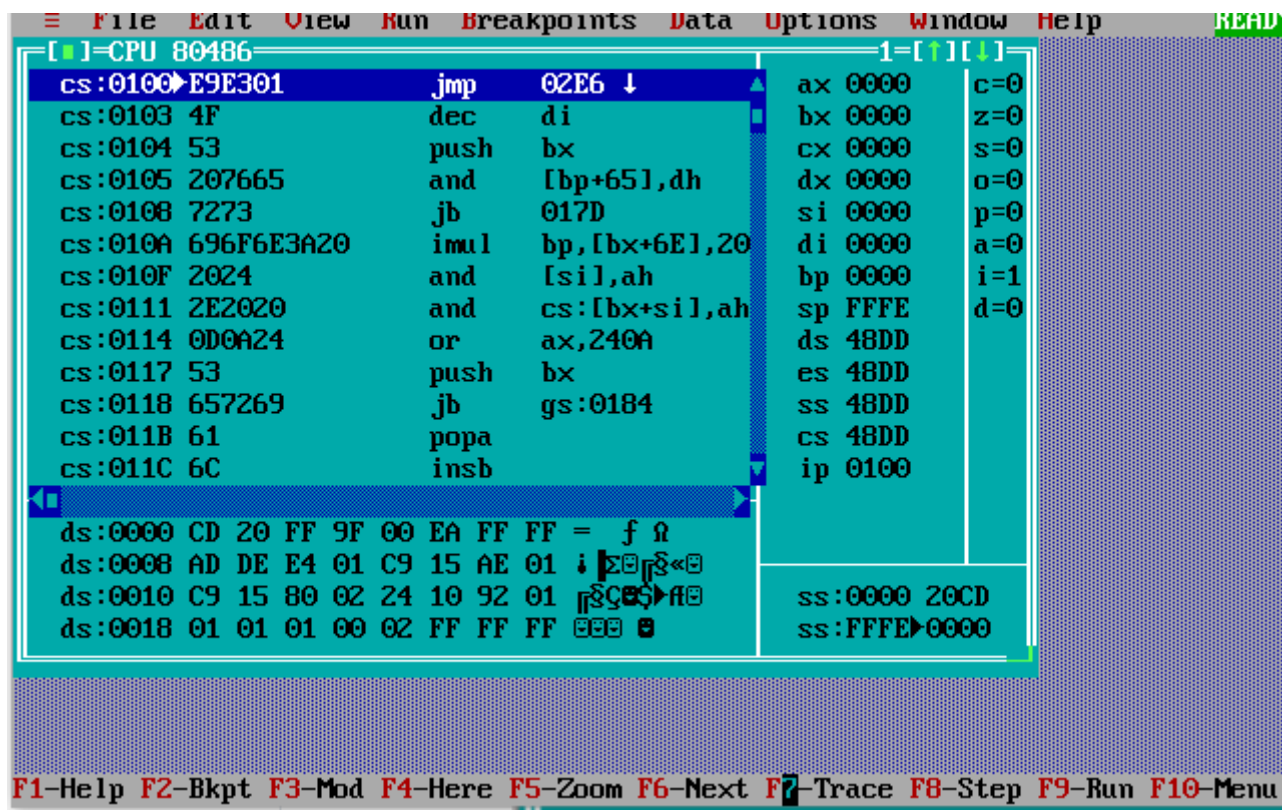
Ответ: PSP

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

Ответ: ds, es, ss, cs имеют значение 48DD и указывают на PSP. SP имеет значение FFFE и указывает на конец сегмента.

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

Ответ: Стек определяется автоматически. Стек занимает всю область. Адреса расположены от 0h до FFFEh.



Изображение 7 - Загрузка модуля COM

Шаг 6. Загрузка “хорошего” EXE модуля в память

1) Как загружается “хороший” EXE? Какие значения имеют сегментные регистры?

Ответ: .EXE загружается по смещению 100h от PSP, считывает информацию заголовка и осуществляет перемещение адресов сегментов. DS – 48DD, ES – 48DD, SS – 48DD, CS – 4918.

2) На что указывают регистры DS и ES?

Ответ: PSP

3) Как определяется стек?

Ответ: с помощью директивы .stack

4) Как определяется точка входа?

Ответ: с помощью директивы END

Заключение.

В ходе лабораторной работы были исследованы загрузочные модули .COM и .EXE, их структура и загрузка в основную память.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab1.asm

```
; .COM and .EXE differences in structures
; 15.02.2021
; Nistratov Dmitry

; Шаблон текста программы на ассемблере для модуля типа .COM
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START:  JMP BEGIN
; Данные
VERSION DB "OS version: $"; 12
DOT_VERSION DB ". ", 0DH, 0AH, '$' ; 1
SERIAL DB "Serial number: ", 0DH, 0AH, '$' ; 15 + 6 - 2
OEM DB "OEM: ", 0DH, 0AH, '$'; 5

PC_TYPE_MSG DB "PC TYPE: $"
PC_TYPE_1 DB "PC", 0DH, 0AH, '$'
PC_TYPE_2 DB "PC/XT", 0DH, 0AH, '$'
PC_TYPE_3 DB "AT", 0DH, 0AH, '$'
PC_TYPE_4 DB "PS2 model 30", 0DH, 0AH, '$'
PC_TYPE_5 DB "PS2 model 50 or 60", 0DH, 0AH, '$'
PC_TYPE_6 DB "PS2 model 80", 0DH, 0AH, '$'
PC_TYPE_7 DB "PCjr", 0DH, 0AH, '$'
PC_TYPE_8 DB "PC Convertible", 0DH, 0AH, '$'
PC_TYPE_9 DB " ", 0DH, 0AH, '$'
; 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 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 цифра
    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
;-----
; КОД
WRITE PROC NEAR
        push ax
        mov ah, 09h
        int 21h
        pop ax
        ret
WRITE ENDP

OS_TYPE PROC NEAR
        mov ah, 30h
        int 21H

        push ax
        push bx
        push cx
        push dx
        push si
        push di

```

```

    mov si, offset VERSION
    add si, 12
    call BYTE_TO_DEC
    mov dx, offset VERSION
    call WRITE

    mov si, offset DOT_VERSION
    add si, 1
    mov al, ah
    call BYTE_TO_DEC
    mov dx, offset DOT_VERSION
    call WRITE

    mov si, offset OEM
    add si, 7
    mov al, bh
    call BYTE_TO_DEC
    mov dx, offset OEM
    call WRITE

    mov di, offset SERIAL
    add di, 20
    mov al, bl
    call BYTE_TO_HEX
    mov ax, cx
    call WRD_TO_HEX
    sub di, 2
    mov [di], ax
    mov dx, offset SERIAL
    call WRITE

    pop di
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    ret
OS_TYPE ENDP

PC_TYPE PROC NEAR
    push ax
    push bx
    push cx
    push dx
    push si
    push di
    push es

    mov ax, 0f000h
    mov es, ax
    mov al, es:[0ffffh]
    mov dx, offset PC_TYPE_MSG
    call WRITE
    cmp al, 0ffh
    jz pc_1
    cmp al, 0feh

```

```

        jz pc_2
        cmp al, 0fbh
        jz pc_2
        cmp al, 0fch
        jz pc_3
        cmp al, 0fah
        jz pc_4
        cmp al, 0fch
        jz pc_5
        cmp al, 0f8h
        jz pc_6
        cmp al, 0fdh
        jz pc_7
        cmp al, 0f9h
        jz pc_8
        jmp pc_unknown

pc_1:
        mov dx, offset PC_TYPE_1
        jmp print

pc_2:
        mov dx, offset PC_TYPE_2
        jmp print

pc_3:
        mov dx, offset PC_TYPE_3
        jmp print

pc_4:
        mov dx, offset PC_TYPE_4
        jmp print

pc_5:
        mov dx, offset PC_TYPE_5
        jmp print

pc_6:
        mov dx, offset PC_TYPE_6
        jmp print

pc_7:
        mov dx, offset PC_TYPE_7
        jmp print

pc_8:
        mov dx, offset PC_TYPE_8
        jmp print

pc_unknown:
        call BYTE_TO_HEX
        mov bx, offset PC_TYPE_9
        mov [bx], al
        mov [bx+1], ah
        mov dx, bx

print:
        call WRITE

        pop di
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
        pop es

```

```

        ret
PC_TYPE ENDP
BEGIN:
        call PC_TYPE
        call OS_TYPE

; Выход в DOS
        xor AL,AL
        mov AH,4Ch
        int 21H
TESTPC ENDS
        END START ; Конец модуля, START - точка входа

```

Название файла: lab1_exe.asm

```

; .COM and .EXE differences in structures
; 15.02.2021
; Nistratov Dmitry

STACKSG SEGMENT STACK
        DW 256 DUP(?)

STACKSG ENDS

DATASG SEGMENT
        VERSION DB "OS version: $"; 12
        DOT_VERSION DB ". ", 0DH, 0AH, '$' ; 1
        SERIAL DB "Serial number: ", 0DH, 0AH, '$' ; 15 + 6 - 2
        OEM DB "OEM: ", 0DH, 0AH, '$'; 5

        PC_TYPE_MSG DB "PC TYPE: $"
        PC_TYPE_1 DB "PC", 0DH, 0AH, '$'
        PC_TYPE_2 DB "PC/XT", 0DH, 0AH, '$'
        PC_TYPE_3 DB "AT", 0DH, 0AH, '$'
        PC_TYPE_4 DB "PS2 model 30", 0DH, 0AH, '$'
        PC_TYPE_5 DB "PS2 model 50 or 60", 0DH, 0AH, '$'
        PC_TYPE_6 DB "PS2 model 80", 0DH, 0AH, '$'
        PC_TYPE_7 DB "PCjr", 0DH, 0AH, '$'
        PC_TYPE_8 DB "PC Convertible", 0DH, 0AH, '$'
        PC_TYPE_9 DB " ", 0DH, 0AH, '$'

DATASG ENDS

CODESG SEGMENT
        ASSUME SS:STACKSG, DS:DATASG, CS:CODESG

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 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 цифра
    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_1
    or AL,30h
    mov [SI],AL
end_1: pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP
;-----
; КОД
WRITE PROC NEAR
    push ax
    mov ah, 09h
    int 21h
    pop ax

```

```

        ret
WRITE ENDP

OS_TYPE PROC NEAR
    mov ah, 30h
    int 21h

    push ax
    push bx
    push cx
    push dx
    push si
    push di

    mov si, offset VERSION
    add si, 12
    call BYTE_TO_DEC
    mov dx, offset VERSION
    call WRITE

    mov si, offset DOT_VERSION
    add si, 1
    mov al, ah
    call BYTE_TO_DEC
    mov dx, offset DOT_VERSION
    call WRITE

    mov si, offset OEM
    add si, 7
    mov al, bh
    call BYTE_TO_DEC
    mov dx, offset OEM
    call WRITE

    mov di, offset SERIAL
    add di, 20
    mov al, bl
    call BYTE_TO_HEX
    mov ax, cx
    call WRD_TO_HEX
    sub di, 2
    mov [di], ax
    mov dx, offset SERIAL
    call WRITE

    pop di
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    ret
OS_TYPE ENDP

PC_TYPE PROC NEAR
    push ax
    push bx
    push cx

```

```

push dx
push si
push di
push es

mov ax, 0f000h
mov es, ax
mov al, es:[0ffffh]
mov dx, offset PC_TYPE_MSG
call WRITE
cmp al, 0ffh
jz pc_1
cmp al, 0feh
jz pc_2
cmp al, 0fbh
jz pc_2
cmp al, 0fch
jz pc_3
cmp al, 0fah
jz pc_4
cmp al, 0fch
jz pc_5
cmp al, 0f8h
jz pc_6
cmp al, 0fdh
jz pc_7
cmp al, 0f9h
jz pc_8
jmp pc_unknown

pc_1:
    mov dx, offset PC_TYPE_1
    jmp print
pc_2:
    mov dx, offset PC_TYPE_2
    jmp print
pc_3:
    mov dx, offset PC_TYPE_3
    jmp print
pc_4:
    mov dx, offset PC_TYPE_4
    jmp print
pc_5:
    mov dx, offset PC_TYPE_5
    jmp print
pc_6:
    mov dx, offset PC_TYPE_6
    jmp print
pc_7:
    mov dx, offset PC_TYPE_7
    jmp print
pc_8:
    mov dx, offset PC_TYPE_8
    jmp print
pc_unknown:
    call BYTE_TO_HEX
    mov bx, offset PC_TYPE_9
    mov [bx], al

```

```

        mov [bx+1], ah
        mov dx, bx

print:
        call WRITE

        pop di
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
        pop es
        ret
PC_TYPE ENDP

MAIN PROC FAR
        mov ax,DATASG
        mov ds,ax

        call PC_TYPE
        call OS_TYPE

; Выход в DOS
        xor AL,AL
        mov AH,4Ch
        int 21H
MAIN ENDP
CODESG ENDS
END MAIN

```