МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по практической работе №1

по дисциплине «Операционные системы»

Тема: Исследование структур загрузочных модулей

Студент гр. 8381 _____ Киреев К.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование различий в структурах исходных текстов модулей типов .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 кода и типа компьютера представлено в таблице 1:

Таблица 1 – Идентификация типа компьютера

Код	Тип компьютера
FF	Оригинальный IBM PC
FE	XT, Portable PC
FD	PCjr
FC	AT
FB	XT с памятью 640 К на мат. плате
FA	PS/2 модель 25 или 30
F9	Convertible PC
F8	PS/2 модели 55SX, 70, 80
9A	Compaq XT, Compaq Plus
30	Sperry PC
2D	Compaq PC, Compaq Deskpro

Для определения версии MS DOS следует воспользоваться функцией 30H прерывания 21H. Входным параметром является номер функции в AH:

```
MOV AH, 30h
INT 21h
```

Выходными параметрами являются:

AL – номер основной версии;

AH – номер модификации;

BH – серийный номер OEM (Original Equipment Manufacturer);

BL:CX – 24-битовый серийный номер пользователя.

Постановка задачи.

Требуется реализовать текст исходного .COM модуля, который определяет тип PC и версию системы. Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип PC и выводить строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код переводиться в символьную строку, содержащую запись шестнадцатеричного числа и выводиться на экран в виде соответствующего сообщения.

Затем определяется версия системы. Ассемблерная программа должна по значениям регистров AL и AH формировать текстовую строку в формате xx.yy, где xx - номер основной версии, а yy - номер модификации в десятичной системе счисления, формировать строки с серийным номером OEM (Original Equipment Manufacturer) и серийным номером пользователя. Полученные строки выводятся на экран. Далее производится отладка полученного исходного модуля. Результатом выполнения этого действия будет «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля. Затем нужно написать текст «хорошего» .EXE модуля, который выполняет те же функции, что и модуль .COM, далее его построить, отладить и сравнить исходные тексты для .COM и .EXE модулей, ответить на контрольные вопросы.

Выполнение работы.

Написан текст исходного .COM модуля, который определяет тип PC и версию системы. Полученный исходный модуль был отлажен. В результате был получен «плохой» .EXE модуль и построен «хороший» .COM модуль с помощью программы EXE2BIN.COM. Во время линковки получено предупреждение об отсутствии сегмента стека, представленное на рис. 1.

```
S:\>masm lab_com.asm
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988.  All rights reserved.

Object filename [lab_com.OBJ]:
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  49960 + 455253 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors

S:\>link lab_com.obj

Microsoft (R) Overlay Linker  Version 3.64
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Run File [LAB_COM.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment
```

Рисунок 1 – Предупреждение об отсутствии стека

Запуск «хорошего» .COM модуля представлен на рис. 2.

```
S:\>exe2bin lab_com.exe lab_com.com

S:\>lab_com.com
PC type: AT or PS2 .50/60
MSDOS version: 5.0
OEM serial number: 0
User serial number: 000000
```

Рисунок 2 – «Хороший» .COM модуль

Запуск «плохого» .EXE модуля представлен на рис. 3.

```
S:\>lab_com.exe
<■t▲<ⁿt╥╢
 ≈±Ç╨0ê¶N3╥=
 s±=  ∨♦♀0ê♦XZY├P┤      =!X├PRVWì▬˜☻ŏ€ ┐ ≡Ä└&á■ < t↑<■t▲<ⁿtt*<°t0<²t6<·t<ì520ŏ┬
ŏ=Éì▬7☻ŏ┐ ŏ3Éì▬?☻ŏ« ŏ)Éì▬R☻ŏñ ŏ▼Éì▬\☻ŏÜ ŏ§Éì▬f☻ŏÉ ŏδÉì▬m☻ŏå ŏ☺É┤0=!ì6ê☻â├*ŏ0 â├♥
å─ŏG ì▬ê☻ŏf è║ì6ƒ☻â├‼ŏ4 ì▬ƒ☻ŏS è├ì6╜☻ŏ·■ëD¶â├↓ï■ï┴ŏ²■ì▬╜☻ŏ6 _^ZX┤L=!├PC
Éì▬7☻ŏ┐ ŏ3Éì▬?☻ŏ« ŏ0Éì▬R☻ŏñ ŏ▼Éì▬\☻ŏÜ ŏ§Éì▬f☻ŏÉ ŏδÉì▬m☻ŏå ŏ☺É┤0=!ì6ê☻â├*ŏ0 â├♥å─
ŏG ì▬ê☻ŏf è║ì6ƒ☻â├‼ŏ4 ì▬ƒ☻ŏS è├ì6╜☻ŏ·■ëD¶â├↓ï■ï┴ŏ²■ì▬╜☻ŏ6 _^ZX┤L=!├PC
Éì▬\☻ŏÜ ŏ§Éì▬f☻ŏÉ ŏδ000000å ŏ☺É┤0=!ì6ê☻â├*ŏ0 â├♥å─ŏG ì▬ê☻ŏf è║ì6ƒ☻â├‼ŏ4 ì▬ƒ☻ŏS è
├ì6╜☻ŏ·■ëD¶â├↓ï■ï┴ŏ²■ì▬╜☻ŏ6 _^ZX┤L=!├PC
```

Рисунок 3 – «Плохой» .EXE модуль

Написан текст исходного .EXE модуля, который выполняет те же функции, что и предыдущий модуль. Был получен «хороший» .EXE модуль, запуск которого представлен на рис. 4.

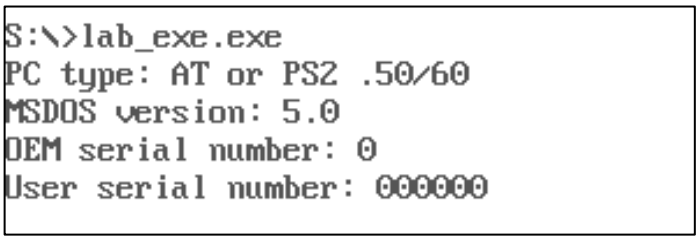

Рисунок 4 – «Хороший» .EXE модуль

Отличия исходных текстов COM и EXE программ.

1. Сколько сегментов должна содержать COM-программа?

Программы типа .COM состоят из единственного сегмента, в котором размещаются программные коды, данные и стек.

2. EXE-программа?

В программах типа .EXE для собственно программы, данных и стека предусматривают отдельные сегменты.

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

Директива ORG 100h, которая резервирует 256 байт для PSP. Заполнять PSP будет по-прежнему система, но место под него в начале сегмента должен отвести программист, код программы располагается только после этого блока.

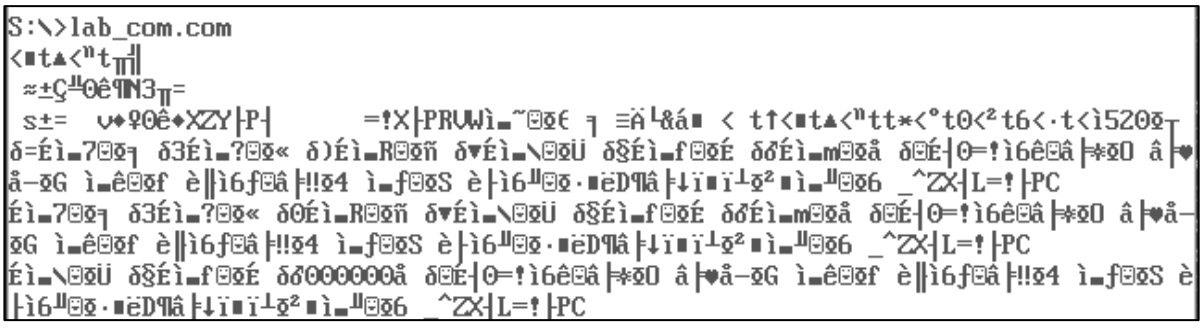

Рисунок 5 – .COM модуль без директивы 100h

Директива ASSUME, ставящая в соответствие адрес сегмента программы сегментам кода и данных.

```
lab_com.asm(11): error A2062: Missing or unreachable CS
lab_com.asm(15): error A2062: Missing or unreachable CS
lab_com.asm(28): error A2062: Missing or unreachable CS
lab_com.asm(48): error A2062: Missing or unreachable CS
lab_com.asm(56): error A2062: Missing or unreachable CS
lab_com.asm(68): error A2062: Missing or unreachable CS
lab_com.asm(75): error A2062: Missing or unreachable CS
lab_com.asm(83): error A2062: Missing or unreachable CS
lab_com.asm(105): error A2062: Missing or unreachable CS
lab_com.asm(105): error A2062: Missing or unreachable CS
lab_com.asm(105): error A2062: Missing or unreachable CS
lab_com.asm(105): error A2062: Missing or unreachable CS
lab_com.asm(105): error A2062: Missing or unreachable CS
lab_com.asm(105): error A2062: Missing or unreachable CS
lab_com.asm(105): error A2062: Missing or unreachable CS
lab_com.asm(107): error A2062: Missing or unreachable CS
lab_com.asm(120): error A2062: Missing or unreachable CS
lab_com.asm(128): error A2062: Missing or unreachable CS

  49960 + 455253 Bytes symbol space free

      0 Warning Errors
     20 Severe  Errors
```

Рисунок 6 – .COM модуль без директивы ASSUME

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

Так как в COM-программе все сегментные регистры определяются в момент запуска программы, а не в момент компиляции (ассемблирования), то невозможно использование, например, таких конструкций:

mov ax, DATA или mov ax, CODE

Нельзя использовать команды вида mov <регистр>, seg <имя сегмента>

Например mov ax, seg CODE

Отличия форматов файлов COM и EXE модулей.

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

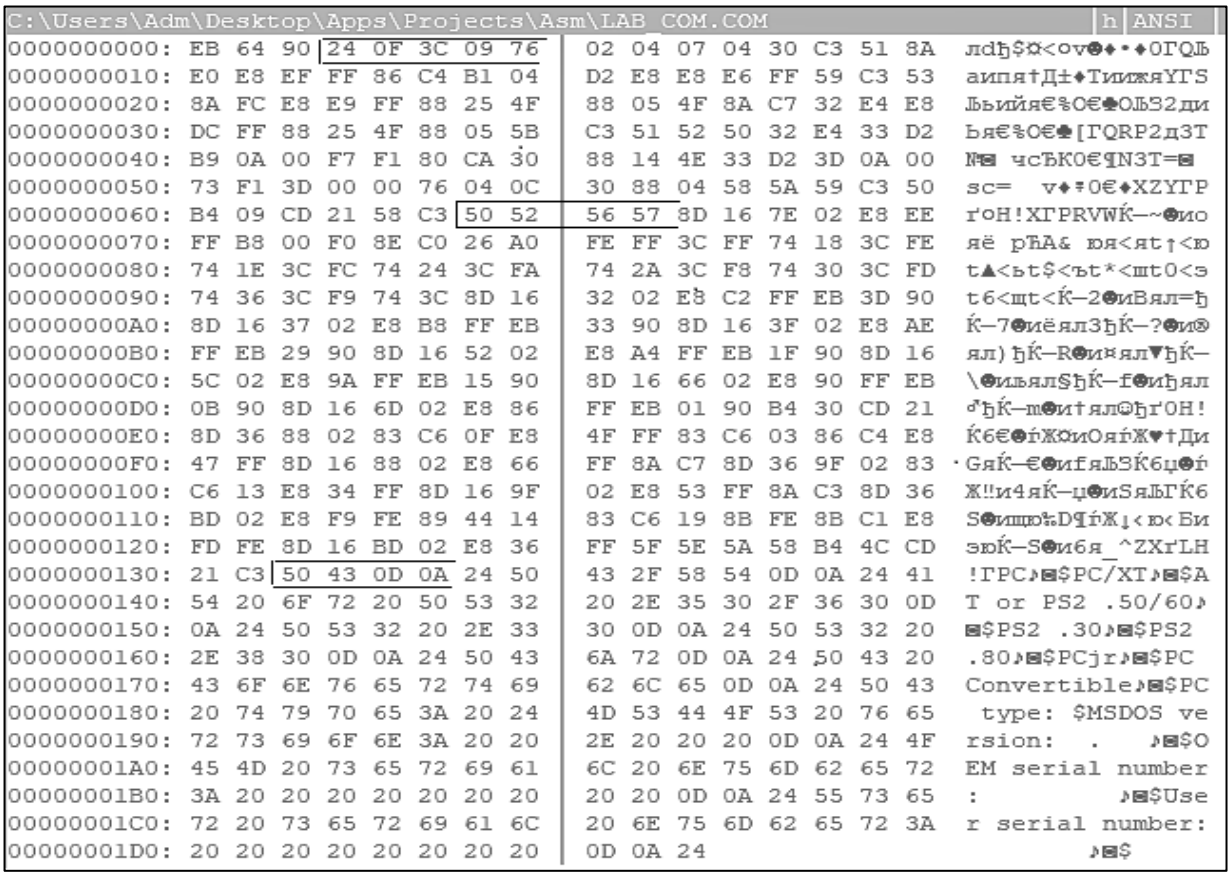

Рисунок 7 - .COM модуль в шестнадцатеричном виде

COM файл состоит из одного сегмента и, независимо от фактического размера программы, ей выделяется 64 Кбайт адресного пространства, Код начинается с адреса 0h, но при загрузке модуля устанавливается смещение в 100h.

Сначала идут байты, отвечающие за код. Например, коды команд, представленные на рис. 8. Данные начинаются на 130h, второй байт.

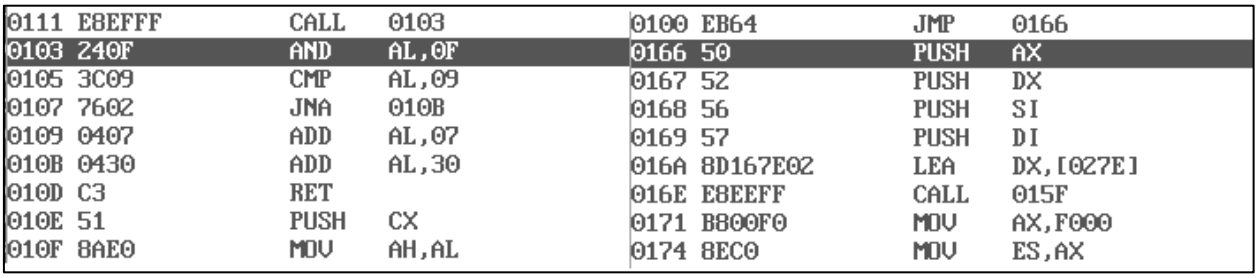

Рисунок 8 – команды .COM модуль

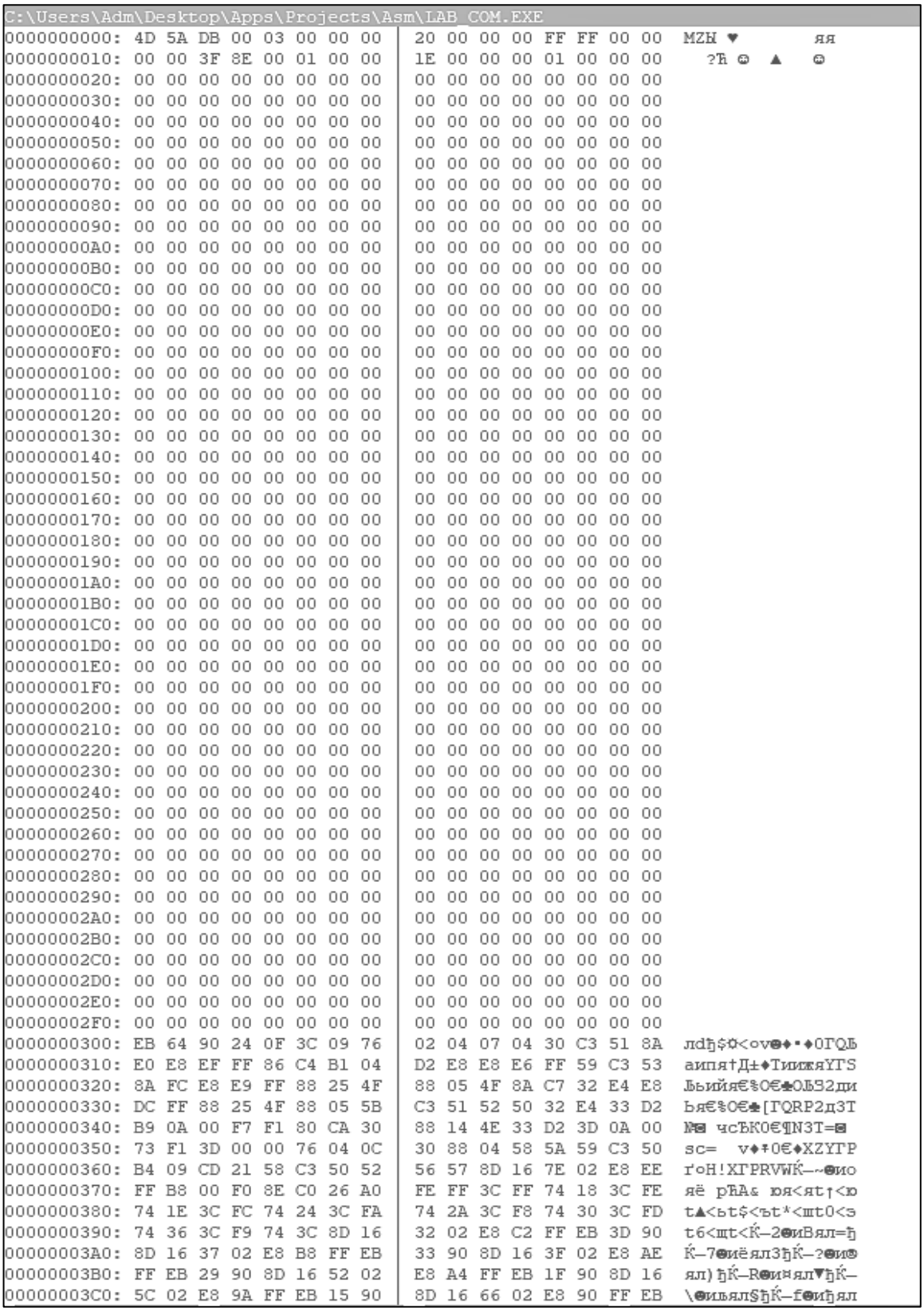

Рисунок 9 - «Плохой» .EXE модуль в шестнадцатеричном виде

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

В «плохом» EXE файле данные и код содержатся в одном сегменте. Код располагается с адреса 300h. С адреса 0h располагается таблица разметки. Также 100h резервируются командой ORG 100h.

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

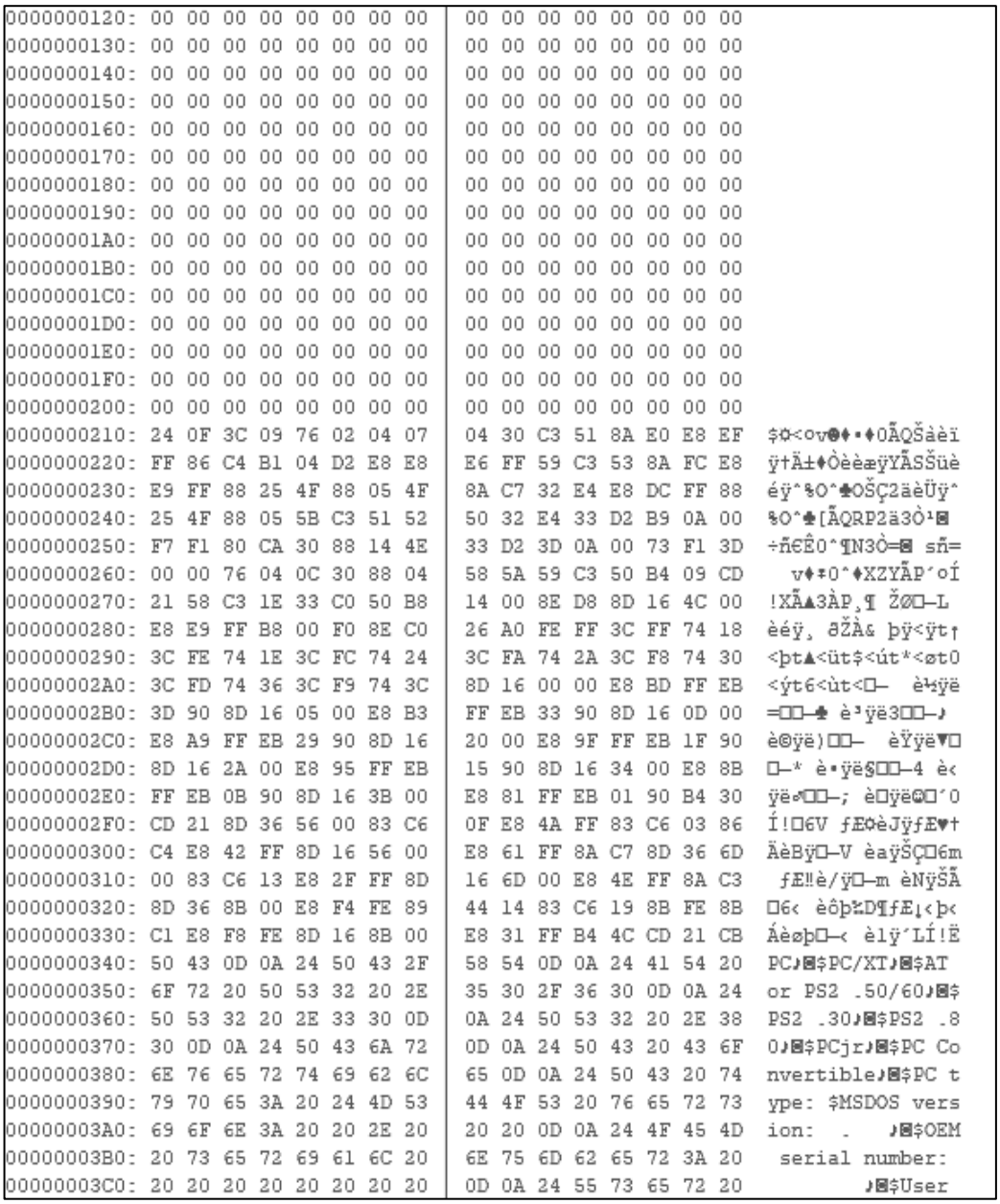

Рисунок 10 - «Хороший» .EXE модуль в шестнадцатеричном виде

В «хорошем» файле EXE содержится информация для загрузчика, сегмент стека, сегмент данных и сегмент кода. Код располагается с адреса 210h в отличие от 300h в «плохом» .EXE файле.

Загрузка COM модуля в основную память.

Рисунок 11 – Образ памяти программы типа .COM

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

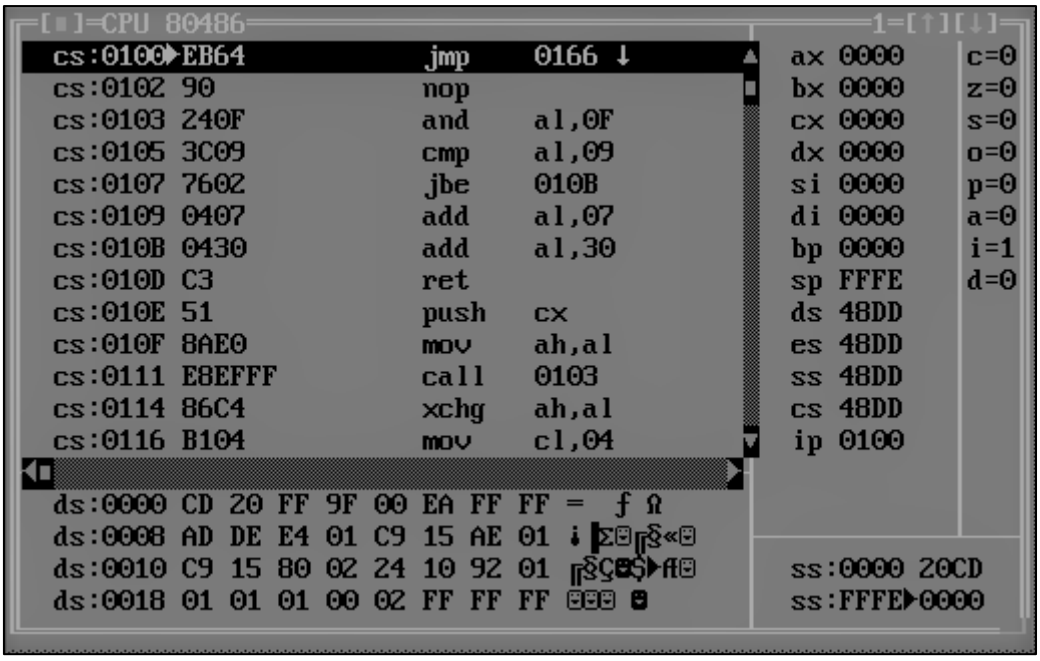

Рисунок 12 – Загрузка .COM модуля в память

После загрузки COM-программы в память сегментные регистры указывают на начало PSP. Код располагается с адреса 100h, IP = 0100h.

2. **Что располагается с 0 адреса?**

С нулевого адреса располагается адрес начала PSP.

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

Сегментные регистры имеют значение 48DDh. Они указывают на начало PSP, что продемонстрировано на рис. 11.

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

Стек определяется автоматически, указатель стека устанавливается на конец сегмента. Если для программы размер сегмента в 64КБ является достаточным, то DOS устанавливает в регистре SP адрес конца сегмента – FFFEh, что также можно увидеть на рис. 11. Адреса расположены в диапазоне от FFFEh до 0000h.

Загрузка «хорошего» EXE модуля в основную память.

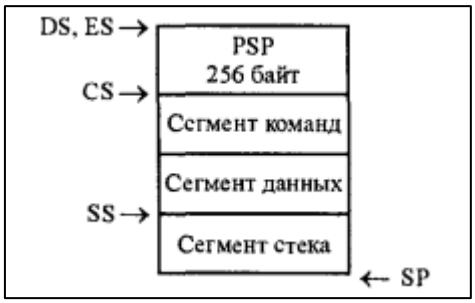

Рисунок 13 – Образ памяти программы типа .EXE

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

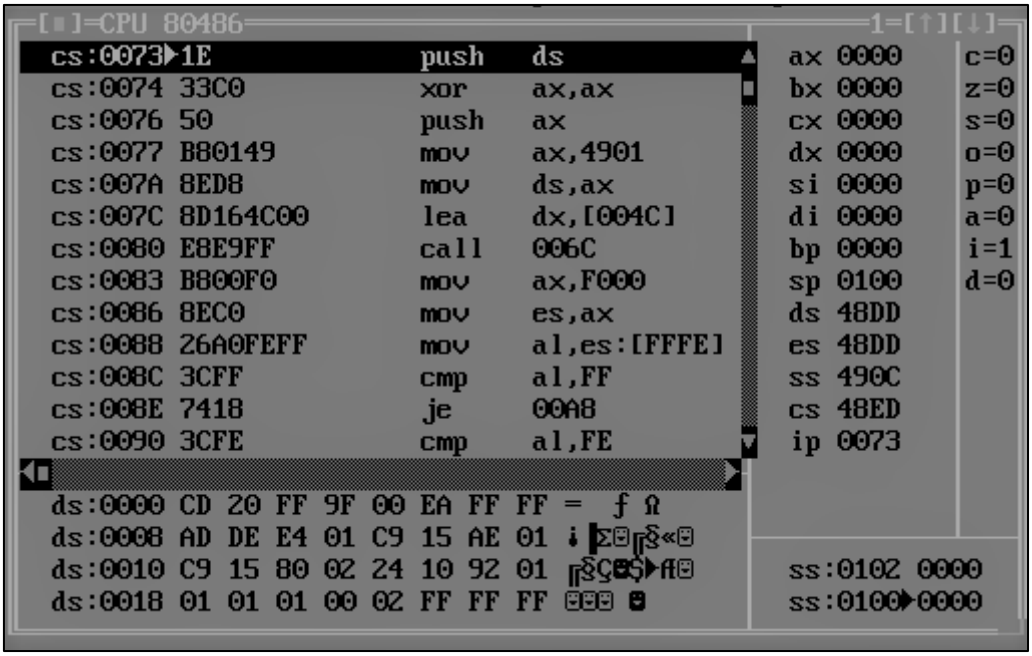

Рисунок 14 – Загрузка «хорошего» .EXE модуля в память

Для PSP и программы выделяется блок памяти. Система, загрузив программу в память, инициализирует сегментные регистры, так что регистры DS и ES указывают на начало PSP(48DDh), CS - на начало сегмента команд(48EDh), а SS - на начало сегмента стека(490Ch). В указатель стека SP кладется смещение конца сегмента стека, а в указатель команд IP загружается смещение точки входа в программу. Таким образом, после загрузки программы в память адресуемыми оказываются все сегменты, кроме сегмента (или сегментов) данных.

2. На что указывают регистры DS и ES?

DS и ES указывают на начало PSP. Инициализация регистра DS в первых строках программы позволяет сделать адресуемым и сегмент данных.

3. Как определяется стек?

В исходном коде модуля стек определяется при помощи директивы .STACK, а при исполнении в регистр SS записывается адрес начала сегмента стека, а в указатель стека SP - смещение конца сегмента стека.

4. Как определяется точка входа?

При загрузке программы в указатель команд IP загружается смещение точки входа в программу (которое берется из операнда директивы END).

Вывод.

В ходе работы было проведено исследование различий в структурах исходных текстов модулей .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. LAB_COM.COM

```
lab segment
assume cs:lab, ds:lab, es:nothing, ss:nothing
org 100h
main: jmp processing

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 16 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 цифра
                    ret
byte_to_hex         endp

word_to_hex         proc near
                    ;перевод в 16 сс 16 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
                    xor ah, ah
                    call byte_to_hex
                    mov [di], ah
                    dec di
                    mov [di], al
                    pop bx
                    ret
word_to_hex         endp
```

```
byte_to_dec         proc near
                    ;перевод в 10 сс, si - адрес поля младшей цифры
                    push cx
                    push dx
                    push ax
                    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  ax, 00h
                    jbe  end_l
                    or al, 30h
                    mov  [si], al
end_l:
                    pop  ax
                    pop  dx
                    pop  cx
                    ret
byte_to_dec         endp

print proc near
                    push ax
                    mov ah, 09h
                    int 21h
                    pop ax
                    ret
print endp

processing:
                    push ax
                    push dx
                    push si
                    push di
                    lea dx, pc_arg
                    call print

                    mov ax, 0F000h ;указывает ES на ПЗУ
                    mov es, ax
                    mov al, es:[0FFFEh]

                    irpc case, FECA8D9
                    cmp al, 0F&case&h
                    je type_&case&
```

```
                    endm

                    irpc met, FECA8D9
                    type_&met&:
                        lea dx, pc_&met&
                        call print
                        jmp OS
                    endm

OS:
                    mov ah, 30h
                    int 21h ;al - осноная версия, ah - модификация,
bh - OEM, bl:cx - номер пользователя

                    lea si, os_arg
                    add si, 15
                    call byte_to_dec ;пишется основная версия
                    add  si, 3
                    xchg al, ah
                    call byte_to_dec ;пишется модификация
                    lea dx, os_arg
                    call print

OEM:
                    mov al, bh
                    lea si, oem_arg
                    add si, 19
                    call byte_to_dec
                    lea dx, oem_arg
                    call print

serial_number:
                    mov  al, bl
                    lea si, user_arg
                    call byte_to_hex
                    mov [si+20], ax
                    add  si, 25
                    mov di, si ;ax - число, di - адрес последнего
символа для word_to_hex
                    mov ax, cx
                    call word_to_hex
                    lea dx, user_arg
                    call print

                    pop di
                    pop si
                    pop dx
                    pop  ax
                    mov ah, 4ch
                    int  21h
                    ret
```

```
pc_F db             'PC', 0dh, 0ah, '$'
pc_E db 'PC/XT', 0dh, 0ah, '$'
pc_C db             'AT or PS2 .50/60', 0dh, 0ah, '$'
pc_A db 'PS2 .30', 0dh, 0ah, '$'
pc_8 db 'PS2 .80', 0dh, 0ah, '$'
pc_D db 'PCjr', 0dh, 0ah, '$'
pc_9 db 'PC Convertible', 0dh, 0ah, '$'
pc_arg db 'PC type: ', '$'
os_arg db 'MSDOS version:  .   ', 0dh, 0ah, '$'
oem_arg             db 'OEM serial number:         ', 0dh, 0ah, '$'
user_arg db         'User serial number:        ', 0dh, 0ah, '$'

lab ends
end main
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. LAB_EXE.EXE

```
dosseg
.model small
.stack 100h
.data
pc_F db             'PC', 0dh, 0ah, '$'
pc_E db 'PC/XT', 0dh, 0ah, '$'
pc_C db             'AT or PS2 .50/60', 0dh, 0ah, '$'
pc_A db 'PS2 .30', 0dh, 0ah, '$'
pc_8 db 'PS2 .80', 0dh, 0ah, '$'
pc_D db 'PCjr', 0dh, 0ah, '$'
pc_9 db 'PC Convertible', 0dh, 0ah, '$'
pc_arg db 'PC type: ', '$'
os_arg db 'MSDOS version:  .   ', 0dh, 0ah, '$'
oem_arg             db 'OEM serial number:         ', 0dh, 0ah, '$'
user_arg db         'User serial number:        ', 0dh, 0ah, '$'
.code
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 16 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 цифра
                    ret
byte_to_hex         endp

word_to_hex         proc near
                    ;перевод в 16 сс 16 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
                    xor ah, ah
                    call byte_to_hex
                    mov [di], ah
                    dec di
                    mov [di], al
                    pop bx
                    ret
word_to_hex         endp

byte_to_dec         proc near
                    ;перевод в 10 сс, si - адрес поля младшей цифры
                    push cx
                    push dx
                    push ax
                    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 ax, 00h
                    jbe end_l
                    or al, 30h
                    mov [si], al
end_l:
                    pop ax
                    pop dx
                    pop cx
                    ret
byte_to_dec         endp

print proc near
                    push ax
                    mov ah, 09h
                    int 21h
                    pop ax
                    ret
print endp

processing proc far
                    push ds
                    xor ax, ax
                    push ax
```

```
                    mov ax, @data
                    mov ds, ax
                    lea dx, pc_arg
                    call print

                    mov ax, 0F000h ;указывает ES на ПЗУ
                    mov es, ax
                    mov al, es:[0FFFEh]

                    irpc case, FECA8D9
                    cmp al, 0F&case&h
                    je type_&case&
                    endm

                    irpc met, FECA8D9
                    type_&met&:
                        lea dx, pc_&met&
                        call print
                        jmp OS
                    endm

OS:
                    mov ah, 30h
                    int 21h ;al - осноная версия, ah - модификация,
bh - OEM, bl:cx - номер пользователя

                    lea si, os_arg
                    add si, 15
                    call byte_to_dec ;пишется основная версия
                    add  si, 3
                    xchg al, ah
                    call byte_to_dec ;пишется модификация
                    lea dx, os_arg
                    call print

OEM:
                    mov al, bh
                    lea si, oem_arg
                    add si, 19
                    call byte_to_dec
                    lea dx, oem_arg
                    call print

serial_number:
                    mov  al, bl
                    lea si, user_arg
                    call byte_to_hex
                    mov [si+20], ax
                    add  si, 25
                    mov  di,  si  ;ax  -  число,  di  -  адрес  последнего
символа для word_to_hex
```

```
                    mov ax, cx
                    call word_to_hex
                    lea dx, user_arg
                    call print

                    mov ah, 4ch
                    int  21h
                    ret
processing endp
end processing
```