МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Операционные системы»

Тема: Исследование структур загрузочных модулей.

Студент гр. 8382 _____ Терехов А.Е.

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.

Ход работы.

1. С помощью данных в методических указаниях функций был написан исходник для загрузочного модуля с расширением .COM. Программа должна выводить сведения о типе IBM PC, версии MS-DOS и серийном номере OEM и пользователя.

2. После трансляции с использованием MASM и линковки была получена программа с расширением .EXE. Программа работает некорректно, как это можно заметить на рис.1.

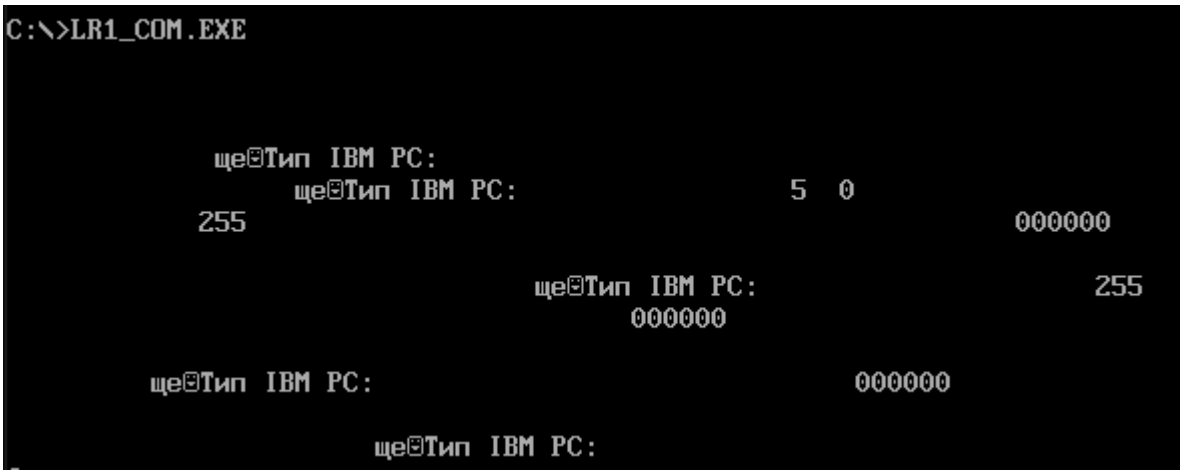

Рис.1. Запуск "плохого" .EXE файла.

3. С помощью утилиты EXE2BIN.EXE был получен "хороший" .COM модуль, работающий корректно. Результат работы программы представлен на рис.2.

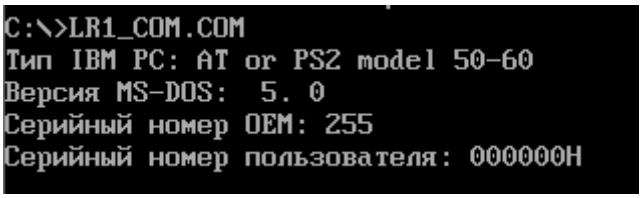

Рис.2. Запуск "хорошего" .COM файла.

4. Был написан исходный код для .EXE модуля, который должен выполнять те же действия, что и модуль на шаге 1.

5. После трансляции с использованием MASM и линковки была получена программа с расширением .EXE, работающая корректно, как это можно заметить на рис.3.

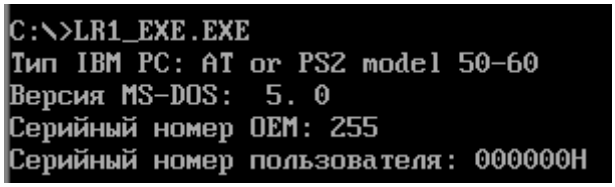

Рис.3. Запуск "хорошего" .EXE файла.

6. С представлениями файлов в шестнадцатеричном виде можно ознакомиться на рисунках 4-6.

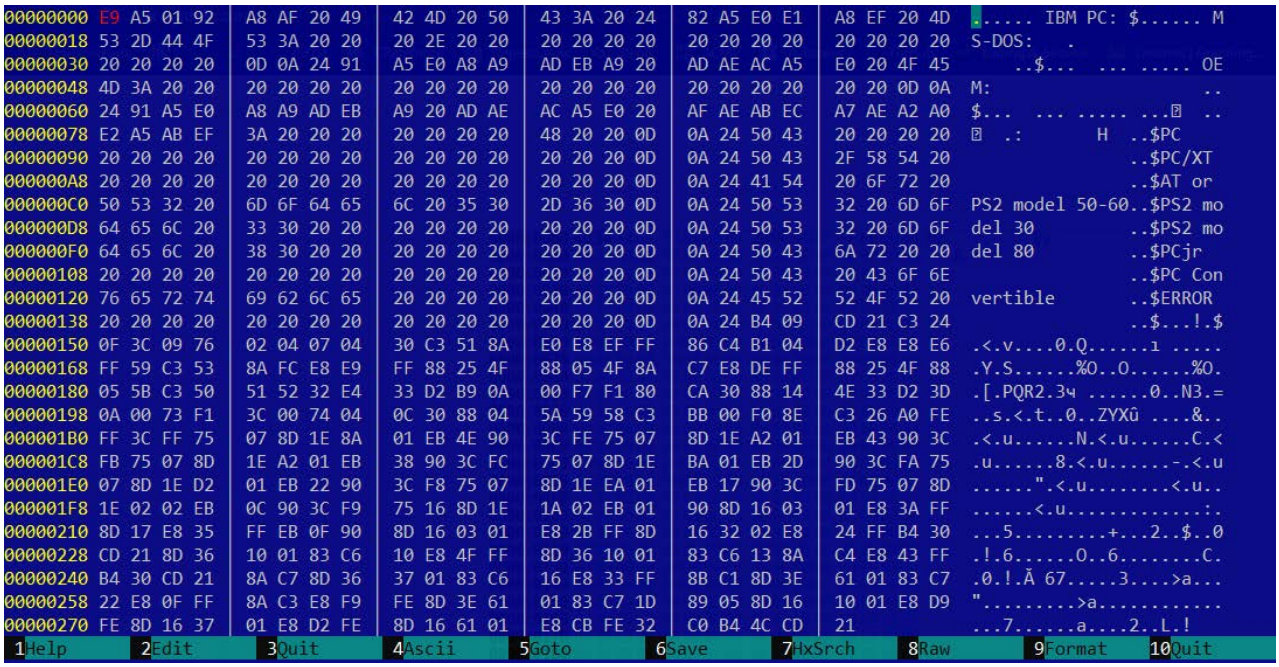

Рис.4. COM-файл в шестнадцатеричном виде.

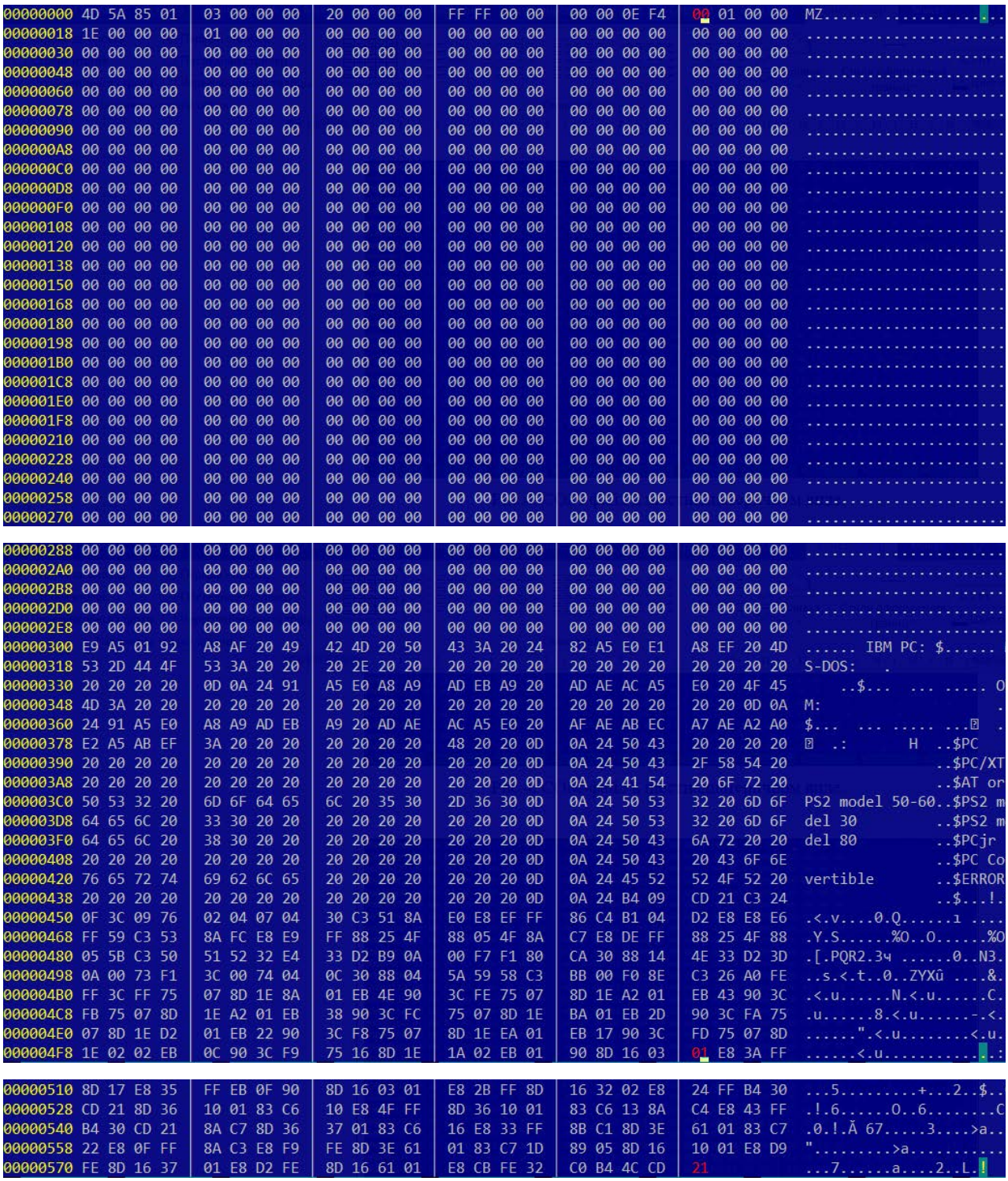

Рис.5. "Плохой" EXE-файл в шестнадцатеричном виде.

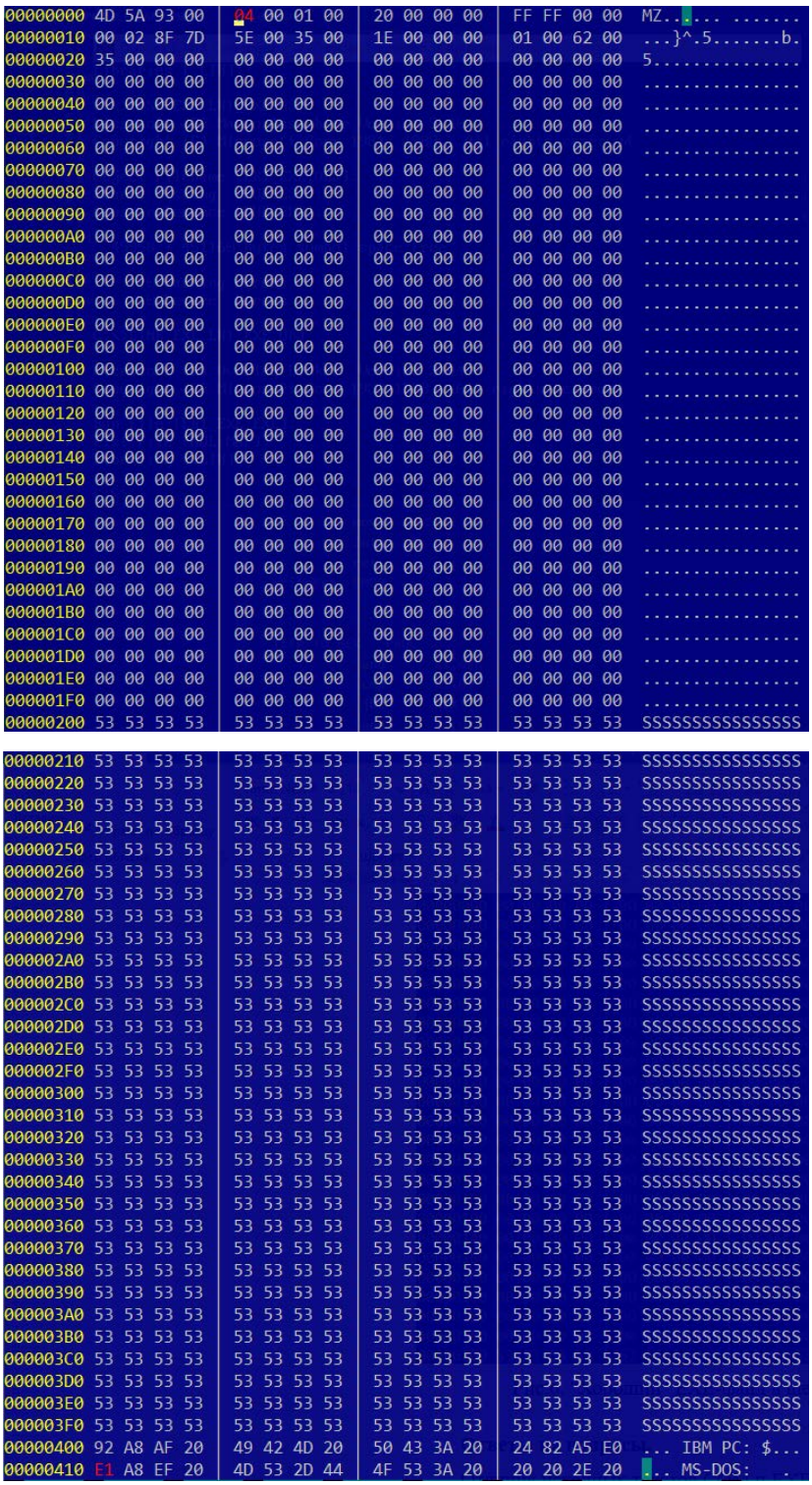

Рис.6 (начало). "Хороший" EXE-файл в шестнадцатеричном виде.

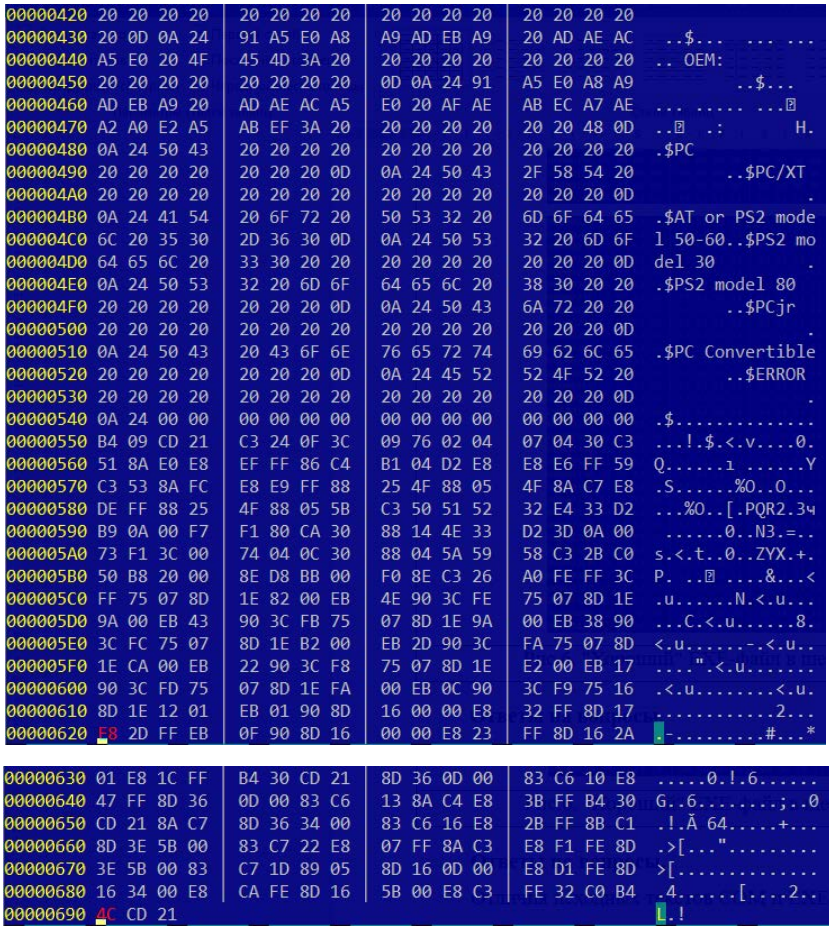

Рис.6 (конец). "Хороший" EXE-файл в шестнадцатеричном виде.

7. Запуск в отладчике TD COM-программы представлен на рисунке 7.

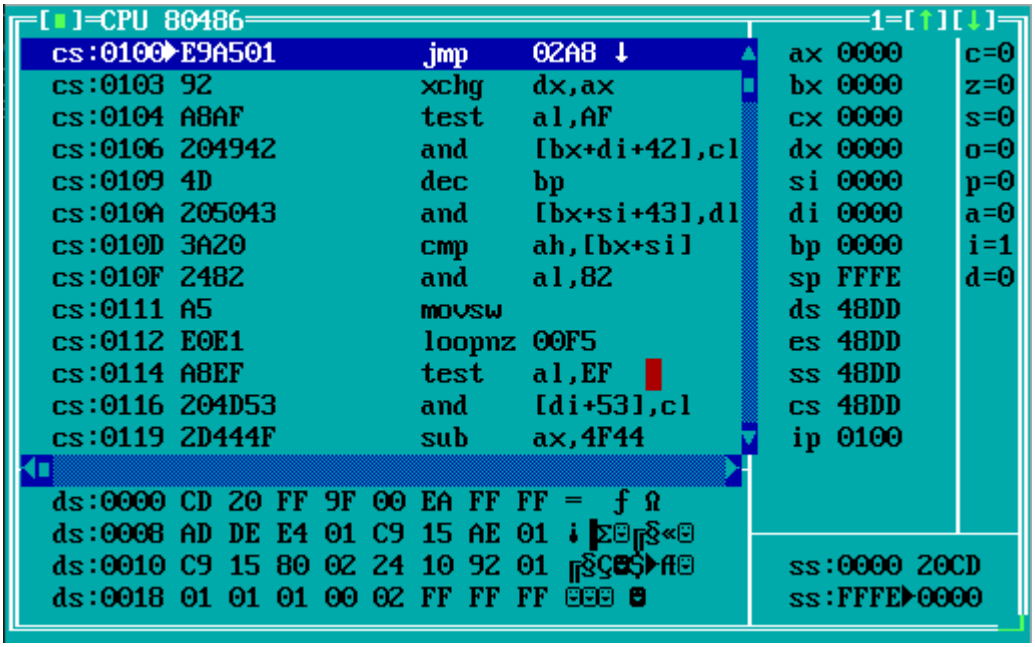

Рис.7. Запуск COM-программы в отладчике.

8. Запуск в отладчике TD EXE-программы представлен на рисунке 8.

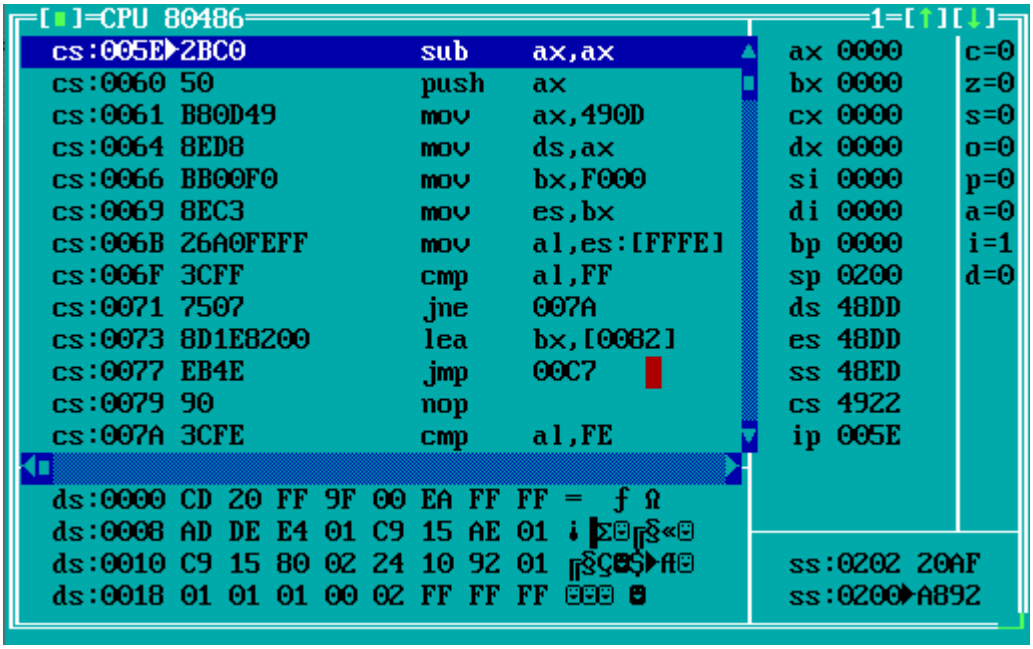

Рис.8. Запуск EXE-программы в отладчике.

Ответы на вопросы.

Отличия исходных текстов COM и EXE программ.

1) Сколько сегментов должна содержать COM-программа?

Один сегмент CODESEG.

2) EXE-программа?

Количество допустимых сегментов определяется используемой моделью памяти:

- small – один сегмент кода, один сегмент данных;
- compact – один сегмент кода, несколько сегментов данных;
- medium – несколько сегментов кода, один сегмент данных;
- large – несколько сегментов кода, несколько сегментов данных;
- huge – много сегментов кода, много сегментов данных.

А также сегмент стека.

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

Директива ASSUME, сообщающая ассемблеру информацию о

соответствии между сегментными регистрами и сегментами (сегментом в данном случае). Директива ORG, сообщающая компилятору о смещении адресации внутри кода.

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

В силу того, что в COM-программе сегментные регистры определяются в момент запуска, а не трансляции (компиляции) нельзя использовать команды, работающие с сегментами, команды, а также команды, размера больше 64Кб, или команды, работающие с 64-битными регистрами.

Отличия форматов файлов COM и EXE модулей.

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

В начале файла можно заметить данные, которые используются в программе, затем следует сам код. COM-программа содержит один сегмент, не превышающий 64Кб. Код располагается с начала программы, но при запуске ему всегда предшествует блок памяти для PSP длиной 100H байт.

2) Какова структура "плохого" EXE файла? С какого адреса располагается код? Что располагается с адреса 0?

"Плохой" EXE-файл содержит только один сегмент. В самом начале идет заголовок EXE-файла, в котором содержится полезная информация, такая как сигнатура, длина заголовка в 16-байтных параграфах, значение SP и IP при входе и т.д. Заголовок занимает адреса 00-1BH (28 байт). За заголовком следует таблица настроек адресов, которая занимает ровно столько места, сколько указано в заголовке. Сам код в данном EXE-файле начинается с адреса 300H.

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

Также, как и в "плохом" в "хорошем" в самом начале идет заголовок и таблица настроек адресов. "Хороший" EXE-файл содержит три сегмента: стека (в шестнадцатеричном виде представлен множеством символов S), данных и кода. Сам код в "хорошем" EXE-файле начинается с адреса 400H, так как в отличие от "плохого" в нем определен стек.

Загрузка COM модуля в основную память.

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

Система выделяет свободный сегмент, в первые 256 байт этого сегмента записывается PSP, непосредственно за ним загружается содержимое COM-файла без изменений, то есть с адреса 0100H, указатель стека (регистр SP) устанавливается на конец сегмента.

2) Что располагается с адреса 0?

Program Segment Prefix (PSP) – область памяти размером 256 (0100h) байт, предшествующая программе при ее загрузке.

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

Все сегментные регистры имеют значения 48DDH. Они указывают на начало выделенной системой памяти, то есть на PSP.

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

Стек расположен в конце выделенной памяти, сразу после содержимого COM-файла. В стек записывается 0000H (адрес возврата для команды ret).

Загрузка "хорошего" EXE модуля в основную память.

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

DS и ES устанавливаются на начало сегмента PSP, SS – на начало сегмента стека, CS – на начало сегмента команд. В IP загружается адрес первой команды.

2) На что указывают регистры DS и ES?

DS и ES указывают на начало префикса программного сегмента (PSP).

3) Как определяется стек?

Стек определяется в коде следующим образом:

<имя сегмента> SEGMENT STACK

DW <количество выделяемой памяти> DUP(<значение по

умолчанию>)

<имя сегмента> ENDS

ASSUME SS:AStack

4) Как определяется точка входа?

Точка входа – начальное значение IP – вычисляется с помощью директивы END <метка>.

Вывод.

В ходе выполнения работы были написаны два варианта программы на языке ассемблера. Программа выводит на экран сообщения, содержащие информацию о некоторых свойствах системы. В процессе были получены три загрузочных модуля: COM из исходника для получения COM-файла, EXE из того же исходника, работающий некорректно, и EXE из исходника для EXE. Были выявлены различия в исходниках, структурах загрузочных модулей, порядке запуска этих двух вариантов.

ПРИЛОЖЕНИЕ А

Исходный код COM-программы.

```
CODESEG SEGMENT
        ASSUME cs:CODESEG, ds:CODESEG, es:NOTHING, ss:NOTHING
        ORG 100H
START: jmp BEGIN
; ===Данные===
TYPE_PC         db'Тип IBM PC: ','$'
MSDOS_VERSION   db'Версия MS-DOS:   .                    ',0DH,0AH,'$'
OEM_NUMBER      db'Серийный номер OEM:                   ',0DH,0AH,'$'
USER_NUMBER     db'Серийный номер пользователя:        H',0DH,0AH,'$'
CASE_FF         db'PC                                    ',0DH,0AH,'$'
CASE_FE_FB      db'PC/XT                                 ',0DH,0AH,'$'
CASE_FC         db'AT or PS2 model 50-60                 ',0DH,0AH,'$'
CASE_FA         db'PS2 model 30                          ',0DH,0AH,'$'
CASE_F8         db'PS2 model 80                          ',0DH,0AH,'$'
CASE_FD         db'PCjr                                  ',0DH,0AH,'$'
CASE_F9         db'PC Convertible                        ',0DH,0AH,'$'
ERROR_MSG       db'ERROR                                 ',0DH,0AH,'$'
; Процедура печати строки
WriteMsg PROC near
        mov ah,09h
        int 21h
        ret
WriteMsg ENDP
;--------------------------------------------------
TETR_TO_HEX PROC near
        and al,0Fh
        cmp al,09
        jbe NEXT
        add al,07
NEXT:   add al,30h; код нуля
        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 байта в 10с/с, si - адрес поля младшей цифры
; al содержит исходный байт
        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
;--------------------------------------------------
BEGIN:
;получим тип IBM PC
        mov bx, 0F000h
        mov es, bx
        mov al, es:[0FFFEh]
        cmp al, 00FFh
        jne FE
        lea bx, CASE_FF
        jmp CONCAT
FE:
        cmp al, 00FEh
        jne FB
        lea bx, CASE_FE_FB
        jmp CONCAT
FB:
        cmp al, 00FBh
        jne FC
        lea bx, CASE_FE_FB
        jmp CONCAT
```

```
FC:
        cmp al, 00FCh
        jne FA
        lea bx, CASE_FC
        jmp CONCAT
FA:
        cmp al, 00FAh
        jne F8
        lea bx, CASE_FA
        jmp CONCAT
F8:
        cmp al, 00F8h
        jne FD
        lea bx, CASE_F8
        jmp CONCAT
FD:
        cmp al, 00FDh
        jne F9
        lea bx, CASE_FD
        jmp CONCAT
F9:
        cmp al, 00F9h
        jne ERROR
        lea bx, CASE_F9
        jmp CONCAT
CONCAT:
        lea dx, TYPE_PC
        call WriteMsg
        lea dx, [bx]
        call WriteMsg
        jmp DOSBOX_VERS
ERROR:
        lea dx, TYPE_PC
        call WriteMsg
        lea dx, ERROR_MSG
        call WriteMsg
; получим версию досбокса
DOSBOX_VERS:
        mov ah, 30h; функция вернет в al старший ah младший номер версии
        int 21h
        lea si, MSDOS_VERSION
        add si, 16
        call BYTE_TO_DEC
        lea si, MSDOS_VERSION
        add si, 19
        mov al, ah
        call BYTE_TO_DEC
; серийный номер OEM
        mov ah, 30h
        int 21h
        mov al, bh
        lea si, OEM_NUMBER
        add si, 22
        call BYTE_TO_DEC
; серийный номер пользователя
        mov ax, cx
        lea di, USER_NUMBER
```

```
        add di, 34
        call WRD_TO_HEX
        mov al, bl
        call BYTE_TO_HEX
        lea di, USER_NUMBER
        add di, 29
        mov [di], ax

        lea dx, MSDOS_VERSION
        call WriteMsg
        lea dx, OEM_NUMBER
        call WriteMsg
        lea dx, USER_NUMBER
        call WriteMsg
        xor al,al
        mov ah,4Ch
        int 21h
    CODESEG ENDS
END START ;конец модуля, START - точка входа
```

ПРИЛОЖЕНИЕ Б

Исходный код EXE-программы.

```
AStack SEGMENT STACK
        DW 100h DUP(5353H)
AStack ENDS
; ===Данные===
DATA SEGMENT
TYPE_PC         db'Тип IBM PC: ','$'
MSDOS_VERSION   db'Версия MS-DOS:   .                  ',0DH,0AH,'$'
OEM_NUMBER      db'Серийный номер OEM:                 ',0DH,0AH,'$'
USER_NUMBER     db'Серийный номер пользователя:       H',0DH,0AH,'$'
CASE_FF         db'PC                                  ',0DH,0AH,'$'
CASE_FE_FB      db'PC/XT                               ',0DH,0AH,'$'
CASE_FC         db'AT or PS2 model 50-60               ',0DH,0AH,'$'
CASE_FA         db'PS2 model 30                        ',0DH,0AH,'$'
CASE_F8         db'PS2 model 80                        ',0DH,0AH,'$'
CASE_FD         db'PCjr                                ',0DH,0AH,'$'
CASE_F9         db'PC Convertible                      ',0DH,0AH,'$'
ERROR_MSG       db'ERROR                               ',0DH,0AH,'$'
DATA ENDS

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

; Процедура печати строки
WriteMsg PROC near
        mov ah,09h
        int 21h
        ret
WriteMsg ENDP
;--------------------------------------------------
TETR_TO_HEX PROC near
        and al,0Fh
        cmp al,09
        jbe NEXT
        add al,07
NEXT:   add al,30h; код нуля
        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 байта в 10с/с, si - адрес поля младшей цифры
; al содержит исходный байт
        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
;--------------------------------------------------
Main PROC FAR
        sub ax, ax
        push ax
        mov ax, DATA
        mov ds, ax
;получим тип IBM PC
        mov bx, 0F000h
        mov es, bx
        mov al, es:[0FFFEh]
        cmp al, 00FFh
        jne FE
        lea bx, CASE_FF
        jmp CONCAT
FE:
        cmp al, 00FEh
        jne FB
        lea bx, CASE_FE_FB
        jmp CONCAT
FB:
        cmp al, 00FBh
        jne FC
        lea bx, CASE_FE_FB
        jmp CONCAT
FC:
        cmp al, 00FCh
        jne FA
```

```
        lea bx, CASE_FC
        jmp CONCAT
FA:
        cmp al, 00FAh
        jne F8
        lea bx, CASE_FA
        jmp CONCAT
F8:
        cmp al, 00F8h
        jne FD
        lea bx, CASE_F8
        jmp CONCAT
FD:
        cmp al, 00FDh
        jne F9
        lea bx, CASE_FD
        jmp CONCAT
F9:
        cmp al, 00F9h
        jne ERROR
        lea bx, CASE_F9
        jmp CONCAT
CONCAT:
        lea dx, TYPE_PC
        call WriteMsg
        lea dx, [bx]
        call WriteMsg
        jmp DOSBOX_VERS
ERROR:
        lea dx, TYPE_PC
        call WriteMsg
        lea dx, ERROR_MSG
        call WriteMsg
; получим версию досбокса
DOSBOX_VERS:
        mov ah, 30h; функция вернет в al старший ah младший номер версии
        int 21h
        lea si, MSDOS_VERSION
        add si, 16
        call BYTE_TO_DEC
        lea si, MSDOS_VERSION
        add si, 19
        mov al, ah
        call BYTE_TO_DEC
; серийный номер OEM
        mov ah, 30h
        int 21h
        mov al, bh
        lea si, OEM_NUMBER
        add si, 22
        call BYTE_TO_DEC
; серийный номер пользователя
        mov ax, cx
        lea di, USER_NUMBER
        add di, 34
        call WRD_TO_HEX
        mov al, bl
        call BYTE_TO_HEX
        lea di, USER_NUMBER
        add di, 29
        mov [di], ax

        lea dx, MSDOS_VERSION
```

```
        call WriteMsg
        lea dx, OEM_NUMBER
        call WriteMsg
        lea dx, USER_NUMBER
        call WriteMsg
        xor al,al
        mov ah,4Ch
        int 21h
Main ENDP
CODE ENDS
END Main
```