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 _____ Черницын П.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

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

Ход работы.

1. Была реализована программа для определения типа PC..

2. После трансляции с использованием MASM и линковки была получена программа с расширением .EXE. Результат работы представлен на рис. 1.

Рис.1. Запуск "плохого" .EXE файла.

3. После был получен "хороший" .COM модуль, работающий корректно. Результат работы программы представлен на рис.2.

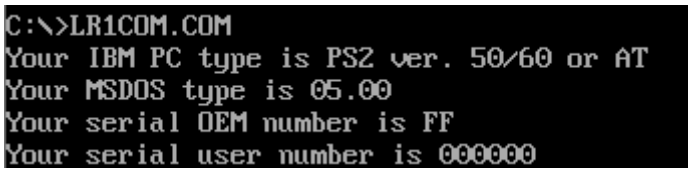

Рис.2. Запуск "хорошего" .COM файла.

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. Результат работы программы представлен на рис.3.

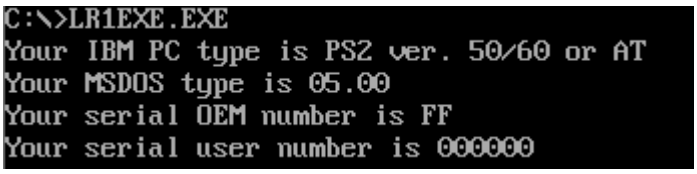

Рис.3. Запуск "хорошего" .EXE файла.

6. Представление файлов в шестнадцатиричном виде представлено на рисунках 4-6.

Рис.4. COM-файл в шестнадцатеричном виде.

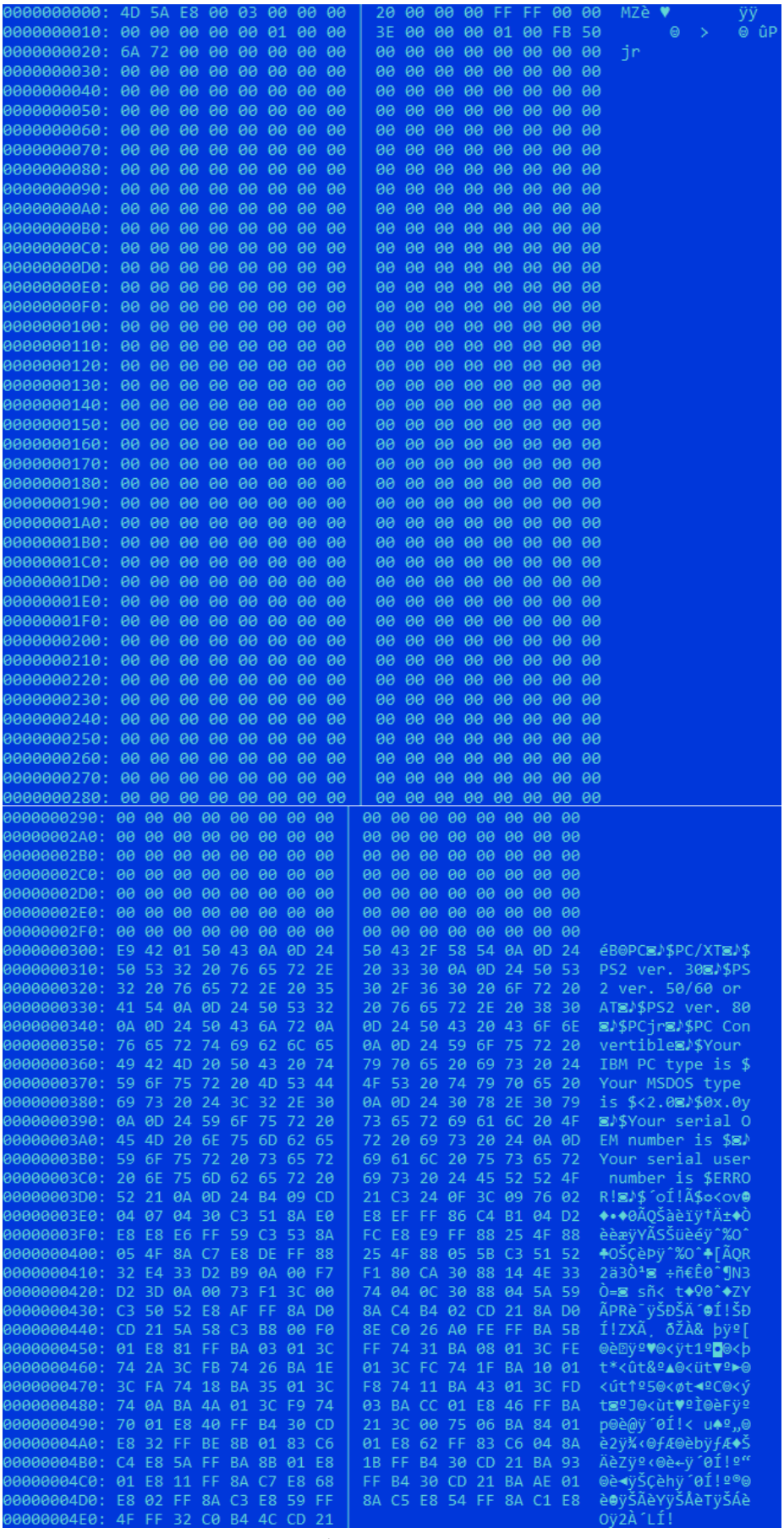

Рис.5. "Плохой" EXE-файл в шестнадцатеричном виде.

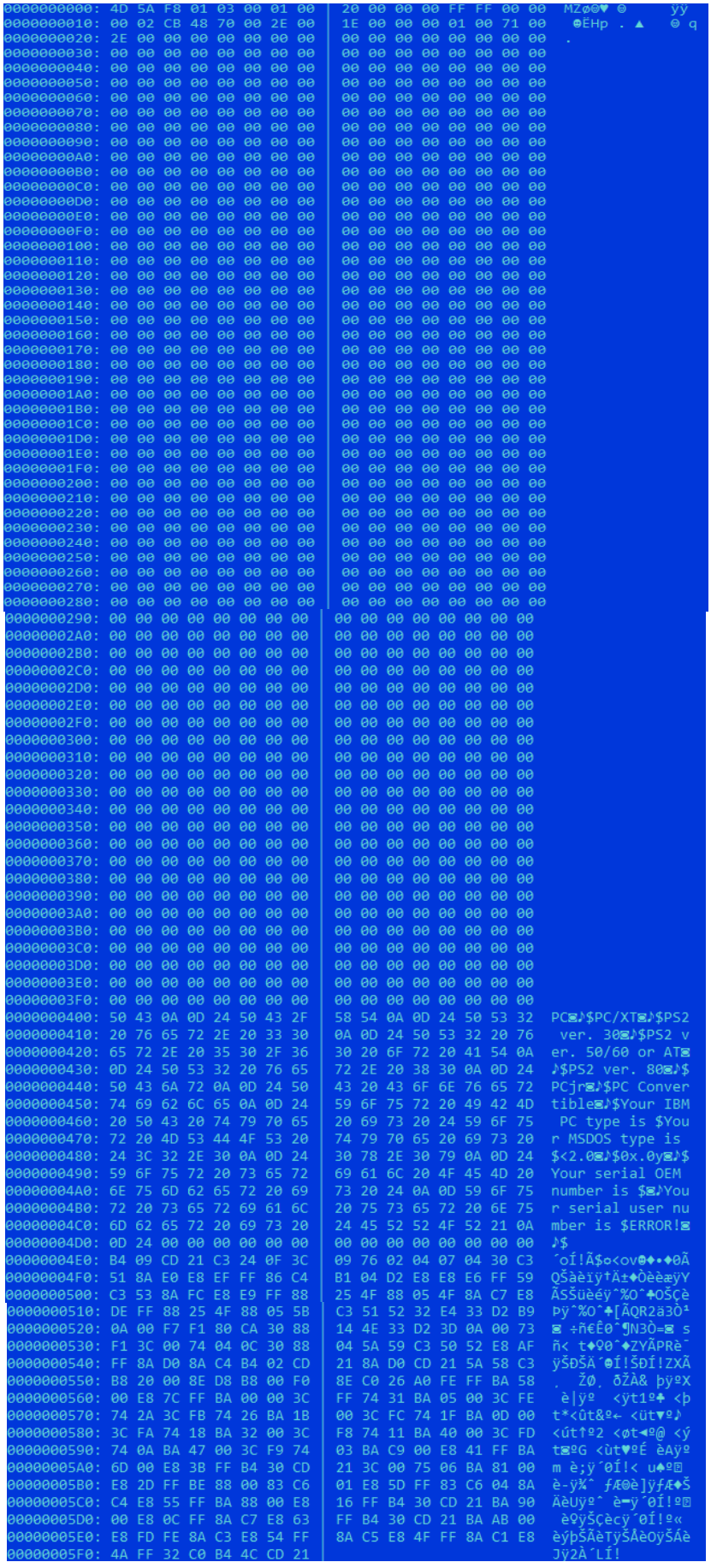

Рис.6 (начало). "Хороший" EXE-файл в шестнадцатеричном виде.

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

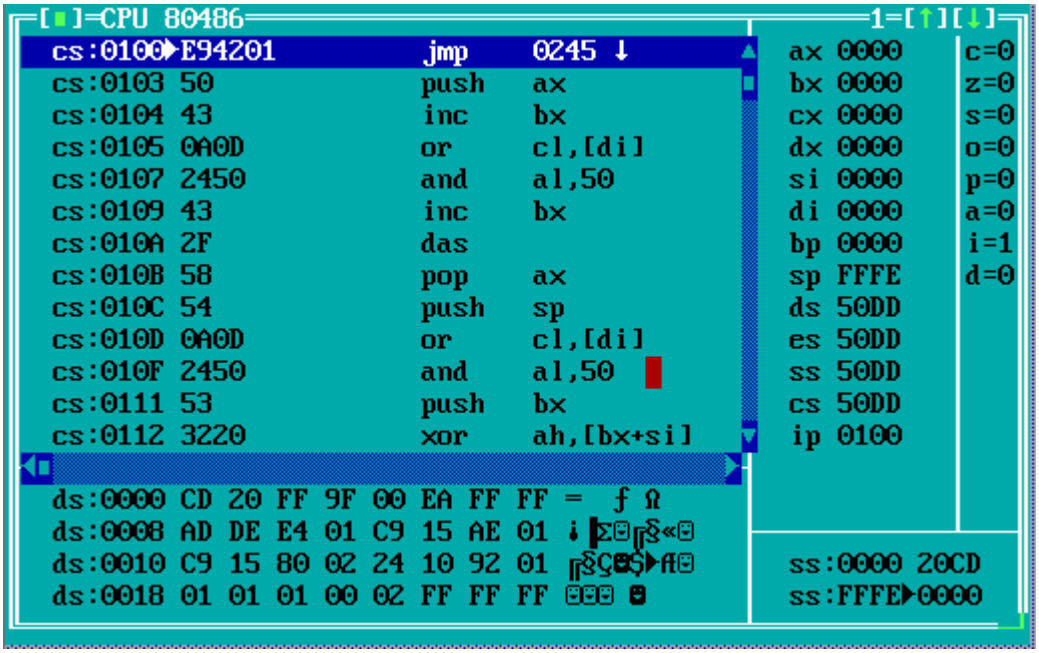

Рис.7. Запуск COM-программы в отладчике.

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

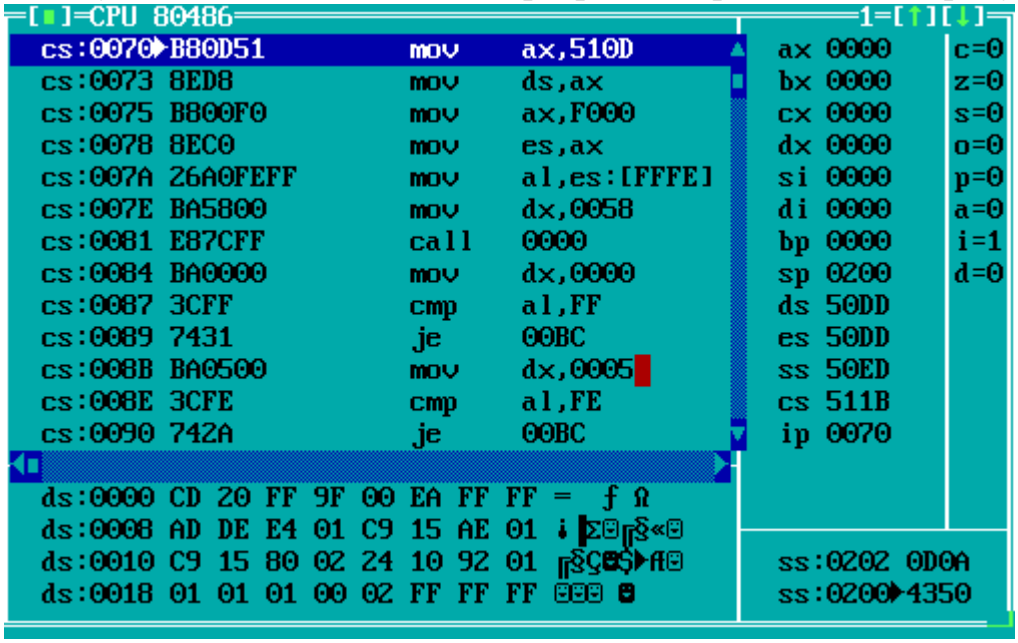

Рис.8. Запуск EXE-программы в отладчике.

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.

Программист может и не использовать перечисленные модели

А также сегмент стека.

3) Какие директивы должны обязательно быть в тексте COM-программы? Директива ASSUME, сообщающая ассемблеру информацию о соответствии между регистрами и сегментами. Директива ORG, сообщающая компилятору о смещении адресации внутри кода.

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

Из-за 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 50DDH. 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 кода. В IP записывается 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 параметрах 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.

```
TESTPC SEGMENT
        ASSUME cs:TESTPC, ds:TESTPC, es:NOTHING, ss:NOTHING
        ORG 100H
START: jmp BEGIN
;DATA
FF_type  db 'PC',10,13,'$'
FE_FB_type          db 'PC/XT',10,13,'$'
FA_type db 'PS2 ver. 30',10,13,'$'
FC_type  db 'PS2 ver. 50/60 or AT',10,13,'$'
F8_type  db 'PS2 ver. 80',10,13,'$'
FD_type db 'PCjr',10,13,'$'
F9_type  db 'PC Convertible',10,13,'$'
typePC              db 'Your IBM PC type is ','$'
typeMSDOS           db 'Your MSDOS type is ','$'
old_ver             db '<2.0',10,13,'$'
new_ver             db '0x.0y',10,13,'$'
serialOEM           db 'Your serial OEM number is ','$'
serialUser          db 10,13,'Your serial user number is ','$'
ErrorMsg            db 'ERROR!',10,13,'$'
;--------------------------------------------------------
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 в шест. сс ах
        push cx
        mov ah,al
        call TETR_TO_HEX
        xchg al,ah
        mov cl,4
        shr al,cl
        call TETR_TO_HEX ;al - старшая
        pop cx ;ah - младшая цифра
        ret
BYTE_TO_HEX ENDP
;--------------------------------------------------------
WRD_TO_HEX PROC near
; перевод в 16сс 16ти разрядного числа
; 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 10сс, si - адрес поля младшей цифры
; al содержит исходный байт
        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
;--------------------------------------------------
WRITE_AL_HEX PROC NEAR
        push ax
        push dx
        call BYTE_TO_HEX

        mov dl, al
        mov al, ah
        mov ah, 02h
        int 21h

        mov dl, al
        int 21h
        pop dx
        pop ax
        ret
WRITE_AL_HEX ENDP
;--------------------------------------------------
BEGIN:
                        mov ax, 0F000h
                        mov es, ax
                        mov al, es:[0FFFEh]

                        mov dx, offset typePC
                        call WriteMsg

                        mov dx, offset FF_type
```

```
                cmp al, 0FFh
                je WRITE_TYPE

                mov dx, offset FE_FB_type
                cmp al, 0FEh
                je WRITE_TYPE
                cmp al, 0FBh
                je WRITE_TYPE

                mov dx, offset FC_type
                cmp al, 0FCh
                je WRITE_TYPE

                mov dx, offset FA_type
                cmp al, 0FAh
                je WRITE_TYPE

                mov dx, offset F8_type
                cmp al, 0F8h
                je WRITE_TYPE

                mov dx, offset FD_type
                cmp al, 0FDh
                je WRITE_TYPE

                mov dx, offset F9_type
                cmp al, 0F9h
                je WRITE_TYPE

                mov dx, offset ErrorMsg

WRITE_TYPE:
                call WriteMsg

                mov dx, offset typeMSDOS
                call WriteMsg

                mov ah, 30h
                int 21h

                cmp al, 0
                jne SKIP_1
                mov dx, offset old_ver
                call WriteMsg

SKIP_1:
                mov si, offset new_ver
                add si, 1
                call BYTE_TO_DEC
                add si, 4
                mov al, ah
                call BYTE_TO_DEC
                mov dx, offset new_ver
                call WriteMsg

                mov ah, 30h
                int 21h

                mov dx, offset serialOEM
                call WriteMsg
```

```
                mov al, bh
                call WRITE_AL_HEX

                mov ah, 30h
                int 21h

                mov dx, offset serialUser
                call WriteMsg
                mov al, bl
                call WRITE_AL_HEX
                mov al, ch
                call WRITE_AL_HEX
                mov al, cl
                call WRITE_AL_HEX

                xor al,al
                mov AH,4Ch
                int 21h
TESTPC ENDS
                END START;
```

ПРИЛОЖЕНИЕ Б

Исходный код EXE-программы.

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

DATA SEGMENT
FF_type  db 'PC',10,13,'$'
FE_FB_type        db 'PC/XT',10,13,'$'
FA_type db 'PS2 ver. 30',10,13,'$'
FC_type db 'PS2 ver. 50/60 or AT',10,13,'$'
F8_type  db 'PS2 ver. 80',10,13,'$'
FD_type db 'PCjr',10,13,'$'
F9_type  db 'PC Convertible',10,13,'$'
typePC            db 'Your IBM PC type is ','$'
typeMSDOS         db 'Your MSDOS type is ','$'
old_ver           db '<2.0',10,13,'$'
new_ver           db '0x.0y',10,13,'$'
serialOEM         db 'Your serial OEM number is ','$'
serialUser        db 10,13,'Your serial user number is ','$'
ErrorMsg          db 'ERROR!',10,13,'$'
DATA ENDS
;--------------------------------------------------------
CODE SEGMENT
                  ASSUME CS:CODE, DS:DATA, SS:ASTACK
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 в шест. сс ах
      push cx
      mov ah,al
      call TETR_TO_HEX
      xchg al,ah
      mov cl,4
      shr al,cl
      call TETR_TO_HEX ;al - старшая
      pop cx ;ah - младшая цифра
      ret
BYTE_TO_HEX ENDP
;--------------------------------------------------------
WRD_TO_HEX PROC near
; перевод в 16сс 16ти разрядного числа
; 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 10сс, si - адрес поля младшей цифры
; al содержит исходный байт
        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
;--------------------------------------------------------
WRITE_AL_HEX PROC NEAR
        push ax
        push dx
        call BYTE_TO_HEX

        mov dl, al
        mov al, ah
        mov ah, 02h
        int 21h

        mov dl, al
        int 21h
        pop dx
        pop ax
        ret
WRITE_AL_HEX ENDP
;--------------------------------------------------------
BEGIN PROC near
                        mov ax, DATA
                        mov ds, ax

                        mov ax, 0F000h
                        mov es, ax
```

```
                mov al, es:[0FFFEh]

                mov dx, offset typePC
                call WriteMsg

                mov dx, offset FF_type
                cmp al, 0FFh
                je WRITE_TYPE

                mov dx, offset FE_FB_type
                cmp al, 0FEh
                je WRITE_TYPE
                cmp al, 0FBh
                je WRITE_TYPE

                mov dx, offset FC_type
                cmp al, 0FCh
                je WRITE_TYPE

                mov dx, offset FA_type
                cmp al, 0FAh
                je WRITE_TYPE

                mov dx, offset F8_type
                cmp al, 0F8h
                je WRITE_TYPE

                mov dx, offset FD_type
                cmp al, 0FDh
                je WRITE_TYPE

                mov dx, offset F9_type
                cmp al, 0F9h
                je WRITE_TYPE

                mov dx, offset ErrorMsg

WRITE_TYPE:
                call WriteMsg

                mov dx, offset typeMSDOS
                call WriteMsg

                mov ah, 30h
                int 21h

                cmp al, 0
                jne SKIP_1
                mov dx, offset old_ver
                call WriteMsg

SKIP_1:
                mov si, offset new_ver
                add si, 1
                call BYTE_TO_DEC
                add si, 4
                mov al, ah
                call BYTE_TO_DEC
                mov dx, offset new_ver
                call WriteMsg
```

```
                mov ah, 30h
                int 21h

                mov dx, offset serialOEM
                call WriteMsg
                mov al, bh
                call WRITE_AL_HEX

                mov ah, 30h
                int 21h

                mov dx, offset serialUser
                call WriteMsg
                mov al, bl
                call WRITE_AL_HEX
                mov al, ch
                call WRITE_AL_HEX
                mov al, cl
                call WRITE_AL_HEX

                xor al,al
                mov AH,4Ch
                int 21h
BEGIN ENDP
CODE ENDS
                END BEGIN
```