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. 8383 _____ Шишкин И.В.

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.

Был написан исходный .COM модуль, который определяет тип PC и версию системы. В результате был получен "хороший" .COM модуль, результат выполнения которого представлен на рис. 1.

Рисунок 1 – Результат выполнения "хорошего" .COM модуля

Был построен "плохой" .EXE файл из исходного текста для .COM модуля. Результат выполнения "плохого" .EXE файла представлен на рис. 2. Как видно на рисунке, после линковки, выдано предупреждение "No stack".

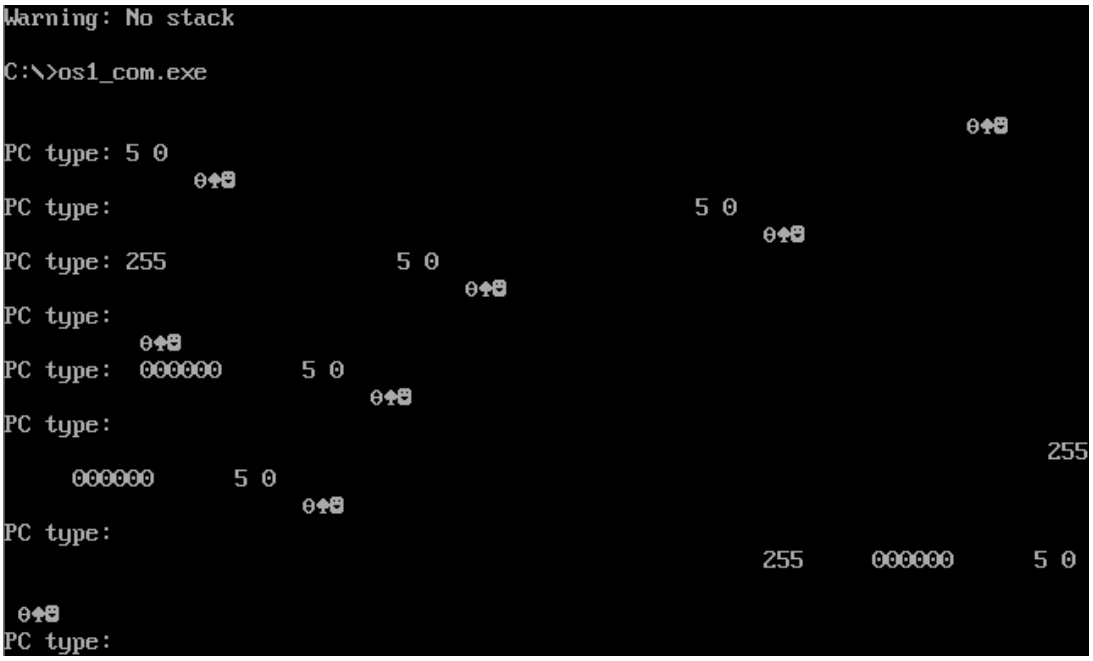

Рисунок 2 – Результат выполнения "плохого" .EXE модуля

Был написан текст исходного .EXE модуля, который выполняет те же функции, что и .COM модуль. Таким образом был получен "хороший" .EXE модуль. Результат его выполнения представлен на рис. 3.

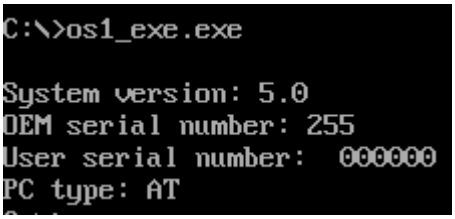

Рисунок 3 – Результат выполнения "хорошего" .EXE модуля

Была запущена программа Far manager, и в ней открыты .COM файл (рис. 4), "плохой" .EXE файл (рис. 5) и "хороший" .EXE файл (рис.6) в шестнадцатеричном виде.

Рисунок 4 – .COM файл в FAR

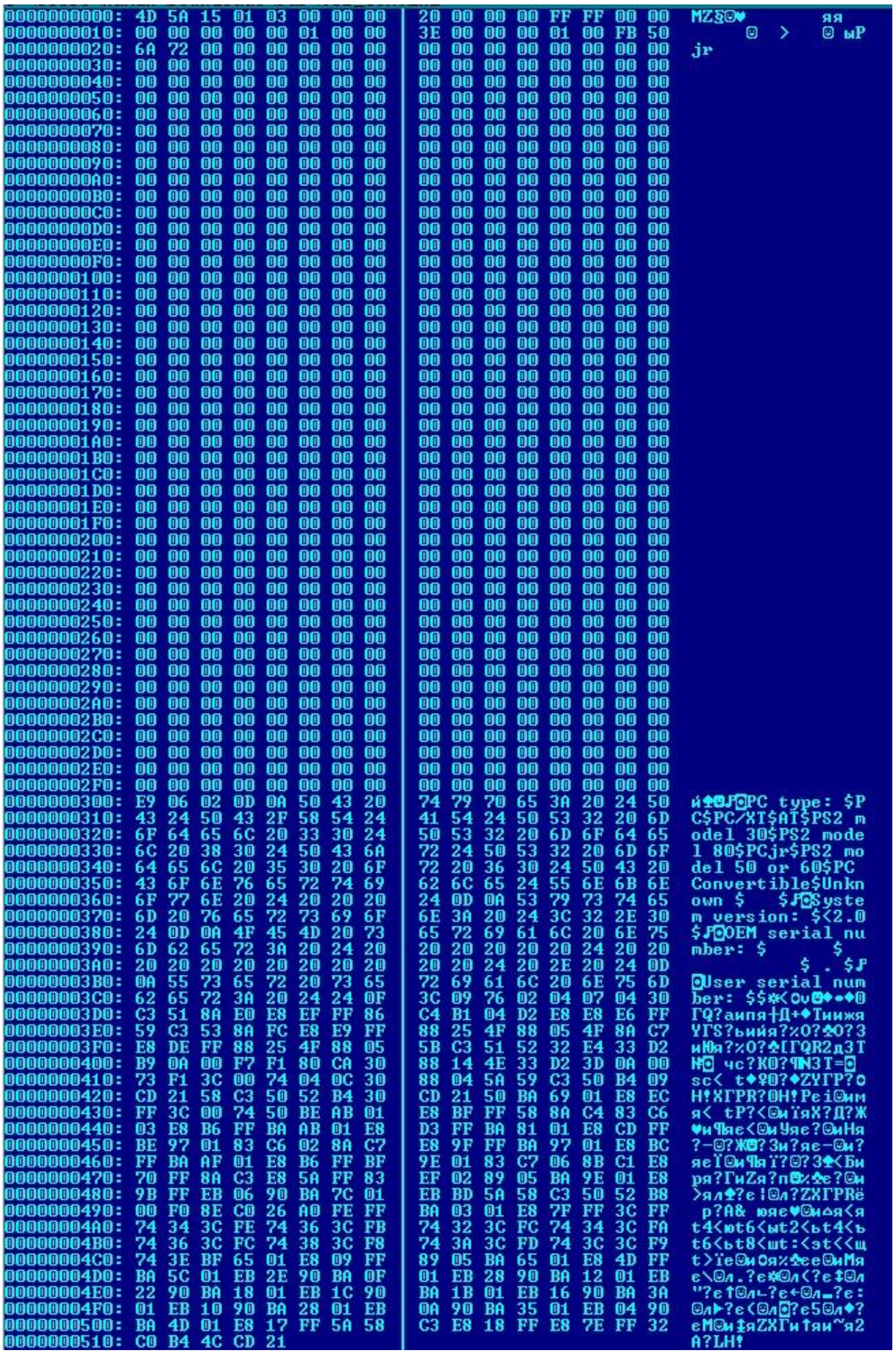

Рисунок 5 – "Плохой" .EXE файл в FAR

Рисунок 6 – "Хороший" .EXE файл в FAR

Был загружен .COM файл и "хороший" .EXE в отладчик TD.EXE. Соответствующие результаты представлены на рис. 7 и рис. 8.

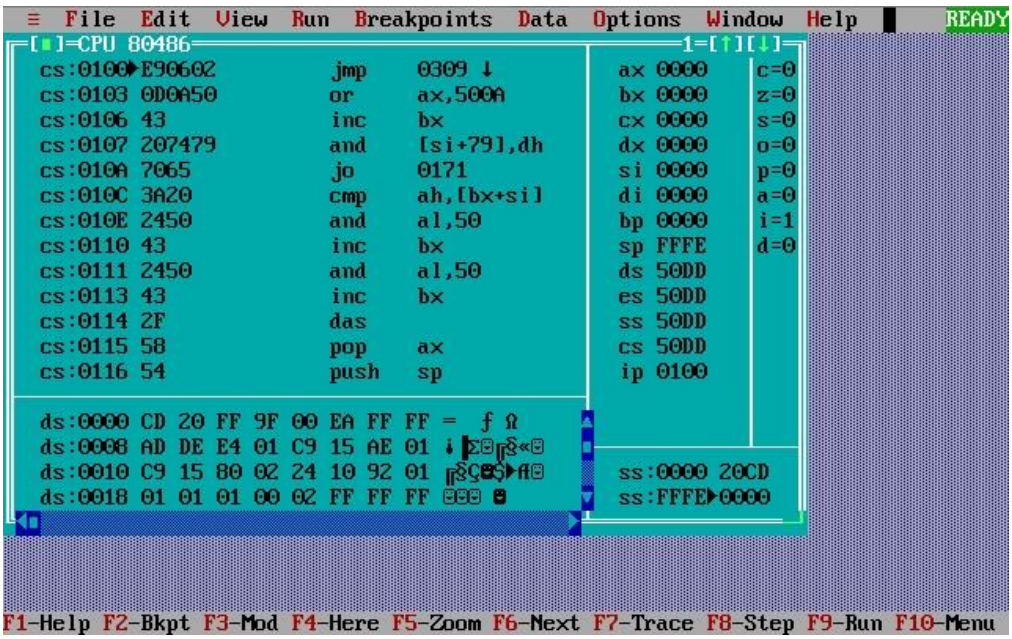

Рисунок 7 – .COM файл в TD.EXE

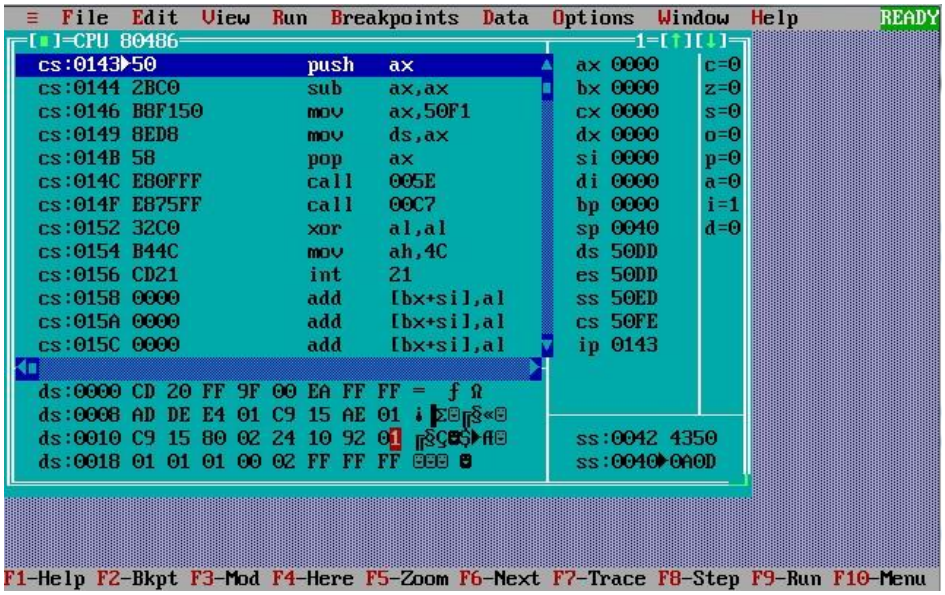

Рисунок 8 – "Хороший" .EXE файл в TD.EXE

Все нужные файлы представлены в приложении.

Контрольные вопросы.

Отличия исходных текстов COM и EXE программ.

1) Сколько сегментов должна содержать COM-программа?

Один. В COM-программе код и данные находятся в одном сегменте, задавать сегмент стека не нужно, так как для COM-программы стек выделяется операционной системой в конце сегмента программы.

2) Сколько сегментов должна содержать EXE-программа?

Количество допустимых сегментов определяется используемой моделью памяти. Сегментов должно быть не менее одного. Верхнего порога нет. Например, модель huge имеет много сегментов кода и сегментов данных.

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

Так как адресация начинается со смещения 100h от начала PSP, COM-программа должна содержать директиву org 100h. Должна быть в конце программы директива END. Так же должна быть директива assume, т.к. эта строка указывает Ассемблеру на привязку сегментного регистра CS к нашему сегменту (CSEG). Пример ошибки, если закомментировать assume, и залинковать программу с помощью TLINK, приведена на рис. 9.

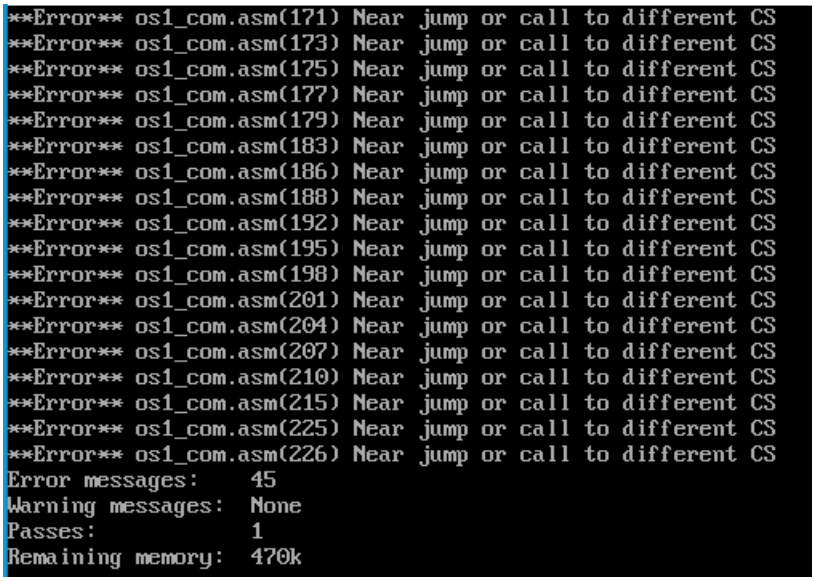

Рисунок 9 – Ошибка при линковке с помощью TLINK.EXE

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

Нет. Нельзя использовать команды с оператором seg, т.к. в COM-программе нет заголовка.

Отличия форматов файлов COM и EXE модулей.

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

В файлах данного типа, обычно не имеющими даже заголовка файла, содержатся только машинный код и данные программы. Код располагается с адреса 100h, т.к. в начале программы происходит смещение.

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

Код и данные содержатся в одном сегменте. Исходя из рис. 5, "плохой" EXE файл начинается с адреса 300h. С адреса 0 располагается заголовок и таблица настроек.

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

Код, данные и стек содержатся в разных сегментах. Так как "плохой" EXE файл получен из исходного текста для COM модуля, то стек, в отличие от "хорошего" EXE, явно не определен. Еще одно отличие в том, что код и данные у "плохого" EXE содержатся в одном сегменте, как было сказано выше. Так же "хороший" EXE начинается с 240h, т.к. 200h для заголовка, а 40h для стека. В "плохом" EXE те же самые 200h для заголовка, а 100h из-за директивы org 100h.

Загрузка COM модуля в основную память.

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

Команды, данные, PSP и стек располагаются в одном сегменте. Код располагается с адреса 100h.

2) Что располагается с адреса 0?

С адреса 0 располагается PSP.

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

Во время загрузки COM-программы выделяется первый свободный сегмент памяти и в его начале размещается PSP. Все сегментные регистры устанавливаются на этот сегмент. Регистры CS и SS указывают на PSP, регистр SP - на конец сегмента PSP (т.е. SP = 0FFFEh).

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

Сегмент стека задавать не нужно, т.к. он выделяется ОС в конце сегмента программы. Все 64 Кбайт занимают данные, коды команд и область стека. В стек задается число 00h. Указатель стека SP устанавливается на конец сегмента программы.

Загрузка "хорошего" EXE модуля в основную память.

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

Во время загрузки "хорошей" EXE-программы выделяется сегмент для PSP. Затем следует сегмент программы. На основании информации в заголовке EXE-программы загрузчик пересчитывает дальние ссылки с учетом реального расположения сегментов. Таким образом, программа заранее не знает, в каких сегментах она будет выполняться. Поэтому для обращения к данным в начале EXE-программы необходимо загрузить в регистр DS значение этого указателя

2) На что указывают регистры DS и ES?

На PSP.

3) Как определяется стек?

EXE-программа по умолчанию записывает нулевое слово в стек. Его можно определить вручную директивой stack.

4) Как определяется точка входа?

По умолчанию точкой входа является начало сегмента кода. Задать точку входа можно, написав в конце программы "end <точка входа>".

Выводы.

В ходе выполнения лабораторной работы были исследованы различия в исходных текстах модулей типов .COM и .EXE. Были разобраны структуры файлов загрузочных модулей и способы их загрузки в основную память.

ПРИЛОЖЕНИЕ А
КОД .COM МОДУЛЯ

```
TESTPC SEGMENT
ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN

     PC_TYPE db 13, 10, "PC type: $"
     TYPE_PC db "PC$"
     TYPE_PCXT db "PC/XT$"
     TYPE_AT db "AT$"
     TYPE_30 db "PS2 model 30$"
     TYPE_80 db "PS2 model 80$"
     TYPE_PCJR db "PCjr$"
     TYPE_5060 db "PS2 model 50 or 60$"
     TYPE_CONVERTIBLE db "PC Convertible$"
     TYPE_UNKNOWN db "Unknown $"
     UNKNOWN_STR db "   $"
     SYSTEM_VERSION db 13,10,"System version: $"
     IF_AL0 db "<2.0$"
     OEM_SN db 13, 10, "OEM serial number: $"
     STR_OEM db "      $"
     STR_USER db "            $"
     SYSVER_STR db " . $"
     USER_SN db 13, 10, "User serial number: $"

;-------------------------------------------------
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 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, в 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 10 с.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
;--------------------------------------------
PRINT PROC near
     push ax
     mov ah, 09h
     int 21h
     pop ax
     ret
PRINT ENDP
;--------------------------------------------
OS_VERSION_DETECTION PROC near
     push ax
     push dx
```

```
        mov ah, 30h
        int 21h
        push ax
        mov dx, offset SYSTEM_VERSION
        call PRINT
        cmp al, 0
        je ifal0
        mov si, offset SYSVER_STR
        call BYTE_TO_DEC
        pop ax
        mov al, ah
        add si, 3
        call BYTE_TO_DEC
        mov dx, offset SYSVER_STR
        exit_from_al0:
             call PRINT

        mov dx, offset OEM_SN
        call PRINT
        mov si, offset STR_OEM
        add si, 2   ;т.к. OEM может состоять макс. из 2 цифр в 16
с.с.
        mov al, bh
        call BYTE_TO_DEC
        mov dx, offset STR_OEM
        call PRINT

        mov dx, offset USER_SN
        call PRINT
        mov di, offset STR_USER  ;т.к. WRD_TO_HEX принимает di
        add di, 6
        mov ax, cx
        call WRD_TO_HEX
        mov al, bl
```

```
      call BYTE_TO_HEX
      sub di, 2
      mov [di], ax
      mov dx, offset STR_USER
      call PRINT
      jmp exit_from_osdetection


      ifal0:
            mov dx, offset IF_AL0
            jmp exit_from_al0

      exit_from_osdetection:
            pop dx
            pop ax
      ret
OS_VERSION_DETECTION ENDP
;-----------------------------------------------
PC_TYPE_DETECTION PROC near
      push ax
      push dx
      mov ax, 0F000h
      mov es, ax
      mov al, es:[0FFFEH]
      mov dx, offset PC_TYPE
      call PRINT

      ;Определение типа
      cmp al, 0FFh ;PC
      je pc_model
      cmp al, 0FEh ;PC/XT
      je xt_model
      cmp al, 0FBh ;PC/XT
      je xt_model
```

```
cmp al, 0FCh ;AT
je at_model
cmp al, 0FAh ;PS2 30
je ps2_model_30
cmp al, 0FCh ;PS2 50 or 60
je ps2_model_5060
cmp al, 0F8h ;PS2 80
je ps2_model_80
cmp al, 0FDh ;PCjr
je pcjr_model
cmp al, 0F9h ;PC Convertible
je pc_convertible

;Если неизвестный тип
mov di, offset UNKNOWN_STR
call BYTE_TO_HEX
mov [di], ax
mov dx, offset UNKNOWN_STR
call PRINT
mov dx, offset TYPE_UNKNOWN
jmp end_of_detection

pc_model:
     mov dx, offset TYPE_PC
     jmp end_of_detection
xt_model:
     mov dx, offset TYPE_PCXT
     jmp end_of_detection
at_model:
     mov dx, offset TYPE_AT
     jmp end_of_detection
ps2_model_30:
     mov dx, offset TYPE_30
     jmp end_of_detection
```

```
      ps2_model_5060:
            mov dx, offset TYPE_5060
            jmp end_of_detection
      ps2_model_80:
            mov dx, offset TYPE_80
            jmp end_of_detection
      pcjr_model:
            mov dx, offset TYPE_PCJR
            jmp end_of_detection
      pc_convertible:
            mov dx, offset TYPE_CONVERTIBLE

      end_of_detection:
            call PRINT
      pop dx
      pop ax
      ret

PC_TYPE_DETECTION ENDP
;-----------------------------------------------

BEGIN:

      call OS_VERSION_DETECTION
      call PC_TYPE_DETECTION

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

ПРИЛОЖЕНИЕ Б
КОД .EXE МОДУЛЯ

```
AStack    SEGMENT  STACK
          DW 20h DUP(?)
AStack    ENDS

DATA SEGMENT
     PC_TYPE db 13, 10, "PC type: $"
     TYPE_PC db "PC$"
     TYPE_PCXT db "PC/XT$"
     TYPE_AT db "AT$"
     TYPE_30 db "PS2 model 30$"
     TYPE_80 db "PS2 model 80$"
     TYPE_PCJR db "PCjr$"
     TYPE_5060 db "PS2 model 50 or 60$"
     TYPE_CONVERTIBLE db "PC Convertible$"
     TYPE_UNKNOWN db "Unknown $"
     UNKNOWN_STR db "   $"
     SYSTEM_VERSION db 13,10,"System version: $"
     IF_AL0 db "<2.0$"
     OEM_SN db 13, 10, "OEM serial number: $"
     STR_OEM db "      $"
     STR_USER db "            $"
     SYSVER_STR db " . $"
     USER_SN db 13, 10, "User serial number: $"
DATA ENDS

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

;--------------------------------------------------
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 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, в 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 10 с.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
;-----------------------------------------------
PRINT PROC near
      push ax
      mov ah, 09h
      int 21h
      pop ax
      ret
PRINT ENDP
```

```
;-----------------------------------------------
OS_VERSION_DETECTION PROC near
     push ax
     push dx
     mov ah, 30h
     int 21h
     push ax
     mov dx, offset SYSTEM_VERSION
     call PRINT
     cmp al, 0
     je ifal0
     mov si, offset SYSVER_STR
     call BYTE_TO_DEC
     pop ax
     mov al, ah
     add si, 3
     call BYTE_TO_DEC
     mov dx, offset SYSVER_STR
     exit_from_al0:
          call PRINT

     mov dx, offset OEM_SN
     call PRINT
     mov si, offset STR_OEM
     add si, 2    ;т.к. OEM может состоять макс. из 2 цифр в 16
с.с.
     mov al, bh
     call BYTE_TO_DEC
     mov dx, offset STR_OEM
     call PRINT

     mov dx, offset USER_SN
     call PRINT
     mov di, offset STR_USER  ;т.к. WRD_TO_HEX принимает di
```

```
      add di, 6
      mov ax, cx
      call WRD_TO_HEX
      mov al, bl
      call BYTE_TO_HEX
      sub di, 2
      mov [di], ax
      mov dx, offset STR_USER
      call PRINT
      jmp exit_from_osdetection


      ifal0:
            mov dx, offset IF_AL0
            jmp exit_from_al0

      exit_from_osdetection:
            pop dx
            pop ax
      ret
OS_VERSION_DETECTION ENDP
;-----------------------------------------------
PC_TYPE_DETECTION PROC near
      push ax
      push dx
      mov ax, 0F000h
      mov es, ax
      mov al, es:[0FFFEH]
      mov dx, offset PC_TYPE
      call PRINT

      ;Определение типа
      cmp al, 0FFh ;PC
      je pc_model
```

```
cmp al, 0FEh ;PC/XT
je xt_model
cmp al, 0FBh ;PC/XT
je xt_model
cmp al, 0FCh ;AT
je at_model
cmp al, 0FAh ;PS2 30
je ps2_model_30
cmp al, 0FCh ;PS2 50 or 60
je ps2_model_5060
cmp al, 0F8h ;PS2 80
je ps2_model_80
cmp al, 0FDh ;PCjr
je pcjr_model
cmp al, 0F9h ;PC Convertible
je pc_convertible

;Если неизвестный тип
mov di, offset UNKNOWN_STR
call BYTE_TO_HEX
mov [di], ax
mov dx, offset UNKNOWN_STR
call PRINT
mov dx, offset TYPE_UNKNOWN
jmp end_of_detection

pc_model:
     mov dx, offset TYPE_PC
     jmp end_of_detection
xt_model:
     mov dx, offset TYPE_PCXT
     jmp end_of_detection
at_model:
     mov dx, offset TYPE_AT
```

```
            jmp end_of_detection
      ps2_model_30:
            mov dx, offset TYPE_30
            jmp end_of_detection
      ps2_model_5060:
            mov dx, offset TYPE_5060
            jmp end_of_detection
      ps2_model_80:
            mov dx, offset TYPE_80
            jmp end_of_detection
      pcjr_model:
            mov dx, offset TYPE_PCJR
            jmp end_of_detection
      pc_convertible:
            mov dx, offset TYPE_CONVERTIBLE

      end_of_detection:
            call PRINT
      pop dx
      pop ax
      ret

PC_TYPE_DETECTION ENDP
;-----------------------------------------------

BEGIN PROC FAR
      push ax
      sub ax, ax
      mov ax, data
      mov ds, ax
      pop ax
      call OS_VERSION_DETECTION
      call PC_TYPE_DETECTION
```

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