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 _____ Хотяков Е.П.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

Цель работы.

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

Задание.

Шаг 1. Написать текст исходного .COM модуля, который определяет тип PC и версию системы. Результатом выполнения этого шага будет «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM.

Шаг 2. Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в шаге 1 и отладить его.

Шаг 3. Сравнить исходные тексты для .COM и .EXE модулей. Ответить на вопросы «Отличия исходных текстов COM и EXE программ».

Шаг 4. Запустить FAR и открыть файл загрузочного модуля .COM и файл «плохого» .EXE в шестнадцатеричном виде. Затем открыть файл загрузочного модуля «хорошего» .EXE и сравнить его с предыдущими файлами. Ответить на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

Шаг 5. Открыть отладчик TD.EXE и загрузить CO. Ответить на контрольные вопросы «Загрузка COM модуля в основную память». Представить в отчете план загрузки модуля .COM в основную память.

Шаг 6. Открыть отладчик TD.EXE и загрузить «хороший» .EXE. Ответить на контрольные вопросы «Загрузка «хорошего» EXE в основную память».

Шаг 7. Оформить отчет в соответствии с требованиями. Привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей — в отладчике.

Выполнение работы.

Шаг 1:

На первом шаге была написана программа на ассемблере, из которой были получены “хороший” .COM модуль и “плохой” .EXE модуль.

Рисунок 1 - работа плохого .EXE

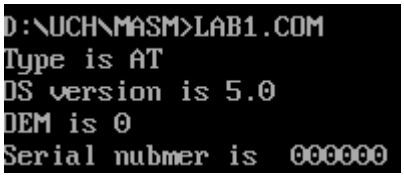


Рисунок 2 - работа хорошего .COM

Шаг 2:

На втором шаге была написана программа на ассемблере, из которой был получен “хороший” .EXE модуль.

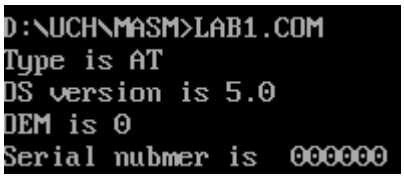


Рисунок 3 - работа хорошего .EXE

Разработанный программный код см. в приложении А.

Шаг 3:

Ответы на контрольные вопросы “Отличия исходных текстов COM и EXE программ” после сравнения исходных текстов для .COM и .EXE модулей.

1) Сколько сегментов должна содержать COM-программа?

В COM программе должен быть сегмент кода(внутри которого также объявляются и данные). Стек в COM-файле генерируется автоматически. Следовательно сама программа содержит только 1 сегмент.

2) EXE-программа?

В EXE программе же может использовать от одного(не менее одного) сегмента, причем сегмент данных, кода и стека записываются отдельно друг от друга.

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

Для смещения от нулевого адреса на 256 байт(смещения до конца PSP) должна использовать директива ORG 100h. Для того, чтобы сегменты данных и кода указывали на один и тот же сегмент необходимо использовать директиву ASSUME с параметрами(где вместо TESTPC должно быть название сегмента) CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING.

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

Нет, не все. Например, команда **MOV AX,CODE** будет некорректна для COM программы, т.к. в момент ассемблирования и редактирования связей сегментное значение для сегмента CODE неизвестно(.COM не может предоставить загрузчику перечня всех сегментных ссылок).

Шаг 4:

Ответы на контрольные вопросы “Отличия форматов COM и EXE модулей” после просмотра .COM и .EXE файлов в шестнадцатеричном виде:

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

COM-программа начинается с самого верха, с ячейки 0h, хотя при загрузке модуля установится смещение в 100h. Сам код располагается в одном сегменте(в котором находятся сегмента кода и данных) и ограничивается 64 Кб памяти.

Рисунок 3 - Шестнадцатеричный вид COM-файла

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

Внутри “плохого” EXE также будет располагаться один сегмент данных. Код(и данные) располагается по адресу 300h. С адреса 0h же располагаются заголовок с MZ-байтами(идентификатор - стандартный формат 16-битных исполняемых файлов с расширением .EXE для DOS) и таблица настроек.

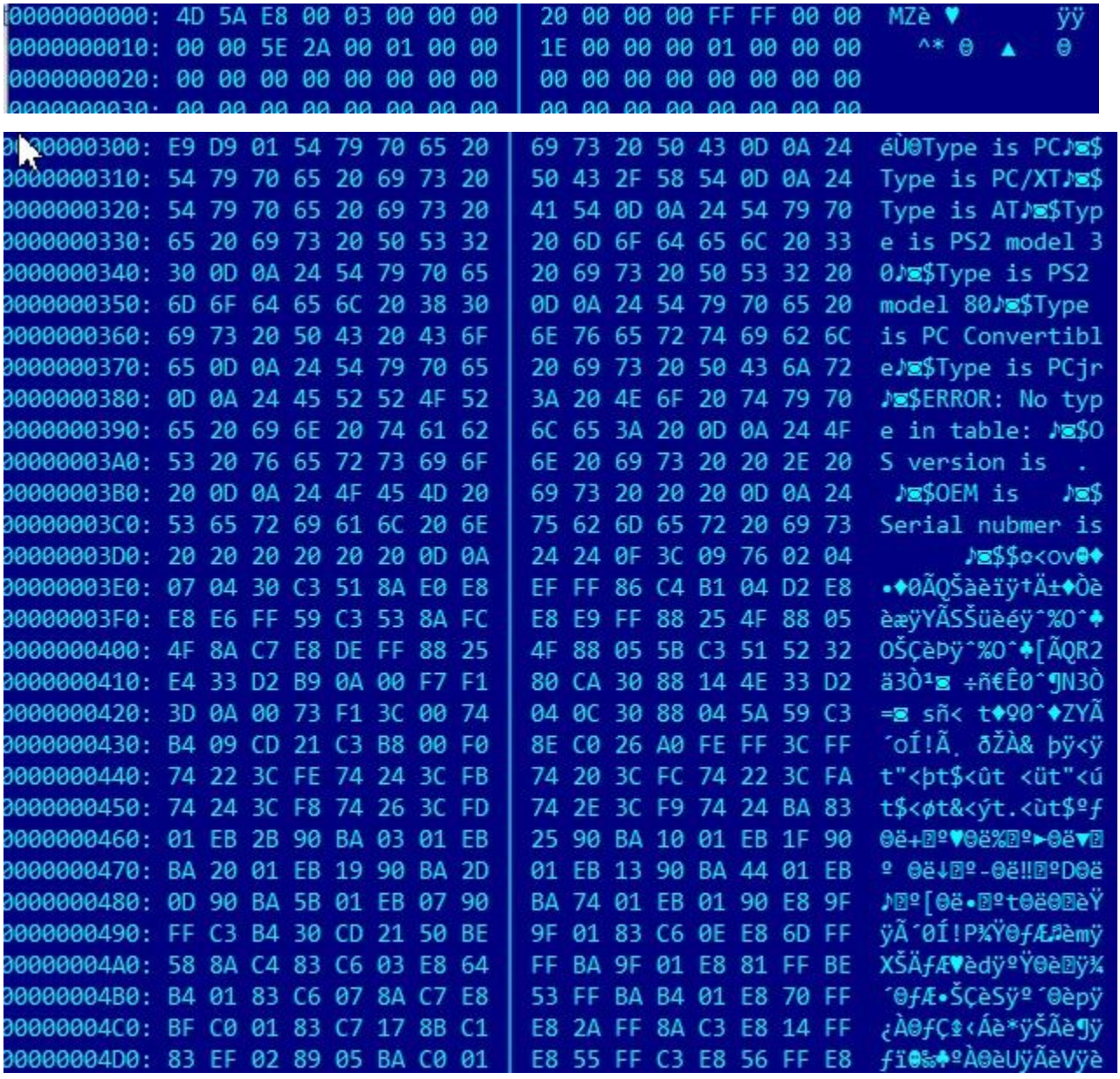


Рисунок 4 - Шестнадцатеричный вид "плохого" EXE-файла

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

В “хорошей” EXE программе сегменты данных, кода и стека разделены друг от друга. В начале файла лежит заголовок, который имеет таблицу для настройки адресов(которая, в отличие от “плохого” EXE-файла, валидна). Адресация кода начинается с 400h, т.к. 200h отводится под заголовок и таблицу настроек и еще 200h под стек.

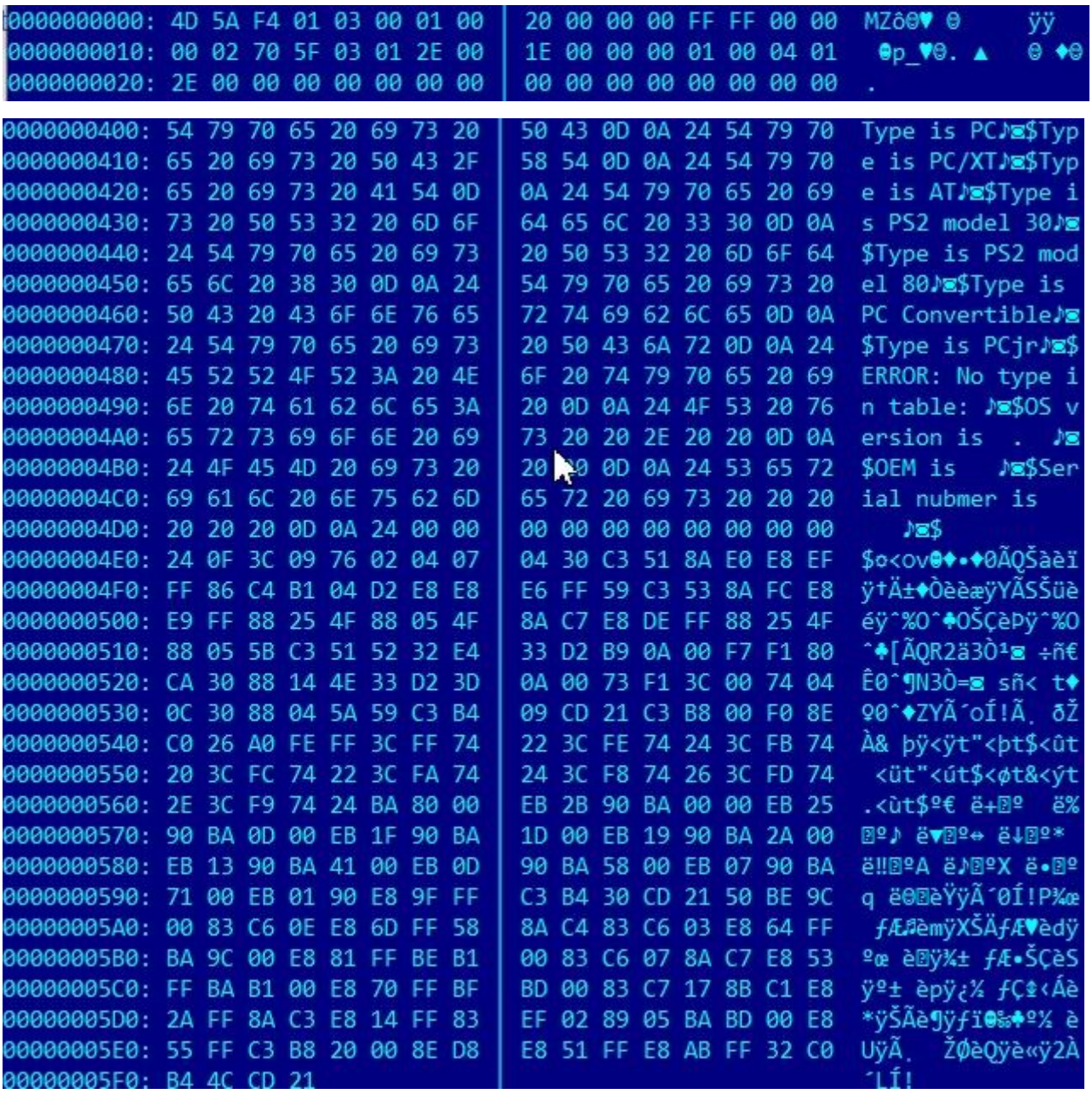
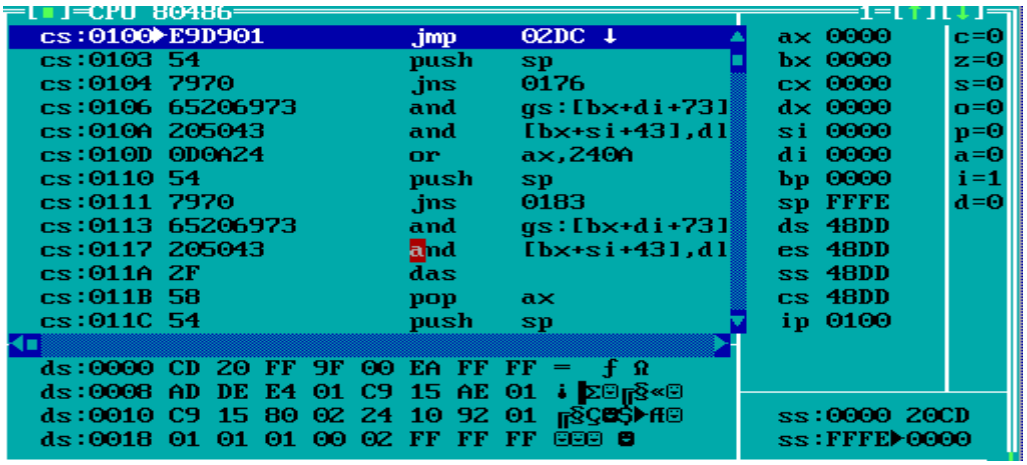


Рисунок 5 - Шестнадцатеричный вид "хорошего" EXE-файла

Шаг 5:



Ответы на контрольные вопросы “Загрузка COM модуля в основную память”:

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

Сначала определяется участок в оперативной памяти, у которого достаточно места для загрузки COM модуля, после чего он помещается в память, начиная с PSP со смещением в 100h.

2) Что располагается с адреса 0?

С адреса 0 располагается сегмент PSP размером 100h

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

CS, DS, ES и SS указывают на PSP и имеют значения 48DD, а SP указывает на конец сегмента FFFE.

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

В COM-программе стек генерируется автоматически в процессе создания.

Стек занимает всю область сегмента, при этом регистр SS указывает на начало(0h), SP – на конец(FFFEh).

Шаг 6:

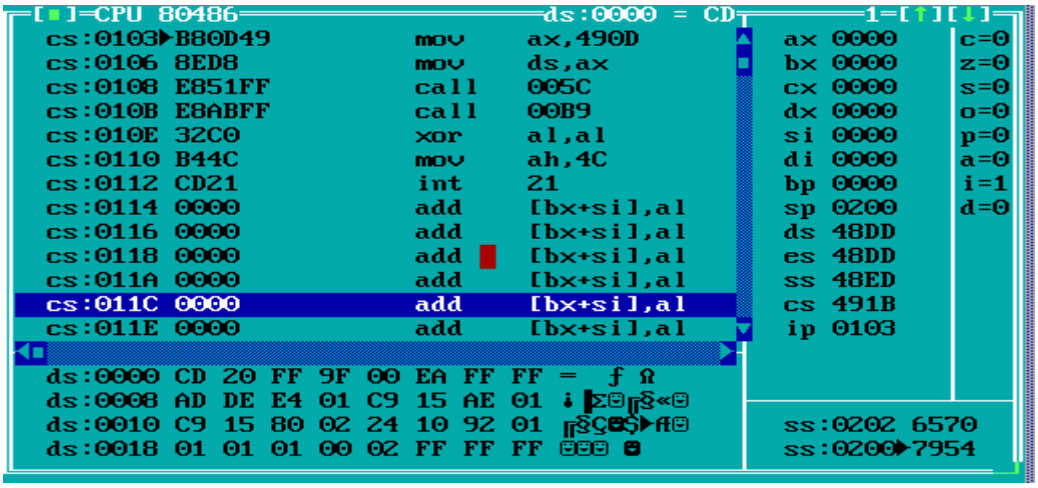


Рисунок 1 - Отладка "хорошего" EXE-файла

Ответы на контрольные вопросы “Загрузка “хорошего” EXE модуля в основную память”:

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

Загрузка файла начинается со смещением 100h(PSP). При этом начинается считывание информации заголовка и выполняется перемещение адресов сегментов: DS и ES указывают на 48DD(начало PSP), SS - 48ED(начало стека), CS – 491B(начало сегмента команд). В IP же загружается смещение точки входа в программу.

2) На что указывают регистры DS и ES?

На начало PSP

3) Как определяется стек?

Стек определяется директивой .stack, при этом SS и SP указывают на начало и конец стека соотвественно.

4) Как определяется точка входа?

Точка входа определяется из метки, расположенной после директивы END.

Выводы.

В результате проделанной работы были написаны программы под COM и EXE-структуры, были изучены их различия и различия их загрузки в память

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab1_com.asm

```
; 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
PC_TYPE db 'Type is PC',0DH,0AH,'$'
PC_XT_TYPE db 'Type is PC/XT',0DH,0AH,'$'
AT_TYPE db 'Type is AT',0DH,0AH,'$'
PS30_TYPE db 'Type is PS2 model 30',0DH,0AH,'$'
PS80_TYPE db 'Type is PS2 model 80',0DH,0AH,'$'
PCCON_TYPE db 'Type is PC Convertible',0DH,0AH,'$'
PCjr_TYPE db 'Type is PCjr',0DH,0AH,'$'
NO_TYPE db 'ERROR: No type in table: ',0DH,0AH,'$'

OS_VERSION db 'OS version is  .  ',0DH,0AH,'$'
OEM db 'OEM is   ',0DH,0AH,'$'
SERIAL db 'Serial nubmer is      ',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

WRITE_STRING PROC near; Вывод строки текста
     mov AH,09h
     int 21h
     ret
WRITE_STRING ENDP

WHAT_TYPE PROC near
    mov AX, 0f000h
    mov ES, AX
    mov AL, es:[0fffeh]

    cmp AL, 0FFh
    je pc_write
    cmp AL, 0FEh
    je pc_xt_write
    cmp AL, 0FBh
    je pc_xt_write
    cmp AL, 0FCh
    je at_write
    cmp AL, 0FAh
    je ps30_write
    cmp AL, 0F8h
    je ps80_write
    cmp AL, 0FDh
    je pcjr_write
    cmp AL, 0F9h
    je pccon_write
    mov dx, offset NO_TYPE
    jmp WRITE_STRING_TYPE



pc_write:
     mov dx, offset PC_TYPE
     jmp WRITE_STRING_TYPE
pc_xt_write:
     mov dx, offset PC_XT_TYPE
```

```
        jmp WRITE_STRING_TYPE
at_write:
        mov dx, offset AT_TYPE
        jmp WRITE_STRING_TYPE
ps30_write:
        mov dx, offset PS30_TYPE
        jmp WRITE_STRING_TYPE
ps80_write:
        mov dx, offset PS80_TYPE
        jmp WRITE_STRING_TYPE
pccon_write:
        mov dx, offset PCCON_TYPE
        jmp WRITE_STRING_TYPE
pcjr_write:
        mov dx, offset PCjr_TYPE
        jmp WRITE_STRING_TYPE
WRITE_STRING_TYPE:
        CALL WRITE_STRING
        ret
WHAT_TYPE ENDP


WHAT_OS_VERSION PROC near
        MOV AH, 30h
        INT 21h
        push AX

        mov SI, offset OS_VERSION
        add SI, 14
        call BYTE_TO_DEC
        pop AX
        mov AL, AH
        add SI, 3
        call BYTE_TO_DEC
        mov DX, offset OS_VERSION
        call WRITE_STRING

        mov SI, offset OEM
        add SI, 7
        mov AL, BH
        call BYTE_TO_DEC
        mov DX, offset OEM
```

```
        call WRITE_STRING

        mov DI, offset SERIAL
        add DI, 23
        mov AX, CX
        call WRD_TO_HEX
        mov AL, BL
        call BYTE_TO_HEX
        sub DI, 2
        mov [DI], AX
        mov DX, offset SERIAL
        call WRITE_STRING
        ret
WHAT_OS_VERSION ENDP

;-------------------------------
; КОД
BEGIN:
        CALL WHAT_TYPE
        CALL WHAT_OS_VERSION
;. . . . . .
; 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

```
AStack SEGMENT STACK
   DW 256 DUP(?)
AStack ENDS
DATA SEGMENT
; Данные
PC_TYPE db 'Type is PC',0DH,0AH,'$'
PC_XT_TYPE db 'Type is PC/XT',0DH,0AH,'$'
AT_TYPE db 'Type is AT',0DH,0AH,'$'
PS30_TYPE db 'Type is PS2 model 30',0DH,0AH,'$'
PS80_TYPE db 'Type is PS2 model 80',0DH,0AH,'$'
PCCON_TYPE db 'Type is PC Convertible',0DH,0AH,'$'
PCjr_TYPE db 'Type is PCjr',0DH,0AH,'$'
```

```
NO_TYPE db 'ERROR: No type in table: ',0DH,0AH,'$'

OS_VERSION db 'OS version is  .  ',0DH,0AH,'$'
OEM db 'OEM is   ',0DH,0AH,'$'
SERIAL db 'Serial nubmer is      ',0DH,0AH,'$'

DATA ENDS

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

WRITE_STRING PROC near; Вывод строки текста
        mov AH,09h
        int 21h
        ret
WRITE_STRING ENDP

WHAT_TYPE PROC near
      mov AX, 0f000h
```

```
    mov ES, AX
    mov AL, es:[0fffeh]

    cmp AL, 0FFh
    je pc_write
    cmp AL, 0FEh
    je pc_xt_write
    cmp AL, 0FBh
    je pc_xt_write
    cmp AL, 0FCh
    je at_write
    cmp AL, 0FAh
    je ps30_write
    cmp AL, 0F8h
    je ps80_write
    cmp AL, 0FDh
    je pcjr_write
    cmp AL, 0F9h
    je pccon_write
    mov dx, offset NO_TYPE
    jmp WRITE_STRING_TYPE



pc_write:
      mov dx, offset PC_TYPE
      jmp WRITE_STRING_TYPE
pc_xt_write:
      mov dx, offset PC_XT_TYPE
      jmp WRITE_STRING_TYPE
at_write:
      mov dx, offset AT_TYPE
      jmp WRITE_STRING_TYPE
ps30_write:
      mov dx, offset PS30_TYPE
      jmp WRITE_STRING_TYPE
ps80_write:
      mov dx, offset PS80_TYPE
      jmp WRITE_STRING_TYPE
pccon_write:
      mov dx, offset PCCON_TYPE
      jmp WRITE_STRING_TYPE
```

```
pcjr_write:
        mov dx, offset PCjr_TYPE
        jmp WRITE_STRING_TYPE
WRITE_STRING_TYPE:
        CALL WRITE_STRING
        ret
WHAT_TYPE ENDP


WHAT_OS_VERSION PROC near
        MOV AH, 30h
        INT 21h
        push AX

        mov SI, offset OS_VERSION
        add SI, 14
        call BYTE_TO_DEC
        pop AX
        mov AL, AH
        add SI, 3
        call BYTE_TO_DEC
        mov DX, offset OS_VERSION
        call WRITE_STRING

        mov SI, offset OEM
        add SI, 7
        mov AL, BH
        call BYTE_TO_DEC
        mov DX, offset OEM
        call WRITE_STRING

        mov DI, offset SERIAL
        add DI, 23
        mov AX, CX
        call WRD_TO_HEX
        mov AL, BL
        call BYTE_TO_HEX
        sub DI, 2
        mov [DI], AX
        mov DX, offset SERIAL
        call WRITE_STRING
        ret
```

```
WHAT_OS_VERSION ENDP

;--------------------------------
; КОД
MAIN PROC FAR
        mov AX, DATA
        mov DS, AX

        call WHAT_TYPE
        call WHAT_OS_VERSION

        xor AL, AL
        mov AH, 4Ch
        int 21h

MAIN ENDP

CODE ENDS
END MAIN
```