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. 0382 _____ Крючков А.М.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2022

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.

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.
2. Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в шаге 1 и отладить его. Таким образом, будет получен «хороший» .EXE.
3. Сравнить исходные тексты для .COM и .EXE модулей. Ответить на вопросы «Отличия исходных текстов COM и EXE программ».
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».
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.
6. Открыть отладчик TD.EXE и загрузить «хороший» .EXE. Ответить на контрольные вопросы «Загрузка «хорошего» EXE в основную память».
7. Оформить отчет в соответствии с требованиями. Привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей — в отладчике.

Выполнение работы.

Был написан текст исходного .COM модуля, который определяет тип PC и версию системы. Для этого определения типа PC, просматривается число находящееся по адресу 0F000h:0FFFEh и сравнивается со значениями из таблицы, данной в методических указаниях. Для определения версии системы, используется прерывание 21h при AH = 30h, результат которого выводится в консоль.

Был написан текст исходного .EXE модуля, который выполняет те же функции, что и модуль .COM. Для этого было использовано разделение программы на сегменты.

Контрольные вопросы.

Отличия исходных текстов COM и EXE программ:

1. Сколько сегментов должна содержать COM-программа?
   Один.
2. EXE-программа?
   Два и более. EXE-программа должна содержать сегмент кода и сегмент данных. Если сегмент стэка не объявлен, то используется стэк DOS.
3. Какие директивы должны быть обязательно в тексте COM-программы?
   ORG и ASSUME. ORG устанавливает смещение относительно области PSP. ASSUME позволяет сегменту данных и сегменту кода указывать на один сегмент.
4. Все ли форматы команд можно использовать в COM-программе?
   Нет. Нельзя использовать команды с указанием сегментов, так как отсутствует relocation table.

Отличия форматов файлов .com и .exe модулей:

1. Какова структура файла .COM? С какого адреса располагается код?
   .COM файл состоит из одного 16-битного сегмента, где данные, код и стек находятся в одном сегменте. Код располагается по адресу 0, но благодаря директиве ORG 100h вся адресация смещается на 100h байт.

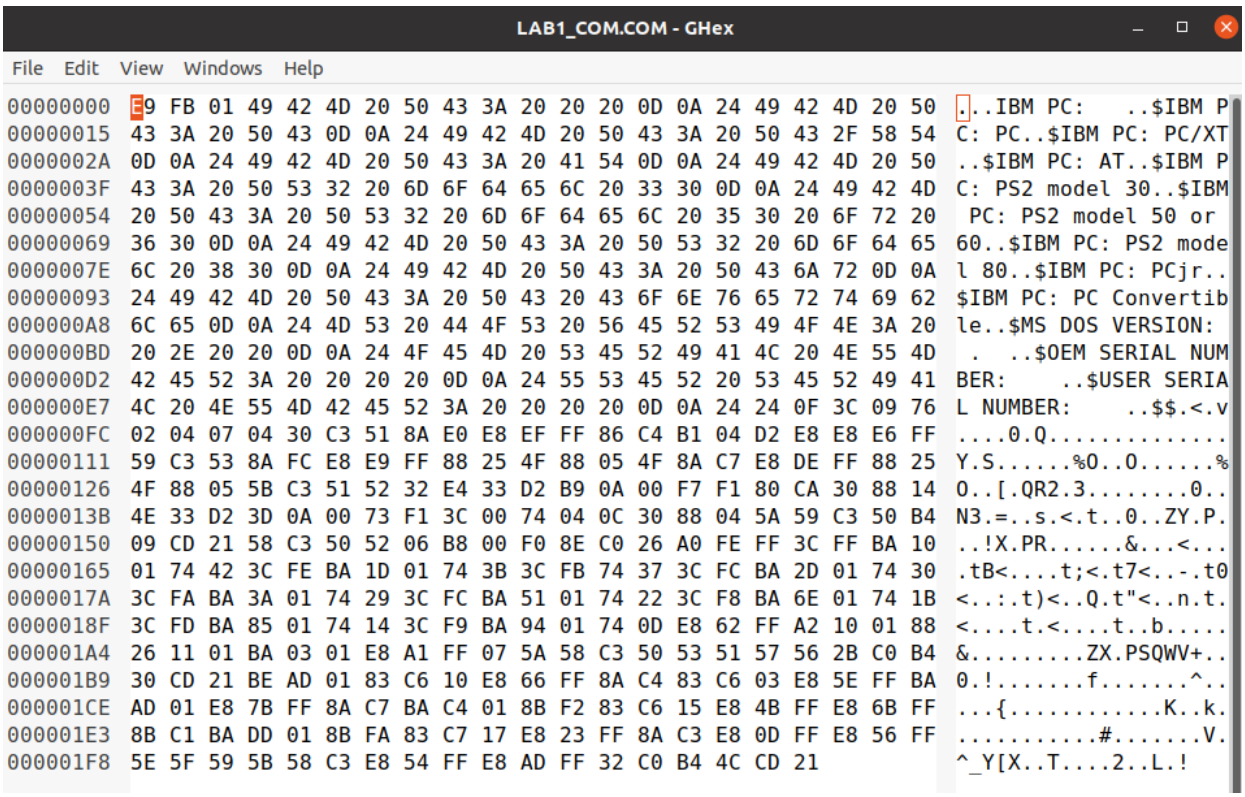

Рис. 1 16-ичный вид «хорошего» .com файла

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

   «Плохой» .EXE файл так же состоит из одного сегмента. Код располагается по адресу 300h. По адресу 0 располагается заголовок .EXE модуля.

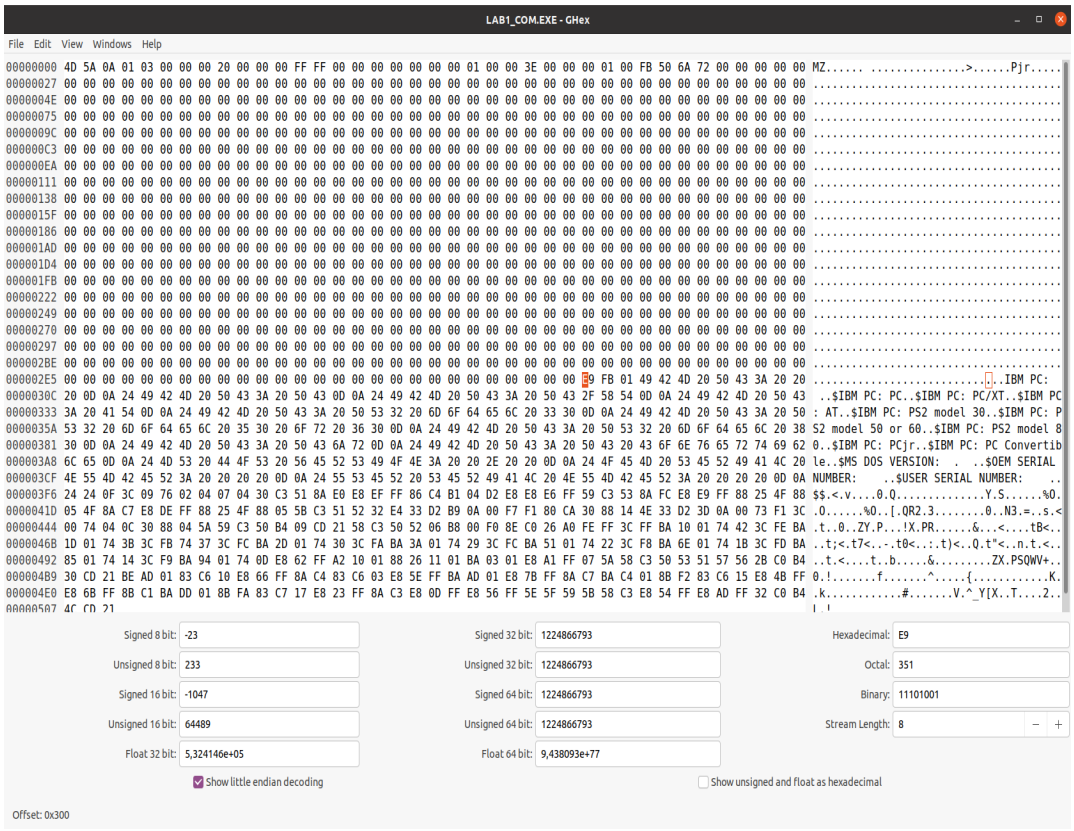

Рис. 2. 16-ичный вид «плохого» exe файла

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

   Файл состоит из двух частей. Управляющей информации для загрузчика и загрузочного модуля. Отличие от «плохого» состоит в разнице загрузочных модулей. В «хорошем» .EXE файле вдля данных, кода и стека отводится по сегменту, когда в плохом все находится в одном сегменте.

Рис. 3 16-ичный вид «хорошего» .EXE

Загрузка .com модуля в основную память:

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

   Загрузка .com модуля в память происходит следующим образом: в основной памяти выделяется свободный сегмент, в первых 256 байтах этого сегмента генерируется PSP, далее записывается сама программа. Код располагается с адреса CS:0100 = 48DD:0100.

2. Что располагается с адреса 0?

   PSP.

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

   Сегментные регистры имеют значение 48DD. Они указывают на начало PSP.

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

   Стек определяется автоматически. Стек находится находится в общем сегменте, т. к. для всей программы сегмент только один. Sp в начале равен FFFE, т. е. Последнему чётному доступному адресу. Таким образом стэк заполняется «с конца» сегмента. Стэк имеет адреса от FFFE до 0.

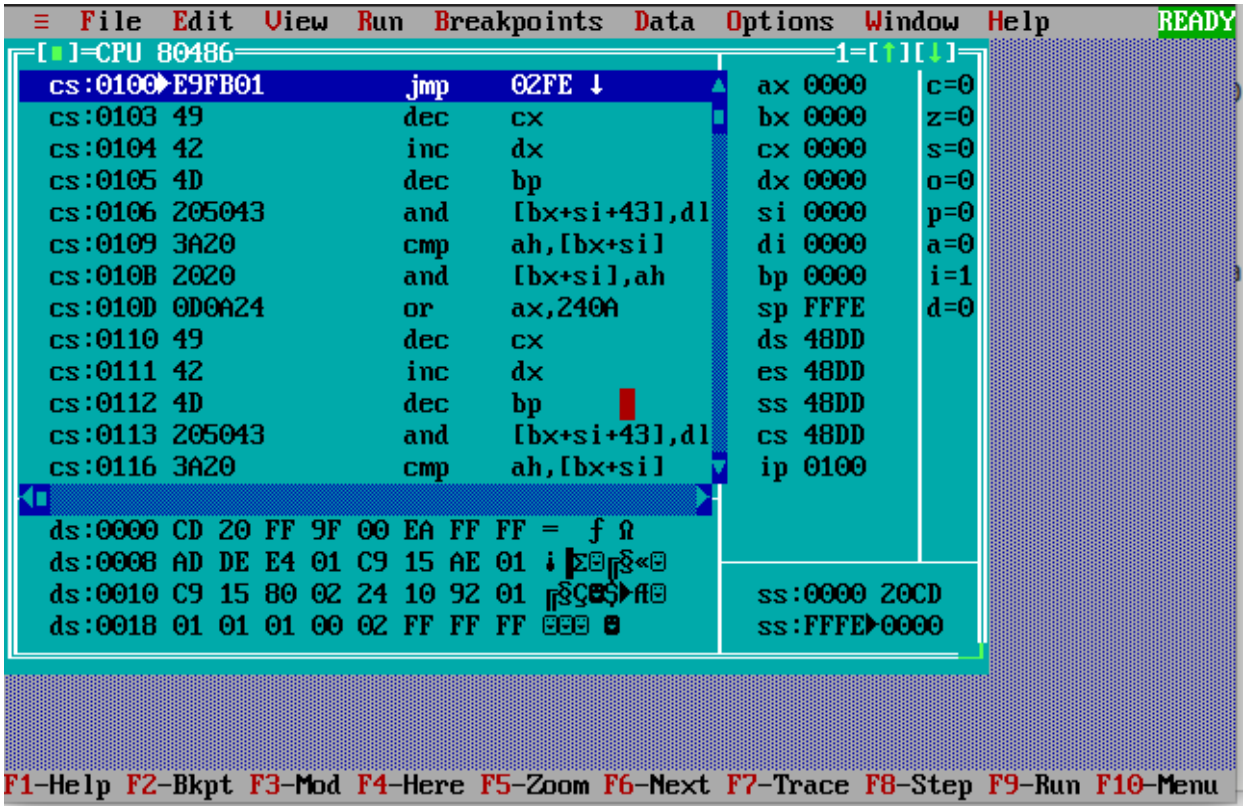

Рис. 4 – загрузочный модуль .com

Загрузка «хорошего» .exe модуля в основную память:

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

   В начале определяется сегментный адрес свободного участка памяти, размер которого достаточен для размещения программы. Затем создается и заполняется блок памяти для переменных среды. Создается блок памяти

для PSP и программы (сегмент:0000h – PSP; сегмент+0010h:0000h – программа). В поля PSP заносятся соответствующие значения. В рабочую область загрузчика считывается форматированная часть заголовка EXE-файла. Инициализируются регистры, выполняется программа. ES=DS=PSP=48DD. SS = 48ED. CS=497D.

2. На что указывают DS и ES?

   На начало PSP.

3. Как определяется стэк?

   В программе при помощи регистров SS и SP. В коде стэк определяется при помощи описания стэкового сегмента в коде и директивы ASSUME.

4. Как определяется точка входа?

   Точка входа определяется при помощи директивы END.

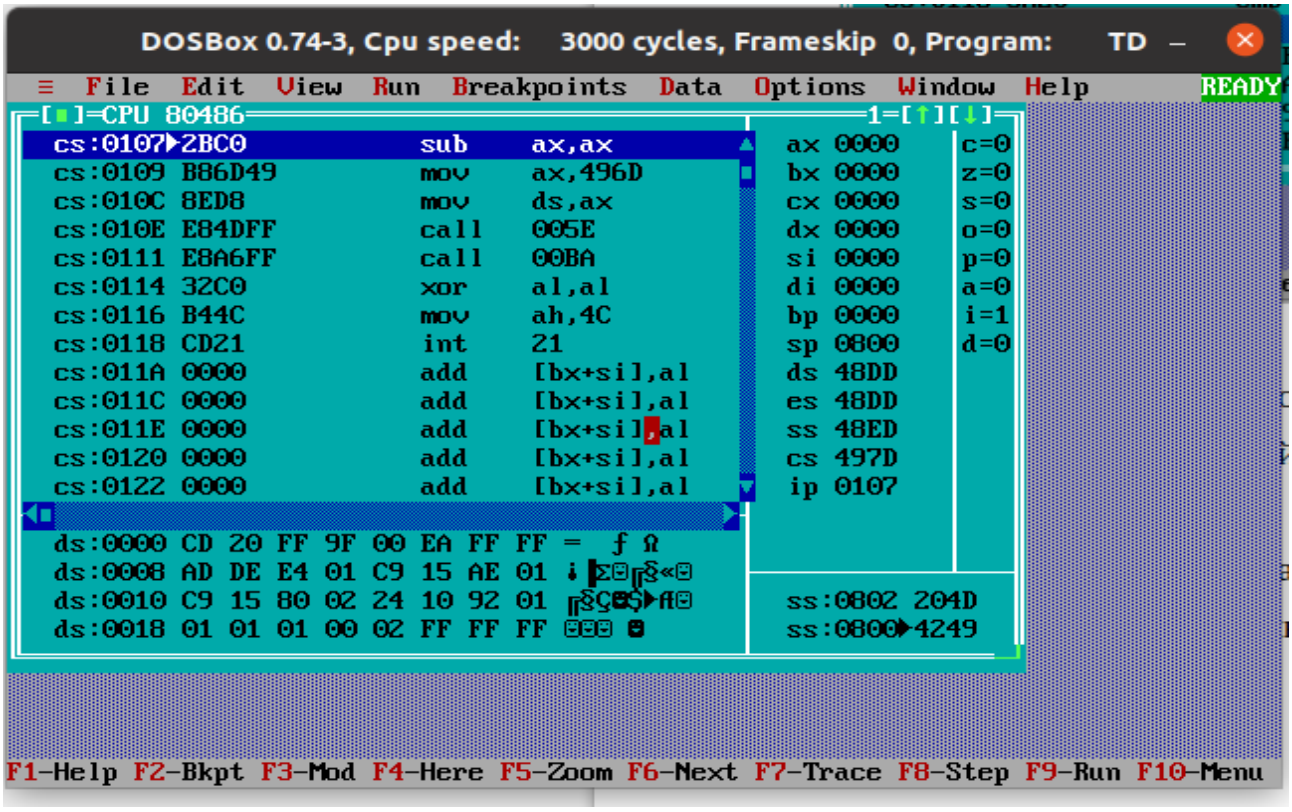

Рис. 5 – загрузочный модуль .exe

Выводы.

В ходе работы были изучены основные принципы устройства .com и .exe исполняемых модулей, их сходства и различия, алгоритм загрузки в память и инициализации начального состояния. Написана программа, выводящая строковую информацию о версии операционной системы DOS.

ПРИЛОЖЕНИЕ А.

Исходный код модулей

lab1_com.asm:

```
comprogramm SEGMENT
 ASSUME CS:comprogramm, DS:comprogramm, ES:NOTHING, SS:NOTHING
 ORG 100H
START: JMP BEGIN
; ДАННЫЕ
      MODEL0 DB 'IBM PC:   ',0DH,0AH,'$'
      MODEL1 DB 'IBM PC: PC',0DH,0AH,'$'
      MODEL2 DB 'IBM PC: PC/XT',0DH,0AH,'$'
      MODEL3 DB 'IBM PC: AT',0DH,0AH,'$'
      MODEL4 DB 'IBM PC: PS2 model 30',0DH,0AH,'$'
      MODEL5 DB 'IBM PC: PS2 model 50 or 60',0DH,0AH,'$'
      MODEL6 DB 'IBM PC: PS2 model 80',0DH,0AH,'$'
      MODEL7 DB 'IBM PC: PCjr',0DH,0AH,'$'
      MODEL8 DB 'IBM PC: PC Convertible',0DH,0AH,'$'
      VER    DB 'MS DOS VERSION:  .  ',0DH,0AH,'$'
      OEM    DB 'OEM SERIAL NUMBER:    ',0DH,0AH,'$'
      USER  DB 'USER SERIAL NUMBER:    ',0DH,0AH,'$'
;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 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 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
;--------------------------------
PRINT PROC NEAR
  push AX
  mov AH, 09h
  int 21h
  pop AX
  ret
PRINT ENDP
;--------------------------------
PC_MODEL PROC NEAR
           push AX
           push DX
           push ES
           mov AX, 0F000h
           mov ES, AX
           mov AL, ES:[0FFFEh]

           cmp AL, 0FFh
           mov DX, offset MODEL1
           je result
           cmp AL, 0FEh
           mov DX, offset MODEL2
           je result
           cmp AL, 0FBh
           je result
           cmp AL, 0FCh
           mov DX, offset MODEL3
           je result
           cmp AL, 0FAh
           mov DX, offset MODEL4
           je result
```

```
            cmp AL, 0FCh
            mov DX, offset MODEL5
            je result
            cmp AL, 0F8h
            mov DX, offset MODEL6
            je result
            cmp AL, 0FDh
            mov DX, offset MODEL7
            je result
            cmp AL, 0F9h
            mov DX, offset MODEL8
            je result

            call BYTE_TO_HEX
            mov MODEL0[13], AL
            mov MODEL0[14], AH
            mov DX, offset MODEL0

            result:
            call PRINT
            pop ES
            pop DX
            pop AX
            ret
      PC_MODEL ENDP
;-------------------------------
SYSTEM_VER PROC NEAR
            push AX
            push BX
            push CX
            push DI
            push SI

            sub AX,AX
            mov AH, 30h
            int 21h

            ;Ver
            mov SI, offset VER
            add SI, 16
            call BYTE_TO_DEC
            mov AL, AH
            add SI, 3
            call BYTE_TO_DEC
            mov DX, offset VER
            call PRINT

            ; OEM
            mov AL, BH
            mov DX, offset OEM
            mov SI, DX
            add SI, 21
            call BYTE_TO_DEC
            call PRINT

            ; User
            mov AX, CX
            mov DX, offset USER
            mov DI, DX
            add DI, 23
```

```
            call WRD_TO_HEX
            mov AL, BL
            call BYTE_TO_HEX
            call PRINT

            pop SI
            pop DI
            pop CX
            pop BX
            pop AX
            ret
      SYSTEM_VER ENDP
;-------------------------------
; КОД
BEGIN:

; Вывод строки текста из поля STRING
  call PC_MODEL
  call SYSTEM_VER
  ; Выход в DOS
  xor AL,AL
  mov AH,4Ch
  int 21H
comprogramm ENDS
 END START
```

lab1_exe.asm:

```
ASSUME CS:exeprogramm, DS:DATA, SS:ASTACK

ASTACK SEGMENT STACK
      DW 1024 DUP(?)
ASTACK ENDS

; ДАННЫЕ
DATA SEGMENT
      MODEL0 DB 'IBM PC:   ',0DH,0AH,'$'
      MODEL1 DB 'IBM PC: PC',0DH,0AH,'$'
      MODEL2 DB 'I BM PC: PC/XT',0DH,0AH,'$'
      MODEL3 DB 'IBM PC: AT',0DH,0AH,'$'
      MODEL4 DB 'IBM PC: PS2 model 30',0DH,0AH,'$'
      MODEL5 DB 'IBM PC: PS2 model 50 or 60',0DH,0AH,'$'
      MODEL6 DB 'IBM PC: PS2 model 80',0DH,0AH,'$'
      MODEL7 DB 'IBM PC: PCjr',0DH,0AH,'$'
      MODEL8 DB 'IBM PC: PC Convertible',0DH,0AH,'$'
      VER    DB 'MS DOS VERSION:  .  ',0DH,0AH,'$'
      OEM    DB 'OEM SERIAL NUMBER:    ',0DH,0AH,'$'
      USER  DB 'USER SERIAL NUMBER:    ', 0DH,0AH,'$'
DATA ENDS

exeprogramm SEGMENT
;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 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 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
;--------------------------------
PRINT PROC NEAR
  push AX
```

```
  mov AH, 09h
  int 21h
  pop AX
  ret
PRINT ENDP
;-------------------------------
PC_MODEL PROC NEAR
          push AX
          push DX
          push ES
          mov AX, 0F000h
          mov ES, AX
          mov AL, ES:[0FFFEh]

          cmp AL, 0FFh
          mov DX, offset MODEL1
          je result
          cmp AL, 0FEh
          mov DX, offset MODEL2
          je result
          cmp AL, 0FBh
          je result
          cmp     AL, 0FCh
          mov DX, offset MODEL3
          je result
          cmp AL, 0FAh
          mov DX, offset MODEL4
          je result
          cmp AL, 0FCh
          mov DX, offset MODEL5
          je result
          cmp AL, 0F8h
          mov DX, offset MODEL6
          je result
          cmp AL, 0FDh
          mov DX, offset MODEL7
          je result
          cmp AL, 0F9h
          mov DX, offset MODEL8
          je result

          call BYTE_TO_HEX
          mov MODEL0[13], AL
          mov MODEL0[14], AH
          mov DX, offset MODEL0

          result:
          call PRINT
          pop ES
          pop DX
          pop AX
          ret
     PC_MODEL ENDP
;-------------------------------
SYSTEM_VER PROC NEAR
          push AX
          push BX
          push CX
          push DI
          push SI
```

```
            sub AX,AX
            mov AH, 30h
            int 21h

            ; Version
            mov SI, offset VER
            add SI, 16
            call BYTE_TO_DEC
            mov AL, AH
            add SI, 3
            call BYTE_TO_DEC
            mov DX, offset VER
            call PRINT

            ; OEM Serial Number
            mov AL, BH
            mov DX, offset OEM
            mov SI, DX
            add SI, 21
            call BYTE_TO_DEC
            call PRINT

            ; User Serial Number
            mov AX, CX
            mov DX, offset USER
            mov DI, DX
            add DI, 23
            call WRD_TO_HEX
            mov AL, BL
            call BYTE_TO_HEX
            call PRINT

            pop SI
            pop DI
            pop CX
            pop BX
            pop AX
            ret
      SYSTEM_VER ENDP
;-------------------------------
; КОД
BEGIN PROC NEAR
    sub AX, AX
    mov AX, DATA
    mov DS, AX

    call PC_MODEL
    call SYSTEM_VER


    ; Выход в DOS
    xor AL,AL
    mov AH,4Ch
    int 21H
BEGIN ENDP
exeprogramm ENDS
 END BEGIN ;конец модуля, START - точка входа
```