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 _____ Санников В.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Исследовать 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 будет «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля.

2) Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Пункте 1 и построить и отладить его. Таким образом, будет получен «хороший» .EXE.

3) Сравнить исходные тексты для .COM и .EXE модулей. Ответить на контрольные вопросы.

4) С помощью FAR ответить на контрольные вопросы.

5) С помощью отладчика TD.exe дать ответы на контрольные вопросы с помощью .COM и .EXE модулей.

6) Оформление отчета.

Ход работы.

В ходе данной лабораторной работы были выполнены следующие пункты:

1) Для выполнения первого пункта был написан .COM модуль, в котором присутствуют функции преобразования двоичных кодов в символы шестнадцатеричных и десятичных чисел, функции определения типа компьютера и версии системы DOS. Следовательно мы имеем «хороший» .COM модуль и «плохой» .EXE модуль. Результаты первого шага приведенные на рисунках 1 и 2.

Рисунок 1 — Выполнение «хорошего» .COM модуля

Рисунок 2 — Выполнение «плохого» .EXE модуля

2) Чтобы написать «хороший» .EXE модуль мы разбиваем программу на сегмент данных, кода, стека и добавляем главную процедуру MAIN. Результат запуска данного модуля представлен на рисунке 3.

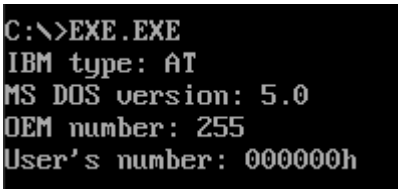

Рисунок 3 — Выполнение «хорошего» .EXE модуля

3) Теперь сравним данные модули и ответим на контрольные вопросы:

- Сколько сегментов должна содержать COM-программа?

  Программа должна содержать один сегмент (код, даннные, стек, который генерируется сам)

- EXE-программа?

  Обязательно только сегмент кода, сегмент стека и данных описываются отдельно друг от друга.

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

  Директива ORG 100h нужна обязательно, так как первые 256 байт занимает PSP. Нам необходимо смещение адресации в 256 байт от нулевого адреса. И директива ASSUME, которая указывает, с каким сегментом нужно связать регистры сегмента кода и сегмента данных.

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

  Нельзя использовать команды с указанием сегментов, так как сегментные регистры в .COM-программе определяются в момент ее запуска, а не при линковке, поскольку не имеет relocation table.

  4) На основе шестнадцатеричных модулей ответим на следующие контрольные вопросы:

- Какова структура файла .COM? С какого адреса располагается код?

  .COM модуль состоит из одного сегмента, который содержит код и данные. Код располагается с нулевого адреса. PSP занимает 100h байт, поэтому устанавливается смещение 100h. (см Рисунок 4)

Рисунок 4 - «хороший» .COM-модуль в 16-м виде (первая строка — начало сегмента, последняя строка — конец сегмента)

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

  .EXE состоит из одного сегмента. Код располагается с адреса 300h (200h — заголовок и relocation table, 100h — смещение). С адреса 0h располагается заголовок .EXE файла. Модуль в 16-м виде представлен на рисунке 5.

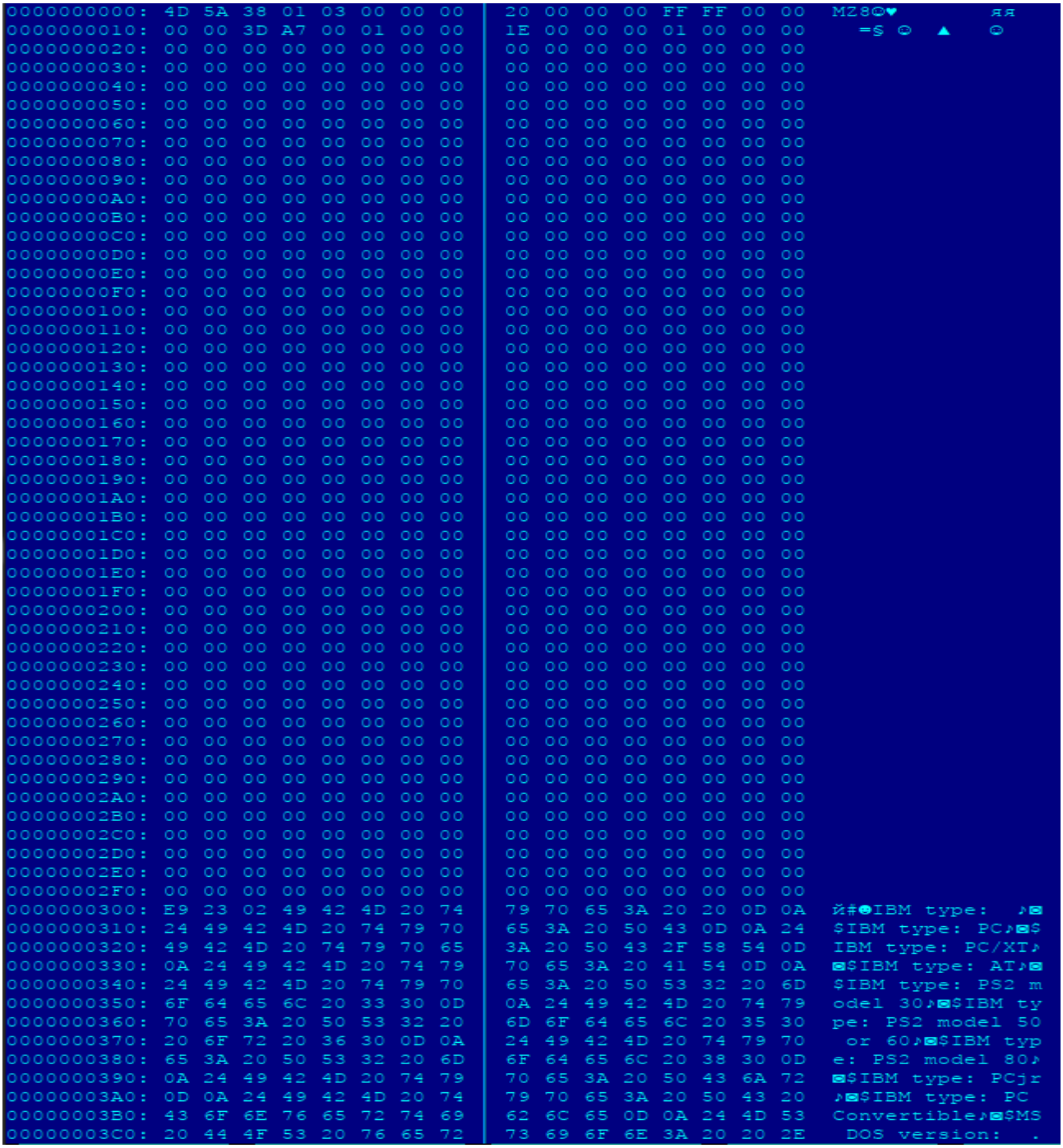

Рисунок 5 - «плохой» .EXE в 16-м виде (строка 0000000300 — начало кода, последняя строка конец сегмента)

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

  «Хороший» .EXE имеет следующие сегмента: стека, данных и кода, также в начале модуля располагается заголовок и relocation table (200h байт). Модуль «хорошего» .EXE см на рисунке 6.

Рисунок 6 - «хороший» .EXE модуль в 16-м виде (0200 — начало сегмента стека, 0400 — начало сегмента данных, 0500 — начало сегмента кода

5) Ответим на контрольные вопросы по отладчику TD.exe. Загрузку в отладчик «хорошего» .COM модуля см на рисунке 7.

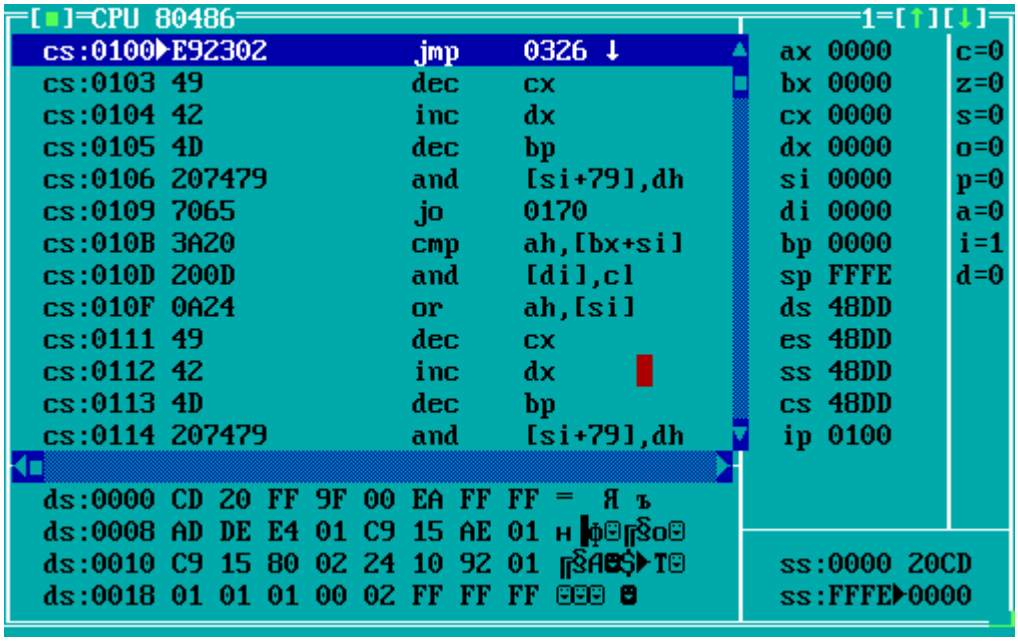

Рисунок 7 - «хороший» .COM модуль в отладчике

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

  Сначала определяется сегментные адрес участка ОП, у которого есть место для загрузки программы. Затем создается блок памяти для PSP и программы. Начиная с PSP:0100h помещается считанные образ COM-файла. Код располгается с адреса CS:0100h.

- Что располагается с адреса 0?

  Сегмент PSP.

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

  Регистры CS, DS, ES, SS указывают на PSP и имеют значение в данном случае 48DDh (см Рисунок 7)

- Как определяется стек? Какую область памяти он занимает? Какие адреса?

  Для .COM файлов стек определяется с FFFEh до 0000h. SS указывается на начало сегмента = 48DDh; SP указывается на последний адрес кратный двум = FFFEh. Значит адреса стека 48DD:0000h — 48DD:FFFEh.

6) Загрузим «хороший» .EXE модуль в отладчик TD.exe и ответим на контрольные вопросы. См рисунок 8.

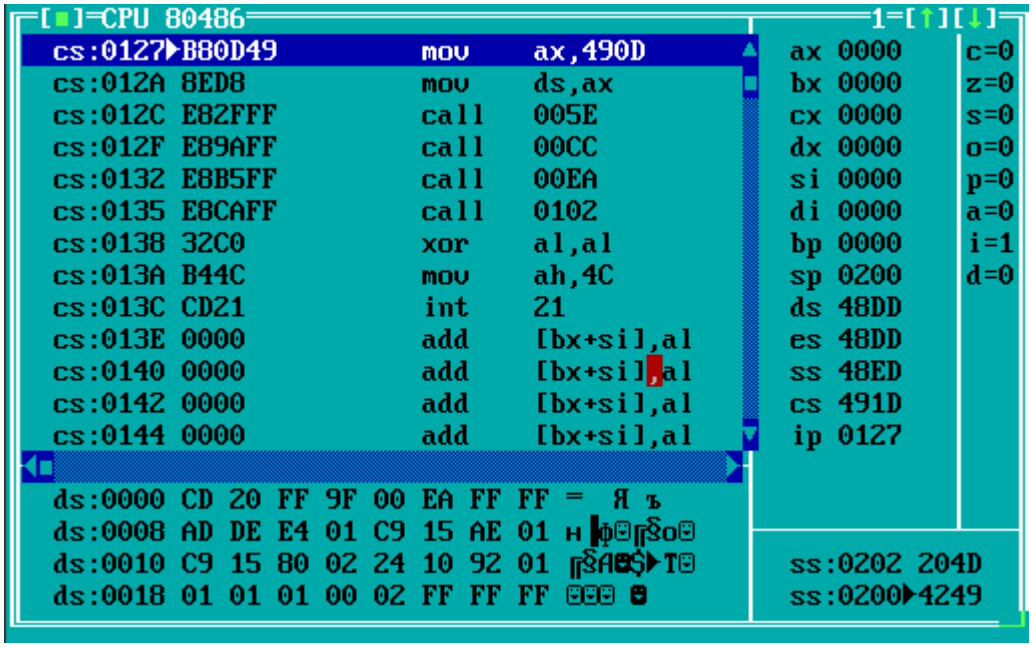

Рисунок 8 - «хороший» .EXE модуль в отладчике

- Как загружается «хороший» .EXE? Какие значения имеют сегментные регистры?

  EXE-файл загружается, начиная с адреса PSP:0100h. В процессе загрузки считывается информация заголовка (PSP) .EXE в начале файла и выполняется перемещение адресов сегментов, то есть DS и ES устанавливаются на начало сегмента PSP(DS=ES=48DD), SS(SS=48ED) – на начало сегмента стека, CS(CS=491D) – на начало сегмента команд. В IP загружается смещение точки входа в программу, которая берётся из метки после директивы END.

- На что указывают регистры DS и ES?

  На начало PSP.

- Как определяется стек?

  С помощью директивы .stack либо вручную через segmant.

- Как определяется точка входа?

Директивой END. Она определяет адрес, с которого начинается выполнение программы.

Вывод.

В ходе данной лабораторной работы был написан текст .COM и .EXE модулей, были изучены способы загрузки данных модулей в основную память и различия между двумя типами модулей. Данные файлы отвечали за вывод на экран информации о типе PC и версии системы DOS.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММ

Файл COM.asm

```
TESTPC SEGMENT
     ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
     ORG  100H
START:    jmp BEGIN
;-------------------------------
NOTYPE db 'IBM type:  ', 0DH, 0AH, '$'
PC db 'IBM type: PC', 0DH, 0AH, '$'
XT db 'IBM type: PC/XT', 0DH, 0AH, '$'
AT db 'IBM type: AT', 0DH, 0AH, '$'
PS230 db 'IBM type: PS2 model 30', 0DH, 0AH, '$'
PS250_60 db 'IBM type: PS2 model 50 or 60', 0DH, 0AH, '$'
PS280 db 'IBM type: PS2 model 80', 0DH, 0AH, '$'
PCjr db 'IBM type: PCjr', 0DH, 0AH, '$'
PC_Convertible db 'IBM type: PC Convertible', 0DH, 0AH, '$'
DOSV db 'MS DOS version:  .  ', 0DH, 0AH, '$'
OEM db 'OEM number:   ', 0DH, 0AH, '$'
USR db "User's number:       h", 0DH, 0AH, "$"
;-------------------------------
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 16-го 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, в 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_1
      or AL, 30h
      mov [SI], AL
end_1:  pop DX
      pop CX
      ret
BYTE_TO_DEC ENDP
;----------------------
PRINT PROC near
      push AX
      mov AH, 09h
      int 21h
      pop AX
      ret
PRINT ENDP
;----------------------
; Определение одной из систем
PC_VERSION PROC near
      mov AX, 0F000h
      mov ES, AX
      mov AL, ES:[0FFFEh]

      cmp AL, 0FFh
      je print_pc
```

```
      cmp AL, 0FBh
      je print_XT

      cmp AL, 0FCh
      je print_AT

      cmp AL, 0FAh
      je print_ps230

      cmp AL, 0FCh
      je print_ps250_60

      cmp AL, 0F8h
      je print_ps280

      cmp AL, 0FDh
      je print_PCjr

      cmp AL, 0F9h
      je print_pc_convertible

      mov DI, offset NOTYPE
      add DI, 10
      call BYTE_TO_HEX
      mov [DI], AX
      mov DX, offset NOTYPE
      jmp final

print_pc:
      mov DX, offset PC
      jmp final

print_XT:
      mov DX, offset XT
      jmp final

print_AT:
      mov DX, offset AT
      jmp final

print_ps230:
      mov DX, offset PS230
      jmp final

print_ps250_60:
      mov DX, offset ps250_60
      jmp final

print_ps280:
      mov DX, offset ps280
      jmp final
```

```
print_PCjr:
      mov DX, offset PCjr
      jmp final

print_pc_convertible:
      mov DX, offset PC_Convertible
      jmp final

final:
      call PRINT
      ret
PC_VERSION ENDP
;-----------------------------------
DOS_VERSION PROC near
      xor AX, AX
      mov AH, 30h
      int 21h

      mov SI, offset DOSV
      add SI, 16
      call BYTE_TO_DEC
      mov AL, AH
      add SI, 3
      call BYTE_TO_DEC
      mov DX, offset DOSV
      call PRINT
      ret
DOS_VERSION ENDP
;-----------------------------------
OEM_NUMBER PROC near
      xor AX, AX
      mov AH, 30h
      int 21h

      mov SI, offset OEM
      add SI, 14
      mov AL, BH
      call BYTE_TO_DEC
      mov DX, offset OEM
      call PRINT
      ret
OEM_NUMBER ENDP
;-----------------------------------
USER_NUMBER PROC near
      xor AX, AX
      mov AH, 30h
      int 21h

      mov DI, offset USR
      add DI, 20
      mov AX, CX
```

```
      call WRD_TO_HEX
      mov AL, BL
      call BYTE_TO_HEX
      mov di, offset USR
      add DI, 15
      mov [DI], AX
      mov DX, offset USR
      call PRINT
      ret
USER_NUMBER ENDP
;---------------------------------
BEGIN:
      call PC_VERSION
      call DOS_VERSION
      call OEM_NUMBER
      call USER_NUMBER
      xor AL, AL
      mov AH, 4Ch
      int 21h
TESTPC ENDS
      END START
```

Файл EXE.asm

```
AStack SEGMENT STACK
      DW 256 DUP(?)
AStack ENDS
;-------------------------------
DATA SEGMENT
      NOTYPE db 'IBM type:  ', 0DH, 0AH, '$'
      PC db 'IBM type: PC', 0DH, 0AH, '$'
      XT db 'IBM type: PC/XT', 0DH, 0AH, '$'
      AT db 'IBM type: AT', 0DH, 0AH, '$'
      PS230 db 'IBM type: PS2 model 30', 0DH, 0AH, '$'
      PS250_60 db 'IBM type: PS2 model 50 or 60', 0DH, 0AH, '$'
      PS280 db 'IBM type: PS2 model 80', 0DH, 0AH, '$'
      PCjr db 'IBM type: PCjr', 0DH, 0AH, '$'
      PC_Convertible db 'IBM type: PC Convertible', 0DH, 0AH, '$'
      DOSV db 'MS DOS version:  .  ', 0DH, 0AH, '$'
      OEM db 'OEM number:   ', 0DH, 0AH, '$'
      USR db "User's number:       h", 0DH, 0AH, "$"
DATA ENDS
;-------------------------------
TESTPC SEGMENT
      ASSUME CS:TESTPC, 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 16-го 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, в 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_1
```

```
      or AL, 30h
      mov [SI], AL
end_1:  pop DX
      pop CX
      ret
BYTE_TO_DEC ENDP
;----------------------
PRINT PROC near
      push AX
      mov AH, 09h
      int 21h
      pop AX
      ret
PRINT ENDP
;----------------------
; Определение одной из систем
PC_VERSION PROC near
      mov AX, 0F000h
      mov ES, AX
      mov AL, ES:[0FFFEh]

      cmp AL, 0FFh
      je print_pc

      cmp AL, 0FBh
      je print_XT

      cmp AL, 0FCh
      je print_AT

      cmp AL, 0FAh
      je print_ps230

      cmp AL, 0FCh
      je print_ps250_60

      cmp AL, 0F8h
      je print_ps280

      cmp AL, 0FDh
      je print_PCjr

      cmp AL, 0F9h
      je print_pc_convertible

      mov DI, offset NOTYPE
      add DI, 10
      call BYTE_TO_HEX
      mov [DI], AX
      mov DX, offset NOTYPE
      jmp final
```

```
print_pc:
      mov DX, offset PC
      jmp final

print_XT:
      mov DX, offset XT
      jmp final

print_AT:
      mov DX, offset AT
      jmp final

print_ps230:
      mov DX, offset PS230
      jmp final

print_ps250_60:
      mov DX, offset ps250_60
      jmp final

print_ps280:
      mov DX, offset ps280
      jmp final

print_PCjr:
      mov DX, offset PCjr
      jmp final

print_pc_convertible:
      mov DX, offset PC_Convertible
      jmp final

final:
      call PRINT
      ret
PC_VERSION ENDP
;-----------------------------------
DOS_VERSION PROC near
      xor AX, AX
      mov AH, 30h
      int 21h

      mov SI, offset DOSV
      add SI, 16
      call BYTE_TO_DEC
      mov AL, AH
      add SI, 3
      call BYTE_TO_DEC
      mov DX, offset DOSV
      call PRINT
      ret
DOS_VERSION ENDP
```

```
;----------------------------------
OEM_NUMBER PROC near
      xor AX, AX
      mov AH, 30h
      int 21h

      mov SI, offset OEM
      add SI, 14
      mov AL, BH
      call BYTE_TO_DEC
      mov DX, offset OEM
      call PRINT
      ret
OEM_NUMBER ENDP
;----------------------------------
USER_NUMBER PROC near
      xor AX, AX
      mov AH, 30h
      int 21h

      mov DI, offset USR
      add DI, 20
      mov AX, CX
      call WRD_TO_HEX
      mov AL, BL
      call BYTE_TO_HEX
      mov di, offset USR
      add DI, 15
      mov [DI], AX
      mov DX, offset USR
      call PRINT
      ret
USER_NUMBER ENDP
;----------------------------------
MAIN PROC far
      mov ax, data
      mov ds, ax
      call PC_VERSION
      call DOS_VERSION
      call OEM_NUMBER
      call USER_NUMBER

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