МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по практической работе №1

по дисциплине «Операционные системы»

Тема: Исследование структур загрузочных модулей

Студент гр. 9382 _____ Дерюгин Д.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

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

Задание.

Шаг 1. Напишите текст исходного .COM модуля, который определяет тип PC и версию системы. Это довольно простая задача и для тех, кто уже имеет опыт программирования на ассемблере, это будет небольшой разминкой. Для тех, кто раньше не сталкивался с программированием на ассемблере, это неплохая задача для первого опыта.

За основу возьмите шаблон, приведенный в разделе «Основные сведения». Необходимые сведения о том, как извлечь требуемую информацию, представлены в следующем разделе.

Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип PC и выводить строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код переводиться в символьную строку, содержащую запись шестнадцатеричного числа и выводиться на экран в виде соответствующего сообщения.

Затем определяется версия системы. Ассемблерная программа должна по значениям регистров AL и AH формировать текстовую строку в формате xx.yy, где xx - номер основной версии, а yy - номер модификации в десятичной системе счисления, формировать строки с серийным номером OEM и серийным номером пользователя. Полученные строки выводятся на экран.

Отладьте полученный исходный модуль.

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

Шаг 2. Напишите текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1 и постройте и отладьте его. Таким образом, будет получен «хороший» .EXE.

Шаг 3. Сравните исходные тексты для .COM и .EXE модулей. Ответьте на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

Шаг 4. Запустите FAR и откройте (F3/F4) файл загрузочного модуля .COM и файл «плохого» .EXE в шестнадцатеричном виде. Затем откройте (F3/F4) файл загрузочного модуля «хорошего» .EXE и сравните его с предыдущими файлами. Ответьте на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

Шаг 5. Откройте отладчик TD.EXE и загрузите .COM. Ответьте на контрольные вопросы «Загрузка COM модуля в основную память». Представьте в отчете план загрузки модуля .COM в основную память.

Шаг 6. Откройте отладчик TD.EXE и загрузите «хороший» .EXE. Ответьте на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

Шаг 7. Оформление отчета в соответствии с требованиями. В отчете необходимо привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей – в отладчике.

Необходимые сведения для составления программы

Тип IBM PC хранится в байте по адресу 0F000:0FFFEh, в предпоследнем байте ROM BIOS. Соответствие кода и типа в таблице:

PC	FF
PC/XT	FE,FB
AT	FC
PS2 модель 30	FA
PS2 модель 50 или 60	FC
PS2 модель 80	F8
PCjr	FD

PC Convertible F9

Для определения версии MS DOS следует воспользоваться функцией 30H прерывания 21H. Входным параметром является номер функции в AH:

MOV AH,30h

INT 21h

Выходными параметрами являются:

AL - номер основной версии. Если 0, то < 2.0

AH - номер модификации

BH - серийный номер OEM (Original Equipment Manufacturer) BL:CX - 24-битовый серийный номер пользователя.

Выполнение работы.

1.Написан «хороший» модуль .com, который определяет тип PC и версию системы, а также на его базе был написан «плохой» .exe модуль.

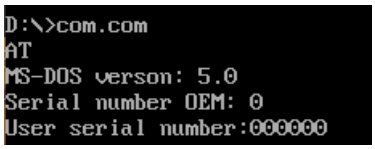


Рис.1 Результат работы «хорошего» .com модуля

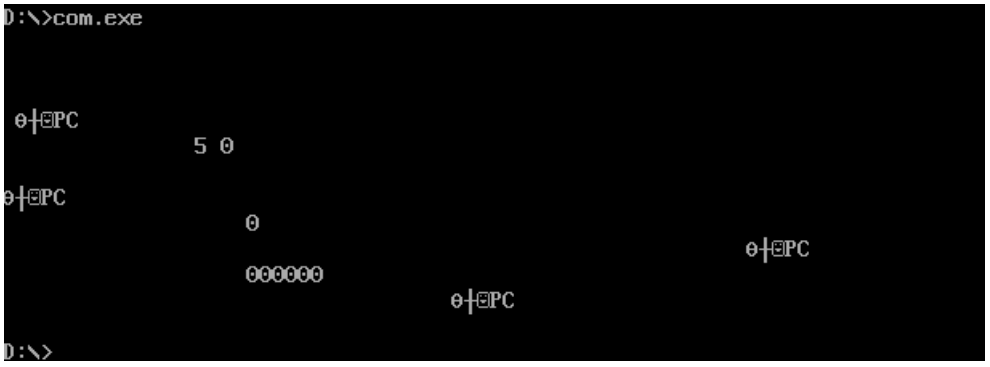


Рис.2 Результат работы «плохого» .exe модуля

2. Написан текст «хорошего» .exe модуля на основе .com модуля

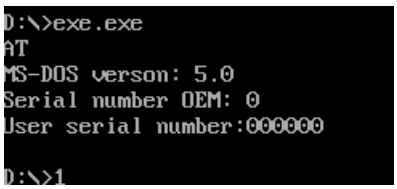


Рис.3 Результат работы «хорошего» .exe модуля

3.

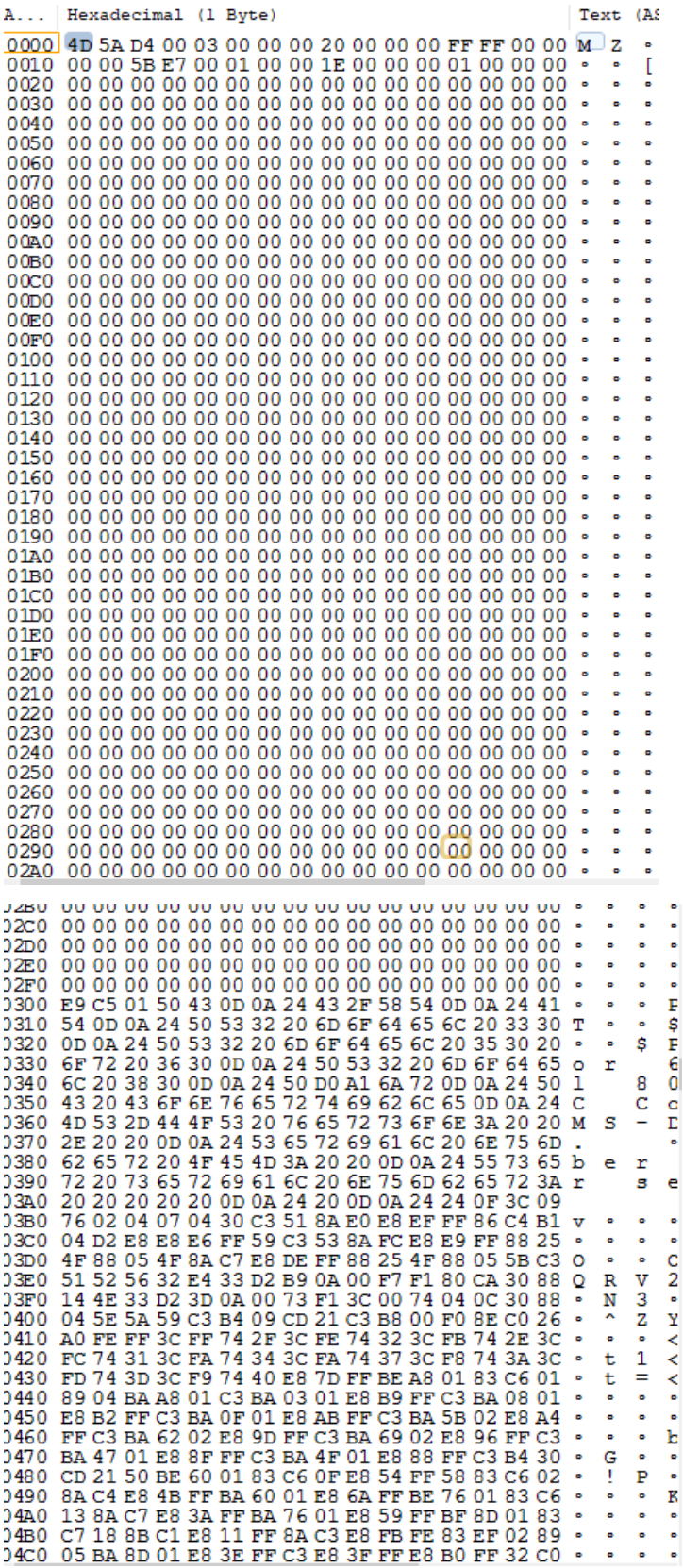
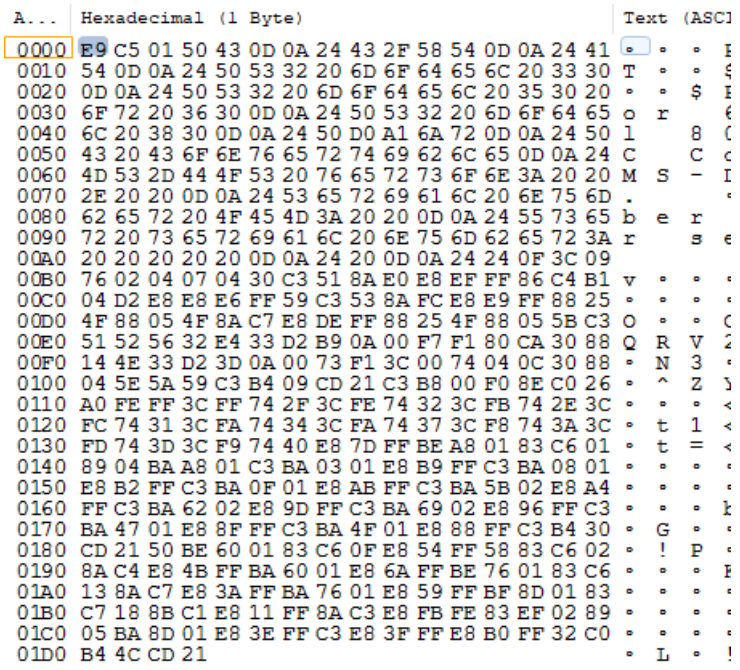


Рис.4 Файл загрузочного модуля «плохого» .EXE в 16-ном виде



A...	Hexadecimal (1 Byte)	Text (ASCI
0000	E9 C5 01 50 43 0D 0A 24 43 2F 58 54 0D 0A 24 41	∘ ∘ ∘
0010	54 0D 0A 24 50 53 32 20 6D 6F 64 65 6C 20 33 30	T ∘ ∘
0020	0D 0A 24 50 53 32 20 6D 6F 64 65 6C 20 35 30 20	∘ ∘ $
0030	6F 72 20 36 30 0D 0A 24 50 53 32 20 6D 6F 64 65	o r
0040	6C 20 38 30 0D 0A 24 50 D0 A1 6A 72 0D 0A 24 50	l 8
0050	43 20 43 6F 6E 76 65 72 74 69 62 6C 65 0D 0A 24	C C
0060	4D 53 2D 44 4F 53 20 76 65 72 73 6F 6E 3A 20 20	M S -
0070	2E 20 20 0D 0A 24 53 65 72 69 61 6C 20 6E 75 6D	.
0080	62 65 72 20 4F 45 4D 3A 20 20 0D 0A 24 55 73 65	b e r
0090	72 20 73 65 72 69 61 6C 20 6E 75 6D 62 65 72 3A	r s
00A0	20 20 20 20 20 0D 0A 24 20 0D 0A 24 24 0F 3C 09	
00B0	76 02 04 07 04 30 C3 51 8A E0 E8 EF FF 86 C4 B1	v ∘ ∘
00C0	04 D2 E8 E8 E6 FF 59 C3 53 8A FC E8 E9 FF 88 25	∘ ∘ ∘
00D0	4F 88 05 4F 8A C7 E8 DE FF 88 25 4F 88 05 5B C3	O ∘ ∘
00E0	51 52 56 32 E4 33 D2 B9 0A 00 F7 F1 80 CA 30 88	Q R V
00F0	14 4E 33 D2 3D 0A 00 73 F1 3C 00 74 04 0C 30 88	∘ N 3
0100	04 5E 5A 59 C3 B4 09 CD 21 C3 B8 00 F0 8E C0 26	∘ ^ Z
0110	A0 FE FF 3C FF 74 2F 3C FE 74 32 3C FB 74 2E 3C	∘ ∘ ∘
0120	FC 74 31 3C FA 74 34 3C FA 74 37 3C F8 74 3A 3C	∘ t 1
0130	FD 74 3D 3C F9 74 40 E8 7D FF BE A8 01 83 C6 01	∘ t =
0140	89 04 BA A8 01 C3 BA 03 01 E8 B9 FF C3 BA 08 01	∘ ∘ ∘
0150	E8 B2 FF C3 BA 0F 01 E8 AB FF C3 BA 5B 02 E8 A4	∘ ∘ ∘
0160	FF C3 BA 62 02 E8 9D FF C3 BA 69 02 E8 96 FF C3	∘ ∘ ∘
0170	BA 47 01 E8 8F FF C3 BA 4F 01 E8 88 FF C3 B4 30	∘ G ∘
0180	CD 21 50 BE 60 01 83 C6 0F E8 54 FF 58 83 C6 02	∘ ! P
0190	8A C4 E8 4B FF BA 60 01 E8 6A FF BE 76 01 83 C6	∘ ∘ ∘
01A0	13 8A C7 E8 3A FF BA 76 01 E8 59 FF BF 8D 01 83	∘ ∘ ∘
01B0	C7 18 8B C1 E8 11 FF 8A C3 E8 FB FE 83 EF 02 89	∘ ∘ ∘
01C0	05 BA 8D 01 E8 3E FF C3 E8 3F FF E8 B0 FF 32 C0	∘ ∘ ∘
01D0	B4 4C CD 21	∘ L ∘ !

Рис.5 Файл загрузочного модуля .COM в 16-ном виде

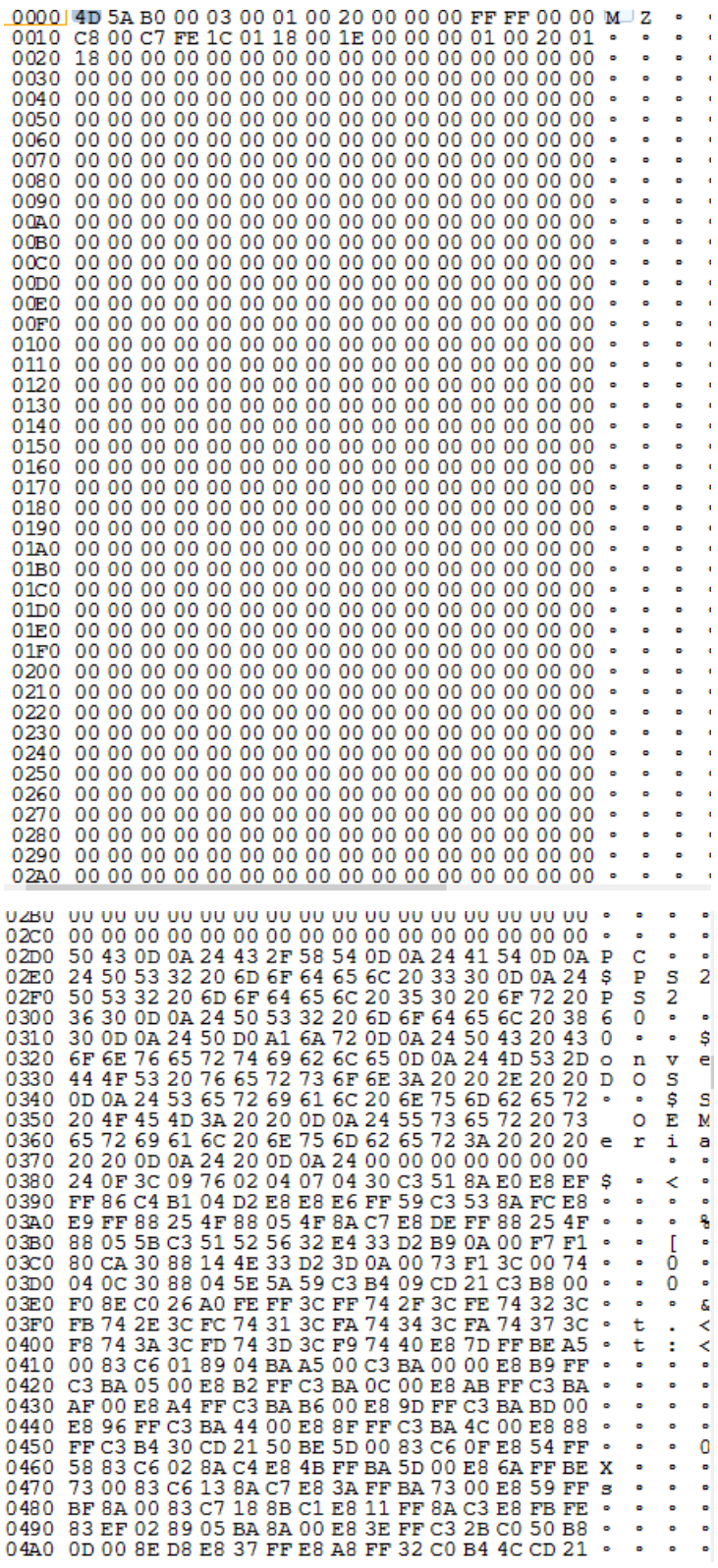


Рис.6 Файл загрузочного модуля «хорошего» .EXE в 16-ном виде

4.

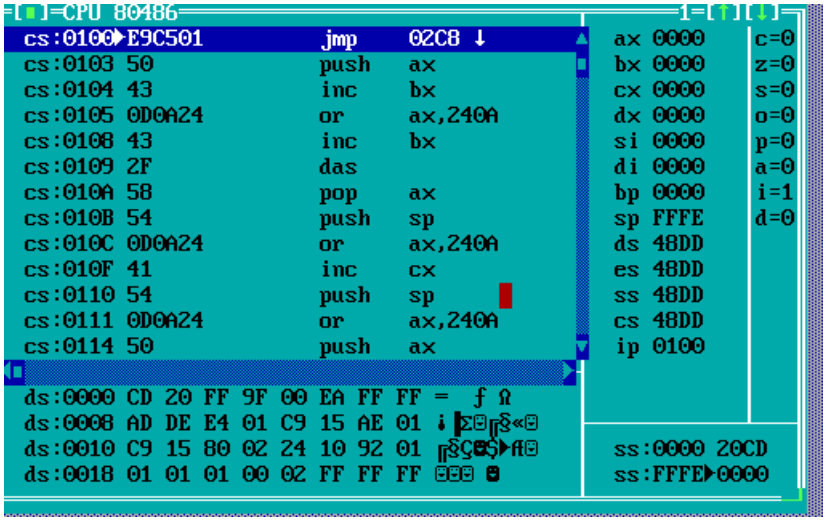


Рис. 7 Отладчик TD.EXE для .COM

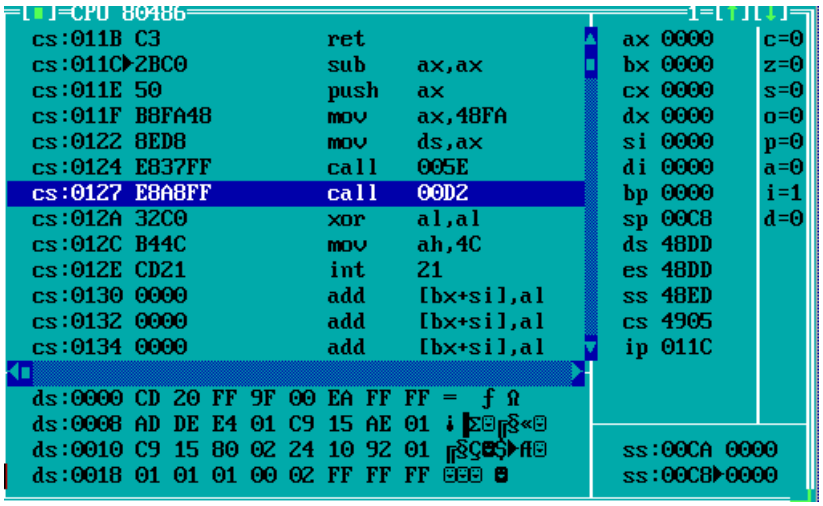


Рис. 8 Отладчик TD.EXE для .EXE

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

Отличия исходных текстов COM и EXE программ:

1. Сколько сегментов должна содержать COM-программа?

   COM-программа должна содержать один сегмент.

2. EXE-программа?

   >=1 сегмента.

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

   org 100h – обязательная директива для COM-программы, подгружается PSP размером в 100h.

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

Не все.

Нельзя использовать перемещение сегмента

Инструкция `MOV AX, DATA` ссылается на сегмент `DATA`, а значение `DATA` зависит от того, где MS-DOS загружает исполняемый файл. Это требует перемещения в исполняемом файле, чтобы MS-DOS знала, как изменить значение с правильным значением сегмента при его загрузке.

Отличия форматов файлов .COM и.EXE программ:

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

   Файл .COM состоит из одного сегмента и содержит как данные, так и сам код программы. Код начинается с адреса 0h.

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

   В «плохом» EXE все находится в одном сегменте.

   Код располагается с адреса 300h.

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

В «хорошем» EXE данные, стек и код находятся в разных сегментах. Код располагается с адреса 200h а не с 300h в «плохом» EXE.

Загрузка COM модуля в основную память:

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

   При загрузки модуля сегментные регистры указывают на начало PSP. Код начинается с адреса 100h.

2. Что располагается с адреса 0?

   Начало PSP

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

   Регистры имеют значения 48DD, указывают на PSP.

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

   Стек определяется автоматически при загрузки com.

Загрузка «хорошего» EXE модуля в основную память:

1. На что указывают регистры DS и ES?

   Регистры DS и ES указывают на начало PSP.

2. Как определяется стек?

   При помощи директивы .STACK, задав задается размер стека. Регистр SS указывет на начало сегмента стека, а SP на его конец.

3. Как определяется точка входа?

   Точка входа определяется при помощи END.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Com.asm

```
TESTPC SEGMENT
   ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
   ORG 100H
START: JMP BEGIN
; ДАННЫЕ
  PC_MODEL                        db 'PC',0DH,0AH,'$'
  XT_MODEL                        db 'C/XT',0DH,0AH,'$'
  AT_MODEL_STR                    db 'AT',0DH,0AH,'$'
  PS2_MODEL_MODEL_30          db 'PS2 model 30',0DH,0AH,'$'
  PS2_MODEL_MODEL_50_60       db 'PS2 model 50 or 60',0DH,0AH,'$'
  PS2_MODEL_MODEL_80          db 'PS2 model 80',0DH,0AH,'$'
  PCJS_MODEL                      db 'PCjr',0DH,0AH,'$'
  PC_CONVERTIBLE_MODEL        db 'PC Convertible',0DH,0AH,'$'

  MS_DOS_VERSION              db 'MS-DOS verson:  .  ',0DH, 0AH, '$'
  SERIAL_OEM                      db 'Serial number OEM:  ',0DH, 0AH, '$'
  USER_SERIAL_NUMER           db 'User serial number:     ',0DH, 0AH,'$'

  NOT_FOUND_MODEL             db    ' ',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 младцая
   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
   push SI
   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 SI
   pop DX
   pop CX

   ret
BYTE_TO_DEC ENDP
;--------------------------------
; КОД

PRINT PROC near
   mov AH,09h
   int 21h
   ret
PRINT ENDP



PC_INFO PROC near
    mov ax, 0f000h
      mov es, ax
      mov al, es:[0fffeh]

      cmp al, 0ffh
      je PC

      cmp al, 0feh
      je XT

      cmp al, 0fbh
```

```
        je XT

        cmp al, 0fch
        je AT_MODEL

        cmp al, 0fah
        je PS2_MODEL_30

        cmp al, 0fah
        je PS2_MODEL_50_60

        cmp al, 0f8h
        je PS2_MODEL_80

        cmp al, 0fdh
        je PSJR

        cmp al, 0f9h
        je PS_CONVERTIBLE


        call BYTE_TO_HEX
        mov   si, offset NOT_FOUND_MODEL
        add   si, 1
        mov   [si], ax
        mov dx,offset NOT_FOUND_MODEL
        ret

PC:
        mov dx, offset PC_MODEL
        call PRINT
        ret

XT:
        mov dx, offset XT_MODEL
        call PRINT
        ret

AT_MODEL:
        mov dx, offset AT_MODEL_STR
        call PRINT
```

```
      ret

PS2_MODEL_30:
      mov dx, offset PS2_MODEL_30
      call PRINT
      ret

PS2_MODEL_50_60:
      MOV dx, offset PS2_MODEL_50_60
      call PRINT
      ret

PS2_MODEL_80:
      mov dx, offset PS2_MODEL_80
      call PRINT
      ret

PSJR:
      mov dx, offset PCJS_MODEL
      call PRINT
      ret

PS_CONVERTIBLE:
      mov dx, offset PC_CONVERTIBLE_MODEL
      call PRINT
      ret

PC_INFO ENDP


OS_INFO PROC near
      mov ah, 30h
      int 21h

      push ax

      mov si, offset MS_DOS_VERSION
      add si, 15
      call BYTE_TO_DEC

    pop ax
```

```
      add si, 2
    mov al, ah

      call BYTE_TO_DEC
      mov dx, offset MS_DOS_VERSION
      call PRINT

      mov si, offset SERIAL_OEM
      add si, 19
      mov al, bh
      call BYTE_TO_DEC
      mov dx, offset SERIAL_OEM
      call PRINT

      mov di, offset USER_SERIAL_NUMER
      add di, 24
      mov ax, cx
      call WRD_TO_HEX
      mov al, bl
      call BYTE_TO_HEX
      sub di, 2
      mov [di], ax
      mov dx, offset USER_SERIAL_NUMER
      call PRINT
      ret
OS_INFO ENDP

BEGIN:
   call PC_INFO
   call OS_INFO

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

Exe.asm

```
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 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 младцая
   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
   push SI
   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 SI
   pop DX
   pop CX

   ret
BYTE_TO_DEC ENDP
;--------------------------------
; КОД

PRINT PROC near
   mov AH,09h
   int 21h
   ret
PRINT ENDP
```

```
PC_INFO PROC near
    mov ax, 0f000h
      mov es, ax
      mov al, es:[0fffeh]

      cmp al, 0ffh
      je PC

      cmp al, 0feh
      je XT

      cmp al, 0fbh
      je XT

      cmp al, 0fch
      je AT_MODEL

      cmp al, 0fah
      je PS2_MODEL_30

      cmp al, 0fah
      je PS2_MODEL_50_60

      cmp al, 0f8h
      je PS2_MODEL_80

      cmp al, 0fdh
      je PSJR

      cmp al, 0f9h
      je PS_CONVERTIBLE


      call BYTE_TO_HEX
      mov   si, offset NOT_FOUND_MODEL
      add   si, 1
      mov   [si], ax
      mov dx,offset NOT_FOUND_MODEL
      ret
```

```
PC:
      mov dx, offset PC_MODEL
      call PRINT
      ret

XT:
      mov dx, offset XT_MODEL
      call PRINT
      ret

AT_MODEL:
      mov dx, offset AT_MODEL_STR
      call PRINT
      ret

PS2_MODEL_30:
      mov dx, offset PS2_MODEL_30
      call PRINT
      ret

PS2_MODEL_50_60:
      MOV dx, offset PS2_MODEL_50_60
      call PRINT
      ret

PS2_MODEL_80:
      mov dx, offset PS2_MODEL_80
      call PRINT
      ret

PSJR:
      mov dx, offset PCJS_MODEL
      call PRINT
      ret

PS_CONVERTIBLE:
      mov dx, offset PC_CONVERTIBLE_MODEL
      call PRINT
      ret

PC_INFO ENDP
```

```
OS_INFO PROC near
      mov ah, 30h
      int 21h

      push ax

      mov si, offset MS_DOS_VERSION
      add si, 15
      call BYTE_TO_DEC

    pop ax
      add si, 2
    mov al, ah

      call BYTE_TO_DEC
      mov dx, offset MS_DOS_VERSION
      call PRINT

      mov si, offset SERIAL_OEM
      add si, 19
      mov al, bh
      call BYTE_TO_DEC
      mov dx, offset SERIAL_OEM
      call PRINT

      mov di, offset USER_SERIAL_NUMER
      add di, 24
      mov ax, cx
      call WRD_TO_HEX
      mov al, bl
      call BYTE_TO_HEX
      sub di, 2
      mov [di], ax
      mov dx, offset USER_SERIAL_NUMER
      call PRINT
      ret
OS_INFO ENDP

Main PROC FAR
```

```
    sub   ax, ax
    push  ax
    mov   ax,DATA
    mov   ds,ax
    call PC_INFO
    call OS_INFO
    xor al,al
    mov ah,4Ch
    int 21h
Main ENDP
CODE ENDS
      END Main
```