

**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**

Студентка гр. 8382

\_\_\_\_\_

Ефимова М.А.

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 выполнения.**

1. Для определения типа PC и версии системы были написаны тексты .COM и .EXE модулей (см. Приложение А и В). Тип PC). Тип PC определяется предпоследним байтом ROM B). Тип PCIOS (табл. 1).

Таблица 1 – Соответствие кода и типа

PC	FF
PC/XT	FE, FB), Тип PC
AT	FC
PS2 модель 30	FA
PS2 модель 50 или 60	FC
PS2 модель 80	F8
PCjr	FD
PC Convertible	F9

Версия системы определяется значением регистров AL, AH, B). Тип PCN, B). Тип PCL:CX, полученных после выполнения функции 30H прерывания 21H. (в AL – номер основной версии, в AH – номер модификации, в B). Тип PCN – серийный номер OEM, в B). Тип PCL:CX – 24-битовый серийный номер пользователя).

Результат выполнения .COM модуля представлен на рис. 1. Результат выполнения «плохого» .EXE модуля, полученного из исходного текста для .COM модуля, представлен на рис. 2. Результат выполнения «хорошего» .EXE модуля представлен на рис. 3.

```

C:\>exe2bin first.exe first.com

C:\>first.com
AT
Version of MS-DOS:0 .
DEM number :
User serial number: 000000H
C:\>

```

рис. 1 – результат выполнения .COM модуля



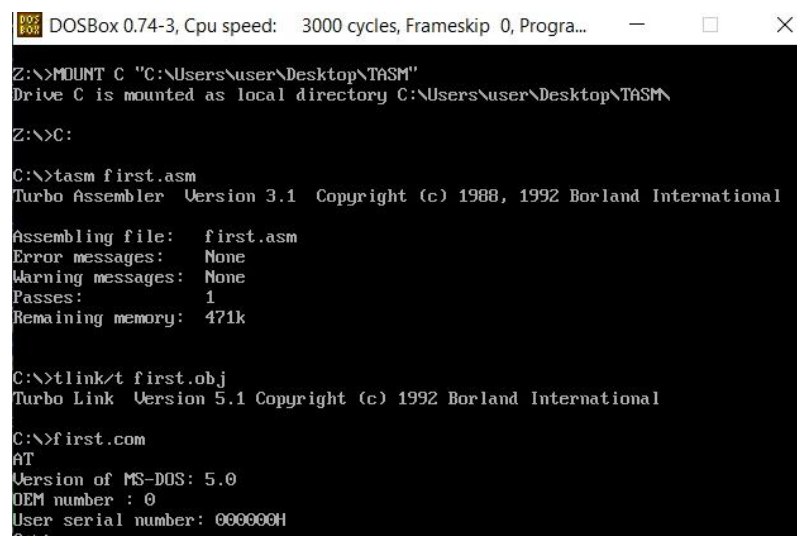
```

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
:\>MOUNT C "C:\Users\user\Desktop\MASM"
rive C is mounted as local directory C:\Users\user\Desktop\MASM\
:\>C:
:\>exe2bin first.exe first.com
:\>first.com
T
ersion of MS-DOS:0 .
EM number :
sOr serial number: 000000H
:\>first.exe

T@PC
5 0
0
0T@PC
0
000000
0T@PC
:\>

```

рис. 2 – результат выполнения «плохого» .EXE модуля



```

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
Z:\>MOUNT C "C:\Users\user\Desktop\TASM"
Drive C is mounted as local directory C:\Users\user\Desktop\TASM\
Z:\>C:
C:\>tasm first.asm
Turbo Assembler Version 3.1 Copyright (c) 1988, 1992 Borland International
Assembling file: first.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 471k
C:\>tlink/t first.obj
Turbo Link Version 5.1 Copyright (c) 1992 Borland International
C:\>first.com
AT
Version of MS-DOS: 5.0
DEM number : 0
User serial number: 000000H
C:\>

```

рис. 3 – результат выполнения «хорошего» .EXE модуля

## **Отличия исходных текстов COM и EXE программ**

1. Сколько сегментов должна содержать COM-программа?

У файла типа .COM есть один сегмент команд.

2. Сколько сегментов должна содержать EXE-программа?

У файла типа .EXE может содержать не менее одного сегмента.

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

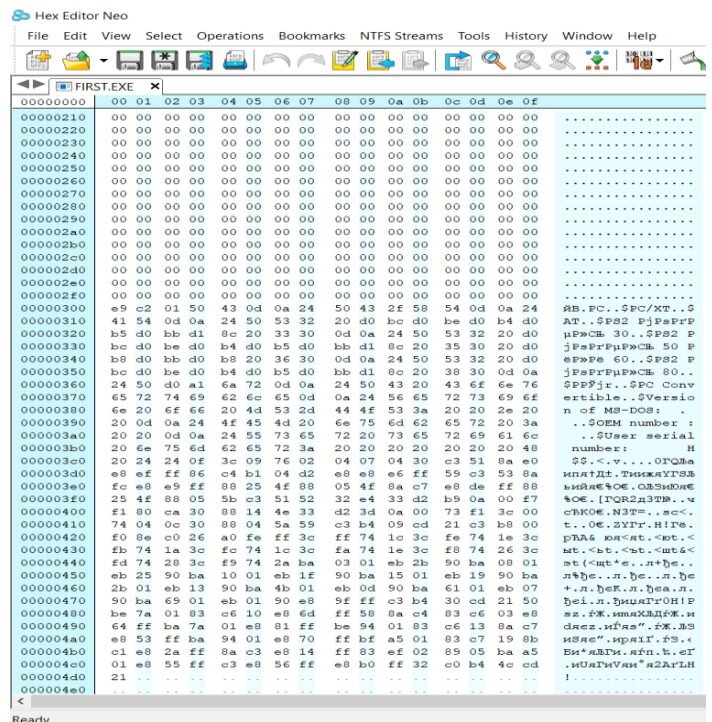
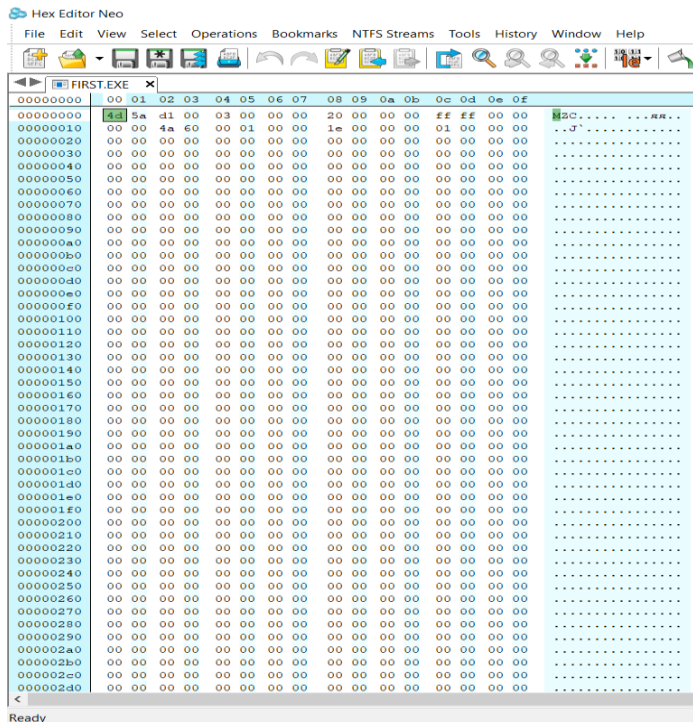
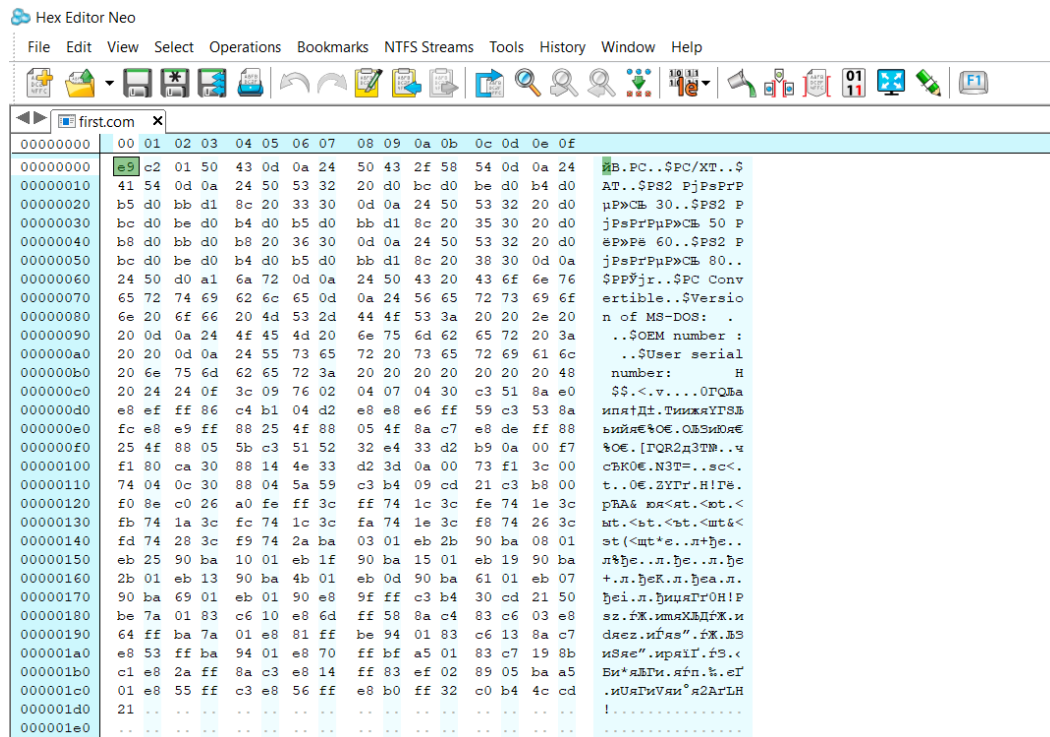
DOS передает управление в сегмент памяти, отведенный для команд, в точку со смещением 100H. Должна присутствовать директива `ORG 100h` и директива `ASSUME` для того, чтобы сегмент кода и сегмент данных указывали на общий сегмент.

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

Нет, не все форматы команд можно использовать в COM-программе.

В COM – файле есть только один сегмент, а команды, в которых присутствует адрес сегмента, использовать нельзя.

2. Файл загрузочного модуля .COM, «плохого» .EXE и «хорошего» в шестнадцатеричном виде представлены ниже.





## **Отличия форматов файлов COM и EXE модулей**

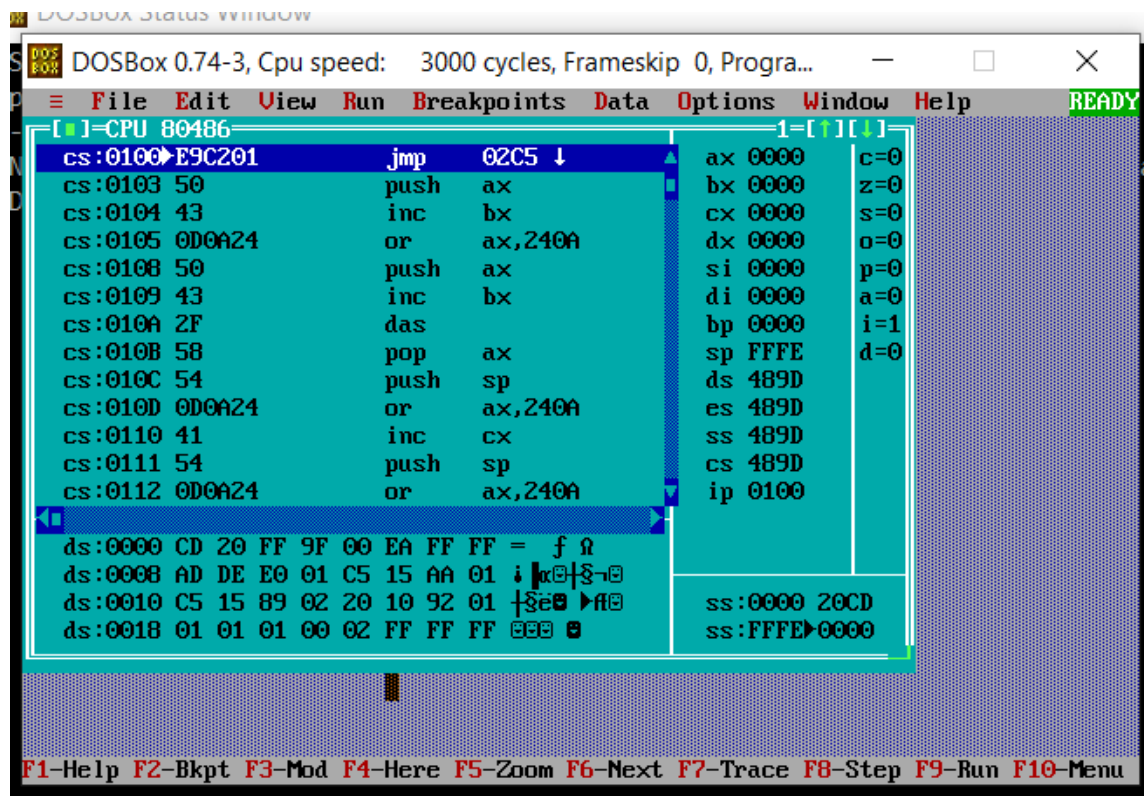
1. Какова структура файла COM? С какого адреса располагается код?  
Содержит один сегмент. Не превышает 64 Кб. Код располагается с адреса 0h. 2. Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с адреса 0?  
«Плохой» EXE файл, так же, как и COM файла, имеет один сегмент. Код располагается с адреса 300h. С адреса 0 располагается таблица настроек.

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

В «хорошем» EXE файле есть сегменты данных, кода и стека, в отличии от «плохого», в котором только один сегмент.

В «плохом» файле адресация всегда начинается с адреса 300h, а в «хорошем» EXE файле адресация кода начнется с 200h + размер памяти, выделенной под стек. Также в «хорошем» EXE файле заголовок располагается с адреса 0h и далее таблица настройки адресов.

Результат работы отладчика для COM файла представлен ниже



### Загрузка COM модуля в основную память

1. Какой формат загрузки модуля COM? С какого адреса располагается код?
  - Смещение в сегменте команд равно 100h, сегментные регистры указывают на PSP.
2. Что располагается с адреса 0?
  - С адреса 0 до адреса 100h располагается PSP.
3. Какие значения имеют сегментные регистры? На какие области памяти они указывают?
  - Сегментные регистры имеют значения 48DD и указывают на PSP.
4. Как определяется стек? Какую область памяти он занимает? Какие адреса?
  - Значение регистра SP устанавливается (автоматически) так, чтобы он указывал на последнюю доступную в сегменте ячейку памяти (SP указывает на FF FE). Таким образом программа занимает начало, а стек - конец сегмента.



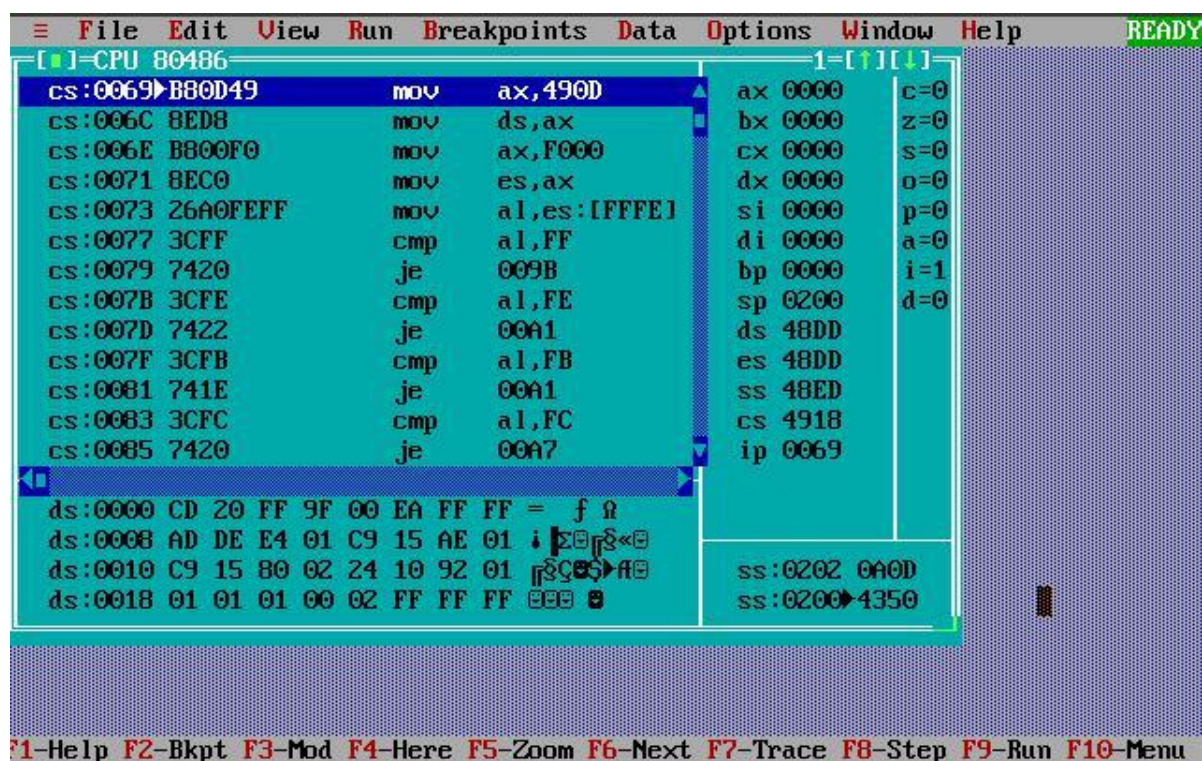


рис 9 – файл «хорошего» .EXE в отладчике

## **Загрузка «хорошего» EXE модуля в основную память**

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

- Значения DS, ES (48DD) устанавливаются на начало PSP, SS (48ED) - на начало сегмента стека, CS (4918) – на начало сегмента команд.

2. На что указывают регистры DS и ES?

- Регистры DS и ES указывают на начало

PSP. 3. Как определяется стек?

- Стек определяется при помощи директивы .STACK или при помощи директивы ASSUME, которая установит сегментный регистр SS на начало сегмента стека.

4. Как определяется точка входа?

- Точка входа определяется при помощи директивы END (за ней идет название функции или метки, с которой нужно начать выполнение программы).

## **Выводы**

В ходе выполнения работы были изучены COM и EXE файлы и их различия. Так же были получены две программы `typescom.com` и `typeexe.exe`.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД TYPECOM.ASM

```
TESTPC      SEGMENT
              ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
              ORG 100H ; резервирование места для PSP
START:      JMP BEGIN

PC db  'PC',0DH,0AH,'$'
XT db  'PC/XT',0DH,0AH,'$'
tAT db  'AT',0DH,0AH,'$'
PS2_30 db  'PS2 model 30',0DH,0AH,'$'
PS2_80 db  'PS2 model 80',0DH,0AH,'$'
PCJR db  'PCjr',0DH,0AH,'$'
PC_CONVERTIBLE db  'PC Convertible',0DH,0AH,'$'
OTHER_TYPE db      'Other type:',0DH,0AH,'$'
END_LINE db  '          ', 0DH,0AH,'$'

VERSION db  'Version: ',0DH,0AH,'$'
VERS db  ' $'
MODIFICATION db      '.$'
VERSION2 db  'Version <2.0',0DH,0AH,'$'
OEM db  'OEM number:',0DH,0AH,'$'
SERIAL_NUMBER db 'User serial number:', 0AH, '          ', 0DH,0AH,'$'

BEGIN:
              mov ax,0F000H
              mov es,ax
              mov al,es:[0FFFEH]

              cmp al, 0FFH
              je itIsPC
              cmp al, 0FEH
              je itIsPC_XT
              cmp al, 0FBH
              je itIsPC_XT
              cmp al, 0FCH
              je itIsAT
              cmp al, 0FAH
              je itIsPS2_30
              cmp al, 0F8H
              je itIsPS2_80
              cmp al, 0FDH
              je itIsPCjr
              cmp al, 0F9H
```

```

        je itIsPCconvertible

        cmp al, 0F9H
        jne itIsOther

;-----
; Для вывода типа
itIsPC:
        mov dx, offset PC
        jmp writeType

itIsPC_XT:
        mov dx, offset XT
        jmp writeType

itIsAT:
        mov dx, offset tAT
        jmp writeType

itIsPS2_30:
        mov dx, offset PS2_30
        jmp writeType

itIsPS2_80:
        mov dx, offset PS2_80
        jmp writeType

itIsPCjr:
        mov dx, offset PCJR
        jmp writeType

itIsPCconvertible:
        mov dx, offset PC_CONVERTIBLE
        jmp writeType

itIsOther:
        mov dx, offset OTHER_TYPE
        mov ah, 09h
        int 21h
        call BYTE_TO_HEX
        call PRINT_NUM
        mov al, ah
        call PRINT_NUM
        call PRINT_END_LINE
        jmp OS_VERSION

writeType:
        mov ah, 09h

```

```

        int 21h
        jmp OS_VERSION
;-----

;----- ;
Для вывода версии системы

OS_VERSION:
        mov ah, 30h
        int 21h

printVer:
        push ax
        cmp al, 0
        je ver2

        mov dx, offset VERSION
        push ax
        mov ah, 09h
        int 21h
        pop ax

        mov si, offset VERS
        call BYTE_TO_DEC
        add si, 1
        mov dx, offset VERS
        mov ah, 09h
        int 21h
        pop ax
        jmp numMod

ver2:
        mov dx, offset VERSION2
        mov ah, 09h
        int 21h
        pop ax

numMod:
        mov dx, offset MODIFICATION
        push ax
        mov ah, 09h
        int 21h
        pop ax
        mov si, offset END_LINE
        mov al, ah
        call BYTE_TO_DEC
        add si, 1

```

```

        mov dx, offset END_LINE
        mov ah, 09h
        int 21h

numOEM:
        mov dx, offset OEM
        push ax
        mov ah, 09h
        int 21h
        pop ax
        mov si, offset END_LINE
        mov al, bh
        call BYTE_TO_DEC
        add si, 1
        mov dx, offset END_LINE
        mov ah, 09h
        int 21h

serialNumb:

        mov di, offset SERIAL_NUMBER
        add di, 25
        mov ax, cx
        call WRD_TO_HEX
        mov al, bl
        call BYTE_TO_HEX
        sub di, 2
        mov [di], ax
        mov dx, offset SERIAL_NUMBER
        mov ah, 09h
        int 21h

        xor al, al
        mov ah, 4Ch
        int 21h

;-----

PRINT_END_LINE PROC near
        push ax
        mov dx, offset END_LINE
        mov ah, 09h
        int 21h
        pop ax
PRINT_END_LINE ENDP

```

PRINT\_NUM PROC near

```
    ; вывод a1
    push ax mov
    dx, ax mov
    ah, 02h int
    21h pop ax
    ret
```

PRINT\_NUM ENDP

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 шест. 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_1
        or AL,30h
        mov [SI],AL
end_1:
        pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP

TESTPC      ENDS
                END START

```

## ПРИЛОЖЕНИЕ В

### ИСХОДНЫЙ КОД FIRST.ASM

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN
; Данные
TYPE_PC db 'PC',0DH,0AH,'$'
TYPE_PC_XT db 'PC/XT',0DH,0AH,'$'
TYPE_AT db 'AT',0DH,0AH,'$'
TYPE_PS2_M30 db 'PS2 модель 30',0DH,0AH,'$'
TYPE_PS2_M50_60 db 'PS2 модель 50 или 60',0DH,0AH,'$'
TYPE_PS2_M80 db 'PS2 модель 80',0DH,0AH,'$'
TYPE_PC_JR db 'PCjr',0DH,0AH,'$'
TYPE_PC_CONV db 'PC Convertible',0DH,0AH,'$'

VERSIONS db 'Version of MS-DOS: . ',0DH,0AH,'$'
SERIAL_NUMBER db 'OEM number : ',0DH,0AH,'$'
USER_NUMBER db 'User serial number:      H $'

; 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 шест. 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
;-----
WRITESTRING PROC near
    mov AH,09h
    int 21h
    ret
WRITESTRING ENDP
;-----
PC_TYPE PROC near
    mov ax, 0F000H ; получаем номер модели
    mov es, ax
    mov al, es:[0FFEH]

    cmp al, 0FFH ; начинаем сравнивать
    je isPc
    cmp al, 0FEH
    je isPc_xt
    cmp al, 0FBH
    je isPc_xt
    cmp al, 0FCH
    je isPc_at
    cmp al, 0FAH
    je isPs2_m30
    cmp al, 0F8H
    je isPs2_m80
    cmp al, 0FDH
    je isPc_jr
    cmp al, 0F9H
    je isPc_cv
isPc:
    mov dx, offset TYPE_PC
    jmp writetype
isPc_xt:
    mov dx, offset TYPE_PC_XT
    jmp writetype
isPc_at:
    mov dx, offset TYPE_AT
    jmp writetype
isPs2_m30:
    mov dx, offset TYPE_PS2_M30

```

```

        jmp writetype
isPs2_m50_60:
        mov dx, offset TYPE_PS2_M50_60
        jmp writetype
isPs2_m80:
        mov dx, offset TYPE_PS2_M80
        jmp writetype
isPc_jr:
        mov dx, offset TYPE_PC_JR
        jmp writetype
isPc_cv:
        mov dx, offset TYPE_PC_CONV
        jmp writetype
writetype:
        call WRITESTRING
        ret
PC_TYPE ENDP
;-----
OS_VER PROC near
        mov ah, 30h
        int 21h
        push ax
        ;si - индекс источника данных, операции над строками, с ds
        mov si, offset VERSIONS
        add si, 19
        call BYTE_TO_DEC
        pop ax
        mov al, ah ; в al -> 30h
        add si, 3
        call BYTE_TO_DEC
        mov dx, offset VERSIONS
        call WRITESTRING

        mov si, offset SERIAL_NUMBER
        add si, 13
        mov al, bh
        call BYTE_TO_DEC
        mov dx, offset SERIAL_NUMBER
        call WRITESTRING

        ; di - индекс назначения, используется с es
        mov di, offset USER_NUMBER
        add di, 25
        mov ax, cx
        call WRD_TO_HEX
        mov al, bl
        call BYTE_TO_HEX
        sub di, 2
        mov [di], ax
        mov dx, offset USER_NUMBER
        call WRITESTRING
        ret
OS_VER ENDP
;-----
; Код
BEGIN:
        call PC_TYPE
        call OS_VER

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

```