**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра математического обеспечения и применения ЭВМ**

# ОТЧЕТ

**по практической работе № 1**

**по дисциплине «Операционные системы»**

**Тема: Исследование структур загрузочных модулей**

Студентка гр. 9383 _____ Лапина А.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.

**Постановка задачи.**

Требуется написать текст исходного .COM модуля, который определяет тип PC и версию системы. 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 (Original Equipment Manufacturer) и серийным номером пользователя. Полученные строки выводятся на экран. Далее необходимо отладить полученный исходный модуль и получить «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля. Затем нужно написать текст «хорошего» .EXE модуля, который выполняет те же функции, что и модуль .COM, далее его построить, отладить и сравнить исходные тексты для .COM и .EXE модулей.

**Выполнение работы.**

Были реализованы следующие функции:

* PC_TYPE — определяет тип ПК (по содержимому регистра AL) согласно таблице (Рисунок 1).

| | |
|---|---|
| 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 |

Рисунок 1 – Соответствие кода предпоследнего байта ROM BIOS и типа PC

Выполнение программы начинается с занесения в регистр AL данных предпоследнего байта ROM BIOS для определения типа PC. Происходит сравнение регистра AL с числами, соответствующих различным типам PC (Рисунок 1). В случае совпадения происходит вывод на экран соответствующей числу строки, а в противном случае – замена числа на код ASCII символа и вывод его на экран. Все выводы на экран в программе реализованы с помощью функции write, в которой использована функция 09h прерывание 21h.

*Функция OS, определяет номер основной версии MS-DOS, лежащий в регистре AL, номер её модификации в регистре AH, серийный номер OEM в регистре BH и 24-битный серийный номер пользователя в регистрах BL:CX. Программа последовательно переводит числа в соответствующие им строки и выводит на экран. Завершается программа вызовом функции 4Ch прерывания 21h с кодом 0. Разработанный программный код см. в приложении А.

Были выделены строки для вывода информации:

- T_PC db 'Type: PC',0DH,0AH,'$' - для типа PC;
- T_PC_XT db 'Type: PC/XT',0DH,0AH,'$'- для типа PC/XT;
- T_AT db 'Type: AT',0DH,0AH,'$'- для типа AT;
- T_PS2_M30 db 'Type: PS2 модель 30',0DH,0AH,'$' - для типа PS2 модель 30;
- T_PS2_M50_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'$' для типа PS2 модель 50 или 60;

- T_PS2_M80 db 'Type: PS2 модель 80',0DH,0AH,'$' - для типа PS2 модель 80;
- T_PC_JR db 'Type: PCjr',0DH,0AH,'$' для типа PC jr;
- T_PC_C db 'Type: PC Convertible',0DH,0AH,'$' - для типа PC Convertible;
- VERSION db 'Version: . ',0DH,0AH,'$' - для версии и модификации;
- OEM db 'OEM: ',0DH,0AH,'$' - для серийного номера OEM;
- USER db 'User: $' - для серийного номера пользователя;

Результаты работы «хороших» .COM и .EXE модулей представлены на рисунках 2 и 3. Результат работы «плохого» .EXE представлен на рисунке 4.

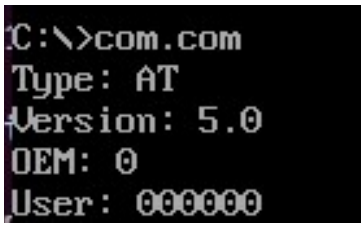


Рисунок 2 –Результат .COM модуля

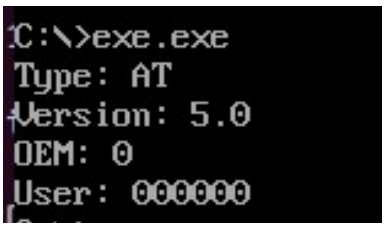


Рисунок 3 – Результат .EXE модуля


Рисунок 4 – Результат «плохого» .EXE модуля

**Контрольные вопросы.**

Отличия исходных текстов COM и EXE программ:

1. Сколько сегментов должна содержать COM-программа?

COM-программа должна содержать один сегмент, в котором располагаются команды и данные.

2. Сколько сегментов должна содержать EXE-программа?

EXE-программа может содержать от одного до трёх сегментов: сегмент команд, данных и стека (последние два могут отсутствовать).

3. Какие директивы должны обязательно быть в тексте COMпрограммы?

Должна быть обязательна директива ORG 100h, так как при загрузке модуля все сегментные регистры содержат адрес префикса программного сегмента (PSP), который является 256-байтовым(100H) блоком, поэтому адресация имеет смещение в 256 байт от нулевого адреса. Также необходима процедура ASSUME для того, чтобы сегмент данных и сегмент кода указывали на один общий сегмент. (ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING)

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

Не все форматы команд можно использовать в COM-программе. Так, например, ввиду отсутствия таблицы настроек нельзя использовать команды вида mov <register>, seg .

Отличия форматов файлов .COM и .EXE модулей:

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

Файл .COM состоит из программного сегмента PSP размером 256 байт, одного сегмента, содержащего команды и данные, а также стека, генерируемого автоматически. Структура файла представлена на рисунке 5. Код располагается с адреса 100h (256).

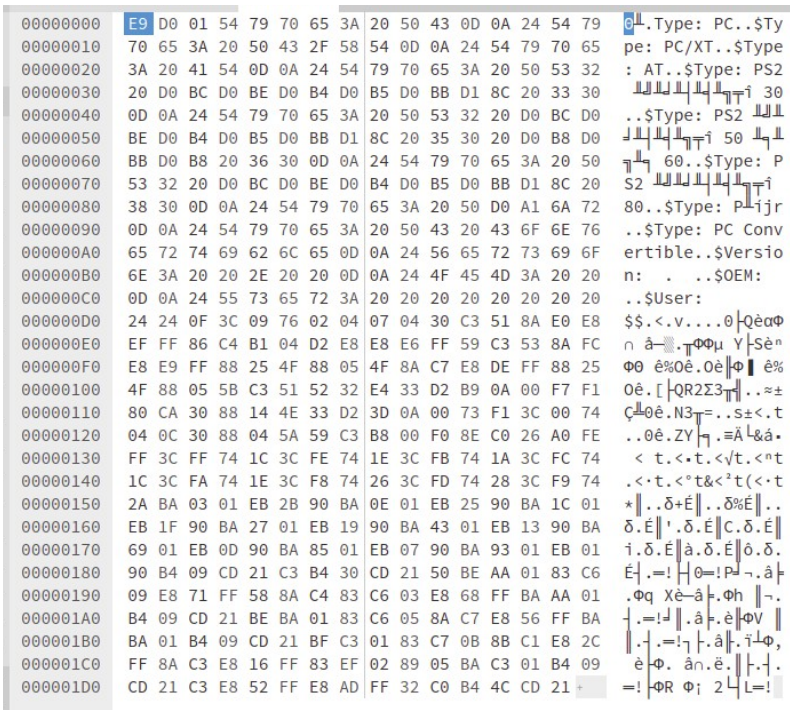

Рисунок 5 – Структура файла .COM

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

«Плохой» файл .EXE состоит из единственного сегмента, что некорректно для файлов данного типа. Структура файла представлена на рисунке 6. Код располагается с адреса 300h, а с адреса 0h – таблица настроек.

Рисунок 6 – Структура «плохого» файла .EXE

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

«Хороший» файл .EXE состоит из программного сегмента PSP, за которым идут несколько сегментов: команд, данных и стека, – что и отличает его от «плохого» файла.

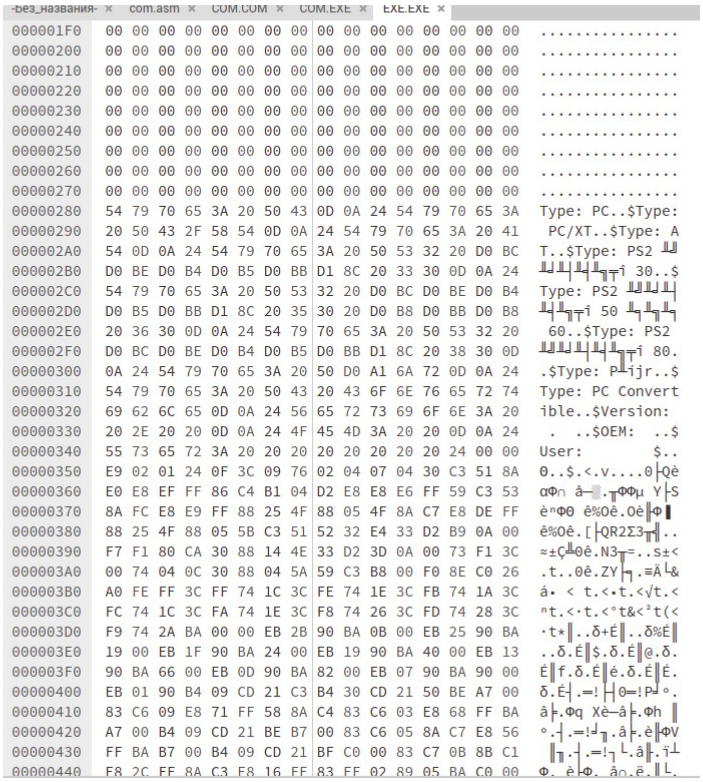


Рисунок 7 – Структура «хорошего» файла .EXE

<u>Загрузка .COM модуля в основную память:</u>

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

После загрузки программы все 4 сегментных регистра указывают на начало единственного сегмента, то есть фактически на начало PSP. Указатель стека SP автоматически инициализируется числом 0FFFEh. Таким образом, независимо от фактического размера программы, ей выделяется 64 Кбайт адресного пространство, всю нижнюю часть которого занимает стек (рисунок

8). Указатель команд на момент начала программы указывает на адрес 100h, с которого, соответственно, располагается код.

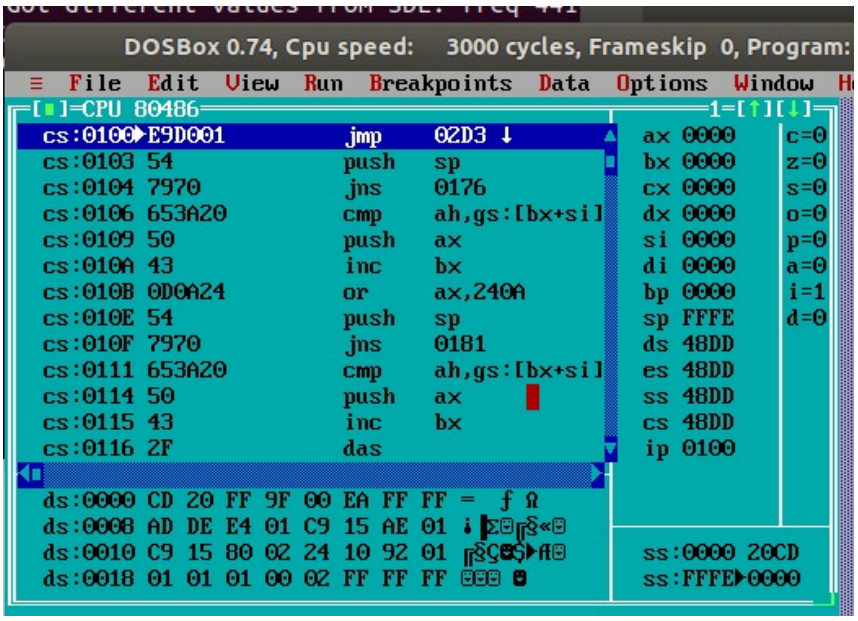


Рисунок 8 – Отладка .COM модуля (с помощью TD.EXE)

2. Что располагается с адреса 0?

С адреса 0 располагается программный сегмент PSP, размером 256 байт, зарезервируемый операционной системой.

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

Сегментные регистры имеют значения 48DDh и указывают на программный сегмент PSP.

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

Стек занимает всю нижнюю часть адресного пространства. При этом сегментный регистр SS в начале программы, как и остальные регистры, указывает на PSP (48DDh), а регистр SP – на конец стека с адресом 0FFFEh.

Загрузка «хорошего» .EXE модуля в основную память:

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

Система, загрузив программу в память, инициализирует сегментные регистры так, что регистры DS и ES указывают на начало PSP, CS – на начало сегмента команд, а SS – на начало сегмента стека. В указатель команд IP загружается смещение точки входа в программу, а в указатель стека SP – смещение конца сегмента стека. Таким образом, после загрузки программы в память адресуемыми оказываются все регистры, кроме сегментов данных.

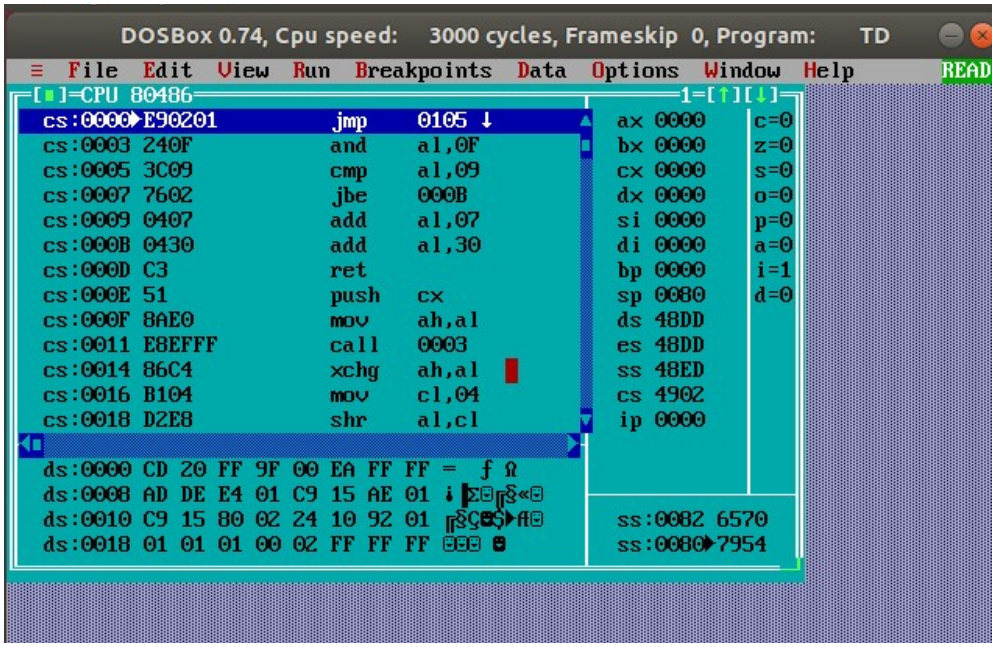


Рисунок 9 – Отладка «хорошего» .EXE модуля (с помощью TD.EXE)

2. На что указывают регистры DS и ES?

При запуске программы регистры DS и ES указывают на начало PSP.

3. Как определяется стек?

Оператор segment, начинающий сегмент стека, имеет описатель stack, что приводит к тому, что при загрузке программы в память регистр SS будет настроен на начало сегмента стека, а указатель стека SP -–на его конец.

4. Как определяется точка входа?

Точка входа определяется с помощью директивы END.

**Выводы.**

В ходе лабораторной работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: com.asm

```
TESTPC SEGMENT
   ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
   ORG 100H    ;так как адресация начинается со смещением 100 в .com
START: JMP BEGIN      ;точка входа (метка)


; Данные
T_PC db  'Type: PC',0DH,0AH,'$'
T_PC_XT db 'Type: PC/XT',0DH,0AH,'$'
T_AT db  'Type: AT',0DH,0AH,'$'
T_PS2_M30 db 'Type: PS2 модель 30',0DH,0AH,'$'
T_PS2_M50_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'$'
T_PS2_M80 db 'Type: PS2 модель 80',0DH,0AH,'$'
T_PC_JR db 'Type: PCjr',0DH,0AH,'$'
T_PC_C db 'Type: PC Convertible',0DH,0AH,'$'

VERSION db 'Version:  .  ',0DH,0AH,'$'
OEM db  'OEM:  ',0DH,0AH,'$'
USER db  'User:        $'

; 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
;-------------------------------
PC_TYPE PROC near
    mov ax, 0f000h       ;получаем номер модели
    mov es, ax
    mov al, es:[0fffeh]  ;смещение

    cmp al, 0ffh         ;сравниваем
    je pc
    cmp al, 0feh
    je pc_xt
    cmp al, 0fbh
    je pc_xt
    cmp al, 0fch
    je pc_at
    cmp al, 0fah
    je pc_ps2_m30
    cmp al, 0f8h
    je pc_ps2_m80
    cmp al, 0fdh
    je pc_jr
    cmp al, 0f9h
    je pc_conv
pc:
    mov dx, offset T_PC
    jmp write
pc_xt:
    mov dx, offset T_PC_XT
    jmp write
pc_at:
    mov dx, offset T_AT
    jmp write
pc_ps2_m30:
    mov dx, offset T_PS2_M30
    jmp write
pc_ps2_m50_60:
    mov dx, offset T_PS2_M50_60
    jmp write
pc_ps2_m80:
    mov dx, offset T_PS2_M80
```

```
    jmp write
pc_jr:
    mov dx, offset T_PC_JR
    jmp write
pc_conv:
    mov dx, offset T_PC_C
    jmp write
write:
    mov AH,09h
    int 21h
    ret
PC_TYPE ENDP

OS PROC near
    ;версия
    mov ah, 30h
    int 21h
    push ax
    mov si, offset VERSION
    add si, 9       ;смещение
    call BYTE_TO_DEC
    pop ax
    mov al, ah
    add si, 3       ;смещение
    call BYTE_TO_DEC
    mov dx, offset VERSION
    mov AH,09h      ;вывод
    int 21h

    ;серийный номер OEM
    mov si, offset OEM
    add si, 5      ;смещение
    mov al, bh
    call BYTE_TO_DEC
    mov dx, offset OEM
    mov AH,09h     ;вывод
    int 21h

    ;серийный номер пользователя
    mov di, offset USER
    add di, 11     ;смещение
    mov ax, cx
    call WRD_TO_HEX
```

```
    mov al, bl
    call BYTE_TO_HEX
    sub di, 2
    mov [di], ax
    mov dx, offset USER
    mov AH,09h     ;вывод
    int 21h
    ret

OS ENDP

; Код
BEGIN:
   call PC_TYPE
   call OS

   xor AL,AL
   mov AH,4Ch
   int 21H
TESTPC ENDS
END START; конец модуля, START - точка выхода
```

**Название файла: exe.asm**

```
STK SEGMENT STACK
   DB 128 DUP (0) ;выделяем память
STK ENDS

DATA SEGMENT
; Данные
T_PC db  'Type: PC',0DH,0AH,'$'
T_PC_XT db 'Type: PC/XT',0DH,0AH,'$'
T_AT db  'Type: AT',0DH,0AH,'$'
T_PS2_M30 db 'Type: PS2 модель 30',0DH,0AH,'$'
T_PS2_M50_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'$'
T_PS2_M80 db 'Type: PS2 модель 80',0DH,0AH,'$'
T_PC_JR db 'Type: PCjr',0DH,0AH,'$'
T_PC_C db 'Type: PC Convertible',0DH,0AH,'$'

VERSION db 'Version:  .  ',0DH,0AH,'$'
OEM db  'OEM:  ',0DH,0AH,'$'
USER db  'User:        $'
```

```
DATA ENDS

CODE SEGMENT
   ASSUME CS:CODE, DS:DATA
START: JMP BEGIN      ;точка входа (метка)

; 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
;-------------------------------
PC_TYPE PROC near
    mov ax, 0f000h       ;получаем номер модели
    mov es, ax
    mov al, es:[0fffeh]  ;смещение

    cmp al, 0ffh         ;сравниваем
    je pc
    cmp al, 0feh
    je pc_xt
    cmp al, 0fbh
    je pc_xt
```

```
    cmp al, 0fch
    je pc_at
    cmp al, 0fah
    je pc_ps2_m30
    cmp al, 0f8h
    je pc_ps2_m80
    cmp al, 0fdh
    je pc_jr
    cmp al, 0f9h
    je pc_conv
pc:
    mov dx, offset T_PC
    jmp write
pc_xt:
    mov dx, offset T_PC_XT
    jmp write
pc_at:
    mov dx, offset T_AT
    jmp write
pc_ps2_m30:
    mov dx, offset T_PS2_M30
    jmp write
pc_ps2_m50_60:
    mov dx, offset T_PS2_M50_60
    jmp write
pc_ps2_m80:
    mov dx, offset T_PS2_M80
    jmp write
pc_jr:
    mov dx, offset T_PC_JR
    jmp write
pc_conv:
    mov dx, offset T_PC_C
    jmp write
write:
    mov AH,09h
    int 21h
    ret
PC_TYPE ENDP

OS PROC near
    ;версия
    mov ah, 30h
    int 21h
```

```
    push ax
    mov si, offset VERSION
    add si, 9       ;смещение
    call BYTE_TO_DEC
    pop ax
    mov al, ah
    add si, 3       ;смещение
    call BYTE_TO_DEC
    mov dx, offset VERSION
    mov AH,09h      ;вывод
    int 21h

    ;серийный номер OEM
    mov si, offset OEM
    add si, 5      ;смещение
    mov al, bh
    call BYTE_TO_DEC
    mov dx, offset OEM
    mov AH,09h     ;вывод
    int 21h

    ;серийный номер пользователя
    mov di, offset USER
    add di, 11     ;смещение
    mov ax, cx
    call WRD_TO_HEX
    mov al, bl
    call BYTE_TO_HEX
    sub di, 2
    mov [di], ax
    mov dx, offset USER
    mov AH,09h     ;вывод
    int 21h
    ret

OS ENDP

; Код
BEGIN:
    mov AX, DATA
    mov DS, AX
   call PC_TYPE
   call OS
```

```
   xor AL,AL
   mov AH,4Ch
   int 21H
CODE ENDS
END START; конец модуля, START - точка выхода
```