

**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.**

Факультет: КТИ

Дата выполнения работы: 14.02.2021

Студент гр. 9381

Семенов А. Н.

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.

Написать тексты исходных **.COM** и **.EXE** модулей на языке ассемблера, которые определяют тип РС и версию системы. Сравнить исходные тексты. Произвести трансляцию и сборку этих модулей с получением хороших **.COM** и **.EXE** модулей и плохого **.EXE** модуля. Изучить полученные загрузочные модули, произвести их сравнительный анализ. Изучить загрузки модулей в основную память с помощью отладчика **TD.EXE**.

Функции и структуры данных.

TETR_TO_HEX – переводит значение 4 младших битов регистра **al** в цифру 16-ричной системы счисления в виде символа, которая кладется в регистр **al**.

BYTE_TO_HEX – переводит значение байта, содержащегося в регистре **al** в двухсимвольное число в шестнадцатеричной системе счисления, которые кладется в регистр **ax**: код первого символа в **al**, второго в **ah**.

BYTE_TO_DEC – переводит значение байта, содержащегося в регистре **al** в символьное представление числа в десятичной системе счисления, которое кладется в память по адресу **si**.

WRD_TO_HEX – переводит значение регистра **AX** в шестнадцатеричное число в виде 4 символов, которые кладутся в память по адресу **di**.

Последовательность действий, выполняемых программой.

1. В регистр **ES** и **BX** записываются соответственно адреса места в памяти, где хранится шестнадцатеричный байт – тип РС. При обращении к данному месту памяти в регистр **AL** кладется соответствующий байт.

2. Производятся сравнения байта в AL с существующими значениями, определение и печать на экран информации о типе РС в строковом виде.

3. Далее в регистр AH кладется число 30h и вызывается прерывание int 21h, которое при данном аргументе раскидывает следующие значения по регистрам:

AL - номер основной версии. Если 0, то < 2.0
AH - номер модификации
BH - серийный номер OEM (Original Equipment Manufacturer)
BL:CH - 24-битовый серийный номер пользователя.

4. Затем в регистры SI и DI кладутся адреса памяти в строках вывода, предназначенные для вставки соответствующих номеров.

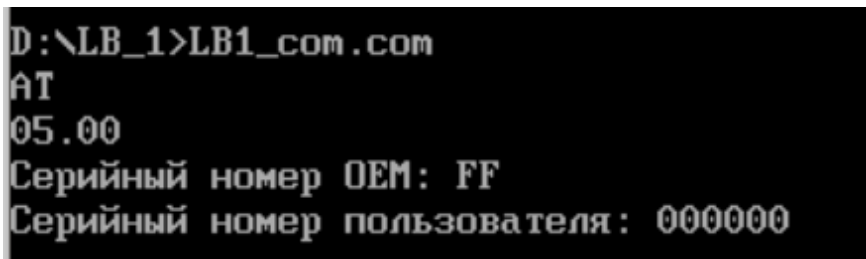
5. Производятся вызовы процедур, переводящих значения номеров в строковые форматы и записывающих их в соответствующие ячейки памяти по адресам, содержащимся в регистрах DI и SI. Перед каждым вызовом процедуры необходимый для перевода в символьную информацию номер кладется в регистр AL или AH.

6. Производится печать на экран строки с информацией о серийных номерах, номера версии и модификации.

Ход работы и результаты исследования проблем.

1. Написание исходного текста исходного .COM модуля на языке ассемблера, файл: *LBI_COM.ASM* (см. в приложении).
2. Трансляция исходного кода командой: *masm lbl_com.asm*, отладка и получение объектного модуля *LBI_COM.OBJ*.
3. Сборка объектного модуля командой: *link lbl_com.obj*, и получение загрузочного модуля *LBI_COM.EXE* – плохого .EXE модуля.
4. Получение хорошего .COM модуля: *LBI_COM.COM*, с помощью команды: *exe2bin lbl_com.exe*.

5. Запуск программы в терминале ДОС (рис. 1):



```
D:\LB_1>LB1_com.com
AT
05.00
Серийный номер OEM: FF
Серийный номер пользователя: 000000
```

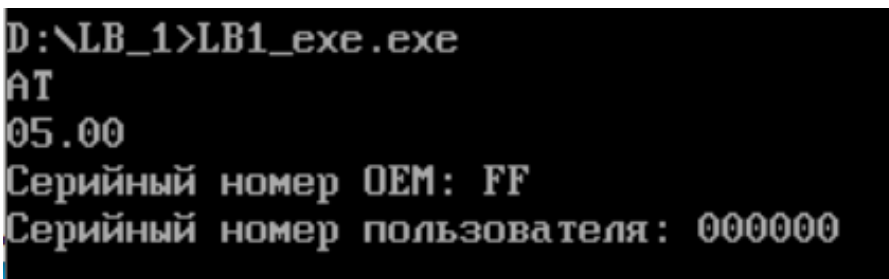
Рис. 1. Запуск программы lb1_com.com в терминале ДОС

6. Написание исходного текста исходного .EXE модуля на языке ассемблера, файл: *LB1_EXE.ASM* (см. в приложении).

7. Трансляция исходного кода командой: *masm lb1_exe.asm*, отладка и получение объектного модуля *LB1_EXE.OBJ*.

8. Сборка объектного модуля командой: *link lb1_exe.obj*, и получение загрузочного модуля *LB1_EXE.EXE* – хорошего .EXE модуля.

9. Запуск программы в терминале ДОС (рис. 2):



```
D:\LB_1>LB1_exe.exe
AT
05.00
Серийный номер OEM: FF
Серийный номер пользователя: 000000
```

Рис. 2. Запуск программы lb1_exe.exe в терминале ДОС

10. Сравнение исходных текстов: *LB1_COM.ASM* и *LB1_EXE.ASM* (ответы на вопросы «Отличие исходных текстов .COM и .EXE программ»):

1) Сколько сегментов должна содержать COM-программа? *COM-программа содержит всего один сегмент.*

2) Сколько сегментов должна содержать EXE-программа? *EXE-программа может содержать от одного до нескольких сегментов. Например, в DOS программа содержит 3 сегмента: сегменты данных, кода и стека.*

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

Директивы *ORG* и *ASSUME* должны обязательно присутствовать в тексте COM-программы. Команда *ORG 100h* помещает в регистр *IP* смещение 256 байт относительно начала *PSP*-сектора, находящегося в начале программы. Таким образом, в *IP* записывается адрес начала кода программы, так как размер *PSP* – 256 байт. Без этой директивы программа начнет исполнять код *PSP*. Директива же *ASSUME* указывает, с каким сегментом связан тот или иной сегментный регистр. Позволяет ассемблеру проверить допустимость ссылок. Без нее программа не скомпилируется.

4) Все ли форматы команд можно использовать в COM-программе? В COM-программе можно использовать не все виды команд. Например, запрещается использовать команду *SEG*, так как в COM-программе отсутствует таблица настройки и данных о каких-либо иных сегментах, кроме единственного основного, нет.

11. Запуск FAR и исследование файла загрузочного модуля *LBI_COM.COM* и плохого *LBI_COM.EXE* в шестнадцатеричном виде. Их сравнение с загрузочным модулем *LBI_EXE.EXE*: (Ответы на вопросы «Отличия форматов файлов COM и EXE модулей»):

1) Какова структура файла COM? С какого адреса располагается код? Код (данные) располагаются с нулевого адреса 0h, так как код, данные и стек расположены в одном сегменте. Структура файла COM отображена на рисунке 3 (см. файл *LBI_COM.COM*, рис. 3).

2) Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с адреса 0? В файле плохого EXE код данные и стек расположены в одном сегменте. С нулевого адреса располагается заголовок и таблица настройки адресов. Код с данными начинается с адреса 300h. Структура файла «плохого» EXE отображена на рисунке 4 (см. файл *LBI_COM.EXE*, рис. 4).

12. Запуск отладчика TD.EXE и загрузка .COM модуля в основную память.

Ответы на вопросы «Загрузка COM-модуля в основную память»:

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

Модуль .COM имеет следующий формат (рис. 6):

- С нулевого адреса располагается специальный блок PSP (префикс программного сегмента), размер которого – 100h (256 байт).
- Далее с адреса 100h располагается код и данные в одном сегменте.
- Все сегментные регистры DS, CS, ES и SS указывают на нулевой адрес относительно начала памяти под программу, т. е. на PSP.
- Директива ORG 100h заносит в регистр IP значение 100h, чтобы точка входа в программу была в месте начала кода: CS:IP.
- Размер всей памяти, отведенный под программу – 64 кб, при этом стек заполняется «сверху вниз», начиная с адреса FFFFh (64 кб), относительно начала сегмента программы. Сегментный регистр стека имеет адрес 0h, а регистр SP указывает на верхушку стека в начале программы, т. е. имеет адрес FFFFh, относительно начала сегмента программы.

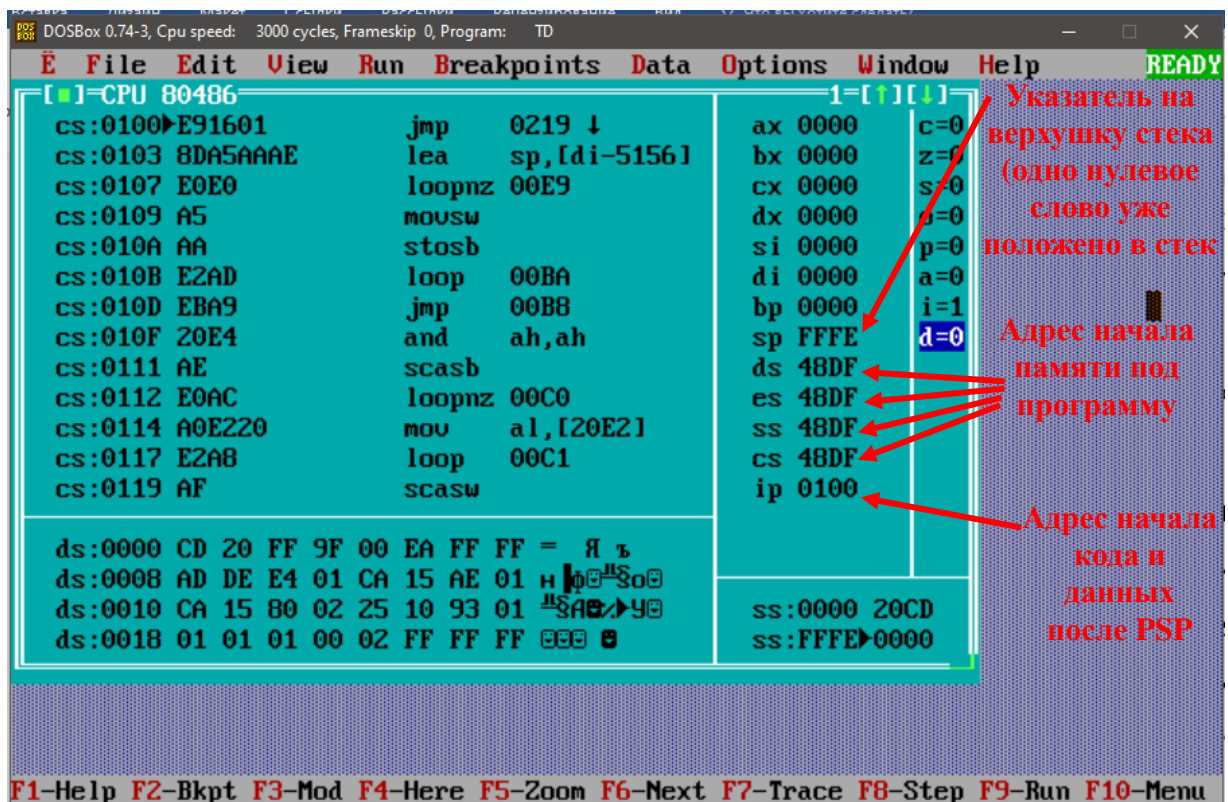


Рис. 6. Запуск программы .COM в отладчике TD

2) Что располагается с адреса 0? С нулевого адреса располагается специальный блок PSP (префикс программного сегмента), размер которого – 100h (256 байт).

3) Какие значения имеют сегментные регистры? На какие области памяти они указывают? Все сегментные регистры DS, CS, ES и SS указывают на нулевой адрес относительно начала памяти под программу, т. е. на PSP (см. рис. 6).

4) Как определяется стек? Какую область памяти он занимает? Какие адреса? Размер всей памяти, отведенный под программу – 64 кб, при этом стек заполняется «сверху вниз», начиная с адреса FFFFh (64 кб), относительно начала сегмента программы. Сегментный регистр стека имеет адрес 0h, а регистр SP указывает на верхушку стека в начале программы, т. е. имеет адрес FFFFh, относительно начала сегмента программы.

План загрузки модуля .COM в основную память:

- I.** Определяется сегментный адрес свободного участка памяти для загрузки программы (обычно MS-DOS загружает программу в младшие адреса памяти, если при редактировании не указана загрузка в старшие адреса);
- II.** Создаются два блока памяти - блок памяти для переменных среды, а также блок памяти для PSP и программы;
- III.** В блок памяти переменных среды помещается путь к файлу программы;
- IV.** Заполняются поля префикса сегмента программы PSP в соответствии с характеристиками программы (количество памяти, доступное программе, адрес сегмента блока памяти, содержащего переменные среды и т. д.);
- V.** Адрес области Disk Transfer Area (DTA) устанавливается на вторую половину PSP (PSP:0080);
- VI.** Анализируются параметры запуска программы на предмет наличия в первых двух параметрах идентификаторов дисковых устройств. По результатам анализа устанавливается содержимое регистра AX при входе в программу. Если первый или второй параметры не содержат правильного идентификатора дискового устройства, то соответственно в регистры AL и AH записывается значение FFh.
- VII.** Сегментные регистры CS, DS, ES, SS устанавливаются на начало PSP;
- VIII.** Регистр SP устанавливается на конец сегмента PSP;
- IX.** Вся область памяти после PSP распределяется программе;
- X.** В стек записывается слово 0000;
- XI.** Указатель команд IP устанавливается на 100h (начало программы).

13. Запуск отладчика TD.EXE и загрузка хорошего .EXE модуля в основную память. Ответы на вопросы «Загрузка хорошего EXE модуля в основную память:

1) Как загружается «хороший» EXE? Какие значения имеют сегментные регистры? Хороший .EXE после его загрузки в основную память имеет следующую структуру (рис. 7):

- С нулевого адреса начинается PSP-блок размером 256 байт.
- С адреса 100h начинается блок сегмента стека. Адрес начала этого блока записывается в сегментный регистр SS.
- За сегментом стека следует сегмент данных. Его адрес необходимо положить в регистр DS в процессе выполнения программы.
- За сегментом данных следует сегмент кода, адрес которого кладется в регистр CS.

Регистры DS и ES перед началом работы программы указывают на начало блока PSP.

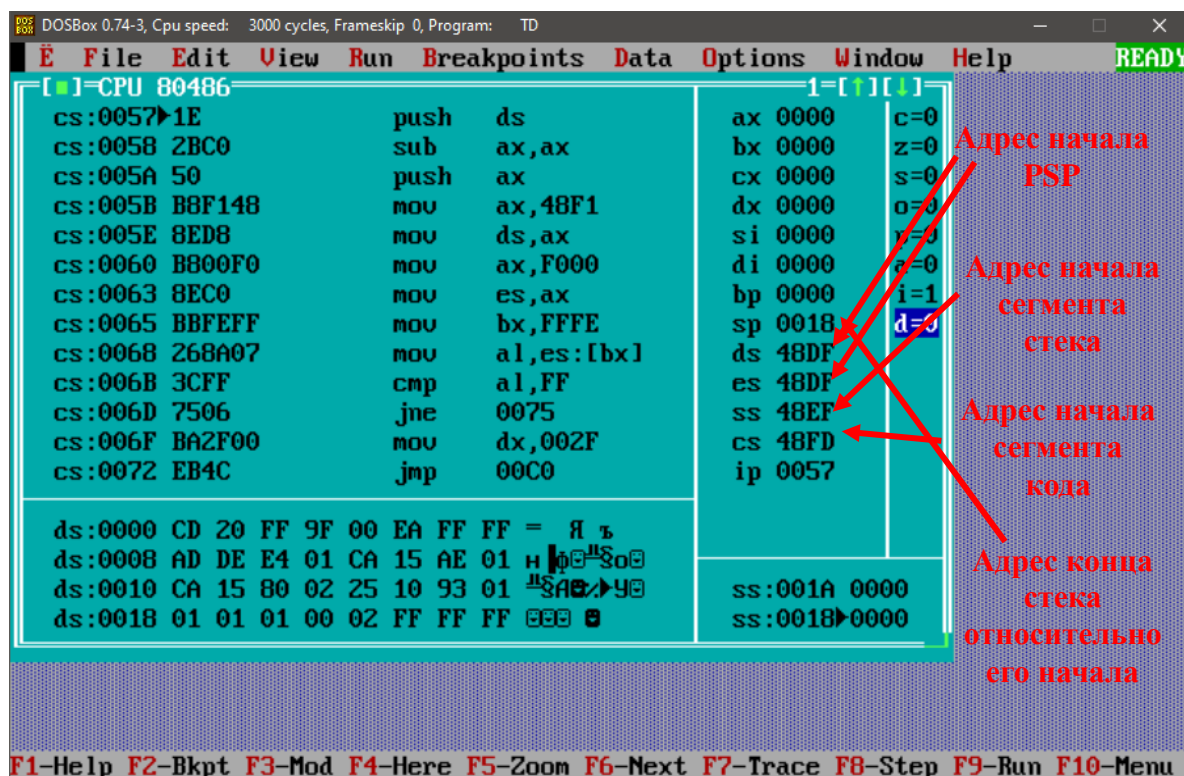


Рис. 7. Запуск программы .EXE в отладчике TD

- 2) На что указывают регистры DS и ES? Регистры DS и ES указывают на начало блока PSP.
- 3) Как определяется стек? Стек определяется следующим образом: регистр SS указывает на начало сегмента стека, а регистр SP – на конец сегмента стека относительно его начала.
- 4) Как определяется точка входа? Точка входа определяется названием метки в коде программы, стоящей после директивы END, с которой требуется начать выполнение программы. В данном случае такой меткой является начала главной процедуры Main.

Вывод.

В ходе лабораторной работы было проведено исследование различий в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память на примере собственной программы на языке Ассемблера, определяющей тип PC и версию системы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

1. Файл *LB1_COM.ASM* (исходный текст для *.COM* модуля):

```
LB_1 SEGMENT
    ASSUME CS:LB_1, DS:LB_1, ES:NOTHING, SS:NOTHING
    ORG 100H

START: JMP BEGIN

; Данные:
Message db 'Некорректный формат типа PC: встретилось: '
Number  dw 0
        db 0DH,0AH,'$'

FF db 'PC', 0DH,0AH,'$'
FE db 'PC/XT', 0DH,0AH,'$'
FC db 'AT', 0DH,0AH,'$'
FA db 'PC2 модель 30', 0DH,0AH,'$'
;FC db 'PC2 модель 50 или 60', 0DH,0AH,'$'
F8 db 'PC2 модель 80', 0DH,0AH,'$'
FD db 'PCjr', 0DH,0AH,'$'
F9 db 'PC Convertible', 0DH,0AH,'$'

Version_xx dw '00'
        db '.'
Version_yy dw '00'
        db 0DH,0AH,'$'

Serial_m db 'Серийный номер OEM: '
Serial_n dw 0
db 0DH,0AH, 'Серийный номер пользователя: '
Serial_user db 6 dup(0)
db 0DH,0AH, '$'

; Код (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
    push CX
    mov AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX

    pop CX
    ret
BYTE_TO_HEX ENDP

BYTE_TO_DEC PROC near
    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

WRD_TO_HEX PROC near
    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

```

; Код (программа):

```

BEGIN:
        mov AX, 0F000h
        mov ES, AX
        mov BX, 0FFFEh
        mov AL, ES:[BX]
cmp_1:   cmp AL, 0FFh
        jne cmp_2
        mov dx, offset FF
        jmp end_case
cmp_2:   cmp AL, 0FEh
        jne cmp_3
dop:    mov dx, offset FE
        jmp end_case
cmp_3:   cmp AL, 0FBh
        jne cmp_4
        jmp dop
cmp_4:   cmp AL, 0FCh
        jne cmp_5
        mov dx, offset FC
        jmp end_case
cmp_5:   cmp AL, 0FAh
        jne cmp_6
        mov dx, offset FA
        jmp end_case
cmp_6:   cmp AL, 0F8h
        jne cmp_7

```

```

                                mov dx, offset F8
                                jmp end_case
cmp_7:                          cmp AL, 0FDh
                                jne cmp_8
                                mov dx, offset FD
                                jmp end_case
cmp_8:                          cmp AL, 0F9h
                                jne default
                                mov dx, offset F9
                                jmp end_case
default:
                                call BYTE_TO_HEX
                                mov DI, offset number
                                mov word ptr DS:[DI], AX
                                mov dx, offset Message
end_case:
                                mov AH,09h
                                int 21h

                                mov AH, 30h
                                int 21h
                                mov DX, AX
                                mov SI, offset Version_xx
                                inc SI
                                call BYTE_TO_DEC
                                mov AL, DH
                                mov SI, offset Version_yy
                                inc SI
                                call BYTE_TO_DEC

                                mov DX, offset Version_xx
                                mov AH,09h
                                int 21h

                                mov AL, BH
                                call BYTE_TO_HEX
                                mov DI, offset Serial_n
                                mov word ptr DS:[DI], AX

                                mov AL, BL
                                call BYTE_TO_HEX

```

```

        mov DI, offset Serial_user
        mov word ptr DS:[DI], AX
        mov AX, CX
        mov DI, offset Serial_user
        add DI, 5
        call WRD_TO_HEX

        mov DX, offset Serial_m
        mov AH, 09h
        int 21h

        xor AL, AL
        mov AH, 4Ch
        int 21h

LB_1 ENDS
END START

```

2. Файл *LBI_EXE.ASM* (исходный текст для *.EXE* модуля):

```

AStack    SEGMENT    STACK
            DW 12 DUP(?)    ; Отводится 12 слов памяти
AStack    ENDS

; Данные программы

DATA      SEGMENT

        Message db 'Некорректный формат типа PC: встретилось: '
        Number  dw 0
            db 0DH, 0AH, '$'

        FF db 'PC', 0DH, 0AH, '$'
        FE db 'PC/XT', 0DH, 0AH, '$'
        FC db 'AT', 0DH, 0AH, '$'
        FA db 'PC2 модель 30', 0DH, 0AH, '$'
        ;FC db 'PC2 модель 50 или 60', 0DH, 0AH, '$'
        F8 db 'PC2 модель 80', 0DH, 0AH, '$'
        FD db 'PCjr', 0DH, 0AH, '$'
        F9 db 'PC Convertible', 0DH, 0AH, '$'

        Version_xx dw '00'
            db '.'

```

```

Version_yy dw '00'
            db 0DH,0AH,'$'

Serial_m db 'Серийный номер OEM: '
Serial_n dw 0
db 0DH,0AH, 'Серийный номер пользователя: '
Serial_user db 6 dup(0)
db 0DH,0AH, '$'

DATA        ENDS

; Код программы

CODE        SEGMENT
            ASSUME CS:CODE, 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
            push CX
            mov AH,AL
            call TETR_TO_HEX
            xchg AL,AH
            mov CL,4
            shr AL,CL
            call TETR_TO_HEX

            pop CX
            ret
BYTE_TO_HEX ENDP

BYTE_TO_DEC PROC near
            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

WRD_TO_HEX PROC near
        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

Main      PROC  FAR
            push  DS
            sub   AX,AX
            push  AX
            mov   AX, DATA

```

```

        mov     DS, AX

        mov AX, 0F000h
        mov ES, AX
        mov BX, 0FFFEh
        mov AL, ES:[BX]
cmp_1:   cmp AL, 0FFh
        jne cmp_2
        mov dx, offset FF
        jmp end_case
cmp_2:   cmp AL, 0FEh
        jne cmp_3
dop:     mov dx, offset FE
        jmp end_case
cmp_3:   cmp AL, 0FBh
        jne cmp_4
        jmp dop
cmp_4:   cmp AL, 0FCh
        jne cmp_5
        mov dx, offset FC
        jmp end_case
cmp_5:   cmp AL, 0FAh
        jne cmp_6
        mov dx, offset FA
        jmp end_case
cmp_6:   cmp AL, 0F8h
        jne cmp_7
        mov dx, offset F8
        jmp end_case
cmp_7:   cmp AL, 0FDh
        jne cmp_8
        mov dx, offset FD
        jmp end_case
cmp_8:   cmp AL, 0F9h
        jne default
        mov dx, offset F9
        jmp end_case
default:
        call BYTE_TO_HEX
        mov DI, offset number
        mov word ptr DS:[DI], AX

```

```

                                mov dx, offset Message
end_case:
                                mov AH,09h
                                int 21h

                                mov AH, 30h
                                int 21h
                                mov DX, AX
                                mov SI, offset Version_xx
                                inc SI
                                call BYTE_TO_DEC
                                mov AL, DH
                                mov SI, offset Version_yy
                                inc SI
                                call BYTE_TO_DEC

                                mov DX, offset Version_xx
                                mov AH,09h
                                int 21h

                                mov AL, BH
                                call BYTE_TO_HEX
                                mov DI, offset Serial_n
                                mov word ptr DS:[DI], AX

                                mov AL, BL
                                call BYTE_TO_HEX
                                mov DI, offset Serial_user
                                mov word ptr DS:[DI], AX
                                mov AX, CX
                                mov DI, offset Serial_user
                                add DI, 5
                                call WRD_TO_HEX

                                mov DX, offset Serial_m
                                mov AH,09h
                                int 21h

                                ret
Main      ENDP
CODE      ENDS

```


END Main