

**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. 8383

Кормщикова А. О.

Преподаватель

Ефремов М. А.

Санкт-Петербург

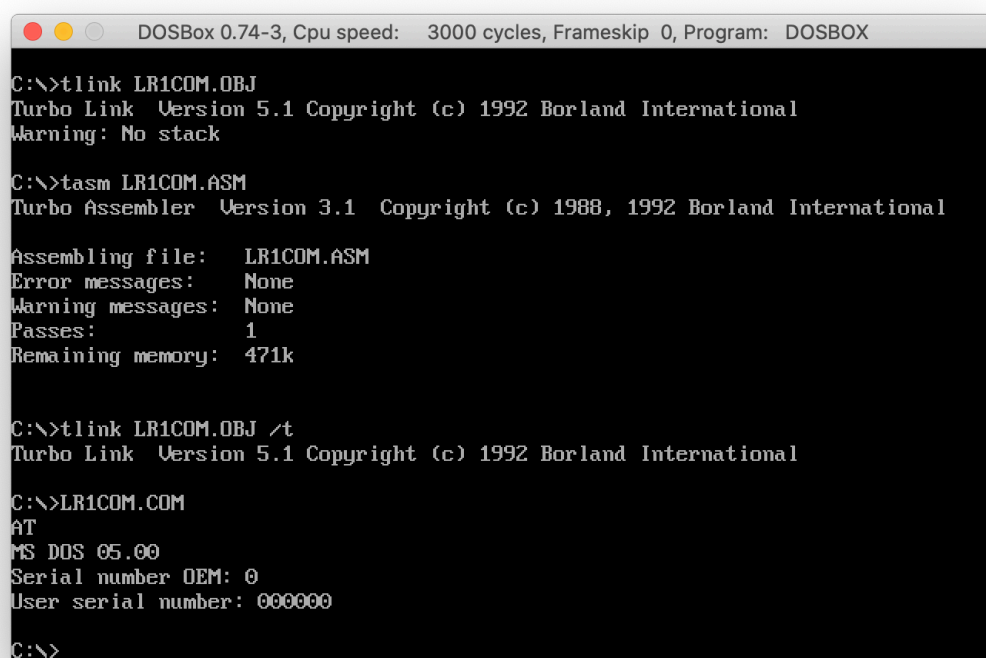
2020

### 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.

Был написан код исходного **.COM** модуля (LR1COM.ASM представлен в приложении А), который определяет тип PC и версию системы. Ассемблерная программа читает содержимое предпоследнего байта ROM BIOS, по коду определяет тип PC и выводит строку с названием модели. Из данного кода были собраны "хороший" **.COM** модуль и "плохой" **.EXE** модуль. Во время линковки "плохого" модуля было выведено предупреждение об отсутствии стека. Результаты выполнения **.COM** модуля представлены на рис. 1, **.EXE** модуля на рис. 2.



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

C:\>tlink LR1COM.OBJ
Turbo Link Version 5.1 Copyright (c) 1992 Borland International
Warning: No stack

C:\>tasm LR1COM.ASM
Turbo Assembler Version 3.1 Copyright (c) 1988, 1992 Borland International

Assembling file: LR1COM.ASM
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 471k

C:\>tlink LR1COM.OBJ /t
Turbo Link Version 5.1 Copyright (c) 1992 Borland International

C:\>LR1COM.COM
AT
MS DOS 05.00
Serial number OEM: 0
User serial number: 000000

C:\>
```

Рисунок 1 - Результат выполнения **.COM** модуля



## Отличия исходных текстов COM и EXE программ

### 1. Сколько сегментов должна содержать COM-программа?

COM-программа содержит ровно один сегмент

### 2. EXE-программа?

EXE-программа должна содержать не менее одного сегмента

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

Должна присутствовать директива `assume` в которой инициализируются регистры, `CS` и `DS` указывают на общий сегмент. `ORG 100h`, которая резервирует первые 256 байт под `PSP`, директива говорит, что вся адресация внутри кода смещена на эти байты.

`ORG 100h`, для размещения `PSP`. `Assume` для инициализации регистров

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

Модуль такого типа не содержит таблицы настроек, поэтому некорректно указание адреса сегмента.

При помощи программы `FAR` были открыты загрузочные модули. Вид модулей в шестнадцатеричном виде представлен на рис. 4-6.

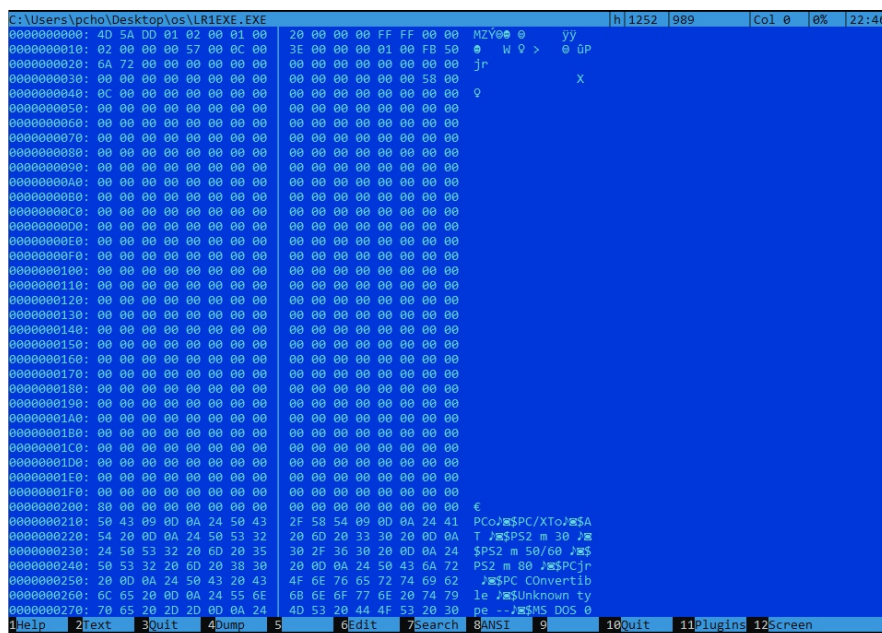


Рисунок 4 - Содержимое файла "хорошего" .EXE модуля

```

C:\Users\pcho\Desktop\os\LRICOM.COM h|1252|455 Col 0 100% 22:47
00000000: E9 03 01 50 43 09 00 0A 24 50 43 2F 58 54 09 0D 6V0PC0;W$PC/XTo>
00000001: 0A 24 41 54 20 00 0A 24 50 53 32 20 60 20 35 30 2F 36 30 20 W$AT /W$P$2 m 30
00000002: 20 00 0A 24 50 53 32 20 60 20 38 30 20 00 0A 24 50 /W$P$2 m 50/60
00000003: 00 0A 24 50 53 32 20 60 20 38 30 20 00 0A 24 50 /W$P$2 m 80 /W$P
00000004: 43 6A 72 20 00 0A 24 50 43 20 43 4F 6E 76 65 72 Cjr /W$PC Conver
00000005: 74 69 62 6C 65 20 00 0A 24 55 6E 68 6E 6F 77 6E tible /W$Unknown
00000006: 20 74 79 70 65 20 2D 2D 00 0A 24 4D 53 20 44 4F type --/W$MS DO
00000007: 53 20 30 30 2E 30 30 00 0A 24 53 65 72 69 61 6C S 00.00/W$Serial
00000008: 20 6E 75 6D 62 65 72 20 4F 45 4D 3A 20 20 20 20 number OEM:
00000009: 00 0A 24 55 73 65 72 20 73 65 72 69 61 6C 20 6E /W$User serial n
0000000A: 75 6D 62 65 72 3A 20 20 20 20 20 00 0A 24 24 umber: /W$S
0000000B: 0F 3C 09 76 02 04 07 04 30 C3 51 8A E0 E8 EF FF 0cov0+0AQ$Aaiy
0000000C: 86 C4 B1 04 D2 E8 E8 E6 FF 59 C3 53 8A FC E8 E9 tAz+0eeyyV$Suee
0000000D: FF 88 25 4F 88 05 4F 8A C7 E8 DE FF 88 25 4F 88 y"0"05Capy"0"
0000000E: 05 5B C3 51 52 32 E4 33 D2 B9 0A 00 F7 F1 80 CA +[AQ2a30"e +nE
0000000F: 30 88 14 4E 33 D2 3D 0A 00 73 F1 3C 00 74 04 0C 0"n30=8 sn< t+0
00000010: 30 88 04 5A 59 C3 B8 00 F0 8E C0 26 A1 FE FF 3C 0-*ZYA, 07A8;by<
00000011: FF 74 31 3C FE 74 33 3C FB 74 2F 3C FC 74 31 3C yt1<pt3<0t/<0t1<
00000012: FA 74 33 3C FC 74 35 3C F8 74 37 3C FD 74 39 3C 0t3<0t5<0t7<yt9<
00000013: F9 74 3B E8 84 FF BB 59 01 88 47 0E 88 67 0F 88 0t;B,y>Y0"G"ge<
00000014: D3 EB 2E 90 BA 03 01 EB 28 90 BA 09 01 EB 22 90 0e.0e0e(0e0e0e"0
00000015: BA 12 01 EB 1C 90 BA 18 01 EB 16 90 BA 24 01 EB 0+0eL0e=0e$0e
00000016: 10 90 BA 33 01 EB 0A 90 BA 3F 01 EB 04 90 BA 47 >0e30e0e0e70e+0eG
00000017: 01 B4 09 CD 21 B4 30 CD 21 50 BE 6B 01 83 C6 08 0'oi! 0i!PKkefA
00000018: E8 60 FF 58 8A C4 83 C6 03 E8 57 FF BA 6B 01 B4 0'yx$A$efWwy0k0'
00000019: 09 CD 21 BE 7A 01 83 C6 13 8A C7 E8 45 FF BA 7A 0i!kzofA!!5C0Eyz
0000001A: 01 B4 09 CD 21 BF 93 01 83 C7 19 8B C1 E8 1B FF 0'oi!;"0fC1;Ae-y
0000001B: 8A C3 E8 05 FF 83 EF 02 89 05 BA 93 01 B4 09 CD 0Aeyfies+e"0'oi
0000001C: 21 32 C0 B4 4C CD 21 12A"LI!
```

Рисунок 5 - Содержимое модуля .COM

```

C:\Users\pcho\Desktop\os\LRICOM.EXE h|1252|1223 Col 0 0% 23:09
00000000: 4D 5A C7 00 03 00 00 00 20 00 00 00 FF FF 00 00 MZC y
00000001: 00 00 00 00 00 01 00 00 3E 00 00 00 01 00 FB 50 0 > 0 0P
00000002: 6A 72 00 00 00 00 00 00 00 00 00 00 00 00 00 00 jr
00000003: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000004: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000005: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000006: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000007: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000008: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000009: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000011: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000012: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000013: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000014: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000015: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000017: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000018: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000019: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000021: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000022: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000023: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000024: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000025: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000026: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000027: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000028: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000029: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030: E9 03 01 50 43 09 00 0A 24 50 43 2F 58 54 09 0D 6V0PC0;W$PC/XTo>
00000031: 0A 24 41 54 20 00 0A 24 50 53 32 20 60 20 35 30 2F 36 30 20 W$AT /W$P$2 m 30
00000032: 20 00 0A 24 50 53 32 20 60 20 38 30 20 00 0A 24 50 /W$P$2 m 50/60
00000033: 00 0A 24 50 53 32 20 60 20 38 30 20 00 0A 24 50 /W$P$2 m 80 /W$P
00000034: 43 6A 72 20 00 0A 24 50 43 20 43 4F 6E 76 65 72 Cjr /W$PC Conver
00000035: 74 69 62 6C 65 20 00 0A 24 55 6E 68 6E 6F 77 6E tible /W$Unknown
00000036: 20 74 79 70 65 20 2D 2D 00 0A 24 4D 53 20 44 4F type --/W$MS DO
00000037: 53 20 30 30 2E 30 30 00 0A 24 53 65 72 69 61 6C S 00.00/W$Serial
00000038: 20 6E 75 6D 62 65 72 20 4F 45 4D 3A 20 20 20 20 number OEM:
00000039: 00 0A 24 55 73 65 72 20 73 65 72 69 61 6C 20 6E /W$User serial n
0000003A: 75 6D 62 65 72 3A 20 20 20 20 20 00 0A 24 24 umber: /W$S
0000003B: 0F 3C 09 76 02 04 07 04 30 C3 51 8A E0 E8 EF FF 0cov0+0AQ$Aaiy
0000003C: 86 C4 B1 04 D2 E8 E8 E6 FF 59 C3 53 8A FC E8 E9 tAz+0eeyyV$Suee
```

Рисунок 6 - Содержимое файла " плохого" .EXE модуля



## Отличие форматов файлов COM и EXE модулей

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

COM файл содержит единственный сегмент с кодом и данными. В памяти код располагается начиная с 0h.

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

"Плохой" EXE файл содержит один сегмент с кодом и данными, который начинается с адреса 300h. До машинного кода идет таблица настроек.

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

Данные, стек, код находятся в разных сегментах, в отличие от "плохого" файла с одним сегментом и отсутствием стека.

С помощью отладчика TD были загружены модули COM и EXE. см. рис 7-8.

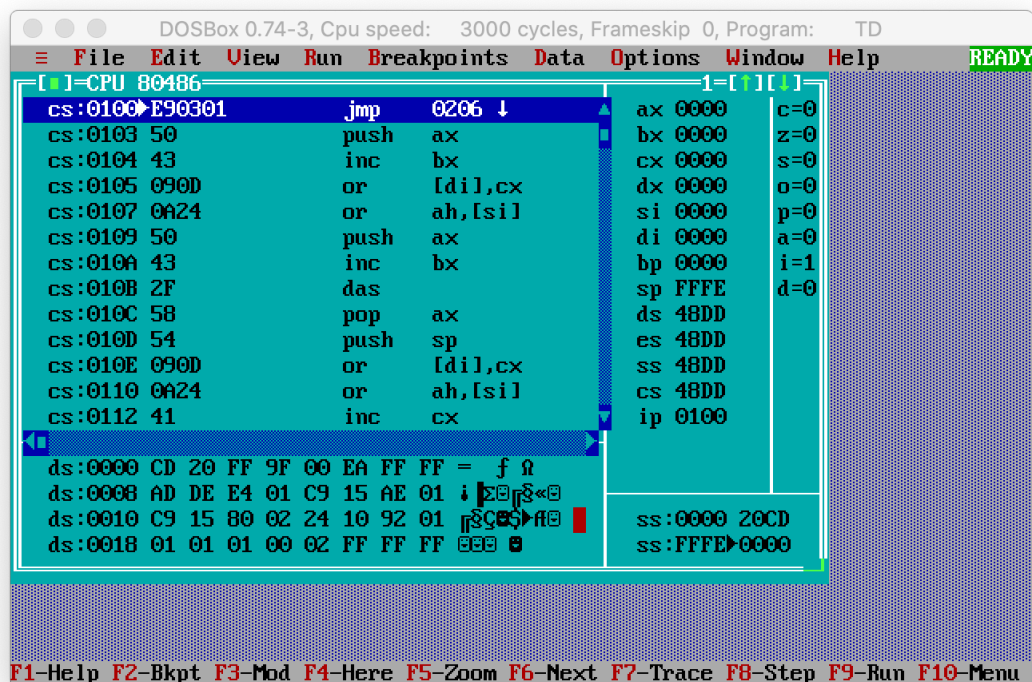


Рисунок 7 - модуль COM в отладчике TD

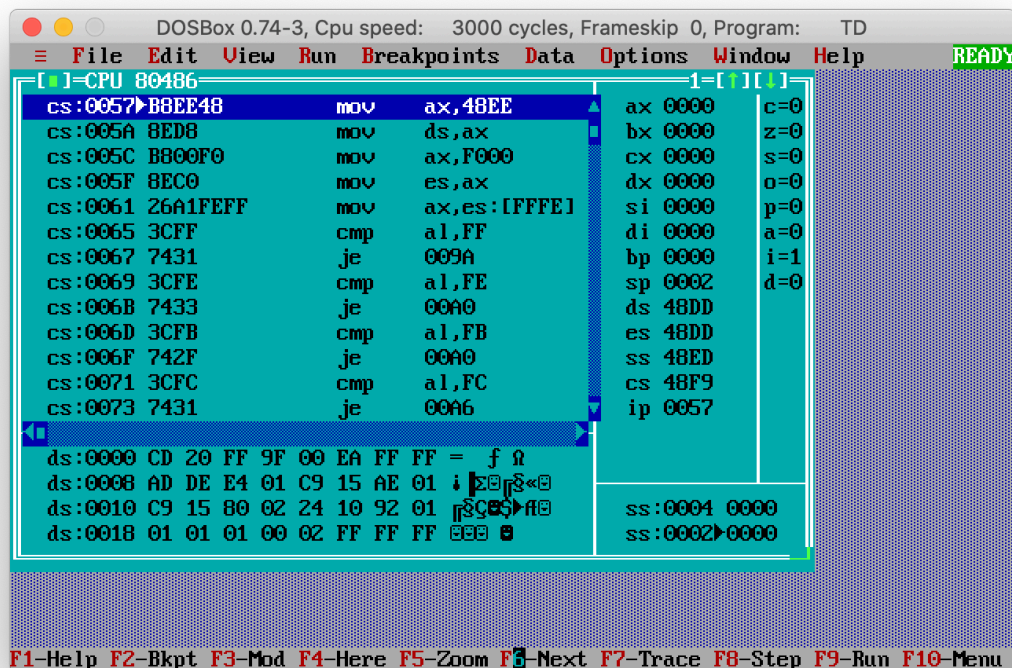


Рисунок 8 - модуль EXE в отладчике TD

### Загрузка COM модуля в основную память

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

COM файл загружается со смещением 100h, код и данные располагаются в памяти с этого адреса.

2. Что располагается с адреса 0?

С адреса 0h при загрузке ОС располагает PSP

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

Все сегментные регистры имеют одинаковое значение и указывают на PSP.

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

Стек занимает всю доступную память после кода. При загрузке SP устанавливается в FFFh.

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

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

При загрузке сегментные регистры CS и SS устанавливаются в начало соответствующего сегмента, DS и ES устанавливаются на начало PSP. В IP загружается смещение точки входа в программу.

2. На что указывают регистры DS и ES?

DS и ES устанавливаются на начало PSP

3. Как определяется стек?

Стек определяется с помощью директивы `assume SS:SSTACK`. Где SSTACK - сегмент, отведенный под стек.

4. Как определяется точка входа?

Если точка входа не указана явно, то ей является начало сегмента кода. В программе точка входа указывается при помощи директивы `END <метка>`, где метка - точка входа в программу.

### **Выводы.**

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



## ПРИЛОЖЕНИЕ

### Содержимое файла "хорошего" EXE модуля:

```
SSTACK SEGMENT STACK
    DW 128
SSTACK ENDS

DATA SEGMENT

    pcff db 'PC ', 0DH, 0AH, '$'
    pcxt db 'PC/XT ', 0DH, 0AH, '$'
    atfc db 'AT ', 0DH, 0AH, '$'
    ps30 db 'PS2 m 30 ', 0DH, 0AH, '$'
    ps50 db 'PS2 m 50/60 ', 0DH, 0AH, '$'
    ps80 db 'PS2 m 80 ', 0DH, 0AH, '$'
    pcjr db 'PCjr ', 0DH, 0AH, '$'
    pccm db 'PC CONvertible ', 0DH, 0AH, '$'
    unkt db 'Unknown type --', 0DH, 0AH, '$'
    ver db 'MS DOS 00.00', 0DH, 0AH, '$'
    oem db 'Serial number OEM: ', 0DH, 0AH, '$'
    usn db 'User serial number: ', 0DH, 0AH, '$'

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, SS:SSTACK

;-----
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 в 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 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 10 с/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

MAIN PROC FAR

        mov ax, DATA
        mov ds, ax
;type
        mov ax,0F000h
        mov es, ax
        mov ax, es:[0FFFEh]

        cmp al, 0FFh
        je PC

        cmp al, 0FEh
        je XT

        cmp al, 0FBh
        je XT

        cmp al, 0FCh
        je AT

        cmp al, 0FAh
        je PS230

        cmp al, 0FCh
        je PS250

        cmp al, 0F8h
        je PS280

```

```

        cmp al, 0FDh
        je PCJ

        cmp al, 0F9h
        je PCC

;UNKNOWN TYPE

        call BYTE_TO_HEX
        mov bx, offset unkt
        mov [bx+14], al
        mov [bx+15], ah
        mov dx, bx
        jmp res

PC:
        mov dx, offset pcff
        jmp res

XT:
        mov dx, offset pcxt
        jmp res

AT:
        mov dx, offset atfc
        jmp res

PS230:
        mov dx, offset ps30
        jmp res

PS250:
        mov dx, offset ps50
        jmp res

PS280:
        mov dx, offset ps80
        jmp res

PCJ:
        mov dx, offset pcjr
        jmp res

PCC:
        mov dx, offset pccm

RES:
        mov ah, 09h
        int 21h

;os ver
        mov ah, 30h
        int 21h

        push ax
        mov si, offset ver
        add si, 8
        call BYTE_TO_DEC
        pop ax
        mov al, ah
        add si, 3
        call BYTE_TO_DEC
        mov dx, offset ver

```

```

mov ah, 09h
int 21h

mov si, offset oem
add si, 19
mov al, bh
call BYTE_TO_DEC
mov dx, offset oem
mov ah, 09h
int 21h

mov di, offset usn
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 usn
mov ah, 09h
int 21h

;exit to dos

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

MAIN      ENDP
CODE ENDS
END MAIN

```

## Содержимое .COM модуля:

```

TESTPC SEGMENT
        ASSUME  CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
        ORG     100H
START:   JMP     BEGIN

;DATA

pcff db 'PC ', 0DH, 0AH, '$'
pcxt db 'PC/XT ', 0DH, 0AH, '$'
atfc db 'AT ', 0DH, 0AH, '$'
ps30 db 'PS2 m 30 ', 0DH, 0AH, '$'
ps50 db 'PS2 m 50/60 ', 0DH, 0AH, '$'
ps80 db 'PS2 m 80 ', 0DH, 0AH, '$'
pcjr db 'PCjr ', 0DH, 0AH, '$'
pccm db 'PC COnvertible ', 0DH, 0AH, '$'
unkt db 'Unknown type --', 0DH, 0AH, '$'
ver db 'MS DOS 00.00', 0DH, 0AH, '$'
oem db 'Serial number OEM: ', 0DH, 0AH, '$'
usn db 'User serial number: ', 0DH, 0AH, '$'

;-----
TETR_TO_HEX PROC near
        and     AL,0Fh
        cmp     AL,09
        jbe     NEXT
        add     AL,07
NEXT:

```

```

NEXT:      add      AL,30h
           ret
TETR_TO_HEX ENDP

;-----
BYTE_TO_HEX PROC near
; байт в AL переводится в два символа в 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 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 10 с/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
```

```
BEGIN:
```

```
;type
```

```
    mov ax,0F000h
    mov es, ax
    mov ax, es:[0FFFEh]
```

```
    cmp al, 0FFh
    je PC
```

```
    cmp al, 0FEh
    je XT
```

```
    cmp al, 0FBh
    je XT
```

```
    cmp al, 0FCh
    je AT
```

```
    cmp al, 0FAh
    je PS230
```

```
    cmp al, 0FCh
    je PS250
```

```
    cmp al, 0F8h
    je PS280
```

```
    cmp al, 0FDh
    je PCJ
```

```
    cmp al, 0F9h
    je PCC
```

```
;UNKNOWN TYPE
```

```
    call BYTE_TO_HEX
    mov bx, offset unkt
    mov [bx+14], al
    mov [bx+15], ah
    mov dx, bx
    jmp res
```

```
PC:
```

```
    mov dx, offset pcff
    jmp res
```

```
XT:
```

```
    mov dx, offset pcxt
    jmp res
```

```
AT:
```

```
    mov dx, offset atfc
    jmp res
```



```

PS230:
    mov dx, offset ps30
    jmp res

PS250:
    mov dx, offset ps50
    jmp res

PS280:
    mov dx, offset ps80
    jmp res

PCJ:
    mov dx, offset pcjr
    jmp res

PCC:
    mov dx, offset pccm

RES:
    mov ah, 09h
    int 21h

;os ver
    mov ah, 30h
    int 21h

    push ax
    mov si, offset ver
    add si, 8
    call BYTE_TO_DEC
    pop ax
    mov al, ah
    add si, 3
    call BYTE_TO_DEC
    mov dx, offset ver
    mov ah, 09h
    int 21h

    mov si, offset oem
    add si, 19
    mov al, bh
    call BYTE_TO_DEC
    mov dx, offset oem
    mov ah, 09h
    int 21h

    mov di, offset usn
    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 usn
    mov ah, 09h
    int 21h

;exit to dos

```

```
        xor AL,AL
        mov AH,4Ch
        int 21H

TESTPC      ENDS
END START
```