

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

**ОТЧЕТ**  
**по лабораторной работе №1**  
**по дисциплине «Организация ЭВМ и систем»**  
**Тема: «Трансляции, отладка и выполнение программ**  
**на языке Ассемблера»**

Студент гр. 8381

Преподаватель

\_\_\_\_\_  
\_\_\_\_\_

Переверзев Д.Е.

Ефремов М.А.

Санкт-Петербург

2019

### **Цель работы.**

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

### **Основные теоретические положения.**

Тип IBM PC хранится в байте по адресу 0F000:0FFFFh, в предпоследнем байте ROM BIOS. Соответствие кода и типа представлены в табл.1.

Таблица 1 – Соответствие кода и типа PC

Тип IBM PC	Код
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. Выходными параметрами являются:

- 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.

### Выполнение работы.

Выполнение работы производилось на базе операционной системы Mac OS в редакторе VS Code. Сборка и отладка модулей производились с помощью компилятора MASM и отладчика AFD.

Был написан текст исходного .COM модуля, который определяет тип РС и информацию о системе. Полученные модуль были отлажены, и в результате были получены «плохой» .EXE модуль и «хороший» .COM модуль (с помощью программы exe2bin).

Результат выполнения «плохого» .EXE модуля представлен на рис. 1.

```
C:\USERS\DMITRI~1\DESKTOP\OS\LR1>com.exe
```

	Modification number:					
	5 0		255			000000
	Modification number:					
255						000000
	Modification n					
umber:	.					
		000000				
	Modification number:					

Рисунок 1 – Вывод «плохого» .EXE модуля

Результат выполнения «хорошего» .COM модуля представлен на рис. 2.

```
C:\USERS\DMITRI~1\DESKTOP\OS\LR1>com.com
TypePC: AT
Modification number: 5.0
DEM:255
User serial number: 000000
```

Рисунок 2 – Вывод «хорошего» .COM модуля

Результат выполнения «хорошего» .EXE модуля представлен на рис. 3.

```
C:\USERS\DMITRI~1\DESKTOP\OS\LR1>exe.exe
TypePC: AT
Modification number: 5.0
DEM:255
User serial number: 000000
```

Рисунок 3 – Вывод «хорошего» .EXE модуля

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

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

.COM - программы содержат только один сегмент. Модель памяти tiny - код, данные и стек объединены в один физический сегмент, максимальный размер которого не мог превышать 64 Кбайта без 256 байтов .

### **2. EXE-программа?**

EXE-программа может содержать несколько сегментов. При использовании модели памяти small, в программе должен содержаться один сегмент данных и один сегмент кода в разных физических сегментах и каждый из них не может превосходить 64 Кбайта. При других моделях памяти (например large) есть возможность использования нескольких сегментов данных и (или) нескольких сегментов кода.

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

При запуске COM-программы первые 256 байт необходимо зарезервировать для префикса программного сегмента (PSP). Для этого используется команда ORG 100h, которая устанавливает относительный адрес для начала выполнения программы.

Также обязательна директива ASSUME, которая устанавливает значения регистров CS, DS на адрес сегмента программы.

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

Нельзя использовать команды, связанные с адресом сегмента, так как он неизвестен до загрузки этого сегмента в память. Это связано с тем, что в .COM файле отсутствует таблица настройки с информацией о типе адресов и их местоположении.

## Отличия форматов файлов COM и EXE модулей.

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

COM-файл состоит из одного сегмента, а размер файла не превышает 64 КБ. Код располагается с адреса 0h.

[illegible]

Рисунок 4 – Вид COM файла в шестнадцатеричном виде.

СОМ-файл состоит из одного сегмента, а размер файла не превышает 64 КБ. Код располагается с адреса 0h. Данные оканчиваются В4. На В5 начинается сегмент кода, в котором байты отвечают за команды.

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

COM.EXE x																	
00000000	4D	5A	2C	01	03	00	00	00	20	00	00	00	FF	FF	00	00	Z,.....
00000010	00	00	6F	E3	00	01	00	00	1E	00	00	00	01	00	00	00	...оп.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000200	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000210	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000220	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000230	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000240	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000250	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000260	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000270	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000280	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000290	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000002A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000002B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000002C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000002D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000002E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000002F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000300	E9	96	01	4D	6F	64	69	66	69	63	61	74	69	6F	6E	20	e0.Modification
00000310	6E	75	6D	62	65	72	3A	20	20	20	2E	20	20	20	0D	0A	number: . . .
00000320	24	4F	45	4D	3A	20	20	20	20	0D	0A	24	55	73	65	72	\$OEM: ..\$User
00000330	20	73	65	72	69	61	6C	20	6E	75	6D	62	65	72	3A	20	serial number:
00000340	20	20	20	20	20	20	0D	0A	24	3F	3F	68	24	54	79	70	..\$??h\$Typ
00000350	65	50	43	3A	20	50	43	0D	0A	24	54	79	70	65	50	43	ePC: PC..\$TypePC
00000360	3A	20	50	43	2F	58	54	0D	0A	24	54	79	70	65	50	43	: PC/XT..\$TypePC
00000370	3A	20	41	54	0D	0A	24	54	79	70	65	50	43	3A	20	50	: AT..\$TypePC: P

Рисунок 4 – Вид «плохого» EXE файла в шестнадцатеричном виде.

У «плохого» .EXE модуля данные и код располагаются в одном сегменте, так как он был получен текста для .COM модуля. Код располагается по адресу 300h, так как перед ним начиная с адреса 0h находится таблица настроек.

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

[illegible]

Рисунок 5 – Вид «хорошего» EXE файла

В «хорошем» EXE с нулевого адреса также располагается управляющая информация для загрузчика. Также перед кодом располагается сегмент стека.

Так, при размере стека 200h код располагается с адреса 400h. Отличие от «плохого» EXE в том, что в «хорошем» не резервируется дополнительно 100h, которые в COM файле требовались для PSP.

## Загрузка COM модуля в основную память.

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

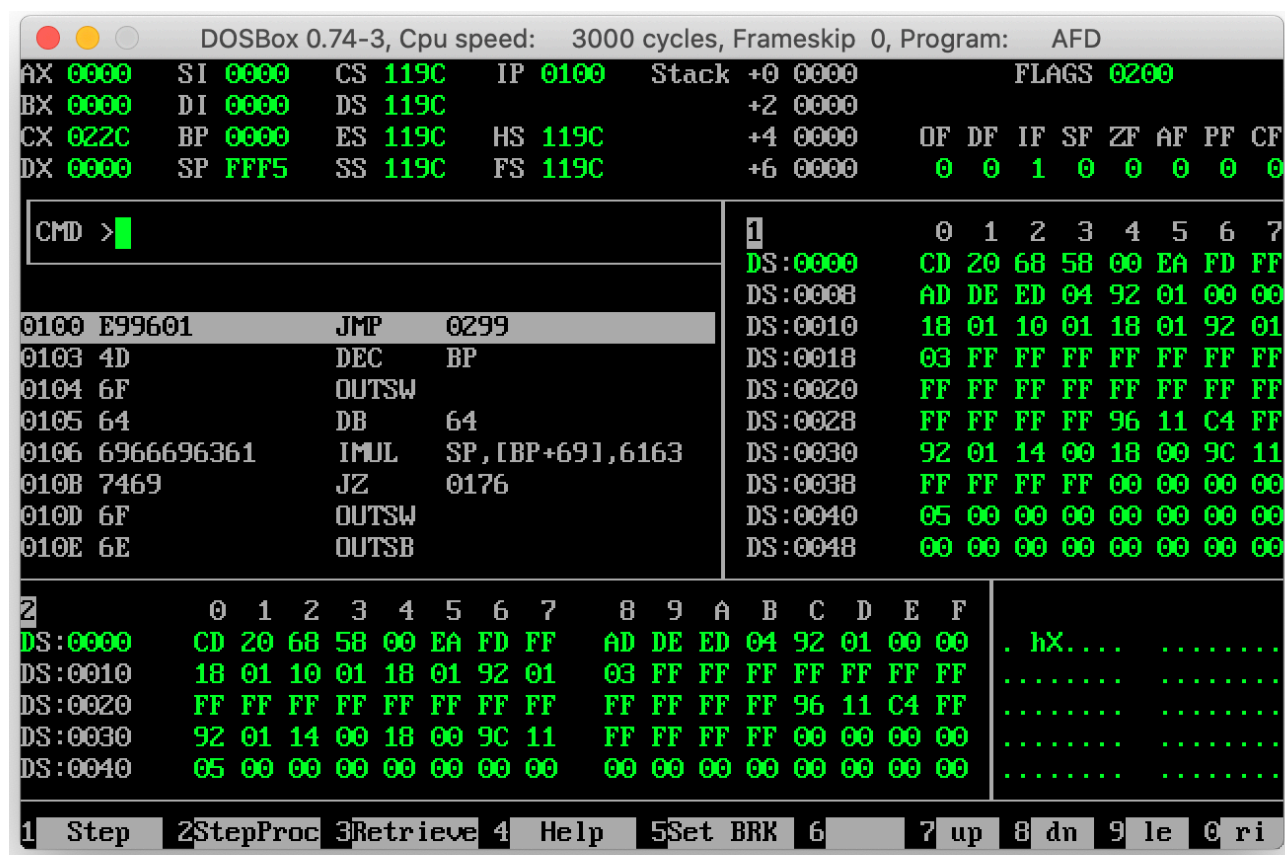


Рисунок 7 – Отладка файла .COM

Определяется сегментный адрес свободного участка ОП, в который можно загрузить программу. Создается блок памяти. В поля PSP заносятся значения. Загружается COM файл со смещением 100h. Сегментные регистры устанавливаются на адрес сегмента PSP, регистр SP указывает на конец сегмента, туда записывается 0000h. С ростом стека значение SP будет уменьшаться. Счетчик команд принимает значение 100h. Программа запускается.



## 2. Что располагается с адреса 0?

С адреса 0 располагается сегмент PSP

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

Сегментные регистры имеют значения 48DDh и указывают на PSP.

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

В COM модуле нельзя объявить стек, он создается автоматически. SP имеет указывает на FFFeh. Стек занимает оставшуюся память, а его адреса изменяются от больших к меньшим, то есть от FFFeh к 0000h.

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

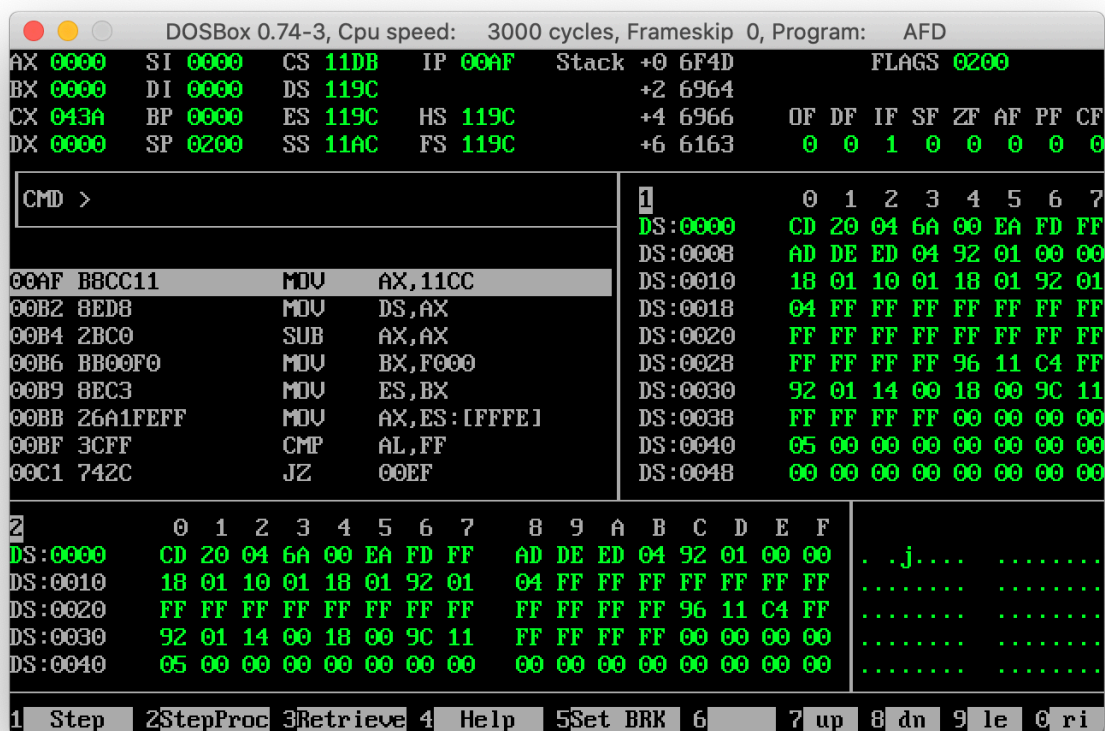


Рисунок 8 – Отладка «хорошего» EXE модуля

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

Для PSP и программы выделяется блок памяти. После запуска программы DS и ES указывают на начало PSP, CS – на начало сегмента команд, а SS – на

начало сегмента стека. IP имеет значение отличное от нуля, так как в программе есть дополнительные процедуры, расположенные до основной.

## **2. На что указывают регистры DS и ES?**

Изначально регистры DS и ES указывают на начало сегмента PSP. Именно поэтому в начале программы для корректной работы с данными необходимо загрузить в DS адрес сегмента данных.

## **3. Как определяется стек?**

Стек может быть объявлен при помощи директивы ASSUME, которая устанавливает сегментный регистр SS на начало сегмента стека, а также задает значение SP, указанное в заголовке. Также стек может быть объявлен с помощью директивы STACK. Если стек не объявлять, то он будет создан автоматически.

## **4. Как определяется точка входа?**

Смещение точки входа в программу загружается в указатель команд IP и определяется операндом (функцией или меткой, с которой необходимо начать программу) директивы END <метка для входа>, который называется точкой входа.

## **Вывод**

В ходе выполнения лабораторной работы были исследованы различия в структурах исходных текстов модулей типов .EXE и .COM . Так же были отлажены и запущены «хорошие» EXE и COM модули и «плохой» EXE.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ. exe.asm

\_STACK SEGMENT STACK

DW 100h DUP(0)

\_STACK ENDS

\_DATA SEGMENT

ModifNum db 'Modification number: . ', 0dh, 0ah, '\$'

OEM db 'OEM: ', 0dh, 0ah, '\$'

UserSerialNum db 'User serial number: ', 0dh, 0ah, '\$'

Type\_PC\_Other db 2 dup ('?'), 'h\$'

Type\_PC db 'TypePC: PC', 0DH, 0AH, '\$'

Type\_PC\_XT db 'TypePC: PC/XT', 0DH, 0AH, '\$'

Type\_AT db 'TypePC: AT', 0DH, 0AH, '\$'

Type\_PS2\_30 db 'TypePC: PC2 model 30', 0DH, 0AH, '\$'

Type\_PS2\_50 db 'TypePC: PC2 model 50 or 60', 0DH, 0AH, '\$'

Type\_PS2\_80 db 'TypePC: PC2 model 80', 0DH, 0AH, '\$'

Type\_PCjr db 'TypePC: PCjr', 0DH, 0AH, '\$'

Type\_PC\_Conv db 'TypePC: PC Convertible', 0DH, 0AH, '\$'

\_DATA ENDS

\_CODE SEGMENT

ASSUME CS:\_CODE, DS:\_DATA, ES:NOTHING, SS:\_STACK

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     al,ah
    call    TETR_TO_HEX
    xchg    al,ah
    mov     cl,4
    shr     al,cl
    call    TETR_TO_HEX
    pop     cx
    ret
```

BYTE\_TO\_HEX 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
    xor     ah,ah
    call    BYTE_TO_HEX
    mov     [di],ah
    dec     di
    mov     [di],al
    pop     bx
    ret
```

WRD\_TO\_HEX ENDP

BYTE\_TO\_DEC PROC NEAR

```
    push    cx
    push    dx
    push    ax
    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     ax,00h
    jbe     end_l
    or      al,30h
    mov     [si],al
end_l: pop    ax
    pop     dx
    pop     cx
    ret
BYTE_TO_DEC ENDP
```

PRINT PROC NEAR

```
    push    ax
    mov     ah,09h
    int     21h
    pop     ax
    ret
```

PRINT ENDP

MOD\_PC PROC NEAR

```
    push    ax
    push    si
    mov     si, offset ModifNum
    add     si, 22
    call    BYTE_TO_DEC
    add     si, 3
    mov     al, ah
    call    BYTE_TO_DEC
    pop     si
    pop     ax
    ret
```

MOD\_PC ENDP

OEM\_PC PROC NEAR

```
    push    ax
    push    bx
    push    si
    mov     al,bh
    lea     si, OEM
    add     si, 6
    call    BYTE_TO_DEC
    pop     si
    pop     bx
    pop     ax
    ret
```

OEM\_PC ENDP

SER\_PC PROC NEAR

```
    push  ax
    push  bx
    push  cx
    push  si
    mov   al,bl
    call  BYTE_TO_HEX
    lea   di,UserSerialNum
    add   di,20
    mov   [di],ax
    mov   ax,cx
    lea   di,UserSerialNum
    add   di,25
    call  WRD_TO_HEX
    pop   si
    pop   cx
    pop   bx
    pop   ax
    ret
```

SER\_PC ENDP

MAIN PROC NEAR

```
    mov   ax, _DATA
    mov   ds, ax
    sub   ax, ax
    mov   bx, 0F000h
    mov   es, bx
    mov   ax, es:[0FFFEh]
    cmp   al, 0FFh
    je    _PC
    cmp   al, 0FEh
```

```
je    _PC_XT
cmp    al, 0FBh
je    _PC_XT
cmp    al, 0FCh
je    _AT
cmp    al, 0FAh
je    _PS2_30
cmp    al, 0F8h
je    _PS2_80
cmp    al, 0FDh
je    _PCjr
cmp    al, 0F9h
je    _PC_Conv
call   BYTE_TO_HEX
lea    di, Type_PC_Other
mov    [di], ax
lea    dx, Type_PC_Other
jmp    _EndPC
```

```
_PC:  lea    dx, Type_PC
      jmp    _EndPC
_PC_XT: lea    dx, Type_PC_XT
      jmp    _EndPC
_AT:   lea    dx, Type_AT
      jmp    _EndPC
_PS2_30: lea    dx, Type_PS2_30
      jmp    _EndPC
_PS2_80: lea    dx, Type_PS2_80
      jmp    _EndPC
_PCjr: lea    dx, Type_PCjr
      jmp    _EndPC
_PC_Conv: lea    dx, Type_PC_Conv
```



\_EndPC: call PRINT

sub ax, ax

mov ah, 30h

int 21h

call MOD\_PC

call OEM\_PC

call SER\_PC

lea dx, ModifNum

call PRINT

lea dx, OEM

call PRINT

lea dx, UserSerialNum

call PRINT

mov ax, 4C00h

int 21h

ret

MAIN ENDP

\_CODE ENDS

END MAIN

## ПРИЛОЖЕНИЕ Б

### ИСХОДНЫЙ КОД ПРОГРАММЫ. com.asm

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

ORG 100H

START: JMP BEGIN

ModifNum db 'Modification number: . ', 0dh, 0ah, '\$'

OEM db 'OEM: ', 0dh, 0ah, '\$'

UserSerialNum db 'User serial number: ', 0dh, 0ah, '\$'

Type\_PC\_Other db 2 dup ('?'), 'h\$'

Type\_PC db 'TypePC: PC', 0DH, 0AH, '\$'

Type\_PC\_XT db 'TypePC: PC/XT', 0DH, 0AH, '\$'

Type\_AT db 'TypePC: AT', 0DH, 0AH, '\$'

Type\_PS2\_30 db 'TypePC: PC2 model 30', 0DH, 0AH, '\$'

Type\_PS2\_50 db 'TypePC: PC2 model 50 or 60', 0DH, 0AH, '\$'

Type\_PS2\_80 db 'TypePC: PC2 model 80', 0DH, 0AH, '\$'

Type\_PCjr db 'TypePC: PCjr', 0DH, 0AH, '\$'

Type\_PC\_Conv db 'TypePC: PC Convertible', 0DH, 0AH, '\$'

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     al,ah
    call    TETR_TO_HEX
    xchg    al,ah
    mov     cl,4
    shr     al,cl
    call    TETR_TO_HEX
    pop     cx
    ret
```

BYTE\_TO\_HEX 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
    xor     ah,ah
    call    BYTE_TO_HEX
    mov     [di],ah
    dec     di
    mov     [di],al
    pop     bx
    ret
```

WRD\_TO\_HEX ENDP

BYTE\_TO\_DEC PROC NEAR

```
    push  cx
    push  dx
    push  ax
    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   ax,00h
    jbe   end_l
    or    al,30h
    mov   [si],al
end_l: pop   ax
    pop   dx
    pop   cx
    ret
```

BYTE\_TO\_DEC ENDP

PRINT PROC NEAR

```
    push  ax
    mov   ah,09h
    int   21h
    pop   ax
    ret
```

PRINT ENDP

MOD\_PC PROC NEAR

```
    push    ax
    push    si
    mov     si, offset ModifNum
    add     si, 22
    call    BYTE_TO_DEC
    add     si, 3
    mov     al, ah
    call    BYTE_TO_DEC
    pop     si
    pop     ax
    ret
```

MOD\_PC ENDP

OEM\_PC PROC NEAR

```
    push    ax
    push    bx
    push    si
    mov     al,bh
    lea     si, OEM
    add     si, 6
    call    BYTE_TO_DEC
    pop     si
    pop     bx
    pop     ax
    ret
```

OEM\_PC ENDP

SER\_PC PROC NEAR

```
    push    ax
    push    bx
    push    cx
    push    si
    mov     al,bl
    call    BYTE_TO_HEX
    lea     di,UserSerialNum
    add     di,20
    mov     [di],ax
    mov     ax,cx
    lea     di,UserSerialNum
    add     di,25
    call    WRD_TO_HEX
    pop     si
    pop     cx
    pop     bx
    pop     ax
    ret
```

SER\_PC ENDP

BEGIN: mov bx, 0F000h

```
    mov     es, bx
    mov     ax, es:[0FFFEh]
    cmp     al, 0FFh
    je      _PC
    cmp     al, 0FEh
    je      _PC_XT
    cmp     al, 0FBh
    je      _PC_XT
    cmp     al, 0FCh
    je      _AT
```

```
    cmp     al, 0FAh
    je      _PS2_30
    cmp     al, 0F8h
    je      _PS2_80
    cmp     al, 0FDh
    je      _PCjr
    cmp     al, 0F9h
    je      _PC_Conv
    call    BYTE_TO_HEX
    lea     di, Type_PC_Other
    mov     [di], ax
    lea     dx, Type_PC_Other
    jmp     _EndPC
```

```
_PC:  lea     dx, Type_PC
      jmp     _EndPC

_PC_XT: lea     dx, Type_PC_XT
      jmp     _EndPC

_AT:   lea     dx, Type_AT
      jmp     _EndPC

_PS2_30: lea     dx, Type_PS2_30
      jmp     _EndPC

_PS2_80: lea     dx, Type_PS2_80
      jmp     _EndPC

_PCjr: lea     dx, Type_PCjr
      jmp     _EndPC

_PC_Conv: lea     dx, Type_PC_Conv
```

```
_EndPC: call PRINT
```

```
    sub     ax, ax
    mov     ah, 30h
```

```
int 21h  
  
call MOD_PC  
  
call OEM_PC  
  
call SER_PC
```

```
lea dx, ModifNum  
  
call PRINT  
  
lea dx, OEM  
  
call PRINT  
  
lea dx, UserSerialNum  
  
call PRINT
```

```
mov ax, 4C00h  
  
int 21h
```

```
TESTPC ENDS
```

```
END START
```