

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

ОТЧЕТ
по лабораторной работе №1
по дисциплине «Операционные системы»
Тема: «Исследование структур загрузочных модулей»

Студент гр. 7381

Тарасенко Е.А.

Преподаватель

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.

Необходимые сведения для составления программы.

Тип IBM PC хранится в байте по адресу 0F000:0FFFE, в предпоследнем байте ROM BIOS. Соответствие кода и типа в таблице:

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:

MOV AH,30h

INT 21h

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.

Постановка задачи.

Требуется реализовать 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.

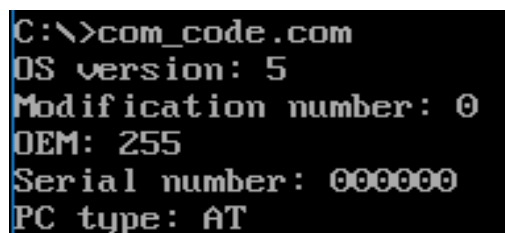
Далее необходимо отладить полученный исходный модуль и получить «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля.

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.

Ход работы.

Шаг 1. Запуск .COM и «плохого» .EXE модулей.

Результаты запуска модулей приведены на рис. 1 и 2.



```
C:\>com_code.com
OS version: 5
Modification number: 0
OEM: 255
Serial number: 000000
PC type: AT
```

Рис. 1 – Запуск .COM модуля

```
LINK : warning L4021: no stack segment
C:\>com_code.exe
5
        OS version: 0
        OS version: 255
version: 000000 255 OS
        OS version:
        OS version: _
```

Рис. 2 – Запуск «плохого» .EXE модуля

Шаг 2. Запуск «хорошего» .EXE модуля.

```
C:\>exe_code.exe
OS version: 5
Modification number: 0
OEM: 255
Serial number: 000000
PC type: AT
```

Рис. 3 – Запуск «хорошего» .EXE модуля

Шаг 3. Ответы на контрольные вопросы. Отличия исходных текстов .COM и .EXE программ.

- 1) Сколько сегментов должна содержать .COM программа?
Один сегмент (сегмент кода).
- 2) Сколько сегментов должна содержать .EXE программа?
3 сегмента (сегменты стека, данных и кода).
- 3) Какие директивы должны быть обязательно в тексте .COM программы?

ORG, которая указывает на то, сколько памяти нужно зарезервировать под PSP, и ASSUME, указывающая на то, что сегменты кода и данных начинаются с одного и того же места.

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

Не все, т. к. в отличие от EXE-программы, в COM-программе нет таблицы настроек (разметки). Адреса сегментов определяются загрузчиком в момент запуска программы на основе информации о местоположении полей адресов в файле из этой таблицы. Следовательно, в связи с отсутствием ее в COM-программах, команды вида mov [регистр], seg [сегмент] недопустимы.

Шаг 4. .COM и оба .EXE модуля в шестнадцатеричном виде.

D:\study\LabOS\LabOS\TASM\TASM\COM_CODE.COM															
00000000:	E9	BF	01	4F	53	20	76	65	72	73	69	6F	6E	3A	20
00000001:	24	0D	0A	4D	6F	64	69	66	69	63	61	74	69	6F	6E
00000002:	6E	75	6D	62	65	72	3A	20	20	24	0D	0A	53	65	72
00000003:	61	6C	20	6E	75	6D	62	65	72	3A	20	20	20	20	20
00000004:	20	24	0D	0A	4F	45	4D	3A	20	20	20	20	20	24	0D
00000005:	50	43	20	74	79	70	65	3A	20	24	50	43	24	50	43
00000006:	58	54	24	41	54	24	50	53	32	20	28	33	30	20	6D
00000007:	64	65	6C	29	24	50	53	32	20	28	35	30	20	6F	72
00000008:	36	30	20	6D	6F	64	65	6C	29	24	50	53	32	20	28
00000009:	30	20	6D	6F	64	65	6C	29	24	50	43	20	6A	72	24
0000000A:	43	20	43	6F	6E	76	65	72	74	69	62	6C	65	24	50
0000000B:	09	CD	21	58	C3	B4	30	CD	21	50	BE	03	01	83	C6
0000000C:	E8	DC	00	BA	03	01	E8	E5	FF	BE	11	01	83	C6	17
0000000D:	8A	C4	E8	CA	00	BA	11	01	E8	D3	FF	BE	42	01	83
0000000E:	09	8A	C7	E8	B9	00	BA	42	01	E8	C2	FF	BF	2A	01
0000000F:	C7	16	8B	C1	E8	90	00	8A	C3	E8	7A	00	83	EF	02
00000010:	05	BA	2A	01	E8	A7	FF	C3	B8	00	F0	8E	C0	26	A0
00000011:	FF	BA	4E	01	E8	97	FF	3C	FF	74	1C	3C	FE	74	1E
00000012:	FB	74	1A	3C	FC	74	1C	3C	FA	74	1E	3C	F8	74	26
00000013:	FD	74	28	3C	F9	74	2A	BA	5A	01	EB	2B	90	BA	5D
00000014:	EB	25	90	BA	63	01	EB	1F	90	BA	66	01	EB	19	90
00000015:	75	01	EB	13	90	BA	8A	01	EB	0D	90	BA	99	01	EB
00000016:	90	BA	9F	01	EB	01	90	E8	44	FF	C3	24	0F	3C	09
00000017:	02	04	07	04	30	C3	51	8A	E0	E8	EF	FF	86	C4	B1
00000018:	D2	E8	E8	E6	FF	59	C3	53	8A	FC	E8	E9	FF	88	25
00000019:	88	05	4F	8A	C7	E8	DE	FF	88	25	4F	88	05	5B	C3
0000001A:	52	32	E4	33	D2	B9	0A	00	F7	F1	80	CA	30	88	14
0000001B:	33	D2	3D	0A	00	73	F1	3C	00	74	04	0C	30	88	04
0000001C:	59	C3	E8	F0	FE	E8	40	FF	B4	10	CD	16	32	C0	B4
0000001D:	CD	21													

Рис. 4 – Представление .COM модуля в шестнадцатеричном виде


```
D:\study\LabOS\LabOS\MASM\MASM\EXE_CODE.EXE
00000000: 4D 5A DF 01 02 00 01 00 20 00 41 00 FF FF 1E 00 MZЯ00 0 A яя▲
000000010: 00 04 6A A3 14 01 0B 00 1E 00 00 00 01 00 19 01 ♦jJg0♂ ▲ 0 ↓0
000000020: 0B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ♂
000000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000200: 4F 53 20 76 65 72 73 69 6F 6E 3A 20 20 24 0D 0A OS version: $J
000000210: 4D 6F 64 69 66 69 63 61 74 69 6F 6E 20 6E 75 6D Modification num
000000220: 62 65 72 3A 20 20 24 0D 0A 53 65 72 69 61 6C 20 ber: $J
000000230: 6E 75 6D 62 65 72 3A 20 20 20 20 20 20 24 0D number: $J
000000240: 0A 4F 45 4D 3A 20 20 20 20 20 24 0D 0A 50 43 20 OEM: $J
000000250: 74 79 70 65 3A 20 24 50 43 24 50 43 2F 58 54 24 type: $PC$PC/XT$
000000260: 41 54 24 50 53 32 20 28 33 30 20 6D 6F 64 65 6C AT$PS2 (30 model
000000270: 29 24 50 53 32 20 28 35 30 20 6F 72 20 36 30 20 )$PS2 (50 or 60
000000280: 6D 6F 64 65 6C 29 24 50 53 32 20 28 38 30 20 6D model)$PS2 (80 m
000000290: 6F 64 65 6C 29 24 50 43 20 6A 72 24 50 43 20 43 odel)$PC jr$PC C
0000002A0: 6F 6E 76 65 72 74 69 62 6C 65 24 00 00 00 00 00 onvertible$
0000002B0: 50 B4 09 CD 21 58 C3 B4 30 CD 21 50 BE 00 00 83 ProH!XГr0H!Ps f
0000002C0: C6 0C E8 DC 00 BA 00 00 E8 E5 FF BE 0E 00 83 C6 ЖФиб є иеяѕ fЖ
0000002D0: 17 58 8A C4 E8 CA 00 BA 0E 00 E8 D3 FF BE 3F 00 фХьДиК єљ иУяѕ?
0000002E0: 83 C6 09 8A C7 E8 B9 00 BA 3F 00 E8 C2 FF BF 27 fЖољзи№ є? иВяi'
0000002F0: 00 83 C7 16 8B C1 E8 90 00 8A C3 E8 7A 00 83 EF fЗ=«Биђ лђиз fп
000000300: 02 89 05 BA 27 00 E8 A7 FF C3 B8 00 F0 8E C0 26 0%«є' иђяѓє рђА&
000000310: A0 FE FF BA 4B 00 E8 97 FF 3C FF 74 1C 3C FE 74 юяєК и-я<ятL<ют
000000320: 1E 3C FB 74 1A 3C FC 74 1C 3C FA 74 1E 3C F8 74 ▲<ыт-><ьтL<ьт▲<шт
000000330: 26 3C FD 74 28 3C F9 74 2A BA 57 00 EB 2B 90 BA &<эт(<шт*єW л+ђе
000000340: 5A 00 EB 25 90 BA 60 00 EB 1F 90 BA 63 00 EB 19 Z л%ђє` лђђєс л↓
000000350: 90 BA 72 00 EB 13 90 BA 87 00 EB 0D 90 BA 96 00 ђєр лђђєf лђђє-
000000360: EB 07 90 BA 9C 00 EB 01 90 E8 44 FF C3 24 0F 3C л•ђєњ л0ђиДяГ$о<
000000370: 09 76 02 04 07 04 00 C3 51 8A E0 E8 EF FF 86 C4 ов0♦♦0ГQљаипя†Д
000000380: B1 04 D2 E8 E8 E6 FF 59 DE C3 53 8A FC E8 E9 FF 88 ±♦ТиижяYГSљыййє
000000390: 25 4F 88 05 4F 8A C7 E8 CE FF 88 25 4F 88 05 5B %0є♦0љзиЮяє%0є♦[
0000003A0: C3 51 52 32 E4 33 D2 B9 0A 00 F7 F1 80 CA 30 88 ГQR2д3TN% чсЂК0є
0000003B0: 14 4E 33 D2 3D 0A 00 73 F1 3C 00 74 04 0C 30 88 ѓNЗT= sс< т♦♀0є
0000003C0: 04 5A 59 C3 1E 2B C0 50 B8 00 00 8E D8 2B C0 E8 ♦ZYГ▲+APё фш+Аи
0000003D0: E5 FE E8 35 FF B4 10 CD 16 32 C0 B4 4C CD 21 еюи5яг•H=2ArLH!
```

Рис. 6 – Представление «хорошего» .EXE модуля в шестнадцатеричном виде

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

.COM файл состоит из одного сегмента и содержит данные и машинные команды. Код начинается с адреса 0h, но при загрузке модуля устанавливается смещение в 100h.

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

В «плохом» .EXE файле данные и код содержатся в одном сегменте. Код располагается с адреса 300h. С адреса 0h располагается Relocation Table (таблица разметки).

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

В «хорошем» файле .EXE содержится информация для загрузчика, сегмент стека, сегмент данных и сегмент кода (3 сегмента вместо одного в «плохом» .EXE). Код располагается с адреса 200h в отличии от 300h в «плохом» .EXE файле.

Шаг 5. Загрузка .COM модуля в основную память.

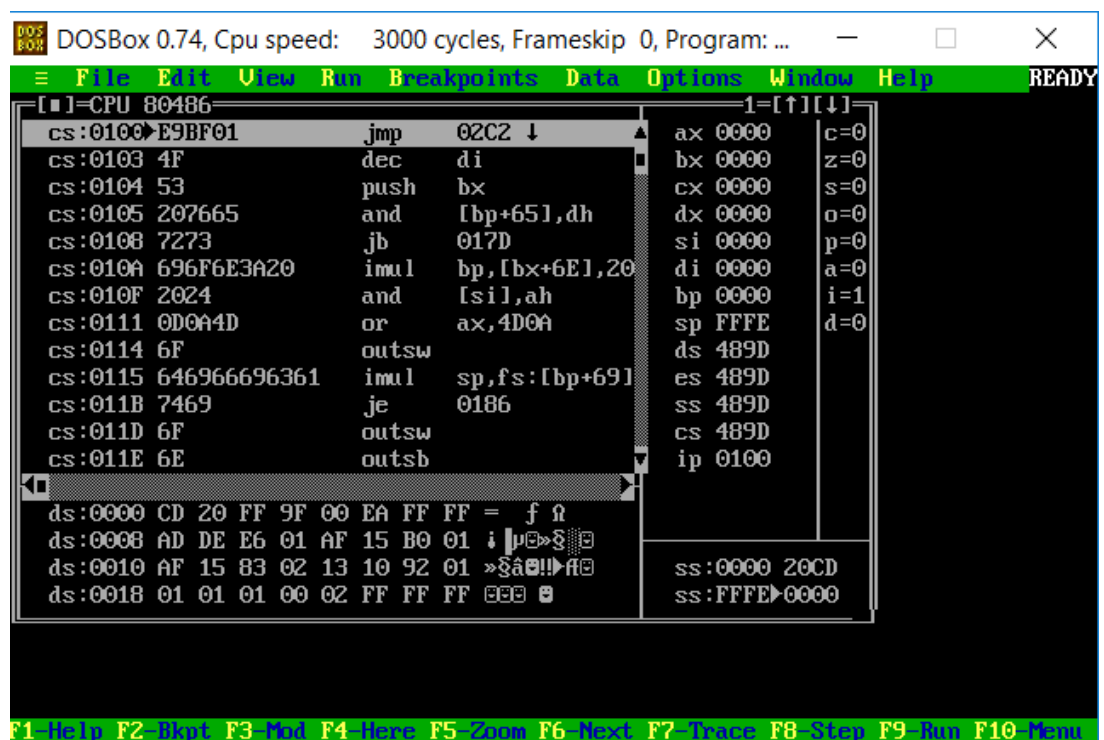


Рис. 7 – Загрузка .COM модуля в основную память

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

Сначала загружается PSP, потом данные и код, потом – стек. Код начинается с адреса 100h.

2) Что располагается с адреса 0?

PSP (Program Segment Prefix).

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

489D. Они указывают на начало PSP. См. рис. 7.

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

Стек определяется автоматически, указатель стека устанавливается на конец сегмента. Если для программы размер сегмента в 64 Кб является достаточным, то MS-DOS устанавливает в регистре SP адрес конца сегмента – FFFEh. Адреса расположены в диапазоне 0000h-FFFEh.

Шаг 6. Загрузка «хорошего» .EXE модуля в память.

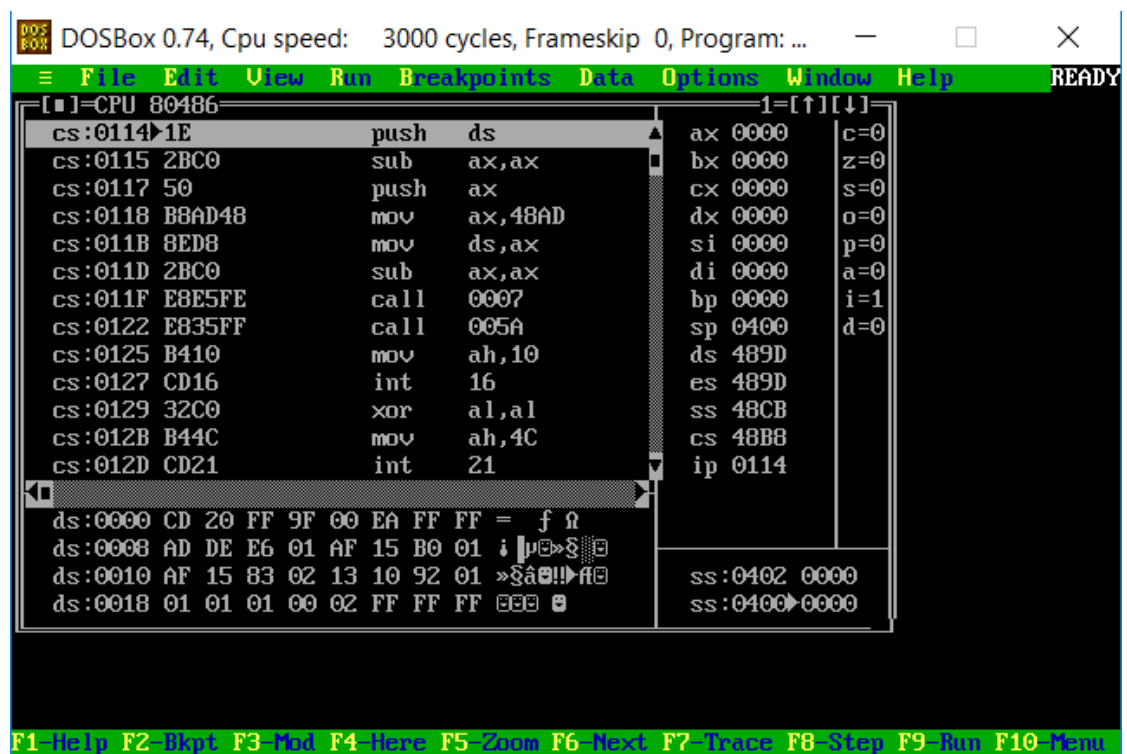


Рис. 8 – Загрузка «хорошего» .EXE модуля в основную память

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

В области памяти строится PSP, потом стандартная часть заголовка считывается в память; определяется длина тела загрузочного модуля и определяется начальный сегмент, позже загрузочный модуль считывается в начальный сегмент, а таблица настройки считывается в рабочую память; определяются значения сегментных регистров. DS и ES устанавливаются на начало PSP, SS - на начало стека, CS - на начало сегмента кода. Значения сегментных регистров – см. рис. 8.

2) На что указывают регистры DS и ES?

На начало PSP.

3) Как определяется стек?

В исходном коде модуля стек определяется при помощи директивы STACK, а при исполнении в регистры SS и SP записываются адрес начала сегмента стека и его вершины соответственно.

4) Как определяется точка входа?

При помощи команды “END”.

Вывод.

В ходе данной 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.

ПРИЛОЖЕНИЕ А.

Исходный текст .COM модуля

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

VERSION db "OS version:  $"
MODIFICATION db 13, 10, "Modification number:  $"
SERIAL db 13, 10, "Serial number:          $"
OEM db 13, 10, "OEM:          $"

TYPEPCSTRING db 13, 10, "PC type: $"
TYPEPC db "PC$"
TYPEPCXT db "PC/XT$"
TYPEAT db "AT$"
TYPEPS2M30 db "PS2 (30 model) $"
TYPEPS2M5060 db "PS2 (50 or 60 model) $"
TYPEPS2M80 db "PS2 (80 model) $"
TYPEPCJR db "PC jr $"
TYPEPCCONV db "PC Convertible $"

WRITE PROC near
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
WRITE ENDP

OSVER PROC near
    mov ah, 30h
    int 21h
    ; al - version number
    ; ah - modification number
    ; bh - OEM serial number
    ; bl:cx - user serial number
    push ax
```

```

mov si, offset VERSION
add si, 12
call BYTE_TO_DEC
mov dx, offset VERSION
call WRITE

mov si, offset MODIFICATION
add si, 23
pop ax
mov al, ah
call BYTE_TO_DEC
mov dx, offset MODIFICATION
call WRITE

mov si, offset OEM
add si, 9
mov al, bh
call BYTE_TO_DEC
mov dx, offset OEM
call WRITE

mov di, offset SERIAL
add di, 22
mov ax, cx
call WRD_TO_HEX
mov al, bl
call BYTE_TO_HEX
sub di, 2
mov [di], ax
mov dx, offset SERIAL
call WRITE
ret

```

OSVER ENDP

PCTYPE PROC near

```

mov ax, 0F000H
mov es, ax
mov al, es:[0FFEH]
mov dx, offset TYPEPCSTRING
call WRITE

```

```

cmp al, 0ffh
jz pc
cmp al, 0feh
jz pcxt
cmp al, 0fbh
jz pcxt
cmp al, 0fch
jz pcat
cmp al, 0fah
jz pcps2m30
cmp al, 0f8h
jz pcps2m80
cmp al, 0fdh
jz pcjr
cmp al, 0f9h
jz pcconv
pc:
    mov dx, offset TYPEPC
    jmp writestring
pcxt:
    mov dx, offset TYPEPCXT
    jmp writestring
pcat:
    mov dx, offset TYPEAT
    jmp writestring
pcps2m30:
    mov dx, offset TYPEPS2M30
    jmp writestring
pcps2m5060:
    mov dx, offset TYPEPS2M5060
    jmp writestring
pcps2m80:
    mov dx, offset TYPEPS2M80
    jmp writestring
pcjr:
    mov dx, offset TYPEPCJR
    jmp writestring
pcconv:
    mov dx, offset TYPEPCCONV
    jmp writestring
writestring:

```

```

        call WRITE
    ret
PCTYPE 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 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 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

BEGIN:

        call OSVER
        call PCTYPE
        mov ah, 10h
        int 16h

; Выход в DOS
        xor        AL,AL
        mov        AH,4Ch
        int        21h

TESTPC    ENDS
END        START

```


ПРИЛОЖЕНИЕ Б.

Исходный текст .EXE модуля

```
EOL EQU '$'
```

```
DATA SEGMENT
```

```
VERSION db "OS version:  $"
```

```
MODIFICATION db 13, 10, "Modification number:  $"
```

```
SERIAL db 13, 10, "Serial number:       $"
```

```
OEM db 13, 10, "OEM:       $"
```

```
TYPEPCSTRING db 13, 10, "PC type:  $"
```

```
TYPEPC db "PC$"
```

```
TYPEPCXT db "PC/XT$"
```

```
TYPEAT db "AT$"
```

```
TYPEPS2M30 db "PS2 (30 model)$"
```

```
TYPEPS2M5060 db "PS2 (50 or 60 model)$"
```

```
TYPEPS2M80 db "PS2 (80 model)$"
```

```
TYPEPCJR db "PC jr$"
```

```
TYPEPCCONV db "PC Convertible$"
```

```
DATA ENDS
```

```
CODE SEGMENT
```

```
    ASSUME CS:CODE, DS:DATA, SS:AStack
```

```
WRITE PROC near
```

```
    push ax
```

```
    mov ah, 09h
```

```
    int 21h
```

```
    pop ax
```

```
    ret
```

```
WRITE ENDP
```

```
AStack SEGMENT  STACK
```

```
    DW 512 DUP(?)
```

```
AStack ENDS
```

```
OSVER PROC near
```

```
    mov ah, 30h
```

```
    int 21h
```

```
    ; al - version number
```

```
    ; ah - modification number
```

```
; bh - OEM serial number
; bl:cx - user serial number
push ax
```

```
mov si, offset VERSION
add si, 12
call BYTE_TO_DEC
mov dx, offset VERSION
call WRITE
```

```
mov si, offset MODIFICATION
add si, 23
pop ax
mov al, ah
call BYTE_TO_DEC
mov dx, offset MODIFICATION
call WRITE
```

```
mov si, offset OEM
add si, 9
mov al, bh
call BYTE_TO_DEC
mov dx, offset OEM
call WRITE
```

```
mov di, offset SERIAL
add di, 22
mov ax, cx
call WRD_TO_HEX
mov al, bl
call BYTE_TO_HEX
sub di, 2
mov [di], ax
mov dx, offset SERIAL
call WRITE
ret
```

OSVER ENDP

```
PCTYPE PROC near
    mov ax, 0F000H
    mov es, ax
```

```

mov al, es:[0FFFEH]
mov dx, offset TYPEPCSTRING
call WRITE
cmp al, 0ffh
jz pc
cmp al, 0feh
jz pcxt
cmp al, 0fbh
jz pcxt
cmp al, 0fch
jz pcat
cmp al, 0fah
jz pcps2m30
cmp al, 0f8h
jz pcps2m80
cmp al, 0fdh
jz pcjr
cmp al, 0f9h
jz pcconv
pc:
    mov dx, offset TYPEPC
    jmp writestring
pcxt:
    mov dx, offset TYPEPCXT
    jmp writestring
pcat:
    mov dx, offset TYPEAT
    jmp writestring
pcps2m30:
    mov dx, offset TYPEPS2M30
    jmp writestring
pcps2m5060:
    mov dx, offset TYPEPS2M5060
    jmp writestring
pcps2m80:
    mov dx, offset TYPEPS2M80
    jmp writestring
pcjr:
    mov dx, offset TYPEPCJR
    jmp writestring
pcconv:

```

```

        mov dx, offset TYPEPCCONV
        jmp writestring
writestring:
        call WRITE
    ret
PCTYPE 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 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 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

```

```

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

        call    OSVER
        call    PCTYPE

```

```
                mov ah, 10h
                int 16h
; Выход в DOS
                xor     AL,AL
                mov     AH,4Ch
                int     21h
Main            ENDP
CODE            ENDS
                END Main
```