

**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. 6382

\_\_\_\_\_

Сергеев А.Д.

Преподаватель

\_\_\_\_\_

Губкин А.Ф.

Санкт-Петербург

2019

## **Цель работы.**

Исследование различий в структурах исходных текстов модулей типа .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 0F000:0FFFEh, в предпоследнем байте ROM BIOS. Соответствия кодов и типов представлены в таблице 1.

Таблица 1 — Соответствие кодов и типов.

Тип 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:

```
mov ah, 30h
```

```
int 21h
```

Выходными параметрами являются:

AL – номер основной версии. Если 0, то < 2.0

AH – номер модификации

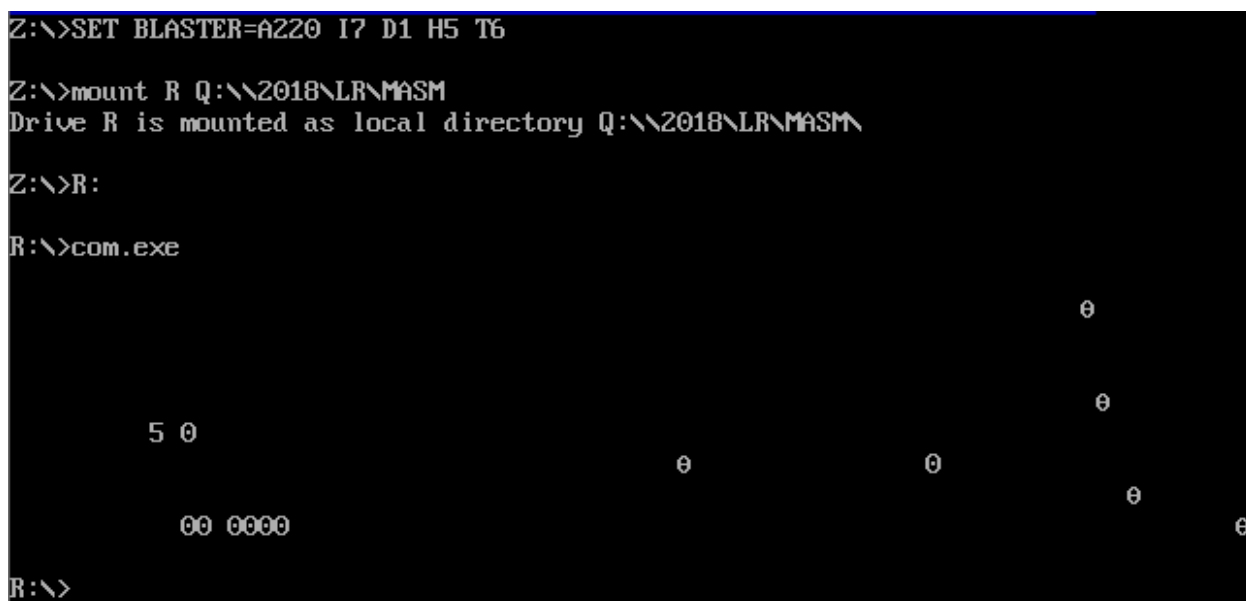
BH – серийный номер OEM (Original Equipment Manufacturer)

BL:CH – 24-битовый серийный номер пользователя

### Порядок выполнения работы.

Написание работы производилось на базе операционной системы Windows 10 в редакторе Atom. Отладка и тестирование проводились в эмуляторе DOSBOX.

Был написан исходный код .COM модуля, он был собран с помощью программ `masm` и `link`. В результате был получен «плохой» .EXE модуль, который был преобразован в «хороший» .COM модуль при помощи программы `exe2bin`. Результат выполнения «плохого» .EXE модуля представлен на рисунке 1.



```
Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount R Q:\\2018\LR\MASM
Drive R is mounted as local directory Q:\\2018\LR\MASM

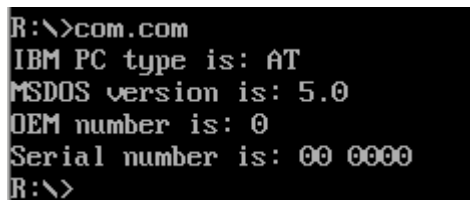
Z:\>R:

R:\>com.exe

                                     0
                                     0
5 0                                     0
                                     0
00 0000                               0
                                     0
R:\>
```

Рисунок 1 — вывод «плохого» .EXE модуля

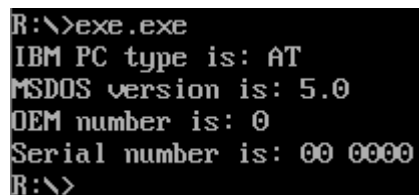
Результат выполнения «хорошего» .COM модуля представлен на рисунке 2.



```
R:\>com.com
IBM PC type is: AT
MSDOS version is: 5.0
OEM number is: 0
Serial number is: 00 0000
R:\>
```

Рисунок 2 — вывод «хорошего» .EXE модуля

Был написан исходный код .EXE модуля, в результате его отладки и сборки был получен «хороший» .EXE модуль. Результат его работы представлен на рисунке 3.



```
R:\>exe.exe
IBM PC type is: AT
MSDOS version is: 5.0
OEM number is: 0
Serial number is: 00 0000
R:\>
```

Рисунок 3 — результат работы «хорошего» .EXE модуля

## Ответы на контрольные вопросы.

### 1. Отличия исходных кодов .COM и .EXE программ

В отличие от .EXE программы, которая может содержать несколько сегментов (при использовании различных моделей памяти), .COM программа содержит только один сегмент (модель памяти tiny). Также в .EXE программе должен содержаться сегмент стека — в случае, если он не был создан программистом, операционная система создаст его автоматически.

В тексте .COM программы обязательно должны быть директивы `org`, которая используется для резервирования первых нескольких байт для префикса программного сегмента (PSP) и устанавливает адрес начала выполнения программы, и `assume`, которая переопределяет системные регистры (в данном случае необходимо, чтобы регистр кода CS и регистр данных DS указывали на единственный сегмент программы).

В .COM программе можно использовать не все форматы команд. Так как таблица настройки с информацией о местоположении адресов отсутству-

ет, нельзя использовать команды, связанные с адресом сегмента, он не известен до загрузки сегмента в память.

## 2. Отличия форматов файлов COM и EXE модулей

На рисунке 4 представлен вид .COM файла в шестнадцатеричном виде.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	
0000000000	e9	24	01	50	43	0d	0a	24	50	43	2f	58	54	0d	0a	24	\$.PC..\$PC/XT..\$
0000000010	41	54	0d	0a	24	50	53	32	20	6d	6f	64	65	6c	20	33	AT..\$PS2 model 3
0000000020	30	0d	0a	24	50	53	32	20	6d	6f	64	65	6c	20	35	30	0..\$PS2 model 50
0000000030	2f	36	30	0d	0a	24	50	53	32	20	6d	6f	64	65	6c	20	/60..\$PS2 model
0000000040	38	30	0d	0a	24	50	53	6a	72	0d	0a	24	50	43	20	63	80..\$PSjr..\$PC c
0000000050	6f	6e	76	65	72	74	69	62	6c	65	0d	0a	24	55	4e	4b	onvertible..\$UNK
0000000060	4e	4f	57	4e	3a	20	20	20	0d	0a	24	49	42	4d	20	50	NOWN: ..\$IBM P
0000000070	43	20	74	79	70	65	20	69	73	3a	20	24	4d	53	44	4f	C type is: \$MSDO
0000000080	53	20	76	65	72	73	69	6f	6e	20	69	73	3a	20	20	2e	S version is: .
0000000090	20	0d	0a	24	4f	45	4d	20	6e	75	6d	62	65	72	20	69	..\$OEM number i
00000000a0	73	3a	20	20	20	20	20	0d	0a	24	53	65	72	69	61	6c	s: ..\$Serial
00000000b0	20	6e	75	6d	62	65	72	20	69	73	3a	20	20	20	20	20	number is:
00000000c0	20	0d	0a	24	50	b4	09	cd	21	58	c3	24	0f	3c	09	76	..\$P...!X.\$.<.v
00000000d0	02	04	07	04	30	c3	51	8a	c4	e8	ef	ff	86	c4	b1	04	....0.Q.....
00000000e0	d2	e8	e8	e6	ff	59	c3	53	8a	fc	e8	e9	ff	88	25	4f	.....Y.S.....%0
00000000f0	88	05	4f	8a	c7	32	e4	e8	dc	ff	88	25	4f	88	05	5b	..0..2.....%0..[
0000000100	c3	51	52	50	32	e4	33	d2	b9	0a	00	f7	f1	80	ca	30	.QRP2.3.....0
0000000110	88	14	4e	33	d2	3d	0a	00	73	f1	3d	00	00	76	04	0c	..N3.=..s.=..v..
0000000120	30	88	04	58	5a	59	c3	52	50	ba	6b	01	e8	95	ff	b8	0..XZY.RP.k.....
0000000130	00	f0	8e	c0	26	a0	fe	ff	3c	ff	74	37	3c	fe	74	39	....&...<.t7<.t9
0000000140	3c	fb	74	35	3c	fc	74	37	3c	fa	74	39	3c	fc	74	3b	<.t5<.t7<.t9<.t;
0000000150	3c	f8	74	3d	3c	fd	74	3f	3c	f9	74	41	8a	e0	8d	36	<.t=<.t?<.tA...6
0000000160	5d	01	83	c6	09	e8	6e	ff	88	04	88	64	01	ba	5d	01	].....n....d..].
0000000170	eb	31	90	ba	03	01	eb	2b	90	ba	08	01	eb	25	90	ba	.1.....+.....%..

Рисунок 4 — .COM файл в шестнадцатеричном виде

Можно заметить, что текст программы начинается с нулевого адреса, секция данных заканчивается на 0b0, на 0c0 начинается код. Между ними есть 6 пустых байт.

На рисунке 5 представлен вид «плохого» .EXE файла в шестнадцатеричном виде (с адреса начала кода).

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	
00000000230	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000000240	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000000250	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000000260	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000000270	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000000280	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000000290	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000002a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000002b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000002c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000002d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000002e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000002f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000000300	e9	24	01	50	43	0d	0a	24	50	43	2f	58	54	0d	0a	24	.\$..PC..\$PC/XT..\$
00000000310	41	54	0d	0a	24	50	53	32	20	6d	6f	64	65	6c	20	33	AT..\$PS2 model 3
00000000320	30	0d	0a	24	50	53	32	20	6d	6f	64	65	6c	20	35	30	0..\$PS2 model 50
00000000330	2f	36	30	0d	0a	24	50	53	32	20	6d	6f	64	65	6c	20	/60..\$PS2 model
00000000340	38	30	0d	0a	24	50	53	6a	72	0d	0a	24	50	43	20	63	80..\$PSjr..\$PC c
00000000350	6f	6e	76	65	72	74	69	62	6c	65	0d	0a	24	55	4e	4b	onvertible..\$UNK
00000000360	4e	4f	57	4e	3a	20	20	20	0d	0a	24	49	42	4d	20	50	NOWN: ..\$IBM P
00000000370	43	20	74	79	70	65	20	69	73	3a	20	24	4d	53	44	4f	C type is: \$MSDO
00000000380	53	20	76	65	72	73	69	6f	6e	20	69	73	3a	20	20	2e	S version is: .
00000000390	20	0d	0a	24	4f	45	4d	20	6e	75	6d	62	65	72	20	69	..\$OEM number i
000000003a0	73	3a	20	20	20	20	20	0d	0a	24	53	65	72	69	61	6c	s: ..\$Serial

Рисунок 5 — «плохой» .EXE файл в шестнадцатеричном виде

С нулевого адреса в файле находится информация для загрузчика, сегменты должны начинаться с адреса 200h, но в данном случае начинаются с 300h, так как дополнительные 100h байтов зарезервировано командой org 100h.

На рисунке 6 представлен вид «хорошего» .EXE файла в шестнадцатеричном виде (с адреса начала кода).

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	
0000000340	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0000000350	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0000000360	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0000000370	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0000000380	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0000000390	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000003a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000003b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000003c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000003d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000003e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000003f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0000000400	50	43	0d	0a	24	50	43	2f	58	54	0d	0a	24	41	54	0d	PC..\$PC/XT..\$AT.
0000000410	0a	24	50	53	32	20	6d	6f	64	65	6c	20	33	30	0d	0a	..\$PS2 model 30..
0000000420	24	50	53	32	20	6d	6f	64	65	6c	20	35	30	2f	36	30	\$PS2 model 50/60
0000000430	0d	0a	24	50	53	32	20	6d	6f	64	65	6c	20	38	30	0d	..\$PS2 model 80.
0000000440	0a	24	50	53	6a	72	0d	0a	24	50	43	20	63	6f	6e	76	..\$PSjr..\$PC conv
0000000450	65	72	74	69	62	6c	65	0d	0a	24	55	4e	4b	4e	4f	57	ertible..\$UNKNOW
0000000460	4e	3a	20	20	20	0d	0a	24	49	42	4d	20	50	43	20	74	N: ..\$IBM PC t
0000000470	79	70	65	20	69	73	3a	20	24	4d	53	44	4f	53	20	76	ype is: \$MSDOS v
0000000480	65	72	73	69	6f	6e	20	69	73	3a	20	20	2e	20	0d	0a	ersion is: . . .
0000000490	24	4f	45	4d	20	6e	75	6d	62	65	72	20	69	73	3a	20	\$OEM number is:
00000004a0	20	20	20	20	0d	0a	24	53	65	72	69	61	6c	20	6e	75	..\$Serial nu
00000004b0	6d	62	65	72	20	69	73	3a	20	20	20	20	20	20	0d	0a	mber is: ..

Рисунок 6 — «хороший» .EXE файл в шестнадцатеричном виде

В отличие от «плохого» .EXE файла, «хороший» содержит в себе сегмент стека, который также начинается с адреса 200h и занимает 100h двойных байтов. Таким образом сегмент данных и кода начинаются по адресу 400h.

### 3. Загрузка COM модуля в основную память

На рисунке 7 представлен вид запущенного .COM модуля в отладчике afd.

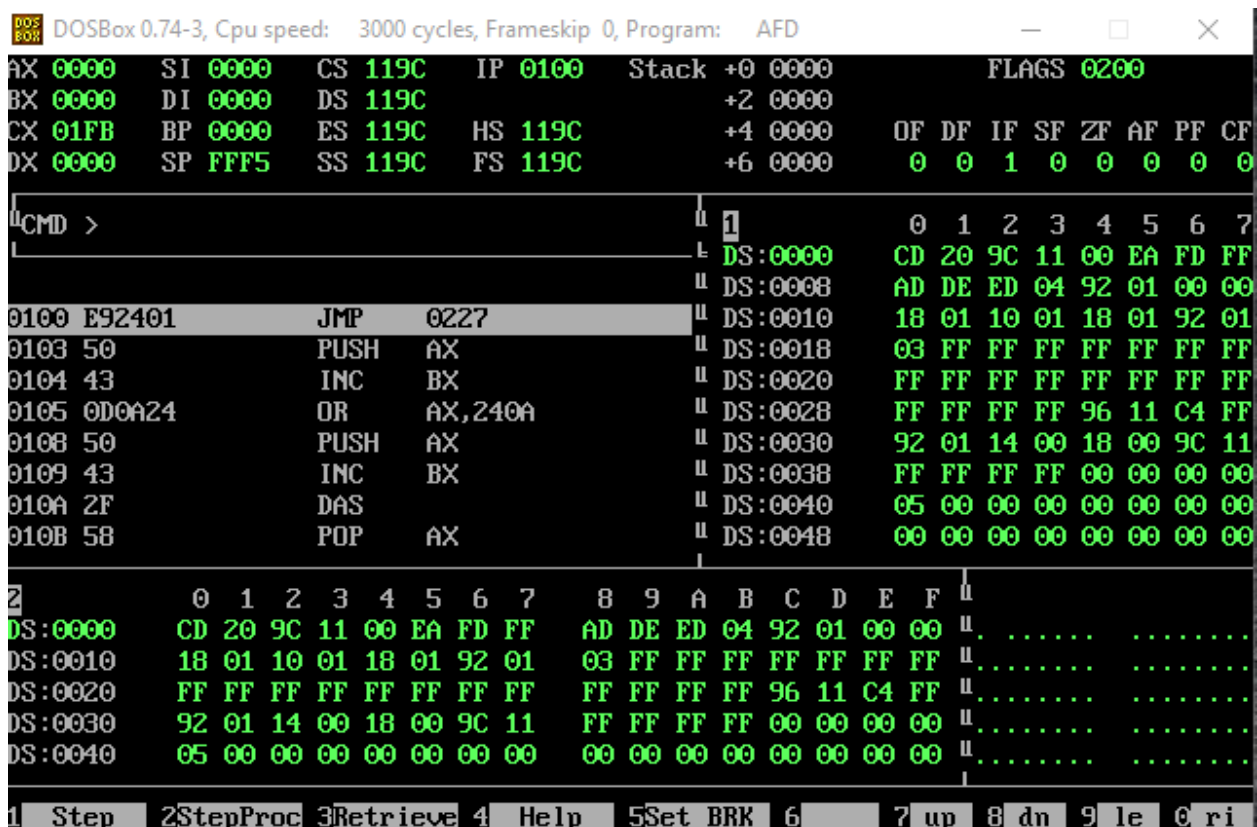


Рисунок 6 — вид запущенного COM модуля в отладчике afd

Все регистры указывают на нулевой адрес, он же - начало PSP, а код и данные располагаются по адресу 100h. Первая команда осуществляет переход к коду.

Стек в .COM модуле создается автоматически и указывает на FFF5 (регистр SP). Адреса в нем нумеруются от больших к меньшим, то есть стек занимает всю память, оставшуюся от 64кб (максимальный размер COM модуля в памяти) после загрузки данных и кода.

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

На рисунке 8 представлен вид запущенного «хорошего» .EXE модуля в отладчике afd.



Рисунок 7 — вид запущенного «хорошего» .EXE модуля в отладчике afd

После запуска регистры DS и ES указывают на PSP, а CS и SS – на начало сегмента кода и стека соответственно. Необходимость в начале работы программы загрузки в DS сегмента данных — одна из отличительных особенностей кода «хорошего» .EXE модуля.

В данном случае стек был объявлен явно, создан, и в начале работы программы его адрес хранится в регистре SS. Если бы он объявлен не был, он был бы создан автоматически так же, как при загрузке .COM модуля.

Точка входа в программу определяется оператором END, который находится в конце кода. Операндом может служить функция или метка, адрес которой в во время загрузки программы в память помещается в регистр IP.

### Вывод.

В ходе выполнения лабораторной работы были изучены структурные различия COM и EXE модулей, а также различия их загрузки в память.

## Приложение А

### Исходный код программы com.asm

TESTPC SEGMENT

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

org 100H

START:     jmp    BEGIN

PC\_TYPE               db     'PC', 0DH, 0AH, '\$'

PC\_XT\_TYPE            db     'PC/XT', 0dh, 0ah, '\$'

AT\_TYPE               db     'AT', 0dh, 0ah, '\$'

PS2\_30\_TYPE           db     'PS2 model 30', 0dh, 0ah, '\$'

PS2\_5060\_TYPE        db     'PS2 model 50/60', 0dh, 0ah, '\$'

PS2\_80\_TYPE           db     'PS2 model 80', 0dh, 0ah, '\$'

PCJR\_TYPE            db     'PSjr', 0dh, 0ah, '\$'

PC\_CONVERTIBLE db     'PC convertible', 0dh, 0ah, '\$'

PC\_UNKNOWN        db 'UNKNOWN: ', 0dh, 0ah, '\$'

IBM\_PC\_NAME           db     'IBM PC type is: ', '\$'

OS\_NAME               db     'MSDOS version is: . ', 0dh, 0ah, '\$'

OEM\_NAME             db     'OEM number is: ', 0dh, 0ah, '\$'

SERIAL\_NAME           db     'Serial number is: ', 0dh, 0ah, '\$'

PRINT\_STRING        PROC near

push   ax

mov    ah, 09h

int                21h

pop    ax

ret

PRINT\_STRING        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
        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_1
    or      al, 30h
    mov     [si], al
end_1: pop          ax
    pop     dx
    pop     cx
    ret
BYTE_TO_DEC          ENDP

BEGIN:
    push    dx
    push    ax

    mov     dx, offset IBM_PC_NAME
    call    PRINT_STRING

    mov     ax, 0F000H
    mov     es, ax

```

```

mov     al, es:[0FFFEH]
cmp     al, 0FFh
je      PC_WRITE
cmp     al, 0FEh
je      PC_XT_WRITE
cmp     al, 0FBh
je      PC_XT_WRITE
cmp     al, 0FCh
je      AT_WRITE
cmp     al, 0FAh
je      PS2_30_WRITE
cmp     al, 0FCh
je      PS2_5060_WRITE
cmp     al, 0F8h
je      PS2_80_WRITE
cmp     al, 0FDh
je      PCJR_WRITE
cmp     al, 0F9H
je      PC_CONVERTIBLE_WRITE

mov     ah, al
lea     si, PC_UNKNOWN
add     si, 9
call    BYTE_TO_HEX
mov     [si], al
mov     [si+1], ah
mov     dx, offset PC_UNKNOWN
jmp     TYPE_WRITE

```

PC\_WRITE:

```

mov     dx, offset PC_TYPE
jmp     TYPE_WRITE

```

PC\_XT\_WRITE:

```

mov     dx, offset PC_XT_TYPE

```

```
        jmp     TYPE_WRITE
```

AT\_WRITE:

```
        mov     dx, offset AT_TYPE
        jmp     TYPE_WRITE
```

PS2\_30\_WRITE:

```
        mov     dx, offset PS2_30_TYPE
        jmp     TYPE_WRITE
```

PS2\_5060\_WRITE:

```
        mov     dx, offset PS2_5060_TYPE
        jmp     TYPE_WRITE
```

PS2\_80\_WRITE:

```
        mov     dx, offset PS2_80_TYPE
        jmp     TYPE_WRITE
```

PCJR\_WRITE:

```
        mov     dx, offset PCJR_TYPE
        jmp     TYPE_WRITE
```

PC\_CONVERTIBLE\_WRITE:

```
        mov     dx, offset PC_CONVERTIBLE
        jmp     TYPE_WRITE
```

TYPE\_WRITE:

```
        call    PRINT_STRING
```

OS\_INFO\_GET:

```
        mov     ah, 30h
        int     21h
```

OS\_VERSION\_SET:

```
    lea        si, OS_NAME
    add        si, 18
    call       BYTE_TO_DEC
    add        si, 3
    mov        al, ah
    call       BYTE_TO_DEC
```

OS\_VERSION\_WRITE:

```
    mov        dx, offset OS_NAME
    call       PRINT_STRING
```

OEM\_SET:

```
    mov        al, BH
    lea        si, OEM_NAME
    add        si, 15
    call       BYTE_TO_DEC
```

OEM\_WRITE:

```
    mov        dx, offset OEM_NAME
    call       PRINT_STRING
```

SERIAL\_SET:

```
    mov        al, bl
    lea        si, SERIAL_NAME
    add        si, 18
    call       BYTE_TO_HEX
    mov        [si], ax
    add        si, 6
    mov        di, si
    mov        ax, cx
    call       WRD_TO_HEX
```

SERIAL\_WRITE:

```
    mov        dx, offset SERIAL_NAME
```

```
call PRINT_STRING
```

ENDING:

```
pop ax
```

```
pop dx
```

```
xor al, al
```

```
mov ah, 4ch
```

```
int 21h
```

```
ret
```

TESTPC ENDS

```
END START
```



## Приложение Б

### Исходный код программы exe.asm

```
AStack    SEGMENT STACK
           DW 100h DUP(?)
AStack    ENDS
```

#### DATA SEGMENT

```
PC_TYPE          db    'PC', 0DH, 0AH, '$'
PC_XT_TYPE       db    'PC/XT', 0dh, 0ah, '$'
AT_TYPE          db    'AT', 0dh, 0ah, '$'
PS2_30_TYPE      db    'PS2 model 30', 0dh, 0ah, '$'
PS2_5060_TYPE    db    'PS2 model 50/60', 0dh, 0ah, '$'
PS2_80_TYPE      db    'PS2 model 80', 0dh, 0ah, '$'
PCJR_TYPE        db    'PSjr', 0dh, 0ah, '$'
PC_CONVERTIBLE   db    'PC convertible', 0dh, 0ah, '$'
PC_UNKNOWN       db    'UNKNOWN: ', 0dh, 0ah, '$'

IBM_PC_NAME      db    'IBM PC type is: ', '$'
OS_NAME          db    'MSDOS version is: . ', 0dh, 0ah, '$'
OEM_NAME         db    'OEM number is:   ', 0dh, 0ah, '$'
SERIAL_NAME      db    'Serial number is:   ', 0dh, 0ah, '$'
```

#### DATA ENDS

#### CODE SEGMENT

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

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

PRINT\_STRING      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

          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_1
        or      al, 30h
        mov     [si], al
end_1: pop        ax
        pop     dx
        pop     cx
        ret
BYTE_TO_DEC        ENDP

Main  PROC far
        push    dx
        push    ax

```

```

mov     ax, DATA ; diff!
mov     ds, ax ; diff!

mov     dx, offset IBM_PC_NAME
call    PRINT_STRING

mov     ax, 0F000H
mov     es, ax
mov     al, es:[0FFFEH]
cmp     al, 0FFh
je      PC_WRITE
cmp     al, 0FEh
je      PC_XT_WRITE
cmp     al, 0FBh
je      PC_XT_WRITE
cmp     al, 0FCh
je      AT_WRITE
cmp     al, 0FAh
je      PS2_30_WRITE
cmp     al, 0FCh
je      PS2_5060_WRITE
cmp     al, 0F8h
je      PS2_80_WRITE
cmp     al, 0FDh
je      PCJR_WRITE
cmp     al, 0F9H
je      PC_CONVERTIBLE_WRITE

mov     ah, al
lea     si, PC_UNKNOWN
add     si, 9
call    BYTE_TO_HEX
mov     [si], al
mov     [si+1], ah

```

```
    mov    dx, offset PC_UNKNOWN
    jmp    TYPE_WRITE
```

PC\_WRITE:

```
    mov    dx, offset PC_TYPE
    jmp    TYPE_WRITE
```

PC\_XT\_WRITE:

```
    mov    dx, offset PC_XT_TYPE
    jmp    TYPE_WRITE
```

AT\_WRITE:

```
    mov    dx, offset AT_TYPE
    jmp    TYPE_WRITE
```

PS2\_30\_WRITE:

```
    mov    dx, offset PS2_30_TYPE
    jmp    TYPE_WRITE
```

PS2\_5060\_WRITE:

```
    mov    dx, offset PS2_5060_TYPE
    jmp    TYPE_WRITE
```

PS2\_80\_WRITE:

```
    mov    dx, offset PS2_80_TYPE
    jmp    TYPE_WRITE
```

PCJR\_WRITE:

```
    mov    dx, offset PCJR_TYPE
    jmp    TYPE_WRITE
```

PC\_CONVERTIBLE\_WRITE:

```
    mov    dx, offset PC_CONVERTIBLE
```

```
        jmp     TYPE_WRITE
```

TYPE\_WRITE:

```
        call    PRINT_STRING
```

OS\_INFO\_GET:

```
        mov     ah, 30h
```

```
        int     21h
```

OS\_VERSION\_SET:

```
        lea     si, OS_NAME
```

```
        add     si, 18
```

```
        call    BYTE_TO_DEC
```

```
        add     si, 3
```

```
        mov     al, ah
```

```
        call    BYTE_TO_DEC
```

OS\_VERSION\_WRITE:

```
        mov     dx, offset OS_NAME
```

```
        call    PRINT_STRING
```

OEM\_SET:

```
        mov     al, BH
```

```
        lea     si, OEM_NAME
```

```
        add     si, 15
```

```
        call    BYTE_TO_DEC
```

OEM\_WRITE:

```
        mov     dx, offset OEM_NAME
```

```
        call    PRINT_STRING
```

SERIAL\_SET:

```
    mov     al, bl
    lea     si, SERIAL_NAME
    add     si, 18
    call    BYTE_TO_HEX
    mov     [si], ax
    add     si, 6
    mov     di, si
    mov     ax, cx
    call    WRD_TO_HEX
```

SERIAL\_WRITE:

```
    mov     dx, offset SERIAL_NAME
    call    PRINT_STRING
```

ENDING:

```
    pop     ax
    pop     dx

    xor     al, al
    mov     ah, 4ch
    int     21h
    ret
```

Main ENDP

CODE ENDS

END Main