

**МИНОБРНАУКИ РОССИИ**  
**Санкт-Петербургский государственный**  
**электротехнический университет**  
**«ЛЭТИ» им. В.И. Ульянова (Ленина)**  
**Кафедра математического обеспечения и применения ЭВМ**

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

Студент гр. 8382

Черницын П.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

### Цель работы.

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

### Ход работы.

1. Была реализована программа для определения типа РС..
2. После трансляции с использованием MASM и линковки была получена программа с расширением .EXE. Результат работы представлен на рис. 1.



Рис.1. Запуск "плохого" .EXE файла.

3. После был получен "хороший" .COM модуль, работающий корректно. Результат работы программы представлен на рис.2.

```
C:\>LR1COM.COM
Your IBM PC type is PS2 ver. 50/60 or AT
Your MSDOS type is 05.00
Your serial OEM number is FF
Your serial user number is 000000
```

Рис.2. Запуск "хорошего" .COM файла.

4. Был написан исходный код для .EXE модуля, который должен выполнять те же действия, что и модуль на шаге 1.
5. После трансляции с использованием MASM и линковки была получена программа с расширением .EXE, работающая корректно. Результат работы программы представлен на рис.3.

```

C:\>LR1EXE.EXE
Your IBM PC type is PS2 ver. 50/60 or AT
Your MSDOS type is 05.00
Your serial OEM number is FF
Your serial user number is 000000

```

Рис.3. Запуск "хорошего" .EXE файла.

6. Представление файлов в шестнадцатиричном виде представлено на рисунках 4-6.

0000000000: E9 42 01 50 43 0A 0D 24	50 43 2F 58 54 0A 0D 24	éB@PC\$PC/XT\$
0000000010: 50 53 32 20 76 65 72 2E	20 33 30 0A 0D 24 50 53	PS2 ver. 30\$PS
0000000020: 32 20 76 65 72 2E 20 35	30 2F 36 30 20 6F 72 20	2 ver. 50/60 or
0000000030: 41 54 0A 0D 24 50 53 32	20 76 65 72 2E 20 38 30	AT\$PS2 ver. 80
0000000040: 0A 0D 24 50 43 6A 72 0A	0D 24 50 43 20 43 6F 6E	\$PCjr\$PC Con
0000000050: 76 65 72 74 69 62 6C 65	0A 0D 24 59 6F 75 72 20	vertible\$Your
0000000060: 49 42 4D 20 50 43 20 74	79 70 65 20 69 73 20 24	IBM PC type is \$
0000000070: 59 6F 75 72 20 4D 53 44	4F 53 20 74 79 70 65 20	Your MSDOS type
0000000080: 69 73 20 24 3C 32 2E 30	0A 0D 24 30 78 2E 30 79	is \$<2.0\$0x.0y
0000000090: 0A 0D 24 59 6F 75 72 20	73 65 72 69 61 6C 20 4F	\$Your serial O
00000000A0: 45 4D 20 6E 75 6D 62 65	72 20 69 73 20 24 0A 0D	EM number is \$
00000000B0: 59 6F 75 72 20 73 65 72	69 61 6C 20 75 73 65 72	Your serial user
00000000C0: 20 6E 75 6D 62 65 72 20	69 73 20 24 45 52 52 4F	number is \$ERRO
00000000D0: 52 21 0A 0D 24 B4 09 CD	21 C3 24 0F 3C 09 76 02	R!\$'oí!Â\$<ov@
00000000E0: 04 07 04 30 C3 51 8A E0	E8 EF FF 86 C4 B1 04 D2	♦♦0AQ\$àèiy†Â±♦0
00000000F0: E8 E8 E6 FF 59 C3 53 8A	FC E8 E9 FF 88 25 4F 88	èèæyYÂS\$uèéy~%0^
0000000100: 05 4F 8A C7 E8 DE FF 88	25 4F 88 05 5B C3 51 52	†0\$çèpy~%0^+ [ÂQR
0000000110: 32 E4 33 D2 B9 0A 00 F7	F1 80 CA 30 88 14 4E 33	2ä30³ ÷ñ€Ê0~ŋN3
0000000120: D2 3D 0A 00 73 F1 3C 00	74 04 0C 30 88 04 5A 59	0= sñ< t♦90~♦ZY
0000000130: C3 50 52 E8 AF FF 8A D0	8A C4 B4 02 CD 21 8A D0	ÂPrè~y\$D\$A'0í!\$D
0000000140: CD 21 5A 58 C3 B8 00 F0	8E C0 26 A0 FE FF BA 5B	í!ZXÃ, ôŽÂ& py°[
0000000150: 01 E8 81 FF BA 03 01 3C	FF 74 31 BA 08 01 3C FE	0èËy°♥0<y†1°□0<þ
0000000160: 74 2A 3C FB 74 26 BA 1E	01 3C FC 74 1F BA 10 01	t* <út&°▲0<út▼°►0
0000000170: 3C FA 74 18 BA 35 01 3C	F8 74 11 BA 43 01 3C FD	<út†°50<ø†◄°C0<y
0000000180: 74 0A BA 4A 01 3C F9 74	03 BA CC 01 E8 46 FF BA	t°J0<út♥°İ0èFy°
0000000190: 70 01 E8 40 FF B4 30 CD	21 3C 00 75 06 BA 84 01	p0è@y'0í!< u♦°,0
00000001A0: E8 32 FF BE 8B 01 83 C6	01 E8 62 FF 83 C6 04 8A	è2y%◄ofÆ0èbyfÆ♦Š
00000001B0: C4 E8 5A FF BA 8B 01 E8	1B FF B4 30 CD 21 BA 93	ÂèZy°◄0è+y'0í!°“
00000001C0: 01 E8 11 FF 8A C7 E8 68	FF B4 30 CD 21 BA AE 01	0è~y\$çèhy'0í!°00
00000001D0: E8 02 FF 8A C3 E8 59 FF	8A C5 E8 54 FF 8A C1 E8	è0y\$Âèyy\$Âèty\$Âè
00000001E0: 4F FF 32 C0 B4 4C CD 21		0y2Â`Lí!

Рис.4. COM-файл в шестнадцатеричном виде.

0000000000:	4D 5A E8 00 03 00 00 00	20 00 00 00 FF FF 00 00	MZè ♥ yy
0000000010:	00 00 00 00 00 01 00 00	3E 00 00 00 01 00 FB 50	è > è ùP
0000000020:	6A 72 00 00 00 00 00 00	00 00 00 00 00 00 00 00	jr
0000000030:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000040:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000050:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000060:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000070:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000080:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000090:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000B0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000100:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000110:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000120:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000130:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000140:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000150:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000160:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000170:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000180:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000190:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000001A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000001B0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000001C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000001D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000001E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000001F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000200:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000210:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000220:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000230:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000240:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000250:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000260:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000270:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000280:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000290:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000002A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000002B0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000002C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000002D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000002E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000002F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000300:	E9 42 01 50 43 0A 0D 24	50 43 2F 58 54 0A 0D 24	éB0PC\$PC/XT\$
0000000310:	50 53 32 20 76 65 72 2E	20 33 30 0A 0D 24 50 53	PS2 ver. 30\$PS
0000000320:	32 20 76 65 72 2E 20 35	30 2F 36 30 20 6F 72 20	2 ver. 50/60 or
0000000330:	41 54 0A 0D 24 50 53 32	20 76 65 72 2E 20 38 30	AT\$PS2 ver. 80
0000000340:	0A 0D 24 50 43 6A 72 0A	0D 24 50 43 20 43 6F 6E	\$PCjr\$PC Con
0000000350:	76 65 72 74 69 62 6C 65	0A 0D 24 59 6F 75 72 20	vertible\$Your
0000000360:	49 42 4D 20 50 43 20 74	79 70 65 20 69 73 20 24	IBM PC type is \$
0000000370:	59 6F 75 72 20 4D 53 44	4F 53 20 74 79 70 65 20	Your MSDOS type
0000000380:	69 73 20 24 3C 32 2E 30	0A 0D 24 30 78 2E 30 79	is \$<2.0\$0x.0y
0000000390:	0A 0D 24 59 6F 75 72 20	73 65 72 69 61 6C 20 4F	\$Your serial 0
00000003A0:	45 4D 20 6E 75 6D 62 65	72 20 69 73 20 24 0A 0D	EM number is \$
00000003B0:	59 6F 75 72 20 73 65 72	69 61 6C 20 75 73 65 72	Your serial user
00000003C0:	20 6E 75 6D 62 65 72 20	69 73 20 24 45 52 52 4F	number is \$ERRO
00000003D0:	52 21 0A 0D 24 B4 09 CD	21 C3 24 0F 3C 09 76 02	R!\$`oí!Á\$<ov0
00000003E0:	04 07 04 30 C3 51 8A E0	E8 EF FF 86 C4 B1 04 D2	♦♦0AQ5àèÿ+Ä±♦0
00000003F0:	E8 E8 E6 FF 59 C3 53 8A	FC E8 E9 FF 88 25 4F 88	èèÿYÄS5ueéy`%0`
0000000400:	05 4F 8A C7 E8 DE FF 88	25 4F 88 05 5B C3 51 52	±0Sçäpy`%0`+ÄQR
0000000410:	32 E4 33 D2 B9 0A 00 F7	F1 80 CA 30 88 14 4E 33	2ä30± ±ñèè0`JN3
0000000420:	D2 3D 0A 00 73 F1 3C 00	74 04 0C 30 88 04 5A 59	0= sñ< t±90`±ZY
0000000430:	C3 50 52 E8 AF FF 8A D0	8A C4 B4 02 CD 21 8A D0	ÄPRè`ÿD5Ä`0í!5D
0000000440:	CD 21 5A 58 C3 B8 00 F0	8E C0 26 A0 FE FF BA 5B	í!ZXÄ, 0ZÄ& pye[
0000000450:	01 E8 81 FF BA 01 3C 00	FF 74 31 BA 08 01 3C FE	0èÿe`♥<ÿt10Q<b
0000000460:	74 2A 3C FB 74 26 BA 1E	01 3C FC 74 1F BA 10 01	t*<út&0<út♥>0
0000000470:	3C FA 74 18 BA 35 01 3C	F8 74 11 BA 43 01 3C FD	<út!±50<0t±0C0<ÿ
0000000480:	74 0A BA 4A 01 3C F9 74	03 BA CC 01 E8 46 FF BA	t±0J0<út♥>I0èFÿ0
0000000490:	70 01 E8 40 FF B4 30 CD	21 3C 00 75 06 BA 84 01	p0è@ÿ`0í!< u±,0
00000004A0:	E8 32 FF BE 8B 01 83 C6	01 E8 62 FF 83 C6 04 8A	è2ÿK<0fÄ0èÿfÄ±S
00000004B0:	C4 E8 5A FF BA 8B 01 E8	1B FF B4 30 CD 21 BA 93	ÄèZÿe<0è±ÿ`0í!=""
00000004C0:	01 E8 11 FF 8A C7 E8 68	FF B4 30 CD 21 BA AE 01	0è±ÿSçèÿ`0í!±00
00000004D0:	E8 02 FF 8A C3 E8 59 FF	8A C5 E8 54 FF 8A C1 E8	à0ÿSÄÿÿSÄèTySÄè
00000004E0:	4F FF 32 C0 B4 4C CD 21		0ÿ2Ä`Lí!

Рис.5. "Плохой" EXE-файл в шестнадцатеричном виде.



7. Запуск в отладчике TD COM-программы представлен на рисунке 7.

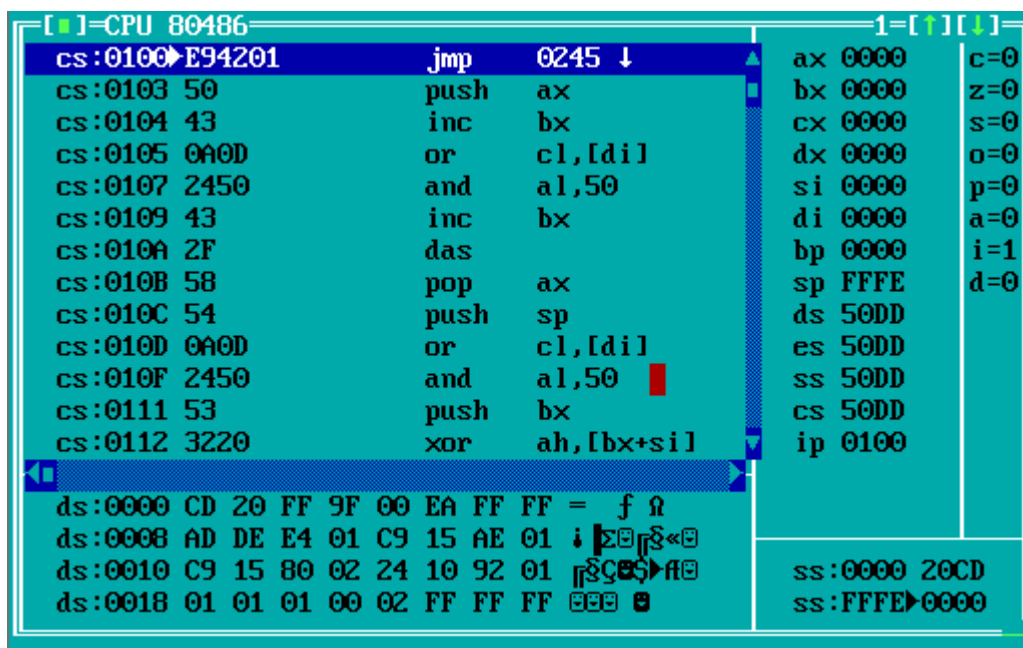


Рис.7. Запуск COM-программы в отладчике.

8. Запуск в отладчике TD EXE-программы представлен на рисунке 8.

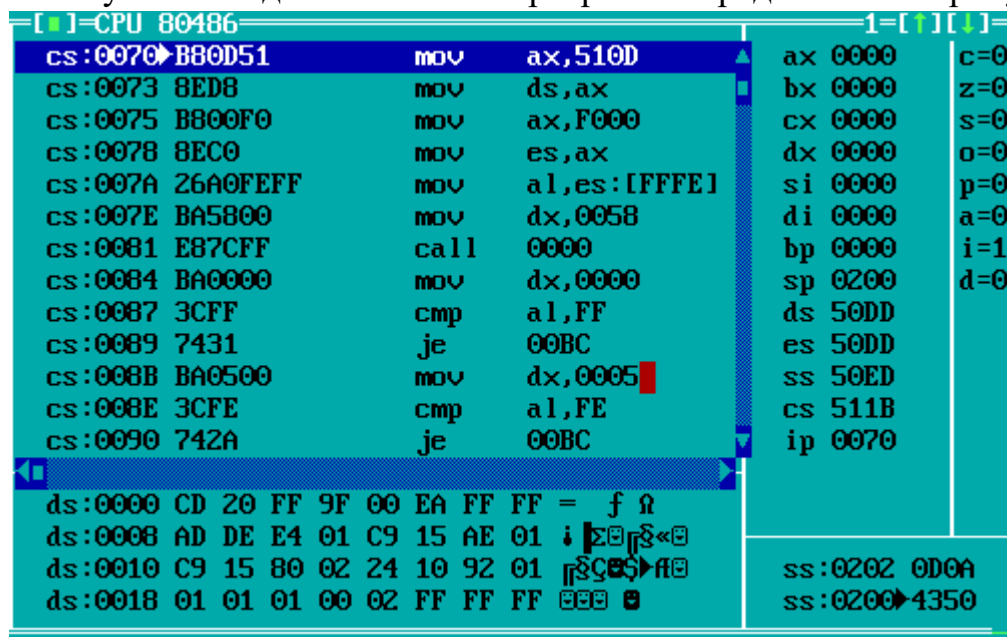


Рис.8. Запуск EXE-программы в отладчике.



## **Ответы на вопросы.**

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

1) Сколько сегментов должна содержать COM-программа?

Один сегмент CODESEG.

2) EXE-программа?

Количество допустимых сегментов определяется используемой моделью памяти:

- small – один сегмент кода, один сегмент данных;
- compact – один сегмент кода, несколько сегментов данных;
- medium – несколько сегментов кода, один сегмент данных;
- large – несколько сегментов кода, несколько сегментов данных;
- huge – много сегментов кода, много сегментов данных.

Программист может и не использовать перечисленные модели

А также сегмент стека.

3) Какие директивы должны обязательно быть в тексте COM-программы? Директива ASSUME, сообщающая ассемблеру информацию о соответствии между регистрами и сегментами. Директива ORG, сообщающая компилятору о смещении адресации внутри кода.

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

Из-за того, что в COM-программе сегментные регистры определяются в момент запуска, а не трансляции (компиляции) нельзя использовать команды, работающие с сегментами, команды, а также команды, размера больше 64Кб, или команды, работающие с 64-битными регистрами.

### **Отличия форматов файлов COM и EXE модулей.**

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

В начале файла можно заметить данные, которые используются в программе, затем следует сам код. COM-программа содержит один сегмент, не превышающий 64Кб. Код располагается с начала программы, но при запуске ему всегда предшествует блок памяти для PSP длиной 100H байт.

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

"Плохой" EXE-файл содержит только один сегмент. В самом начале идет заголовок EXE-файла, в котором содержится полезная информация, такая как сигнатура, длина заголовка в 16-байтных параграфах, значение SP и IP при входе и т.д. Заголовок занимает адреса 00-1ВН (28 байт). За заголовком следует таблица настроек адресов, которая занимает ровно столько места, сколько указано в заголовке. Сам код в данном EXE-файле начинается с адреса 300Н.

3) Какова структура "хорошего" EXE-файла? Чем он отличается от "плохого" EXE-файла?

Также, как и в "плохом" в "хорошем" в самом начале идет заголовок и таблица настроек адресов. "Хороший" EXE-файл содержит три сегмента: стека (в шестнадцатеричном виде представлен множеством символов S), данных и кода. Сам код в "хорошем" EXE-файле начинается с адреса 400Н, так как в отличие от "плохого" в нем определен стек.

### **Загрузка COM модуля в основную память.**

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

Система выделяет свободный сегмент, в первые 256 байт этого сегмента записывается PSP, непосредственно за ним загружается содержимое COM-файла без изменений, то есть с адреса 0100Н, указатель стека (регистр SP) устанавливается на конец сегмента.

2) Что располагается с адреса 0?

Program Segment Prefix (PSP) – область памяти размером 256 (0100h) байт, предшествующая программе при ее загрузке.

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

Все сегментные регистры имеют значения 50DDН. Они указывают на начало выделенной системой памяти, то есть на PSP.

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



адреса?

Стек расположен в конце выделенной памяти, сразу после содержимого COM-файла. В стек записывается 0000H (адрес возврата для команды ret).

### **Загрузка "хорошего" EXE модуля в основную память.**

1) Как загружается "хороший" EXE? Какие значения имеют сегментные регистры?

DS и ES устанавливаются на начало сегмента PSP, SS – на начало сегмента стека, CS – на начало сегмента кода. В IP записывается адрес первой команды.

2) На что указывают регистры DS и ES?

DS и ES указывают на начало префикса программного сегмента (PSP).

3) Как определяется стек?

Стек определяется в коде следующим образом:

```
<имя сегмента>  SEGMENT STACK
```

```
                DW <количество выделяемой памяти> DUP(<значение по  
умолчанию>)
```

```
<имя сегмента>  ENDS
```

```
ASSUME SS:AStack
```

4) Как определяется точка входа?

Точка входа – начальное значение IP – вычисляется с помощью директивы END <метка>.

### **Вывод.**

В ходе выполнения работы были написаны два варианта программы на языке ассемблера. Программа выводит на экран сообщения, содержащие информацию о некоторых параметрах системы. В процессе были получены три загрузочных модуля: COM из исходника для получения COM-файла, EXE из того же исходника, работающий некорректно, и EXE из исходника для EXE. Были выявлены различия в исходниках, структурах загрузочных модулей, порядке запуска этих двух вариантов.

## ПРИЛОЖЕНИЕ А

### Исходный код COM-программы.

```
TESTPC SEGMENT
    ASSUME cs:TESTPC, ds:TESTPC, es:NOTHING, ss:NOTHING
    ORG 100H
START: jmp BEGIN
;DATA
FF_type db 'PC',10,13,'$'
FE_FB_type db 'PC/XT',10,13,'$'
FA_type db 'PS2 ver. 30',10,13,'$'
FC_type db 'PS2 ver. 50/60 or AT',10,13,'$'
F8_type db 'PS2 ver. 80',10,13,'$'
FD_type db 'PCjr',10,13,'$'
F9_type db 'PC Convertible',10,13,'$'
typePC db 'Your IBM PC type is ','$'
typeMSDOS db 'Your MSDOS type is ','$'
old_ver db '<2.0',10,13,'$'
new_ver db '0x.0y',10,13,'$'
serialOEM db 'Your serial OEM number is ','$'
serialUser db 10,13,'Your serial user number is ','$'
ErrorMsg db 'ERROR!',10,13,'$'
;-----
WriteMsg PROC near
    mov ah,09h
    int 21h
    ret
WriteMsg ENDP
;-----
TETR_TO_HEX PROC near
    and al,0Fh
    cmp al,09
    jbe NEXT
    add al,07
NEXT: add al,30h ; код нуля
    ret
TETR_TO_HEX ENDP
;-----
BYTE_TO_HEX PROC near
;байт в al переводится в два символа в шест. сс ax
    push cx
    mov ah,al
    call TETR_TO_HEX
    xchg al,ah
    mov cl,4
    shr al,cl
    call TETR_TO_HEX ;al - старшая
    pop cx ;ah - младшая цифра
    ret
BYTE_TO_HEX ENDP
;-----
WRD_TO_HEX PROC near
; перевод в 16сс 16ти разрядного числа
; 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 байта в 10сс, si - адрес поля младшей цифры
; al содержит исходный байт
    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_l
    or al,30h
    mov [si],al
end_l: pop dx
    pop cx
    ret
BYTE_TO_DEC ENDP
;-----
WRITE_AL_HEX PROC NEAR
    push ax
    push dx
    call BYTE_TO_HEX

    mov dl, al
    mov al, ah
    mov ah, 02h
    int 21h

    mov dl, al
    int 21h
    pop dx
    pop ax
    ret
WRITE_AL_HEX ENDP
;-----
BEGIN:
    mov ax, 0F000h
    mov es, ax
    mov al, es:[0FFFEh]

    mov dx, offset typePC
    call WriteMsg

    mov dx, offset FF_type

```

```

cmp al, 0FFh
je WRITE_TYPE

mov dx, offset FE_FB_type
cmp al, 0FEh
je WRITE_TYPE
cmp al, 0FBh
je WRITE_TYPE

mov dx, offset FC_type
cmp al, 0FCh
je WRITE_TYPE

mov dx, offset FA_type
cmp al, 0FAh
je WRITE_TYPE

mov dx, offset F8_type
cmp al, 0F8h
je WRITE_TYPE

mov dx, offset FD_type
cmp al, 0FDh
je WRITE_TYPE

mov dx, offset F9_type
cmp al, 0F9h
je WRITE_TYPE

mov dx, offset ErrorMessage

```

WRITE\_TYPE:

```

call WriteMsg

mov dx, offset typeMSDOS
call WriteMsg

mov ah, 30h
int 21h

cmp al, 0
jne SKIP_1
mov dx, offset old_ver
call WriteMsg

```

SKIP\_1:

```

mov si, offset new_ver
add si, 1
call BYTE_TO_DEC
add si, 4
mov al, ah
call BYTE_TO_DEC
mov dx, offset new_ver
call WriteMsg

mov ah, 30h
int 21h

mov dx, offset serialOEM
call WriteMsg

```

```
mov al, bh  
call WRITE_AL_HEX
```

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

```
mov dx, offset serialUser  
call WriteMsg  
mov al, bl  
call WRITE_AL_HEX  
mov al, ch  
call WRITE_AL_HEX  
mov al, cl  
call WRITE_AL_HEX
```

```
xor al,al  
mov AH,4Ch  
int 21h
```

```
TESTPC ENDS
```

```
END START;
```

## ПРИЛОЖЕНИЕ Б

### Исходный код EXE-программы.

```
ASTACK SEGMENT STACK
    DW 100 DUP(?)
ASTACK ENDS

DATA SEGMENT
FF_type db 'PC',10,13,'$'
FE_FB_type db 'PC/XT',10,13,'$'
FA_type db 'PS2 ver. 30',10,13,'$'
FC_type db 'PS2 ver. 50/60 or AT',10,13,'$'
F8_type db 'PS2 ver. 80',10,13,'$'
FD_type db 'PCjr',10,13,'$'
F9_type db 'PC Convertible',10,13,'$'
typePC db 'Your IBM PC type is ','$'
typeMSDOS db 'Your MSDOS type is ','$'
old_ver db '<2.0',10,13,'$'
new_ver db '0x.0y',10,13,'$'
serialOEM db 'Your serial OEM number is ','$'
serialUser db 10,13,'Your serial user number is ','$'
ErrorMsg db 'ERROR!',10,13,'$'
DATA ENDS

;-----
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:ASTACK

WriteMsg PROC near
    mov ah,09h
    int 21h
    ret
WriteMsg ENDP

;-----
TETR_TO_HEX PROC near
    and al,0Fh
    cmp al,09
    jbe NEXT
    add al,07
NEXT: add al,30h ; код нуля
    ret
TETR_TO_HEX ENDP

;-----
BYTE_TO_HEX PROC near
;байт в al переводится в два символа в шест. сс ах
    push cx
    mov ah,al
    call TETR_TO_HEX
    xchg al,ah
    mov cl,4
    shr al,cl
    call TETR_TO_HEX ;al - старшая
    pop cx ;ah - младшая цифра
    ret
BYTE_TO_HEX ENDP

;-----
WRD_TO_HEX PROC near
; перевод в 16сс 16ти разрядного числа
; ах - 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 байта в 10сс, si - адрес поля младшей цифры
; al содержит исходный байт
        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_l
        or al,30h
        mov [si],al
end_l: pop dx
        pop cx
        ret
BYTE_TO_DEC ENDP
;-----
WRITE_AL_HEX PROC NEAR
        push ax
        push dx
        call BYTE_TO_HEX

        mov dl, al
        mov al, ah
        mov ah, 02h
        int 21h

        mov dl, al
        int 21h
        pop dx
        pop ax
        ret
WRITE_AL_HEX ENDP
;-----
BEGIN PROC near
        mov ax, DATA
        mov ds, ax

        mov ax, 0F000h
        mov es, ax

```



```

mov al, es:[OFFFEh]

mov dx, offset typePC
call WriteMsg

mov dx, offset FF_type
cmp al, OFFh
je WRITE_TYPE

mov dx, offset FE_FB_type
cmp al, OFEh
je WRITE_TYPE
cmp al, OFBh
je WRITE_TYPE

mov dx, offset FC_type
cmp al, OFCh
je WRITE_TYPE

mov dx, offset FA_type
cmp al, OFAh
je WRITE_TYPE

mov dx, offset F8_type
cmp al, OF8h
je WRITE_TYPE

mov dx, offset FD_type
cmp al, OFDh
je WRITE_TYPE

mov dx, offset F9_type
cmp al, OF9h
je WRITE_TYPE

mov dx, offset ErrorMsg

```

WRITE\_TYPE:

```

call WriteMsg

mov dx, offset typeMSDOS
call WriteMsg

mov ah, 30h
int 21h

cmp al, 0
jne SKIP_1
mov dx, offset old_ver
call WriteMsg

```

SKIP\_1:

```

mov si, offset new_ver
add si, 1
call BYTE_TO_DEC
add si, 4
mov al, ah
call BYTE_TO_DEC
mov dx, offset new_ver
call WriteMsg

```

```

mov ah, 30h
int 21h

mov dx, offset serialOEM
call WriteMsg
mov al, bh
call WRITE_AL_HEX

mov ah, 30h
int 21h

mov dx, offset serialUser
call WriteMsg
mov al, bl
call WRITE_AL_HEX
mov al, ch
call WRITE_AL_HEX
mov al, cl
call WRITE_AL_HEX

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

BEGIN ENDP
CODE ENDS

END BEGIN

```