

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

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

Студент гр.

Аверина О.С.

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.

### **Постановка задачи.**

1. Требуется написать 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.
2. 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.
3. Далее необходимо отладить полученный исходный модуль и получить «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля.
4. Затем нужно написать текст «хорошего» .EXE модуля, который выполняет те же функции, что и модуль .COM, далее его построить, отладить, сравнить исходные тексты для .COM и .EXE модулей и ответить на вопросы.
5. Запустить FAR и открыть файл загрузочного модуля .COM и файл плохого .EXE в шестнадцатеричном виде, сравнить с хорошим файлом. Ответить на вопросы.
6. Запустить в отладчике TD.EXE .COM. Ответить на контрольные вопросы. Представить план загрузки модуля .COM в основную память.

7. Открыть в отладчике TD.EXE «хороший» .EXE. Ответить на вопросы.

### **Выполнение работы.**

В качестве основы программы был взят шаблон из методического пособия, в котором содержатся следующие процедуры:

- TETR\_TO\_HEX - Перевод десятичной цифры в код символа
- BYTE\_TO\_HEX - Перевод байта в 16-ной с/с в символьный код
- WRD\_TO\_HEX - Перевод слова в 16-ной с/с в символьный код
- BYTE\_TO\_DEC – Перевод байта в 10-ю с/с
- WRITE\_PROC – Печать символов

Были объявлены строки для вывода информации:

- TYPE\_PC db 'Type: PC',0DH,0AH,'\$'
- TYPE\_PC\_XT db 'Type: PC/XT',0DH,0AH,'\$'
- TYPE\_AT db 'Type: AT',0DH,0AH,'\$'
- TYPE\_PS2\_30 db 'Type: PS2 модель 30',0DH,0AH,'\$'
- TYPE\_PS2\_50\_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'\$'
- TYPE\_PS2\_80 db 'Type: PS2 модель 80',0DH,0AH,'\$'
- TYPE\_PC\_JR db 'Type: PCjr',0DH,0AH,'\$'
- TYPE\_PC\_CONV db 'Type: PC Convertible',0DH,0AH,'\$'
- VERSION\_OS db 'Version DOS: . ',0DH,0AH,'\$'
- OEM db 'Serial number OEM: ',0DH,0AH,'\$'
- USER\_NUMBER db 'User serial number:     H \$'

Были создана процедура для определения типа ПК PC\_WRITE в соответствии с таблицей:

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

А также функция для определения характеристик ОС VERSION\_OS\_proc:

- номер основной версии системы
- номер модификации;
- серийный номер OEM;
- серийный номер пользователя.

В результате выполнения программы были получены следующие значения:

```
C:\>OS_LAB_1.COM
Type: AT
Version DOS: 5.0
Serial number OEM: 0
User serial number: 0000H
C:\>
```

Рисунок 1 – «хороший» .COM модуль

```
C:\>OS_LAB_1.exe
5 0
0JType: PC
0
0000
0JType: PC
0JType: PC
C:\>
```

Рисунок 2 – «плохой» .EXE модуль

```
C:\>OS_LAB_1.EXE
Type: AT
Version DOS: 5.0
Serial number OEM: 0
User serial number: 0000H
C:\>
```

Рисунок 3 – «хороший» .EXE модуль

### Ответы на вопросы.

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

1) Сколько сегментов должна содержать COM-программа?

В .COM файле все данные, код и стек располагаются в одном сегменте.

2) EXE – программа?

EXE-программа должна содержать не менее одного сегмента. Каждый сегмент определяется отдельно.

3) Какие директивы обязательно должны быть в тексте COM-программе?

В COM-файле обязательно должна быть директива ORG 100h, чтобы зарезервировать память для PSP. Также требуется директива ASSUME, где прописывается, что сегменты кода и данных указывают на один сегмент программы.

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

Нельзя использовать команды вида mov <регистр>, seg <имя сегмента>, так как в .COM-программе отсутствует таблица настроек (содержит описание адресов, которые зависят от размещения загрузочного модуля в ОП).

***Отличия форматов файлов .COM и .EXE программ***

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

В .COM файле все данные, код и стек располагаются в одном сегменте и не могут превышать 64 килобайта. При загрузке COM-файла в память DOS занимает первые 256 байт (100h) блоком данных PSP и располагает код программы после нее. При вызове COM-файла в стек помещаются сегментный адрес и ноль. Вид COM – файла представлен на рисунке 4.

```

C:\Users\Olya\Desktop\OS\MASM\OS_1_COM.COM
00000000: E9 F3 01 54 79 70 65 3A 20 50 43 0D 0A 24 54 79  éó@Type: PC
00000001: 70 65 3A 20 50 43 2F 58 54 0D 0A 24 54 79 70 65  pe: PC/XT
00000002: 3A 20 41 54 0D 0A 24 54 79 70 65 3A 20 50 53 32  : AT
00000003: 20 D0 BC D0 BE D0 B4 D0 B5 D0 BB D1 8C 20 33 30  %D%D'DµD»ÑE 30
00000004: 0D 0A 24 54 79 70 65 3A 20 50 53 32 20 D0 BC D0  Type: PS2 %D
00000005: BE D0 B4 D0 B5 D0 BB D1 8C 20 35 30 20 D0 B8 D0  %D'DµD»ÑE 50 D,D
00000006: BB D0 B8 20 36 30 0D 0A 24 54 79 70 65 3A 20 50  »D, 60
00000007: 53 32 20 D0 BC D0 BE D0 B4 D0 B5 D0 BB D1 8C 20  S2 %D%D'DµD»ÑE
00000008: 38 30 0D 0A 24 54 79 70 65 3A 20 50 D0 A1 6A 72 80
00000009: 0D 0A 24 54 79 70 65 3A 20 50 43 20 43 6F 6E 76  Type: PC Conv
0000000A: 65 72 74 69 62 6C 65 0D 0A 24 56 65 72 73 69 6F  ertible
0000000B: 6E 20 44 4F 53 3A 20 2D 2E 20 0D 0A 24 53 65  n DOS: .
0000000C: 72 69 61 6C 20 6E 75 6D 62 65 72 20 4F 45 4D 3A  rial number OEM:
0000000D: 20 20 0D 0A 24 55 73 65 72 20 73 65 72 69 61 6C  Type
0000000E: 20 6E 75 6D 62 65 72 3A 20 20 20 20 20 20 48  number:
0000000F: 20 24 24 0F 3C 09 76 02 04 07 04 30 C3 51 8A E0  $o<ov
00000010: E8 EF FF 86 C4 B1 04 D2 E8 E8 E6 FF 59 C3 53 8A  èiÿ†Ä±
00000011: FC E8 E9 FF 88 25 4F 88 05 4F 8A C7 E8 DE FF 88  üèéÿ"%O"
00000012: 25 4F 88 05 5B C3 51 52 32 E4 33 D2 B9 0A 00 F7  %O^+
00000013: F1 80 CA 30 88 14 4E 33 D2 3D 0A 00 73 F1 3C 00  ñ€Ê0`
00000014: 74 04 0C 30 88 04 5A 59 C3 B4 09 CD 21 C3 B8 00  t
00000015: F0 8E C0 26 A0 FE FF 3C FF 74 23 3C FE 74 25 3C  ðŽÄ&
00000016: FB 74 21 3C FC 74 23 3C FA 74 25 3C F8 74 27 3C  út!<
00000017: FD 74 29 3C F9 74 2B 32 F6 8A D0 EB 2B 90 BA 03  ýt)<
00000018: 01 EB 25 90 BA 0E 01 EB 1F 90 BA 1C 01 EB 19 90  0ë%
00000019: BA 27 01 EB 13 90 BA 69 01 EB 0D 90 BA 85 01 EB  e'
0000001A: 07 90 BA 93 01 EB 01 90 E8 9E FF C3 B4 30 CD 21  •
0000001B: 50 BE AA 01 83 C6 0D E8 6C FF 58 8A C4 83 C6 03  P%
0000001C: E8 63 FF BA AA 01 E8 80 FF BE BE 01 83 C6 13 8A  ècÿ
0000001D: C3 E8 52 FF BA BE 01 E8 6F FF BF D5 01 83 C7 19  ÀèRÿ
0000001E: 8B C1 E8 29 FF 8A C3 E8 3C FF 83 EF 02 89 05 BA  <Àè)ÿ
0000001F: D5 01 E8 54 FF C3 E8 55 FF E8 B0 FF 32 C0 B4 4C  Õ0èTÿ
00000020: CD 21  í!

```

Рис. 4 – «хороший» .COM – файл

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

В «плохом» EXE - файле данные и код располагаются в одном сегменте, что для EXE - файла некорректно, так как код и данные должны быть разделены на отдельные сегменты. Код располагается с адреса 300h, а с адреса 0h идёт таблица настроек. Вид «плохого» EXE – файла представлен на рисунке 5.

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 F3 01 54 79 70 65 3A	20 50 43 0D 0A 24 54 79	é60Type: PC\
00000000310:	70 65 3A 20 50 43 2F 58	54 0D 0A 24 54 79 70 65	pe: PC/XT\
00000000320:	3A 20 41 54 0D 0A 24 54	79 70 65 3A 20 50 53 32	: AT\
00000000330:	20 D0 BC D0 BE D0 B4 D0	B5 D0 BB D1 8C 20 33 30	0%D%D`DµD»ÑE 30
00000000340:	0D 0A 24 54 79 70 65 3A	20 50 53 32 20 D0 BC D0	\Type: PS2 0%D
00000000350:	BE D0 B4 D0 B5 D0 BB D1	8C 20 35 30 20 D0 B8 D0	%D`DµD»ÑE 50 D,D
00000000360:	BB D0 B8 20 36 30 0D 0A	24 54 79 70 65 3A 20 50	»D, 60\
00000000370:	53 32 20 D0 BC D0 BE D0	B4 D0 B5 D0 BB D1 8C 20	S2 0%D%D`DµD»ÑE
00000000380:	38 30 0D 0A 24 54 79 70	65 3A 20 50 D0 A1 6A 72	80\
00000000390:	0D 0A 24 54 79 70 65 3A	20 50 43 20 43 6F 6E 76	\Type: PC Conv
000000003A0:	65 72 74 69 62 6C 65 0D	0A 24 56 65 72 73 69 6F	ertible\
000000003B0:	6E 20 44 4F 53 3A 20 20	2E 20 20 0D 0A 24 53 65	n DOS: . \Se
000000003C0:	72 69 61 6C 20 6E 75 6D	62 65 72 20 4F 45 4D 3A	rial number OEM:
000000003D0:	20 20 0D 0A 24 55 73 65	72 20 73 65 72 69 61 6C	\User serial
000000003E0:	20 6E 75 6D 62 65 72 3A	20 20 20 20 20 20 20 48	number: H
000000003F0:	20 24 24 0F 3C 09 76 02	04 07 04 30 C3 51 8A E0	\$So<ov0♦♦0ÄQ5à
00000000400:	E8 EF FF 86 C4 B1 04 D2	E8 E8 E6 FF 59 C3 53 8A	èiÿ+Ä±♦0èèÿÿÄ55
00000000410:	FC E8 E9 FF 88 25 4F 88	05 4F 8A C7 E8 DE FF 88	üèéÿ^%0^♦05Çèÿ^

Рис. 5 – «плохой» .EXE – файл

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

В EXE-программе код, данные и стек поделены на сегменты. Заголовок состоит из форматированной части, содержащей сигнатуру и данные, необходимые для загрузки EXE-файла, и таблицы для настройки адресов. В отличие от «плохого» EXE в «хорошем» EXE присутствуют три сегмента: сегмент кода, сегмент данных и сегмент стека, а «плохой» EXE содержит один сегмент, совмещающий код и данные. Вид «хорошего» EXE – файла представлен на рисунке 6.

00000004F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000500:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000510:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000520:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000530:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000540:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000550:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000560:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000570:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000580:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000590:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000005A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000005B0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000005C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000005D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000005E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000005F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000600:	54 79 70 65 3A 20 50 43	0D 0A 24 54 79 70 65 3A	Type: PC
0000000610:	20 50 43 2F 58 54 0D 0A	24 54 79 70 65 3A 20 41	PC/XT
0000000620:	54 0D 0A 24 54 79 70 65	3A 20 50 53 32 20 D0 BC	Type: A
0000000630:	D0 BE D0 B4 D0 B5 D0 BB	D1 8C 20 33 30 0D 0A 24	Type: PS2
0000000640:	54 79 70 65 3A 20 50 53	32 20 D0 BC D0 BE D0 B4	Type: PS2
0000000650:	D0 B5 D0 BB D1 8C 20 35	30 20 D0 B8 D0 BB D0 B8	Type: PS2
0000000660:	20 36 30 0D 0A 24 54 79	70 65 3A 20 50 53 32 20	Type: PS2
0000000670:	D0 BC D0 BE D0 B4 D0 B5	D0 BB D1 8C 20 38 30 0D	Type: PS2
0000000680:	0A 24 54 79 70 65 3A 20	50 D0 A1 6A 72 0D 0A 24	Type: PD
0000000690:	54 79 70 65 3A 20 50 43	20 43 6F 6E 76 65 72 74	Type: PC Convert
00000006A0:	69 62 6C 65 0D 0A 24 56	65 72 73 69 6F 6E 20 44	ible
00000006B0:	4F 53 3A 20 20 2E 20 20	0D 0A 24 53 65 72 69 61	OS:
00000006C0:	6C 20 6E 75 6D 62 65 72	20 4F 45 4D 3A 20 20 20	l number OEM:

Рис. 6 – «хороший» EXE – файл

### Загрузка COM – модуля в память

- 1) Какой формат загрузки модуля COM? С какого адреса располагается код?
  1. Система выделяет свободный сегмент памяти и заносит его адрес во все сегментные регистры (CS, DS, ES и SS).
  2. В первые 256 байт этого сегмента записывается PSP.
  3. Непосредственно за ним загружается содержимое COM-файла без изменений.
  4. Указатель стека (регистр SP) устанавливается на конец сегмента.
  5. В стек записывается 0000h (адрес возврата для команды ret).
  6. Управление передаётся по адресу CS:0100h, где находится первый байт исполняемого файла.

Код располагается сразу после PSP по адресу 100h.



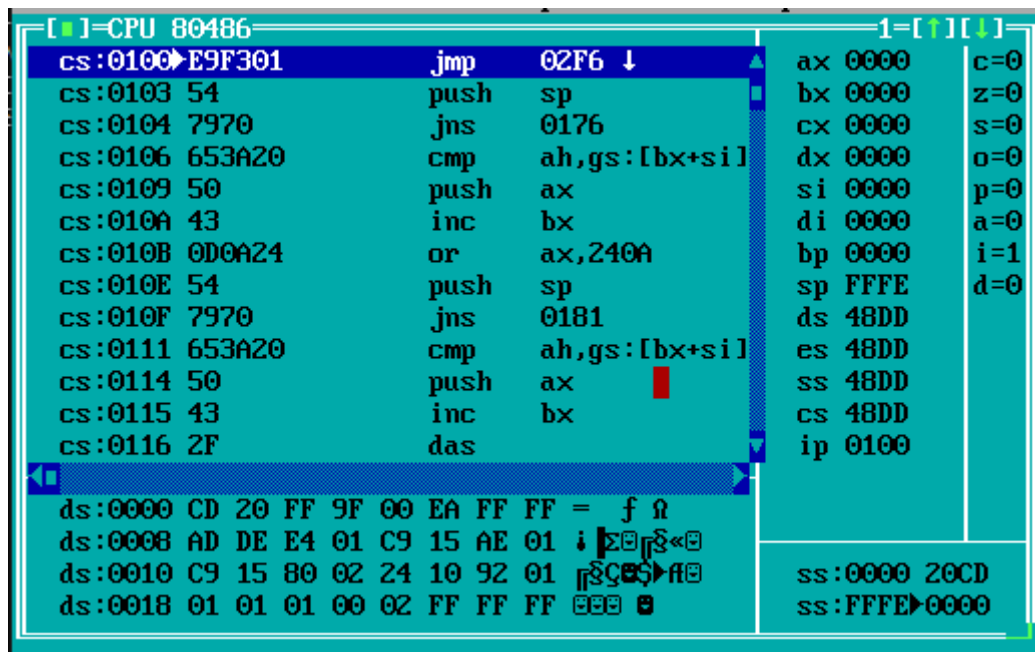


Рис. 7 – COM-файл в отладчике TD

2) Что располагается с адреса 0?

Сегмент PSP.

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

Все сегменты равны нулю. Все они указывают на PSP.

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

В SP содержится адрес FFFEH, т.е. на начало стека в момент загрузки модуля. В com-модулях стек растёт от старших адресов к младшим. SS – на начало.

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

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

DS и ES указывают на начало префикса программного сегмента, SS – на начало сегмента стека, CS – на начало сегмента команд. В IP загружается смещение точки входа в программу, которая берётся из метки после директивы END. Вид «хорошего» EXE – файла в отладчике TD представлен на рисунке 8.

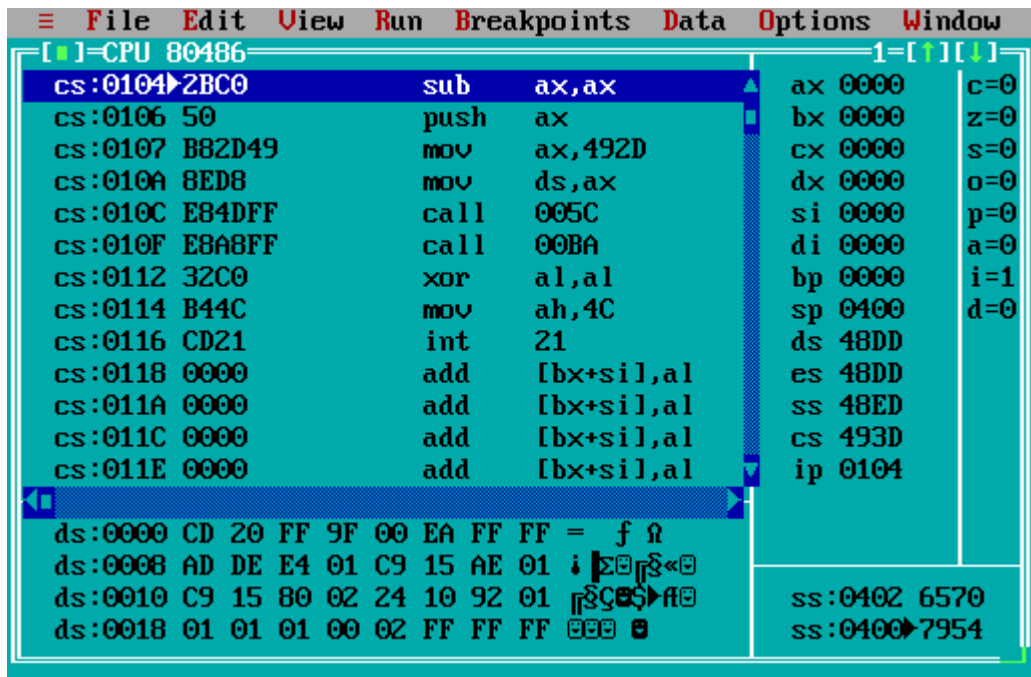


Рис 8 - COM-файл в отладчике TD

2) На что указывают регистры DS и ES?

DS и ES указывают на начало префикса программного сегмента.

3) Как определяется стек?

Стек определяется с помощью директивы `.stack`, после которой задаётся размер стека. При исполнении регистр SS указывает на начало сегмента стека, а SP на конца стека(его смещение).

4) Как определяется точка входа?

Точка входа указывается после директивы `END`, по которой программа переходит при запуске.

**Выводы.**

В ходе лабораторной работы были исследованы различия в структурах исходных текстов модулей типов `.COM` и `.EXE`, структур файлов загрузочных модулей и способов их загрузки в основную память.

## ПРИЛОЖЕНИЕ А (OS\_1\_COM.asm)

```
TESTPC SEGMENT
ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
;Данные
TYPE_PC db 'Type: PC',0DH,0AH,'$'
TYPE_PC_XT db 'Type: PC/XT',0DH,0AH,'$'
TYPE_AT db 'Type: AT',0DH,0AH,'$'
TYPE_PS2_30 db 'Type: PS2 модель 30',0DH,0AH,'$'
TYPE_PS2_50_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'$'
TYPE_PS2_80 db 'Type: PS2 модель 80',0DH,0AH,'$'
TYPE_PC_JR db 'Type: PCjr',0DH,0AH,'$'
TYPE_PC_CONV db 'Type: PC Convertible',0DH,0AH,'$'

VERSION_OS db 'Version DOS: . ',0DH,0AH,'$'
OEM db 'Serial number OEM: ',0DH,0AH,'$'
USER_NUMBER db 'User serial number:      H $'
;Процедуры
;-----
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 шест. 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
;
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_l
    or AL,30h
    mov [SI],AL
end_l: pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP
;-----
WRITE_PROC PROC near
    mov AH,09h
    int 21h
    ret
WRITE_PROC ENDP

PC_WRITE PROC near

    mov ax, 0f000h ; получаем номер модели

```

```

mov es, ax
mov al, es:[0ffffh]

cmp al, 0ffh ; начинаем сравнивать
je pc
cmp al, 0feh
je pc_xt
cmp al, 0fbh
je pc_xt
cmp al, 0fch
je pc_at
cmp al, 0fah
je pc_ps2_m30
cmp al, 0f8h
je pc_ps2_m80
cmp al, 0fdh
je pc_jr
cmp al, 0f9h
je pc_conv
xor dh, dh
mov dl, al
jmp write

pc:
    mov dx, offset TYPE_PC
    jmp write
pc_xt:
    mov dx, offset TYPE_PC_XT
    jmp write
pc_at:
    mov dx, offset TYPE_AT
    jmp write
pc_ps2_m30:
    mov dx, offset TYPE_PS2_30
    jmp write
pc_ps2_m80:
    mov dx, offset TYPE_PS2_80
    jmp write
pc_jr:
    mov dx, offset TYPE_PC_JR
    jmp write
pc_conv:
    mov dx, offset TYPE_PC_CONV
    jmp write
write:
    call WRITE_PROC
ret

```

```
PC_WRITE ENDP
```

```
VERSION_OS_proc PROC near
```

```
    MOV AH,30h
```

```
    INT 21h
```

```
    push ax
```

```
    mov si, offset VERSION_OS
```

```
    add si, 13 ; сдвиг в строке
```

```
    call BYTE_TO_DEC
```

```
    pop ax
```

```
    mov al, ah
```

```
    add si, 3
```

```
    call BYTE_TO_DEC
```

```
    mov dx, offset VERSION_OS ; сдвиг для печати
```

```
    call WRITE_PROC
```

```
    mov si, offset OEM
```

```
    add si, 19
```

```
    mov al, bl
```

```
    call BYTE_TO_DEC
```

```
    mov dx, offset OEM
```

```
    call WRITE_PROC
```

```
    mov di, offset USER_NUMBER
```

```
    add di, 25
```

```
    mov ax, cx
```

```
    call WRD_TO_HEX
```

```
    mov al, bl
```

```
    call BYTE_TO_DEC
```

```
    sub di, 2 ; what is it
```

```
    mov [di], ax
```

```
    mov dx, offset USER_NUMBER
```

```
    call WRITE_PROC
```

```
    ret
```

```
VERSION_OS_proc ENDP
```

```
BEGIN:
```

```
    call PC_WRITE
```

```
    call VERSION_OS_proc
```

```
    xor AL,AL
```

```
    mov AH,4Ch
```

```

int 21H
TESTPC ENDS
END START ;

```

## ПРИЛОЖЕНИЕ Б (OS\_1\_exe.asm)

```

STACK SEGMENT STACK
    DW 200H DUP(?) ; заполнен мусором
STACK ENDS

```

```

;Данные
DATA SEGMENT
TYPE_PC db 'Type: PC',0DH,0AH,'$'
TYPE_PC_XT db 'Type: PC/XT',0DH,0AH,'$'
TYPE_AT db 'Type: AT',0DH,0AH,'$'
TYPE_PS2_30 db 'Type: PS2 модель 30',0DH,0AH,'$'
TYPE_PS2_50_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'$'
TYPE_PS2_80 db 'Type: PS2 модель 80',0DH,0AH,'$'
TYPE_PC_JR db 'Type: PCjr',0DH,0AH,'$'
TYPE_PC_CONV db 'Type: PC Convertible',0DH,0AH,'$'

```

```

VERSION_OS db 'Version DOS: . ',0DH,0AH,'$'
OEM db 'Serial number OEM: ',0DH,0AH,'$'
USER_NUMBER db 'User serial number: H $'
DATA ENDS

```

```

;Процедуры
;-----

```

```

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
    ;байт в AL переводится в два символа шест. 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
;
    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_l

```



```

        or AL,30h
        mov [SI],AL
        end_l: pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP
;-----
WRITE_PROC PROC near
        mov AH,09h
        int 21h
        ret
WRITE_PROC ENDP

PC_WRITE PROC near

        mov ax, 0f000h ; получаем номер модели
        mov es, ax
        mov al, es:[0ffffh]

        cmp al, 0ffh ; начинаем сравнивать
        je pc
        cmp al, 0feh
        je pc_xt
        cmp al, 0fbh
        je pc_xt
        cmp al, 0fch
        je pc_at
        cmp al, 0fah
        je pc_ps2_m30
        cmp al, 0f8h
        je pc_ps2_m80
        cmp al, 0fdh
        je pc_jr
        cmp al, 0f9h
        je pc_conv
        xor dh, dh
        mov dl, al
        jmp write

pc:
        mov dx, offset TYPE_PC
        jmp write
pc_xt:
        mov dx, offset TYPE_PC_XT
        jmp write
pc_at:
        mov dx, offset TYPE_AT

```

```

        jmp write
pc_ps2_m30:
        mov dx, offset TYPE_PS2_30
        jmp write
pc_ps2_m80:
        mov dx, offset TYPE_PS2_80
        jmp write
pc_jr:
        mov dx, offset TYPE_PC_JR
        jmp write
pc_conv:
        mov dx, offset TYPE_PC_CONV
        jmp write
write:
        call WRITE_PROC
        ret
PC_WRITE ENDP

```

```

VERSION_OS_proc PROC near
        MOV AH,30h
        INT 21h

        push ax

        mov si, offset VERSION_OS
        add si, 13 ; сдвиг в строке
        call BYTE_TO_DEC
        pop ax
        mov al, ah
        add si, 3
        call BYTE_TO_DEC
        mov dx, offset VERSION_OS ; сдвиг для печати
        call WRITE_PROC

        mov si, offset OEM
        add si, 19
        mov al, bl
        call BYTE_TO_DEC
        mov dx, offset OEM
        call WRITE_PROC

        mov di, offset USER_NUMBER
        add di, 25
        mov ax, cx

```

```

        call WRD_TO_HEX
        mov al, bl
        call BYTE_TO_DEC
        sub di, 2
        mov [di], ax
        mov dx, offset USER_NUMBER
        call WRITE_PROC

        ret

VERSION_OS_proc ENDP

TYPE_OS PROC near
        sub ax,ax;
        push AX
        mov ax, DATA
        mov ds, ax

        call PC_WRITE
        call VERSION_OS_proc

        xor AL,AL
        mov AH,4Ch
        int 21H
TYPE_OS ENDP
CODE ENDS
END TYPE_OS

```