

**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**

Студентка гр. 9383

Лапина А.А.

Преподаватель

Ефремов М. А.

Санкт-Петербург

2021

### **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.**

Требуется написать текст исходного .COM модуля, который определяет тип РС и версию системы. Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип РС 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.

### **Выполнение работы.**

Были реализованы следующие функции:

\* PC\_TYPE — определяет тип ПК (по содержимому регистра AL) согласно таблице (Рисунок 1).

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

Рисунок 1 – Соответствие кода предпоследнего байта ROM BIOS и типа PC

Выполнение программы начинается с занесения в регистр AL данных предпоследнего байта ROM BIOS для определения типа PC. Происходит сравнение регистра AL с числами, соответствующих различным типам PC (Рисунок 1). В случае совпадения происходит вывод на экран соответствующей числу строки, а в противном случае – замена числа на код ASCII символа и вывод его на экран. Все выводы на экран в программе реализованы с помощью функции write, в которой использована функция 09h прерывание 21h.

\*Функция OS, определяет номер основной версии MS-DOS, лежащий в регистре AL, номер её модификации в регистре AH, серийный номер OEM в регистре BH и 24-битный серийный номер пользователя в регистрах BL:CH. Программа последовательно переводит числа в соответствующие им строки и выводит на экран. Завершается программа вызовом функции 4Ch прерывания 21h с кодом 0. Разработанный программный код см. в приложении А.

Были выделены строки для вывода информации:

- T\_PC db 'Type: PC',0DH,0AH,'\$' - для типа PC;
- T\_PC\_XT db 'Type: PC/XT',0DH,0AH,'\$'- для типа PC/XT;
- T\_AT db 'Type: AT',0DH,0AH,'\$'- для типа AT;
- T\_PS2\_M30 db 'Type: PS2 модель 30',0DH,0AH,'\$' - для типа PS2 модель 30;
- T\_PS2\_M50\_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'\$' для типа PS2 модель 50 или 60;

- T\_PS2\_M80 db 'Type: PS2 модель 80',0DH,0AH,'\$' - для типа PS2 модель 80;
- T\_PC\_JR db 'Type: PCjr',0DH,0AH,'\$' для типа PC jr;
- T\_PC\_C db 'Type: PC Convertible',0DH,0AH,'\$' - для типа PC Convertible;
- VERSION db 'Version: . ',0DH,0AH,'\$' - для версии и модификации;
- OEM db 'OEM: ',0DH,0AH,'\$' - для серийного номера OEM;
- USER db 'User: ' \$ - для серийного номера пользователя;

Результаты работы «хороших» .COM и .EXE модулей представлены на рисунках 2 и 3. Результат работы «плохого» .EXE представлен на рисунке 4.

```
C:\>com.com
Type: AT
Version: 5.0
OEM: 0
User: 000000
```

Рисунок 2 –Результат .COM модуля

```
C:\>exe.exe
Type: AT
Version: 5.0
OEM: 0
User: 000000
```

Рисунок 3 – Результат .EXE модуля

```
C:\>com.exe
Type: PC
5 0
Type: PC
0
000000
Type: PC
```

Рисунок 4 – Результат «плохого» .EXE модуля

## **Контрольные вопросы.**

### Отличия исходных текстов COM и EXE программ:

1. Сколько сегментов должна содержать COM-программа?

COM-программа должна содержать один сегмент, в котором располагаются команды и данные.

2. Сколько сегментов должна содержать EXE-программа?

EXE-программа может содержать от одного до трёх сегментов: сегмент команд, данных и стека (последние два могут отсутствовать).

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

Должна быть обязательна директива `ORG 100h`, так как при загрузке модуля все сегментные регистры содержат адрес префикса программного сегмента (PSP), который является 256-байтовым(100H) блоком, поэтому адресация имеет смещение в 256 байт от нулевого адреса. Также необходима процедура `ASSUME` для того, чтобы сегмент данных и сегмент кода указывали на один общий сегмент. (`ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING`)

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

Не все форматы команд можно использовать в COM-программе. Так, например, ввиду отсутствия таблицы настроек нельзя использовать команды вида `mov <register>, seg <segment>`. Таблица настроек состоит из элементов, число которых записано в байтах 06-07. Элемент таблицы настройки состоит из двух полей: 2-байтного смещения и 2-байтного сегмента, и указывает слова в загрузочном модуле, содержащее адрес, который должен быть настроен на место памяти, в которое загружается задача.

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

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

Файл .COM состоит из программного сегмента PSP размером 256 байт, одного сегмента, содержащего команды и данные, а также

стека, генерируемого автоматически. Структура файла представлена на рисунке 5. Код располагается с адреса 100h (256).

00000000	E9	D8	01	54	79	70	65	3A	20	50	43	0D	0A	24	54	79	04.Type: PC...\$T
00000010		70	65	3A	20	50	43	2F	54	0D	0A	24	54	79	70	65	pe: PC/XT...\$T
00000020		3A	20	41	54	0D	0A	24	54	79	70	65	3A	20	50	53	AT...\$Type: P52
00000030		20	0D	BC	D0	BE	D0	84	D0	B5	D0	B8	D1	8C	20	33	30
00000040		0D	0A	24	54	79	70	65	3A	20	50	53	20	0D	BC	D0	..\$Type: P52
00000050		BE	D0	B4	D0	B5	D0	B8	D1	8C	20	35	30	20	D0	B8	50
00000060		B8	D0	B8	20	36	30	D0	0A	24	54	79	70	65	3A	20	60..\$Type: P
00000070		53	32	20	0D	BC	D0	BE	D0	B4	D0	B5	D0	B8	D1	8C	S2
00000080		38	30	0D	0A	24	54	79	70	65	3A	20	50	D0	A1	6A	72 80..\$Type: P1-jr
00000090		0D	0A	24	54	79	70	65	3A	20	50	43	20	43	6F	6E	76 ..\$Type: PC Conv
000000A0		65	72	74	69	62	6C	65	0D	0A	24	56	65	72	73	69	ertible...\$Versto
000000B0		6E	3A	20	2E	20	20	20	3A	20	4A	2F	45	40	3A	20	n: ...\$OEM:
000000C0		0D	0A	24	55	73	65	72	3A	20	20	20	20	20	20	20	..\$User:
000000D0		24	FF	0F	3C	09	76	02	04	07	04	30	C3	51	8A	E0	\$\$..c.v...0 Qe0n
000000E0		E4	22	86	C4	B1	04	D2	E8	E8	E6	FF	59	C3	53	8A	n a...0000 Y\$E\$
000000F0		E8	E9	FF	F8	25	4F	88	05	4F	8A	C7	E8	DE	FF	28	00 000E.00 0  %
00000100		4F	88	05	38	C3	51	52	32	E4	33	D2	89	0A	00	F1	0E. [QR23Z]...\$.=
00000110		80	CA	30	88	14	43	32	D3	00	00	00	73	F1	3C	00	000E.N3...\$.s.c.t.
00000120		04	0C	30	88	04	5A	59	C3	B8	00	0F	8E	C0	26	0A	..0E.ZY... A&A.
00000130		FF	3C	FF	74	1C	3C	FE	74	1E	3C	FB	74	1A	3C	FC	FE <.t.<.t.<.t.<.t.
00000140		1C	3C	FA	74	1E	3C	F8	74	26	3C	FD	74	28	3C	F9	74 <.t.<.t.<.t.<.t.
00000150		2A	BA	03	01	EB	2B	90	BA	0E	01	EB	25	90	BA	1C	01 * .0.E ...\$.%E
00000160		EB	1F	90	BA	27	01	EB	19	90	BA	43	01	EB	13	90	BA 0.E ...\$.0.E C.0.E
00000170		69	01	EB	0D	90	BA	85	01	EB	07	90	BA	93	01	EB	01 1.0.E A.0.E 0.0.E
00000180		90	BA	09	C0	21	C3	84	30	C0	21	50	BE	AA	01	83	06 E ... 0... A.E
00000190		09	EB	71	FF	58	8A	C4	83	C6	03	E8	68	FF	BA	AA	01 .0q X0-a.h.0h.0h.0h.
000001A0		B4	09	CD	21	BE	BA	01	83	C6	05	8A	C7	E8	56	FF	BA  ... ... .0.E 0.0.E
000001B0		BA	01	B4	09	CD	21	BF	C3	01	83	C7	08	8B	C1	E8	2C  ... ... .0.E 0.0.E
000001C0		FF	8A	C3	E8	16	FF	83	EF	02	89	05	BA	A4	C3	01	BA 0.E.0.0.E ... .0.E
000001D0		C0	21	C3	E8	52	FF	E8	AD	FF	32	08	C4	C4	CD	21	= 0P.0P. 2 ...

### Рисунок 5 – Структура файла .COM

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

«Плохой» файл .EXE состоит из единственного сегмента, что некорректно для файлов данного типа. Структура файла представлена на рисунке 6. Код и данные располагается с адреса 300h, а с адреса 0h лежат заголовок и таблица настроек.

```

000002f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000300 E9 D0 01 54 79 70 65 3A 20 50 43 00 0A 24 54 79 0A
00000310 70 65 3A 20 50 43 2F 58 54 0A 24 54 79 70 65 3A
00000320 3A 20 41 54 00 0A 24 54 79 70 65 3A 20 50 53 32
00000330 20 D8 0C BE D8 BE D8 BE D8 BE D8 BE D8 BE D8 BE D8
00000340 00 0A 24 54 79 70 65 3A 20 50 53 32 20 D8 0C BE
00000350 BE D8 0A D8 BE D8 BB D1 8C 20 53 30 20 D8 00
00000360 BB D8 28 26 36 30 00 0A 24 54 79 70 65 3A 20 50
00000370 53 32 20 D8 0C BE D8 BE D8 BE D8 BB D1 8C 20
00000380 38 30 0D 0A 24 54 79 70 65 3A 20 50 0D 0A 1A 6A
00000390 00 0A 24 54 79 70 65 3A 20 50 43 00 43 6E 76
000003A0 65 72 74 69 62 65 00 0A 24 56 65 72 73 69 6F
000003B0 6E 3A 20 2E 2E 20 0D 0A 24 4F 45 40 3A 20 20
000003C0 00 0A 24 55 73 65 72 3A 20 20 20 20 20 20 20 20
000003D0 24 4F 3C 0B 96 04 04 07 04 3C C3 51 8A E0 E8
000003E0 FF EF 86 C4 B1 04 D2 E8 E6 FF 59 C3 58 8A FC
000003F0 E8 EF 88 25 4F 88 05 4F 8A C7 E8 DE FF 88 25
00000400 4F 88 05 C3 51 52 32 E4 33 02 89 0A 00 F7 F1
00000410 80 3A 08 14 4E 33 02 0A 00 07 F3 F1 3C 00 74
00000420 04 0C 30 08 04 5A 59 53 B8 00 0E 0C 26 AD FE
00000430 FF 3C FF 74 1C 3C FE 74 1E 3C FB 74 1A 3C FC 74
00000440 1C 3A FA 74 1E 3C FB 74 26 3C FD 74 28 3C F9 74
00000450 3C BA 83 01 EB 28 98 BA 0E 01 EB 25 98 BA 1C 01
00000460 EF 1F 90 BA 27 01 E9 90 BA 43 01 EB 13 90 BA
00000470 01 EB 0D 90 BA 85 01 EB 07 90 BA 93 01 EB 01
00000480 90 BA 09 CD 21 C3 B4 30 CD 21 50 BE AA 01 83 06
00000490 E9 E8 71 FF 58 8A C3 06 03 88 FF AA AA 01
000004A0 B4 09 CD 21 BE 0A 01 C3 06 05 A7 C8 EF 56 FF BA
000004B0 B4 01 B4 09 CD 21 BF C3 01 83 C7 08 8B C1 E8 2C
000004C0 FF 8A C3 E8 F6 FF 83 FF 02 89 05 BA C3 01 B4 09
000004D0 CD 21 C3 E8 F2 FF E8 AD 82 0C B4 C4 CD 21

```

Рисунок 6 – Структура «плохого» файла .EXE

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

В «хорошем» EXE файле уже исправлено объединение всех сегментов, теперь они разнесены по разным: сегмент стека, сегмент данных и сегмент кода, в отличие от «плохого». Также отсутствие обязательной для COM файлов директивы ORG 100h приводит к тому, что память под управляющие структуры (PSP) не выделяется.

Адрес	Содержимое	Комментарий
000001F0	00 00 00 00 00 00 00 00	.....
00000200	00 00 00 00 00 00 00 00	.....
00000210	00 00 00 00 00 00 00 00	.....
00000220	00 00 00 00 00 00 00 00	.....
00000230	00 00 00 00 00 00 00 00	.....
00000240	00 00 00 00 00 00 00 00	.....
00000250	00 00 00 00 00 00 00 00	.....
00000260	00 00 00 00 00 00 00 00	.....
00000270	00 00 00 00 00 00 00 00	.....
00000280	54 79 70 65 3A 20 50 43	Type: PC..\$Type:
00000290	20 50 43 2F 58 54 0D 0A	PC/XT..\$Type: A
000002A0	54 0D 0A 24 54 79 70 65	T..\$Type: PS2
000002B0	D0 BE D0 B4 D0 B5 D0 BB	30..\$
000002C0	54 79 70 65 3A 20 50 53	Type: PS2
000002D0	D0 B5 D0 BB D1 8C 20 35	50
000002E0	20 36 30 0D 0A 24 54 79	60..\$Type: PS2
000002F0	D0 BC D0 BE D0 B4 D0 B5	80.
00000300	0A 24 54 79 70 65 3A 20	.\$Type: p1jr..\$
00000310	54 79 70 65 3A 20 50 43	Type: PC Convert
00000320	69 62 6C 65 0D 0A 24 56	ible..\$Version:
00000330	20 2E 20 20 0D 0A 24 4F	..\$OEM: ..\$
00000340	55 73 65 72 3A 20 20 20	User: ..\$
00000350	E9 02 01 24 0F 3C 09 76	0..\$.<.v....0Qè
00000360	E0 E8 EF FF 86 C4 B1 04	αφη à-тФФ Y S
00000370	8A FC E8 E9 FF 88 25 4F	è"ФФ è%0è.0è Ф
00000380	88 25 4F 88 05 5B C3 51	è%0è. [Q23T ..
00000390	F7 F1 80 CA 30 88 14 4E	≈±Ç0è.N3 =..s±<
000003A0	00 74 04 0C 30 88 04 5A	.t..0è.ZY .=A&
000003B0	A0 FE FF 3C FF 74 1C 3C	á < t.<.t.<v/t.<
000003C0	FC 74 1C 3C FA 74 1E 3C	nt.<.t.<^t&<^t(<
000003D0	F9 74 2A BA 00 00 EB 2B	·t* ..δ+É ..δ%É
000003E0	19 00 EB 1F 90 BA 24 00	..δ.É s.δ.É @.δ.
000003F0	90 BA 66 00 EB 0D 90 BA	É f.δ.É è.δ.É É.
00000400	EB 01 90 B4 09 CD 21 C3	δ.É .=: 0= Pδ°.
00000410	83 C6 09 E8 71 FF 58 8A	âf.Фq Xè-âf.Фh
00000420	A7 00 B4 09 CD 21 BE B7	°. .=!д _âf.è Фv
00000430	FF BA B7 00 B4 09 CD 21	_-. =!_L.âf.-iL
00000440	F8 2C FF 8A C3 F8 16 FF	Ф.èФ.â.è.ЛL.

Рисунок 7 – Структура «хорошего» файла .EXE

### Загрузка .COM модуля в основную память:

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

После загрузки программы все 4 сегментных регистра указывают на начало единственного сегмента, то есть фактически на начало PSP. Указатель стека SP автоматически инициализируется числом 0FFFFeh. Таким образом, независимо от фактического размера программы, ей выделяется 64 Кбайт



адресного пространства, всю нижнюю часть которого занимает стек (рисунок 8). Указатель команд на момент начала программы указывает на адрес 100h, с которого, соответственно, располагается код.

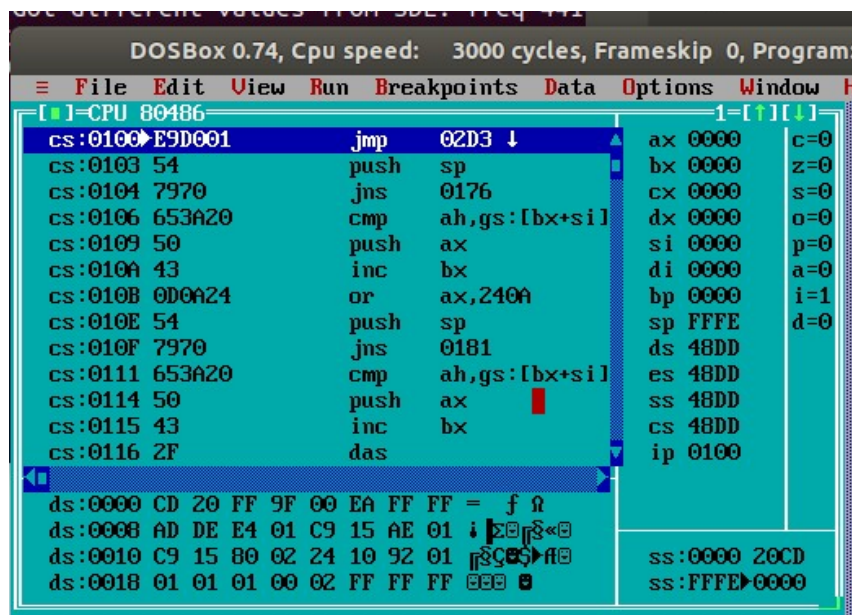


Рисунок 8 – Отладка .COM модуля (с помощью TD.EXE)

2. Что располагается с адреса 0?

С адреса 0 располагается программный сегмент PSP, размером 256 байт, зарезервируемый операционной системой.

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

Сегментные регистры имеют значения 48DDh и указывают на программный сегмент PSP.

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

Стек занимает всю нижнюю часть адресного пространства. При этом сегментный регистр SS в начале программы, как и остальные регистры, указывает на PSP (48DDh), а регистр SP – на конец стека с адресом 0FFFEh.

Загрузка «хорошего» .EXE модуля в основную память:

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



Система, загрузив программу в память, инициализирует сегментные регистры так, что регистры DS и ES указывают на начало PSP, CS – на начало сегмента команд, а SS – на начало сегмента стека. В указатель команд IP загружается смещение точки входа в программу, а в указатель стека SP – смещение конца сегмента стека. Таким образом, после загрузки программы в память адресуемыми оказываются все регистры, кроме сегментов данных.

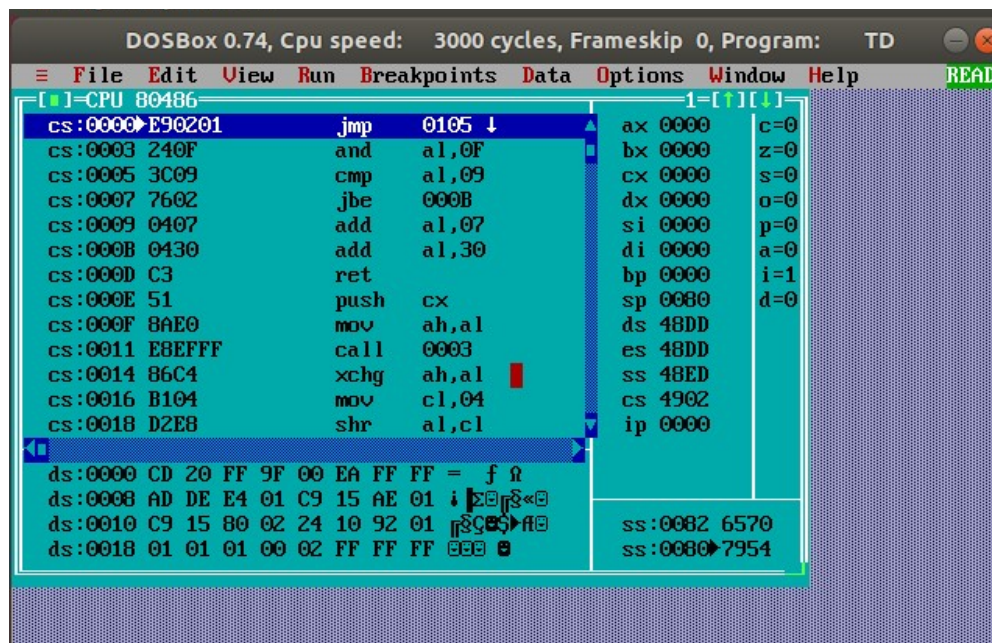


Рисунок 9 – Отладка «хорошего» .EXE модуля (с помощью TD.EXE)

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

При запуске программы регистры DS и ES указывают на начало PSP.

## 3. Как определяется стек?

Оператор segment, начинающий сегмент стека, имеет описатель stack, что приводит к тому, что при загрузке программы в память регистр SS будет настроен на начало сегмента стека, а указатель стека SP —на его конец.

## 4. Как определяется точка входа?

Точка входа определяется с помощью директивы END.

### **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

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: com.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H      ;так как адресация начинается со смещением 100 в .com
START: JMP BEGIN      ;точка входа (метка)

; Данные
T_PC db  'Type: PC',0DH,0AH,'$'
T_PC_XT db 'Type: PC/XT',0DH,0AH,'$'
T_AT db  'Type: AT',0DH,0AH,'$'
T_PS2_M30 db 'Type: PS2 модель 30',0DH,0AH,'$'
T_PS2_M50_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'$'
T_PS2_M80 db 'Type: PS2 модель 80',0DH,0AH,'$'
T_PC_JR db 'Type: PCjr',0DH,0AH,'$'
T_PC_C db 'Type: PC Convertible',0DH,0AH,'$'

VERSION db 'Version:  .  ',0DH,0AH,'$'
OEM db  'OEM:  ',0DH,0AH,'$'
USER db  'User:      $'

; 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
;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_l
        or AL,30h
        mov [SI],AL
end_l:

```

```

        pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP
;-----
PC_TYPE PROC near
    mov ax, 0f000h        ;получаем номер модели
    mov es, ax
    mov al, es:[0ffffh]   ;смещение

    cmp al, 0ffh          ;сравниваем
    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
pc:
    mov dx, offset T_PC
    jmp write
pc_xt:
    mov dx, offset T_PC_XT
    jmp write
pc_at:
    mov dx, offset T_AT
    jmp write
pc_ps2_m30:
    mov dx, offset T_PS2_M30
    jmp write
pc_ps2_m50_60:
    mov dx, offset T_PS2_M50_60
    jmp write
pc_ps2_m80:
    mov dx, offset T_PS2_M80
    jmp write

```

```
pc_jr:
    mov dx, offset T_PC_JR
    jmp write
```

```
pc_conv:
    mov dx, offset T_PC_C
    jmp write
```

```
write:
    mov AH,09h
    int 21h
    ret
```

```
PC_TYPE ENDP
```

```
OS PROC near
```

```
    ;версия
    mov ah, 30h
    int 21h
    push ax
    mov si, offset VERSION
    add si, 9        ;смещение
    call BYTE_TO_DEC
    pop ax
    mov al, ah
    add si, 3        ;смещение
    call BYTE_TO_DEC
    mov dx, offset VERSION
    mov AH,09h      ;вывод
    int 21h
```

```
    ;серийный номер OEM
    mov si, offset OEM
    add si, 5        ;смещение
    mov al, bh
    call BYTE_TO_DEC
    mov dx, offset OEM
    mov AH,09h      ;вывод
    int 21h
```

```
    ;серийный номер пользователя
    mov di, offset USER
    add di, 11       ;смещение
    mov ax, cx
    call WRD_TO_HEX
    mov al, bl
```

```

    call BYTE_TO_HEX
    sub di, 2
    mov [di], ax
    mov dx, offset USER
    mov AH,09h      ;ВЫВОД
    int 21h
    ret

```

OS ENDP

; Код

BEGIN:

```

    call PC_TYPE
    call OS

```

```

    xor AL,AL
    mov AH,4Ch
    int 21H

```

TESTPC ENDS

END START; конец модуля, START - точка выхода

## Название файла: exe.asm

STK SEGMENT STACK

```

    DB 128 DUP (0) ;выделяем память

```

STK ENDS

DATA SEGMENT

; Данные

```

T_PC db  'Type: PC',0DH,0AH,'$'
T_PC_XT db 'Type: PC/XT',0DH,0AH,'$'
T_AT db  'Type: AT',0DH,0AH,'$'
T_PS2_M30 db 'Type: PS2 модель 30',0DH,0AH,'$'
T_PS2_M50_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'$'
T_PS2_M80 db 'Type: PS2 модель 80',0DH,0AH,'$'
T_PC_JR db 'Type: PCjr',0DH,0AH,'$'
T_PC_C db 'Type: PC Convertible',0DH,0AH,'$'

```

```

VERSION db 'Version:  . ',0DH,0AH,'$'

```

```

OEM db  'OEM: ',0DH,0AH,'$'

```

```

USER db  'User:      $'

```

DATA ENDS



```

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START: JMP BEGIN      ; точка входа (метка)

; 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
; 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_l
        or AL,30h
        mov [SI],AL
end_l:
        pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP
;-----
PC_TYPE PROC near
        mov ax, 0f000h          ;получаем номер модели
        mov es, ax
        mov al, es:[0ffffh]    ;смещение

        cmp al, 0ffh           ;сравниваем
        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
pc:
        mov dx, offset T_PC
        jmp write
pc_xt:
        mov dx, offset T_PC_XT
        jmp write
pc_at:
        mov dx, offset T_AT
        jmp write
pc_ps2_m30:
        mov dx, offset T_PS2_M30
        jmp write
pc_ps2_m50_60:
        mov dx, offset T_PS2_M50_60
        jmp write
pc_ps2_m80:
        mov dx, offset T_PS2_M80
        jmp write
pc_jr:
        mov dx, offset T_PC_JR
        jmp write
pc_conv:
        mov dx, offset T_PC_C
        jmp write
write:
        mov AH,09h
        int 21h
        ret
PC_TYPE ENDP

OS PROC near
        ;версия
        mov ah, 30h
        int 21h
        push ax
        mov si, offset VERSION
        add si, 9          ;смещение

```

```

call BYTE_TO_DEC
pop ax
mov al, ah
add si, 3      ;смещение
call BYTE_TO_DEC
mov dx, offset VERSION
mov AH,09h     ;вывод
int 21h

```

```

;серийный номер OEM
mov si, offset OEM
add si, 5      ;смещение
mov al, bh
call BYTE_TO_DEC
mov dx, offset OEM
mov AH,09h     ;вывод
int 21h

```

```

;серийный номер пользователя
mov di, offset USER
add di, 11     ;смещение
mov ax, cx
call WRD_TO_HEX
mov al, bl
call BYTE_TO_HEX
sub di, 2
mov [di], ax
mov dx, offset USER
mov AH,09h     ;вывод
int 21h
ret

```

OS ENDP

```

; Код
BEGIN:
    mov AX, DATA
    mov DS, AX
    call PC_TYPE
    call OS

    xor AL,AL
    mov AH,4Ch

```

```
int 21H  
CODE ENDS  
END START; конец модуля, START - точка выхода
```