

**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.**

Студентка гр. 0382

Чегодаева Е.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

### **Цель работы.**

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

### **Задание.**

**Шаг 1.** Написать текст исходного .COM модуля, который определяет тип РС и версию системы. Результатом выполнения этого шага будет «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля.

**Шаг 2.** Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в *Шаге 1* и отладить его. Таким образом, будет получен «хороший» .EXE.

**Шаг 3.** Сравнить исходные тексты для .COM и .EXE модулей. Ответить на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

**Шаг 4.** Запустить FAR и открыть (F3/F4) файл загрузочного модуля .COM и файл «плохого» .EXE в шестнадцатеричном виде. Затем открыть (F3/F4) файл загрузочного модуля «хорошего» .EXE и сравните его с предыдущими файлами. Ответить на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

**Шаг 5.** Открыть отладчик TD.EXE и загрузить .COM. Ответить на контрольные вопросы «Загрузка COM модуля в основную память». Представить в отчете план загрузки модуля .COM в основную память.

**Шаг 6.** Открыть отладчик TD.EXE и загрузить «хороший» .EXE. Ответить на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

**Шаг 7.** Оформить отчет в соответствии с требованиями. В отчете необходимо привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей – в отладчике.

### **Теоретические сведения.**

Тип IBM PC хранится в байте по адресу 0F000:0FFFFh, в предпоследнем байте ROM BIOS.

Соответствие кода и типа в таблице:

<b>PC</b>	<b>FF</b>
<b>PC/XT</b>	<b>FE,FB</b>
<b>AT</b>	<b>FC</b>
<b>PS2 модель 30</b>	<b>FA</b>
<b>PS2 модель 50 или 60</b>	<b>FC</b>
<b>PS2 модель 80</b>	<b>F8</b>
<b>PCjr</b>	<b>FD</b>
<b>PC Convertible</b>	<b>F9</b>

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:CX - 24-битовый серийный номер пользователя.

### **Выполнение работы.**

На основе шаблона .COM модуля, приведённого в методических указаниях, была реализована программа, определяющая тип PC и версию системы MS DOS. Для этого в шаблон были добавлены сообщения, указывающие на соответствующие данные о системе. А также процедуры «PRINT», «IBM» и «MS\_DOS».

Определение типа IBM PC осуществляется посредством обращения к байту по адресу 0F000:0FFFEh, предпоследнему байту ROM. В зависимости от полученного значения, путём сравнения с данными таблицы, — выводится строка, содержащая информацию о типе PC при помощи прописанной процедуры вывода.

Следующая процедура предназначена для определения версии MS DOS, серийного номера OEM и номера пользователя, реализована на основе функции 30H прерывания 21H. Посредством процедур, приведённых в исходном шаблоне .COM модуля, необходимые данные будут внесены в корректной форме в выводимые сообщения.

Таким образом получен «хороший» .COM модуль и «плохой» .EXE. О чем свидетельствуют результаты запуска:

Рис. 1. Результат запуска lb1\_com.com

Рис. 2. Результат запуска lb1\_com.exe

Далее, для получения «хорошего» .EXE модуля, ранее описанная программа была разделена на соответствующие сегменты.

Рис. 3. Результат запуска lb1\_exe.exe

Исходные коды программ см. в приложении А

### **Контрольные вопросы.**

#### Отличия исходных текстов COM и EXE программ

1) Сколько сегментов должна содержать COM-программа?

- Ответ: COM-программа должна содержать один сегмент (сегмент кода).

2) EXE-программа?

- Ответ: EXE-программа может содержать несколько сегментов (стека, кода, данных), но обязательно наличие как минимум одного (сегмента кода).

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

- Ответ: В тексте COM-программы должны быть следующее директивы:

1.ORG 100h — для смещения на 256 байт (размер PSP) от нулевого адреса;

2. ASSUME — для связи сегментов, указывающих на регистры сегмента данных и сегмента кода.

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

- Ответ: Нет — нельзя использовать команды с указанием адресов сегментов в связи с отсутствием *relocation table* (сегментные регистры определяются при запуске программы, а не на этапе линковки).


#### Отличия форматов файлов COM и EXE модулей

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

- Ответ: В COM-файле все данные заключены внутри одного сегмента, включая: все описания переменных, инструкции процессора и

директивы. Код располагается с адреса 0100h (следом за PSP). см.

Рис. 4.



0000000000:	E9 1C 02 49 42 4D 20 50	43 20 74 79 70 65 20 3A	éL0IBM PC type :
0000000010:	20 50 43 3B 0D 0A 24 49	42 4D 20 50 43 20 74 79	PC;IBM PC ty
0000000020:	70 65 20 3A 20 50 43 2F	58 54 3B 0D 0A 24 49 42	pe : PC/XT;IBM
0000000030:	4D 20 50 43 20 74 79 70	65 20 3A 20 41 54 3B 0D	M PC type : AT;
0000000040:	0A 24 49 42 4D 20 50 43	20 74 79 70 65 20 3A 20	IBM PC type :
0000000050:	50 53 32 20 6D 6F 64 65	6C 20 33 30 3B 0D 0A 24	PS2 model 30;
0000000060:	49 42 4D 20 50 43 20 74	79 70 65 20 3A 20 50 53	IBM PC type : PS
0000000070:	32 20 6D 6F 64 65 6C 20	35 30 20 6F 72 20 36 30	2 model 50 or 60
0000000080:	3B 0D 0A 24 49 42 4D 20	50 43 20 74 79 70 65 20	;IBM PC type
0000000090:	3A 20 50 53 32 20 6D 6F	64 65 6C 20 38 30 3B 0D	: PS2 model 80;
00000000A0:	0A 24 49 42 4D 20 50 43	20 74 79 70 65 20 3A 20	IBM PC type :
00000000B0:	50 43 6A 72 3B 0D 0A 24	49 42 4D 20 50 43 20 74	PCjr;IBM PC t
00000000C0:	79 70 65 20 3A 20 50 43	20 43 6F 6E 76 65 72 74	ype : PC Convert
00000000D0:	69 62 6C 65 3B 0D 0A 24	49 42 4D 20 3A 20 20 3B	ible;IBM :
00000000E0:	0D 0A 24 44 4F 53 20 3A	20 20 2E 20 3B 0D 0A 24	DOS : . ;
00000000F0:	4F 45 4D 20 3A 20 20 3B	0D 0A 24 55 53 45 52 20	OEM : ;USER
0000000100:	3A 20 20 20 20 20 20 20	68 3B 0D 0A 24 24 0F 3C	: h; \$o<
0000000110:	09 76 02 04 07 04 30 C3	51 8A E0 E8 EF FF 86 C4	ov0♦♦♦QŠàèiÿtÄ
0000000120:	B1 04 D2 E8 E8 E6 FF 59	C3 53 8A FC E8 E9 FF 88	±♦ðèèæÿYÄŠüëéÿ^
0000000130:	25 4F 88 05 4F 8A C7 E8	DE FF 88 25 4F 88 05 5B	%O^♦OŠÇèpÿ^%O^†[
0000000140:	C3 51 52 32 E4 33 D2 B9	0A 00 F7 F1 80 CA 30 88	ÄQR2ä3D¹ ÷ñèÊ0^
0000000150:	14 4E 33 D2 3D 0A 00 73	F1 3C 00 74 04 0C 30 88	¶N3D= sñ< t♦Q0^
0000000160:	04 5A 59 C3 B4 09 CD 21	C3 B8 00 F0 8E C0 26 8A	♦ZYÄ´oÍ!Ä, ðŽÄ&Š
0000000170:	26 FE FF 80 FC FF 74 34	80 FC FE 74 35 80 FC FB	&þÿ€üÿt4€üþt5€üü
0000000180:	74 30 80 FC FC 74 31 80	FC FA 74 32 80 FC F8 74	t0€üüÿt1€üüÿt2€üøt
0000000190:	33 80 FC FD 74 34 80 FC	F9 74 35 BF D8 01 83 C7	3€üÿt4€üüÿt5;øøfC
00000001A0:	06 E8 74 FF 89 05 BA D8	01 EB 28 90 BA 03 01 EB	♦èÿ%♦øøë(øø♥øë
00000001B0:	22 90 BA 17 01 EB 1C 90	BA 2E 01 EB 16 90 BA 42	"øø±øëLøø.øë=øøøø

Рис. 4. COM-файл.


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

Что располагается с адреса 0?

- Ответ: «Плохой» EXE, аналогично COM-файлу, представлен в одном сегменте. Код располагается с адреса 300h. С адреса 0 располагается заголовок. см. Рис 5.

Заголовок	0000000000:	4D 5A 2B 01 03 00 00 00	20 00 00 00 FF FF 00 00	MZ+♥	ÿÿ
	0000000010:	00 00 93 EF 00 01 00 00	1E 00 00 00 01 00 00 00	"i 0 ▲	0
	0000000020:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
	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		

.....



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 1C 02 49 42 4D 20 50	43 20 74 79 70 65 20 3A	éL IBM PC type :
0000000310:	20 50 43 3B 0D 0A 24 49	42 4D 20 50 43 20 74 79	PC; IBM PC ty
0000000320:	70 65 20 3A 20 50 43 2F	58 54 3B 0D 0A 24 49 42	pe : PC/XT; IB
0000000330:	4D 20 50 43 20 74 79 70	65 20 3A 20 41 54 3B 0D	M PC type : AT;
0000000340:	0A 24 49 42 4D 20 50 43	20 74 79 70 65 20 3A 20	IBM PC type :
0000000350:	50 53 32 20 6D 6F 64 65	6C 20 33 30 3B 0D 0A 24	PS2 model 30;
0000000360:	49 42 4D 20 50 43 20 74	79 70 65 20 3A 20 50 53	IBM PC type : PS
0000000370:	32 20 6D 6F 64 65 6C 20	35 30 20 6F 72 20 36 30	2 model 50 or 60
0000000380:	3B 0D 0A 24 49 42 4D 20	50 43 20 74 79 70 65 20	; IBM PC type
0000000390:	3A 20 50 53 32 20 6D 6F	64 65 6C 20 38 30 3B 0D	: PS2 model 80;
00000003A0:	0A 24 49 42 4D 20 50 43	20 74 79 70 65 20 3A 20	IBM PC type :
00000003B0:	50 43 6A 72 3B 0D 0A 24	49 42 4D 20 50 43 20 74	PCjr; IBM PC t
00000003C0:	79 70 65 20 3A 20 50 43	20 43 6F 6E 76 65 72 74	ype : PC Convert
00000003D0:	69 62 6C 65 3B 0D 0A 24	49 42 4D 20 3A 20 20 3B	ible; IBM :
00000003E0:	0D 0A 24 44 4F 53 20 3A	20 20 2E 20 3B 0D 0A 24	DOS : . ;
00000003F0:	4F 45 4D 20 3A 20 20 3B	0D 0A 24 55 53 45 52 20	OEM : ; USER
0000000400:	3A 20 20 20 20 20 20 20	68 3B 0D 0A 24 24 0F 3C	: h; \$ \$ <
0000000410:	09 76 02 04 07 04 30 C3	51 8A E0 E8 EF FF 86 C4	ov 0 0 0 0 0 0 0 0

Рис. 5. «Плохой» EXE-файл.

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

- Ответ: Структура «Хорошего» EXE содержит заголовок, *relocation table*, сегмент стека, сегмент данных и сегмент кода, в отличие от «плохого» EXE, содержащего всего лишь один сегмент. см. Рис. 6.

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:	49 42 4D 20 50 43 20 74	79 70 65 20 3A 20 50 43	IBM PC type : PC
0000000230:	3B 0D 0A 24 49 42 4D 20	50 43 20 74 79 70 65 20	;␣\$IBM PC type
0000000240:	3A 20 50 43 2F 58 54 3B	0D 0A 24 49 42 4D 20 50	: PC/XT;␣\$IBM P
0000000250:	43 20 74 79 70 65 20 3A	20 41 54 3B 0D 0A 24 49	C type : AT;␣\$I
0000000260:	42 4D 20 50 43 20 74 79	70 65 20 3A 20 50 53 32	BM PC type : PS2
0000000270:	20 6D 6F 64 65 6C 20 33	30 3B 0D 0A 24 49 42 4D	model 30;␣\$IBM
0000000280:	20 50 43 20 74 79 70 65	20 3A 20 50 53 32 20 6D	PC type : PS2 m
0000000290:	6F 64 65 6C 20 35 30 20	6F 72 20 36 30 3B 0D 0A	odel 50 or 60;␣\$
00000002A0:	24 49 42 4D 20 50 43 20	74 79 70 65 20 3A 20 50	\$IBM PC type : P
00000002B0:	53 32 20 6D 6F 64 65 6C	20 38 30 3B 0D 0A 24 49	S2 model 80;␣\$I
00000002C0:	42 4D 20 50 43 20 74 79	70 65 20 3A 20 50 43 6A	BM PC type : PCj
00000002D0:	72 3B 0D 0A 24 49 42 4D	20 50 43 20 74 79 70 65	r;␣\$IBM PC type
00000002E0:	20 3A 20 50 43 20 43 6F	6E 76 65 72 74 69 62 6C	: PC Convertibl
00000002F0:	65 3B 0D 0A 24 49 42 4D	20 3A 20 20 3B 0D 0A 24	e;␣\$IBM : ;␣\$
0000000300:	44 4F 53 20 3A 20 20 2E	20 3B 0D 0A 24 4F 45 4D	DOS : . ;␣\$OEM

Рис. 6. «Хороший» EXE-файл.

## Загрузка COM модуля в основную память

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

- Ответ: Загрузка модуля COM начинается с загрузки PSP в свободный участок основной памяти, следом загружается сам код. Код располагается с адреса — CS:0100h. см. Рис. 7.

cs:0100	E91C02	jmp	031F ↓	ax	0000	c=0
cs:0103	49	dec	cx	bx	0000	z=0
cs:0104	42	inc	dx	cx	0000	s=0
cs:0105	4D	dec	bp	dx	0000	o=0
cs:0106	205043	and	[bx+si+43],dl	si	0000	p=0
cs:0109	207479	and	[si+79],dh	di	0000	a=0
cs:010C	7065	jo	0173	bp	0000	i=1
cs:010E	203A	and	[bp+si],bh	sp	FFFE	d=0
cs:0110	205043	and	[bx+si+43],dl	ds	48DD	
cs:0113	3B0D	cmp	cx,[di]	es	48DD	
cs:0115	0A24	or	ah,[si]	ss	48DD	
cs:0117	49	dec	cx	cs	48DD	
cs:0118	42	inc	dx	ip	0100	

Рис. 7. COM-файл.



2) Что располагается с адреса 0?

➤ Ответ: С адреса 0 располагается PSP.

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

➤ Ответ: Все сегментные регистры (ES, DS, CS, SS) имеют значение 48DDh и указывают на начало PSP см. Рис. 7.

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

➤ Ответ: В COM модуле стек располагается в сегменте кода, генерируется автоматически при запуске программы. Указатель стека установлен на конец текущего сегмента FFFh и далее «идёт» в меньшую сторону (до 0000h).

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

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

➤ Ответ: Загрузка «хорошего» EXE, аналогично COM модуля, начинается с загрузки PSP в свободный участок основной памяти, далее загружается сам EXE-модуль согласно информации, указанной в заголовке. Сегментные регистры: DS=ES=48DDh, SS=48EDh, CS=490Eh см. Рис. 8.

cs:0109 8905	mov	edi,ax	ax 0000
cs:010B BAF800	mov	dx,00F8	bx 0000
cs:010E E846FF	call	0057	cx 0000
cs:0111 C3	ret		dx 0000
cs:0112 1E	push	ds	si 0000
cs:0113 2BC0	sub	ax,ax	di 0000
cs:0115 50	push	ax	bp 0000
cs:0116 B8FD48	mov	ax,48FD	sp 0100
cs:0119 8ED8	mov	ds,ax	ds 48DD
cs:011B E83EFF	call	005C	es 48DD
cs:011E E8A9FF	call	00CA	ss 48ED
cs:0121 32C0	xor	al,al	cs 490E
cs:0123 B44C	mov	ah,4C	ip 0112

Рис. 8. «Хороший» EXE-файл.

2) На что указывают регистры DS и ES?

➤ Ответ: Регистры DS и ES указывают на начало PSP.

3) Как определяется стек?

➤ Ответ: Стек определяется с помощью директивы SEGMENT STACK с указанием размера.

4) Как определяется точка входа?

➤ Ответ: Точка выхода определяется посредством директивы END.

### **Выводы.**

Были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

Реализованы программы, которые определяют тип РС и версию системы.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: lbl\_com.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START:    jmp BEGIN
; ДАННЫЕ
PC db 'IBM PC type : PC;', 0DH, 0AH, '$'
XT db 'IBM PC type : PC/XT;', 0DH, 0AH, '$'
AT db 'IBM PC type : AT;', 0DH, 0AH, '$'
PS230 db 'IBM PC type : PS2 model 30;', 0DH, 0AH, '$'
PS25060 db 'IBM PC type : PS2 model 50 or 60;', 0DH, 0AH, '$'
PS280 db 'IBM PC type : PS2 model 80;', 0DH, 0AH, '$'
PCjr db 'IBM PC type : PCjr;', 0DH, 0AH, '$'
PC_C db 'IBM PC type : PC Convertible;', 0DH, 0AH, '$'
NT db 'IBM :   ;', 0DH, 0AH, '$'
DOS db 'DOS :   . ;', 0DH, 0AH, '$'
OEM db 'OEM :   ;', 0DH, 0AH, '$'
USER db 'USER :           h.', 0DH, 0AH, '$'

; 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 16-го 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, в 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 10 с/с, в 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
;-----
; КОД

PRINT PROC near
        mov AH, 09h
        int 21h
        ret
PRINT ENDP

IBM PROC near
        mov AX, 0F000h
        mov ES, AX
        mov AH, ES:[0FFFEh]

        cmp AH, 0FFh
        je _PC

```

```

    cmp AH, 0FEh
    je _XT

    cmp AH, 0FBh
    je _XT

    cmp AH, 0FCh
    je _AT

    cmp AH, 0FAh
    je _PS230

    cmp AH, 0F8h
    je _PS280

    cmp AH, 0FDh
    je _PCjr

    cmp AH, 0F9h
    je _PC_C

_OTHER:
    mov DI, offset NT
    add DI, 6
    call BYTE_TO_HEX
    mov [DI], AX
    mov DX, offset NT
    jmp final

_PC:
    mov DX, offset PC
    jmp final

_XT:
    mov DX, offset XT
    jmp final

_AT:
    mov DX, offset AT
    jmp final

_PS230:
    mov DX, offset PS230
    jmp final

_PS280:
    mov DX, offset PS280
    jmp final

_PCjr:
    mov DX, offset PCjr
    jmp final

_PC_C:
    mov DX, offset PC_C

final:
    call PRINT

```

```

        ret
IBM ENDP

MS_DOS PROC near
        mov AH, 30h
        int 21h
_DOS:
        mov SI, offset DOS
        add SI, 6
        call BYTE_TO_DEC
        mov AL, AH
        add SI, 3
        call BYTE_TO_DEC
        mov DX, offset DOS
        call PRINT
_OEM:
        mov SI, offset OEM
        add SI, 6
        mov AL, BH
        call BYTE_TO_DEC
        mov DX, offset OEM
        call PRINT
_USER:
        mov DI, offset USER
        add DI, 12
        mov AX, CX
        call WRD_TO_HEX
        mov AL, BL
        call BYTE_TO_HEX
        sub DI, 2
        mov [DI], AX
        mov DX, offset USER
        call PRINT
        ret
MS_DOS ENDP

BEGIN:
        call IBM
        call MS_DOS
; Выход в DOS
        xor AL, AL
        mov AH, 4Ch
        int 21h
TESTPC ENDS
        END START

```

**Название файла: lbl\_exe.asm**

```

AStack SEGMENT STACK
        DW 12 DUP(?)
AStack ENDS

```

```

DATA SEGMENT
PC db 'IBM PC type : PC;', 0DH, 0AH, '$'
XT db 'IBM PC type : PC/XT;', 0DH, 0AH, '$'
AT db 'IBM PC type : AT;', 0DH, 0AH, '$'
PS230 db 'IBM PC type : PS2 model 30;', 0DH, 0AH, '$'
PS25060 db 'IBM PC type : PS2 model 50 or 60;', 0DH, 0AH, '$'
PS280 db 'IBM PC type : PS2 model 80;', 0DH, 0AH, '$'
PCjr db 'IBM PC type : PCjr;', 0DH, 0AH, '$'
PC_C db 'IBM PC type : PC Convertible;', 0DH, 0AH, '$'
NT db 'IBM :   ;', 0DH, 0AH, '$'
DOS db 'DOS :   . ;', 0DH, 0AH, '$'
OEM db 'OEM :   ;', 0DH, 0AH, '$'
USER db 'USER :           h.', 0DH, 0AH, '$'
DATA ENDS

```

```

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

```

```

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 16-го 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, в 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 10 с/с, в 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
;-----
; КОД

PRINT PROC near
        mov AH, 09h
        int 21h
        ret
PRINT ENDP

IBM PROC near
        mov AX, 0F000h
        mov ES, AX
        mov AH, ES:[0FFFEh]

        cmp AH, 0FFh
        je _PC

        cmp AH, 0FEh
        je _XT

        cmp AH, 0FBh
        je _XT

        cmp AH, 0FCh

```



```

        je _AT

        cmp AH, 0FAh
        je _PS230

        cmp AH, 0F8h
        je _PS280

        cmp AH, 0FDh
        je _PCjr

        cmp AH, 0F9h
        je _PC_C

_OTHER:
        mov DI, offset NT
        add DI, 6
        call BYTE_TO_HEX
        mov [DI], AX
        mov DX, offset NT
        jmp final

_PC:
        mov DX, offset PC
        jmp final

_XT:
        mov DX, offset XT
        jmp final

_AT:
        mov DX, offset AT
        jmp final

_PS230:
        mov DX, offset PS230
        jmp final

_PS280:
        mov DX, offset PS280
        jmp final

_PCjr:
        mov DX, offset PCjr
        jmp final

_PC_C:
        mov DX, offset PC_C

final:
        call PRINT
        ret
IBM ENDP

MS_DOS PROC near
        mov AH, 30h
        int 21h

_DOS:
        mov SI, offset DOS

```

```

        add SI, 6
        call BYTE_TO_DEC
        mov AL, AH
        add SI, 3
        call BYTE_TO_DEC
        mov DX, offset DOS
        call PRINT
_OEM:
        mov SI, offset OEM
        add SI, 6
        mov AL, BH
        call BYTE_TO_DEC
        mov DX, offset OEM
        call PRINT
_USER:
        mov DI, offset USER
        add DI, 12
        mov AX, CX
        call WRD_TO_HEX
        mov AL, BL
        call BYTE_TO_HEX
        sub DI, 2
        mov [DI], AX
        mov DX, offset USER
        call PRINT
        ret
MS_DOS ENDP

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

        call IBM
        call MS_DOS

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

Main ENDP
CODE ENDS
        END Main

```