

**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

Сергиенкова 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.**

Напишите 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 использованы функции:

TETR\_TO\_HEX – перевод десятичной цифры в код символа;

BYTE\_TO\_HEX – перевод байта в 16 с/с в символьный код;

WRD\_TO\_HEX – перевод слова в 16 с/с в символьный код;

BYTE\_TO\_DEC – перевод байта в 16 с/с в символьный код в 10 с/с.

Были составлены функции:

WSTRING – вывод строки на экран;

PC\_TYPE – определение типа PC (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

OS\_VER – определение характеристик OS.

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

Также 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\_MODEL\_30 db 'Type: PS2 модель 30',0DH,0AH,'\$';
- TYPE\_PS2\_MODEL\_50\_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'\$';
- TYPE\_PS2\_MODEL\_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,'\$';
- VERSIONS db 'Version MS-DOS: . ',0DH,0AH,'\$';
- SERIAL\_NUMBER db 'Serial number OEM: ',0DH,0AH,'\$';
- USER\_NUMBER db 'User serial number: H \$'.

В результате выполнения были получены следующие значения(рис.1-3):

```
C:\>LAB1_COM.COM
Type: AT
Version MS-DOS: 5.0
Serial number OEM: 0
User serial number: 000000H
```

Рисунок 1 – «хороший» .COM модуль

```
C:\>LAB1_COM.EXE

                                     0J@Type: PC
5 0
0J@Type: PC
0
000000                                0J@Type: PC
0J@Type: PC
```

Рисунок 2 – «плохой» .EXE модуль

```
C:\>LAB1_EXE.EXE
Type: AT
Version MS-DOS: 5.0
Serial number OEM: 0
User serial number: 000000H
```

Рисунок 3 – «хороший» .EXE модуль

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

1. Сколько сегментов должна содержать COM-программа?

COM-программа должна содержать только один сегмент. Код и данные находятся в одном сегменте, а стек устанавливается автоматически.

2. EXE-программа?

EXE-программа должна содержать не менее одного сегмента. Сегменты кода, данных и стека описываются отдельно друг от друга. Можно не описывать сегмент стека, в таком случае будет использоваться стек DOS.

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

Должна быть директива `ORG 100h` (так как 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.

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.

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

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

COM-файл состоит из одного сегмента.

Также COM-файл ограничен размером одного сегмента и не превышает 64 Кб. Код начинается с адреса `0h`.

192-168-0-111:tools anastasiasergienkova\$ hexyl LAB1\_COM.COM

00000000	e9 0e 02 54 79 70 65 3a	20 50 43 0d 0a 24 54 79	x...Type: PC__\$Ty
00000010	70 65 3a 20 50 43 2f 58	54 0d 0a 24 54 79 70 65	pe: PC/X T__\$Type
00000020	3a 20 41 54 0d 0a 24 54	79 70 65 3a 20 50 53 32	: AT__\$T type: PS2
00000030	20 d0 bc d0 be d0 b4 d0	b5 d0 bb d1 8c 20 33 30	xxxxxxx xxxxx 30
00000040	0d 0a 24 54 79 70 65 3a	20 50 53 32 20 d0 bc d0	__\$Type: PS2 xxx
00000050	be d0 b4 d0 b5 d0 bb d1	8c 20 35 30 20 d0 b8 d0	xxxxxxx x 50 xxx
00000060	bb d0 b8 20 36 30 0d 0a	24 54 79 70 65 3a 20 50	xxx 60__\$Type: P
00000070	53 32 20 d0 bc d0 be d0	b4 d0 b5 d0 bb d1 8c 20	S2 xxxxxx xxxxxxxx
00000080	38 30 0d 0a 24 54 79 70	65 3a 20 50 d0 a1 6a 72	80__\$Type: Pxxjr
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 45 72 72 6f 72 20	ertible__\$Error
000000b0	0d 0a 24 56 65 72 73 69	6f 6e 20 4d 53 2d 44 4f	__\$Versi on MS-DO
000000c0	53 3a 20 20 2e 20 20 0d	0a 24 53 65 72 69 61 6c	S: . ___\$Serial
000000d0	20 6e 75 6d 62 65 72 20	4f 45 4d 3a 20 20 0d 0a	number OEM: __
000000e0	24 55 73 65 72 20 73 65	72 69 61 6c 20 6e 75 6d	\$User se rial num
000000f0	62 65 72 3a 20 20 20 20	20 20 20 48 20 24 24 0f	ber: H \$\$.
00000100	3c 09 76 02 04 07 04 30	c3 51 8a e0 e8 ef ff 86	<_v...0 xQxxxxxx
00000110	c4 b1 04 d2 e8 e8 e6 ff	59 c3 53 8a fc e8 e9 ff	xx...xxx YxSxxxxx
00000120	88 25 4f 88 05 4f 8a c7	e8 de ff 88 25 4f 88 05	x%0x...0xxx xxxxx%0x
00000130	5b c3 51 52 32 e4 33 d2	b9 0a 00 f7 f1 80 ca 30	[xQR2x3x x_0xxxx0
00000140	88 14 4e 33 d2 3d 0a 00	73 f1 3c 00 74 04 0c 30	x...N3x=_0 s<0t..._0
00000150	88 04 5a 59 c3 b4 09 cd	21 c3 b8 00 f0 8e c0 26	x...ZYxx_x  xx0xxx&
00000160	a0 fe ff 3c ff 75 06 ba	03 01 eb 57 90 3c fe 75	xxx<xu... xxWx<xu
00000170	06 ba 0e 01 eb 4d 90 3c	fb 75 06 ba 0e 01 eb 43	xx...Mx< xu...xxC
00000180	90 3c fc 75 06 ba 1c 01	eb 39 90 3c fa 75 06 ba	x<xu...xx x9x<xu...x
00000190	27 01 eb 2f 90 3c fc 75	06 ba 43 01 eb 25 90 3c	'...x/x<xu...xC...x%<
000001a0	f8 75 06 ba 69 01 eb 1b	90 3c fd 75 06 ba 85 01	xu...i... x<xu...xx
000001b0	eb 11 90 3c fd 75 06 ba	93 01 eb 07 90 ba aa 01	xx...<xu...x xxxxxxx
000001c0	eb 01 90 e8 8f ff c3 b4	30 cd 21 be b3 01 83 c6	xx...xxxxx 0x!xx...xx
000001d0	10 50 e8 5d ff 58 8a c4	83 c6 03 e8 54 ff ba b3	...Px]xXxxx xxx...Txxx
000001e0	01 e8 71 ff be ca 01 83	c6 13 8a c7 e8 43 ff ba	...xqxxxxx xxxxxCxx
000001f0	ca 01 e8 60 ff bf e1 01	83 c7 19 8b c1 e8 1a ff	xx...`xxx... xxxxxxxx
00000200	8a c3 e8 04 ff 83 ef 02	89 05 ba e1 01 e8 45 ff	xxx...xxx... xxx...xE
00000210	c3 e8 46 ff e8 b0 ff 32	c0 b4 4c cd 21	xxFxxxx2 xxLx!

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

Что располагается с адреса 0?

EXE файл некорректно работает, так как данные и код располагаются в одном сегменте. Код и данные должны быть разделены на отдельные сегменты. Код располагается с адреса 300h, а с адреса 0h идёт таблица настроек.

[192-168-0-111:tools anastasiasergienkova\$ hexyl LAB1\_COM.EXE

00000000	4d 5a 1d 01 03 00 00 00	20 00 00 00 ff ff 00 00	MZ●●●000	000xx00
00000010	00 00 f2 fd 00 01 00 00	1e 00 00 00 01 00 00 00	00xx0●00	●000●000
00000020	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00000000	00000000
*				
00000300	e9 0e 02 54 79 70 65 3a	20 50 43 0d 0a 24 54 79	x●●Type:	PC__\$Ty
00000310	70 65 3a 20 50 43 2f 58	54 0d 0a 24 54 79 70 65	pe: PC/X	T__\$Type
00000320	3a 20 41 54 0d 0a 24 54	79 70 65 3a 20 50 53 32	: AT__\$T	ype: PS2
00000330	20 d0 bc d0 be d0 b4 d0	b5 d0 bb d1 8c 20 33 30	xxxxxxx	xxxxx 30
00000340	0d 0a 24 54 79 70 65 3a	20 50 53 32 20 d0 bc d0	__\$Type:	PS2 xxx
00000350	be d0 b4 d0 b5 d0 bb d1	8c 20 35 30 20 d0 b8 d0	xxxxxxx	x 50 xxx
00000360	bb d0 b8 20 36 30 0d 0a	24 54 79 70 65 3a 20 50	xxx 60__	\$Type: P
00000370	53 32 20 d0 bc d0 be d0	b4 d0 b5 d0 bb d1 8c 20	S2 xxxxx	xxxxxxx
00000380	38 30 0d 0a 24 54 79 70	65 3a 20 50 d0 a1 6a 72	80__\$Typ	e: Pxxjr
00000390	0d 0a 24 54 79 70 65 3a	20 50 43 20 43 6f 6e 76	__\$Type:	PC Conv
000003a0	65 72 74 69 62 6c 65 0d	0a 24 45 72 72 6f 72 20	ertible_	__\$Error
000003b0	0d 0a 24 56 65 72 73 69	6f 6e 20 4d 53 2d 44 4f	__\$Versi	on MS-DO
000003c0	53 3a 20 20 2e 20 20 0d	0a 24 53 65 72 69 61 6c	S: . _	__\$Serial
000003d0	20 6e 75 6d 62 65 72 20	4f 45 4d 3a 20 20 0d 0a	number	OEM: __
000003e0	24 55 73 65 72 20 73 65	72 69 61 6c 20 6e 75 6d	\$User se	rial num
000003f0	62 65 72 3a 20 20 20 20	20 20 20 48 20 24 24 0f	ber:	H \$\$●
00000400	3c 09 76 02 04 07 04 30	c3 51 8a e0 e8 ef ff 86	<_v●●●●0	xQxxxxxx
00000410	c4 b1 04 d2 e8 e8 e6 ff	59 c3 53 8a fc e8 e9 ff	xx●xxxxxx	YxSxxxxxx
00000420	88 25 4f 88 05 4f 8a c7	e8 de ff 88 25 4f 88 05	x%Q●x0xx	xxxxx%Q●x
00000430	5b c3 51 52 32 e4 33 d2	b9 0a 00 f7 f1 80 ca 30	[xQR2x3x	x_0xxxxx0
00000440	88 14 4e 33 d2 3d 0a 00	73 f1 3c 00 74 04 0c 30	x●N3x=_0	sx<0t●_0
00000450	88 04 5a 59 c3 b4 09 cd	21 c3 b8 00 f0 8e c0 26	x●ZYxx_x	xx0xxx&
00000460	a0 fe ff 3c ff 75 06 ba	03 01 eb 57 90 3c fe 75	xxx<xu●x	●●xW<xu
00000470	06 ba 0e 01 eb 4d 90 3c	fb 75 06 ba 0e 01 eb 43	●x●●xMx<	xu●x●●xC
00000480	90 3c fc 75 06 ba 1c 01	eb 39 90 3c fa 75 06 ba	x<xu●x●●	x9x<xu●x
00000490	27 01 eb 2f 90 3c fc 75	06 ba 43 01 eb 25 90 3c	'●x/x<xu	●xC●x%<
000004a0	f8 75 06 ba 69 01 eb 1b	90 3c fd 75 06 ba 85 01	xu●xi●x●	x<xu●xx●
000004b0	eb 11 90 3c fd 75 06 ba	93 01 eb 07 90 ba aa 01	●x●<xu●x	●x●●●●●●
000004c0	eb 01 90 e8 8f ff c3 b4	30 cd 21 be b3 01 83 c6	●x●●●●●●	0x xx●xx
000004d0	10 50 e8 5d ff 58 8a c4	83 c6 03 e8 54 ff ba b3	●Px]xXxx	xx●xTxxx
000004e0	01 e8 71 ff be ca 01 83	c6 13 8a c7 e8 43 ff ba	●xq●●●●x	●x●●●xCxx
000004f0	ca 01 e8 60 ff bf e1 01	83 c7 19 8b c1 e8 1a ff	●x●`●●●●	●x●●●●●x
00000500	8a c3 e8 04 ff 83 ef 02	89 05 ba e1 01 e8 45 ff	●●●●●●●●	●x●●●●xE
00000510	c3 e8 46 ff e8 b0 ff 32	c0 b4 4c cd 21	xxF●●●●2	xxLx

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

В EXE-программе код, данные и стек поделены на сегменты. Программа в формате EXE может иметь любой размер. EXE-файл имеет заголовок, который используется при его загрузке. 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. Также в



«плохом» EXE адресация кода начинается с 300h, так как он получается из .COM файла, а при создании «плохого» EXE к этому смещению добавляется размер PSP модуля(200h).

[192-168-0-111:tools anastasiasergienkova\$ hexyl LAB1\_EXE.EXE

00000000	4d 5a 27 01 03 00 01 00	20 00 00 00 ff ff 00 00	MZ'....000xx00
00000010	00 01 24 1b 13 01 20 00	1e 00 00 00 01 00 17 01	0.\$...0...0000...
00000020	20 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00000000 00000000
00000030	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00000000 00000000
*			
00000300	54 79 70 65 3a 20 50 43	0d 0a 24 54 79 70 65 3a	Type: PC __\$Type:
00000310	20 50 43 2f 58 54 0d 0a	24 54 79 70 65 3a 20 41	PC/XT __\$Type: A
00000320	54 0d 0a 24 54 79 70 65	3a 20 50 53 32 20 d0 bc	T__\$Type: PS2 xx
00000330	d0 be d0 b4 d0 b5 d0 bb	d1 8c 20 33 30 0d 0a 24	xxxxxxxx xx 30__\$
00000340	54 79 70 65 3a 20 50 53	32 20 d0 bc d0 be d0 b4	Type: PS 2 xxxxxx
00000350	d0 b5 d0 bb d1 8c 20 35	30 20 d0 b8 d0 bb d0 b8	xxxxxx 5 0 xxxxxx
00000360	20 36 30 0d 0a 24 54 79	70 65 3a 20 50 53 32 20	60__\$Type: PS2
00000370	d0 bc d0 be d0 b4 d0 b5	d0 bb d1 8c 20 38 30 0d	xxxxxxxx xxxx 80__
00000380	0a 24 54 79 70 65 3a 20	50 d0 a1 6a 72 0d 0a 24	__\$Type: Pxxjr__\$
00000390	54 79 70 65 3a 20 50 43	20 43 6f 6e 76 65 72 74	Type: PC Convert
000003a0	69 62 6c 65 0d 0a 24 56	65 72 73 69 6f 6e 20 4d	ible__\$Version M
000003b0	53 2d 44 4f 53 3a 20 20	2e 20 20 0d 0a 24 53 65	S-DOS: . __\$Se
000003c0	72 69 61 6c 20 6e 75 6d	62 65 72 20 4f 45 4d 3a	rial num ber OEM:
000003d0	20 20 0d 0a 24 55 73 65	72 20 73 65 72 69 61 6c	__\$Use r serial
000003e0	20 6e 75 6d 62 65 72 3a	20 20 20 20 20 20 20 48	number: H
000003f0	20 24 45 72 72 6f 72 20	0d 0a 24 00 00 00 00 00	\$Error __\$00000
00000400	24 0f 3c 09 76 02 04 07	04 30 c3 51 8a e0 e8 ef	\$<_v...00xQxxxx
00000410	ff 86 c4 b1 04 d2 e8 e8	e6 ff 59 c3 53 8a fc e8	xxxxxxx xxYxSxxx
00000420	e9 ff 88 25 4f 88 05 4f	8a c7 e8 de ff 88 25 4f	xxx%0x0 xxxxxx%0
00000430	88 05 5b c3 51 52 32 e4	33 d2 b9 0a 00 f7 f1 80	x0[xQR2x 3xx_0xxx
00000440	ca 30 88 14 4e 33 d2 3d	0a 00 73 f1 3c 00 74 04	x0xN3x=_0sx<0t0
00000450	0c 30 88 04 5a 59 c3 b4	09 cd 21 c3 b8 00 f0 8e	_0xZYxx _x!xx0xx
00000460	c0 26 a0 fe ff 3c ff 75	06 ba 00 00 eb 57 90 3c	x&xxx<xu 0x00Wx<
00000470	fe 75 06 ba 0b 00 eb 4d	90 3c fb 75 06 ba 0b 00	xu0x0xM x<xu0x0
00000480	eb 43 90 3c fc 75 06 ba	19 00 eb 39 90 3c fa 75	xC<xu0x 0x9x<xu
00000490	06 ba 24 00 eb 2f 90 3c	fc 75 06 ba 40 00 eb 25	0x\$0x/x< xu0x@0x%
000004a0	90 3c f8 75 06 ba 66 00	eb 1b 90 3c fd 75 06 ba	x<xu0xf0 xxx<xu0x
000004b0	82 00 eb 11 90 3c fd 75	06 ba 90 00 eb 07 90 ba	x0x0x<xu 0xx0xxx
000004c0	f2 00 eb 01 90 e8 8f ff	c3 b4 30 cd 21 50 be a7	x0x0xxxx xx0x!Pxx
000004d0	00 83 c6 10 e8 5d ff 58	8a c4 83 c6 03 e8 54 ff	0xx0x]xX xxxxxTx
000004e0	ba a7 00 e8 71 ff be be	00 83 c6 13 8a c7 e8 43	xx0xqxxx 0xxx0xxxC
000004f0	ff ba be 00 e8 60 ff bf	d5 00 83 c7 19 8b c1 e8	xxx0x`xx x0xxx0xxx
00000500	1a ff 8a c3 e8 04 ff 83	ef 02 89 05 ba d5 00 e8	0xxxx0xxx 0xxxx0x
00000510	45 ff c3 2b c0 50 b8 10	00 8e d8 e8 3e ff e8 a8	Exx+Px0 0xxxx>xxx
00000520	ff 32 c0 b4 4c cd 21		x2xxLx!



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

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

Ищется место в оперативной памяти для COM-модуля. Располагается код начиная с PSP:0100h.

2. Что располагается с адреса 0?

Программный сегмент PSP.

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

Все сегментные регистры (CS, DS, ES и SS) указывают на PSP и имеют значения 48DD.

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

Стек генерируется автоматически. SS – на начало (0h), регистр SP указывает на конец стека (FFFEh). Адреса стека расположены в диапазоне 0h – FFFEh.

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

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

EXE-файл загружается, начиная с адреса PSP:0100h. Далее считывается информация заголовка (PSP) EXE в начале файла и выполняется перемещение адресов сегментов, то есть DS и ES устанавливаются на начало сегмента PSP (DS=ES=48DD), SS (SS=48ED) – на начало сегмента стека, CS (CS=490D) – на начало сегмента команд.

2. На что указывают регистры DS и ES?

Указывают на начало сегмента PSP.

### 3. Как определяется стек?

Стек определяется с помощью директивы `.stack`, после которой задаётся размер стека.

### 4. Как определяется точка входа?

Точка входа определяется при помощи директивы `END`.

### **Выводы.**

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

#### LAB1\_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\_MODEL\_30 db 'Type: PS2 модель 30',0DH,0AH,'\$'

TYPE\_PS2\_MODEL\_50\_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'\$'

TYPE\_PS2\_MODEL\_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,'\$'

ERROR db 'Error ',0dh,0ah,'\$'

VERSIONS db 'Version MS-DOS: . ',0DH,0AH,'\$'

SERIAL\_NUMBER db 'Serial number OEM: ',0DH,0AH,'\$'

USER\_NUMBER db 'User serial number:     H \$'

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 AH,AL
    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
    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
    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

```

```

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

```

```

PC_TYPE PROC near
    mov ax, 0f000h
        mov es, ax
        mov al, es:[0fffeh]

    cmp al, 0ffh
    jne pc_xt1
        mov dx, offset TYPE_PC

```

```

        jmp wtype
pc_xt1:
    cmp al, 0feh
    jne pc_xt2
        mov dx, offset TYPE_PC_XT
        jmp wtype
pc_xt2:
    cmp al, 0fbh
    jne pc_at
        mov dx, offset TYPE_PC_XT
        jmp wtype
pc_at:
    cmp al, 0fch
    jne pc_ps2_model_30
        mov dx, offset TYPE_AT
        jmp wtype
pc_ps2_model_30:
    cmp al, 0fah
    jne pc_ps2_model_50_60
        mov dx, offset TYPE_PS2_MODEL_30
        jmp wtype
pc_ps2_model_50_60:
    cmp al, 0fch
    jne pc_ps2_model_80
        mov dx, offset TYPE_PS2_MODEL_50_60
        jmp wtype
pc_ps2_model_80:
    cmp al, 0f8h
    jne pc_jr
        mov dx, offset TYPE_PS2_MODEL_80
        jmp wtype
pc_jr:
    cmp al, 0fdh
    jne pc_conv

```



```

        mov dx, offset TYPE_PC_JR
        jmp wtype

pc_conv:
    cmp al, 0fdh
    jne Err

        mov dx, offset TYPE_PC_CONV
        jmp wtype

Err:
    mov dx, offset ERROR
    jmp wtype

```

```

wtype:
    call WSTRING
    ret
PC_TYPE ENDP

```

```

OS_VER PROC near
    mov ah, 30h
    int 21h

    mov si, offset VERSIONS
    add si, 16
    push ax
    call BYTE_TO_DEC

    pop ax
    mov al, ah
    add si, 3
    call BYTE_TO_DEC

    mov dx, offset VERSIONS
    call WSTRING

    mov si, offset SERIAL_NUMBER

```

```

add si, 19
mov al, bh
call BYTE_TO_DEC

mov dx, offset SERIAL_NUMBER
call WSTRING

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

mov al, bl
call BYTE_TO_HEX

sub di, 2
mov [di], ax
mov dx, offset USER_NUMBER
call WSTRING
ret

```

OS\_VER ENDP

; Код

BEGIN:

```

call PC_TYPE
call OS_VER

```

```

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

```

TESTPC ENDS

END START

## LAB1\_EXE.ASM

AStack SEGMENT STACK

DW 128 DUP(?)

AStack ENDS

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\_MODEL\_30 db 'Type: PS2 модель 30',0DH,0AH,'\$'

TYPE\_PS2\_MODEL\_50\_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'\$'

TYPE\_PS2\_MODEL\_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,'\$'

VERSIONS db 'Version MS-DOS: . ',0DH,0AH,'\$'

SERIAL\_NUMBER db 'Serial number OEM: ',0DH,0AH,'\$'

USER\_NUMBER db 'User serial number: H \$'

ERROR db 'Error ',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

```
    push CX
    mov AH,AL
    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
    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
    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

```

```

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

```

```

PC_TYPE PROC near
    mov ax, 0f000h
    mov es, ax
    mov al, es:[0fffeh]

    cmp al, 0ffh
    jne pc_xt1
    mov dx, offset TYPE_PC
    jmp wtype

```

```

pc_xt1:
    cmp al, 0feh
    jne pc_xt2
        mov dx, offset TYPE_PC_XT
        jmp wtype
pc_xt2:
    cmp al, 0fbh
    jne pc_at
        mov dx, offset TYPE_PC_XT
        jmp wtype
pc_at:
    cmp al, 0fch
    jne pc_ps2_model_30
        mov dx, offset TYPE_AT
        jmp wtype
pc_ps2_model_30:
    cmp al, 0fah
    jne pc_ps2_model_50_60
        mov dx, offset TYPE_PS2_MODEL_30
        jmp wtype
pc_ps2_model_50_60:
    cmp al, 0fch
    jne pc_ps2_model_80
        mov dx, offset TYPE_PS2_MODEL_50_60
        jmp wtype
pc_ps2_model_80:
    cmp al, 0f8h
    jne pc_jr
        mov dx, offset TYPE_PS2_MODEL_80
        jmp wtype
pc_jr:
    cmp al, 0fdh
    jne pc_conv
        mov dx, offset TYPE_PC_JR

```



```

        jmp wtype
pc_conv:
    cmp al, 0fdh
    jne Err
        mov dx, offset TYPE_PC_CONV
        jmp wtype
Err:
    mov dx, offset ERROR
    jmp wtype

```

```

wtype:
    call WSTRING
    ret

```

PC\_TYPE ENDP

```

OS_VER PROC near
    mov ah, 30h
    int 21h
    push ax

    mov si, offset VERSIONS
    add si, 16
    call BYTE_TO_DEC

    pop ax
    mov al, ah
    add si, 3
    call BYTE_TO_DEC

    mov dx, offset VERSIONS
    call WSTRING

    mov si, offset SERIAL_NUMBER

```

```

    add si, 19
    mov al, bh
    call BYTE_TO_DEC

    mov dx, offset SERIAL_NUMBER
    call WSTRING

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

    mov al, bl
    call BYTE_TO_HEX

    sub di, 2
    mov [di], ax
    mov dx, offset USER_NUMBER
    call WSTRING
    ret

```

OS\_VER ENDP

Main PROC FAR

```

    sub  AX,AX
    push AX
    mov  AX,DATA
    mov  DS,AX
    call PC_TYPE
    call OS_VER
    xor  AL,AL
    mov  AH,4Ch
    int  21H

```

Main ENDP

CODE ENDS  
END Main