

**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. 8383

\_\_\_\_\_

Максимова А.А.

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.

### **Выполнение работы.**

Был написан текст исходного .COM модуля, код которого представлен в приложении А, который определяет тип РС и версию системы. Ассемблерная программа читает содержимое предпоследнего байта ROM BIOS. Полученный код сравнивается с известными значениями, выводится тип РС. В случае, когда код не совпадает ни с одним из известных значений, переводится в символьную строку, содержащую запись шестнадцатеричного числа и выводится на экран. После определяется версия системы: используя функцию 30H прерывания 21H, получаем выходные параметры. Формируются соответствующие текстовые строки, выводятся на экран. Строятся "плохой" и "хороший".EXE, код которого представлен в приложении Б. Результаты работы модулей представлены на рис. 1, 2 и 3 соответственно.

Процедуры:

<b>Название:</b>	<b>Предназначение:</b>
TETR_TO_HEX	Процедура перевода тетрады в шестнадцатеричную цифру (символ)
BYTE_TO_HEX	Процедура перевода байта AL в два символа шестнадцатеричного числа
WRD_TO_HEX	Процедура перевода слова в шестнадцатеричную систему счисления
BYTE_TO_DEC	Процедура перевода байта в десятичную систему счисления

PRINTF	Процедура печати
TYPE_PC	Процедура определения и вывода типа PC
CORRECT	Процедура для вывода версии системы в заданном формате хх.уу
VERSION_SYSTEM	Процедура определения и вывода версии системы, серийного номера OEM и номера пользователя

```
C:\>lr1.com
Register value AX = AT
System version: 05.00
Serial number OEM: 00
Serial number user: 000000
C:\>
```

Рисунок 1 - результат работы .com

```
C:\>lr1.exe

      щ>PC

щ>PC

05 00

00
  щ>PC
000000

      щ>PC

C:\>
```

Рисунок 2 - результат работы "плохого" .exe

```
C:\>lr12.exe
Register value AX = AT
System version: 05.00
Serial number OEM: 00
Serial number user: 000000
```

Рисунок 3 - результат работы "хорошего" .exe

## **Ответы на контрольные вопросы.**

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

#### **1. Сколько сегментов должна содержать COM-программа?**

COM программа должна содержать один сегмент - данные, код и стек объединены в одном сегменте, поэтому COM-файл ограничен размером одного сегмента и не превышает 64К.

#### **2. Сколько сегментов должна содержать EXE-программа?**

EXE-программы могут иметь любое число сегментов, минимум 1.

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

Директива `ORG 100h`, устанавливающая значение программного счетчика в `100h` (при загрузке COM файла в память DOS занимает первые 256 байт блоком данных PSP и располагает код программы только после этого блока). Директива `assume`, показывающая соответствие между введенными сегментами программы и сегментными регистрами, обязательна должна быть в тексте Com-программы, состоящей из одного сегмента для привязки сегментных регистров кода (CS) и данных (DS) соответствующему сегменту (TESTPC). Без использования данной директивы программа не будет скомпилирована и будет выведена ошибка "Отсутствует или недоступен CS".

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

Невозможно использовать команды, содержащие `seg <имя сегмента>`, так как адрес сегмента до запуска программы неизвестен. Exe-программа берет адреса из таблицы настроек, в Com-программах таблица настроек отсутствует.

Был запущен FAR, в котором были открыты загрузочные модули файлов .COM и .EXE ("хорошего" и "плохого"). Соответствующие скриншоты представлены на рис. 4, 5 и 6.

0000000000: E9 3E 02 50 43 0D 0A 24	50 43 2F 58 54 0D 0A 24	é>0PC)PC/XT)PC\$
0000000010: 41 54 0D 0A 24 50 53 32	20 6D 6F 64 65 6C 20 33	AT)PC\$PS2 model 3
0000000020: 30 0D 0A 24 50 53 32 20	6D 6F 64 65 6C 20 38 30	0)PC\$PS2 model 80
0000000030: 0D 0A 24 50 43 6A 72 0D	0A 24 50 43 20 43 6F 6E	)PCjr)PC Con
0000000040: 76 65 72 74 69 62 6C 65	0D 0A 24 52 65 67 69 73	vertible)PC\$Regis
0000000050: 74 65 72 20 76 61 6C 75	65 20 41 58 20 3D 20 24	ter value AX = \$
0000000060: 69 6E 64 65 66 69 6E 65	64 2E 0D 0A 24 50 65 6E	undefined.)PC\$Pen
0000000070: 75 6C 74 69 6D 61 74 65	20 62 79 74 65 3A 09 09	ultimate byte:oo
0000000080: 0D 0A 24 53 79 73 74 65	6D 20 76 65 72 73 69 6F	)PC\$System versio
0000000090: 6E 3A 20 20 20 2E 20 20	0D 0A 24 53 65 72 69 61	n: . )PC\$Seria
0000000100: 6C 20 6E 75 6D 62 65 72	20 4F 45 4D 3A 20 20 20	l number OEM:
0000000110: 20 20 0D 0A 24 53 65 72	69 61 6C 20 6E 75 6D 62	)PC\$Serial numb
0000000120: 65 72 20 75 73 65 72 3A	20 20 20 20 20 20 20 20	er user:
0000000130: 20 20 0D 0A 24 24 0F 3C	09 76 02 04 07 04 30 C3	)PC\$PC<ov000000000E0: 51 8A E0 E8 EF FF 86 C4
0000000140: 09 BA 03 01 E8 E5 FF EB	B1 04 D2 E8 E8 E6 FF 59	QŠaëiÿtĀ±00èæÿY
0000000150: 01 E8 D8 FF EB 64 90 3C	25 4F 88 05 4F 8A C7 E8	ĀSŠüèëÿ`%0`0ŠCè
0000000160: FF EB 57 90 3C FC 75 09	C3 51 52 32 E4 33 D2 B9	bÿ`%0`+[ĀQR2ā30¹
0000000170: 90 3C FA 75 09 BA 15 01	14 4E 33 D2 3D 0A 00 73	š ÷ñëĒ0`ĴN30=š s
0000000180: 75 09 BA 24 01 E8 A4 FF	04 5A 59 C3 B4 09 CD 21	ñ< t000`0ZYĀ`oí!
0000000190: 33 01 E8 97 FF EB 23 90	FE FF EB 01 90 3C FF 75	Ā. 0ŽĀ& bÿë00<ÿu
00000001A0: 8A FF EB 16 90 BA 60 01	71 90 3C FE 75 09 BA 08	o0♥0èāÿëq0<buo0
00000001B0: 15 E8 3D FF BA 6D 01 E8	FB 75 09 BA 08 01 E8 CB	0è0ÿëd0<ūuo00èE
00000001C0: 80 C4 30 04 30 5B C3 B4	BA 10 01 E8 BE FF EB 4A	ÿëW0<ūuo00è%ÿëJ
00000001D0: FF BE 83 01 83 C6 11 88	E8 B1 FF EB 3D 90 3C F8	0<ūuo00Š0è±ÿë=0<0
00000001E0: E8 D8 FF 83 C6 04 88 24	EB 30 90 3C FD 75 09 BA	uo0Š0èHÿë00<ÿuo0
00000001F0: FF 33 C0 8A C7 E8 C3 FF	3C F9 75 09 BA 3A 01 E8	30è=ÿë#0<ūuo00:0è
0000000200: 4E 88 04 BA 9B 01 E8 23	E8 81 FF BF 6D 01 83 C7	Šÿë=000`0è0ÿÿm0fC
	72 FF C3 53 B3 10 F6 F3	Šë=ÿ0m0èrÿĀS³0š
	30 CD 21 51 8B C8 E8 EA	€Ā000[Ā`0ĪIQ<Ēèè
	24 4E 88 04 33 C0 8A E5	ÿ%0f0fĀ←`\$N`03ĀSā
	4E 88 04 BA 83 01 E8 3B	è0ÿfĀ0`\$N`00f0è;
	BE 9B 01 83 C6 14 88 24	ÿ3ĀSĈèĀÿ%>0fĀĴ`\$
	FF 59 33 C0 8A C3 E8 AA	N`0000è#ÿÿ3ĀSĀèè

Рисунок 4 - загрузочный модуль .COM

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 3E 02 50 43 0D 0A 24	50 43 2F 58 54 0D 0A 24	é>0PC)PC/XT)PC\$
0000000310: 41 54 0D 0A 24 50 53 32	20 6D 6F 64 65 6C 20 33	AT)PC\$PS2 model 3
0000000320: 30 0D 0A 24 50 53 32 20	6D 6F 64 65 6C 20 38 30	0)PC\$PS2 model 80
0000000330: 0D 0A 24 50 43 6A 72 0D	0A 24 50 43 20 43 6F 6E	)PCjr)PC Con
0000000340: 76 65 72 74 69 62 6C 65	0D 0A 24 52 65 67 69 73	vertible)PC\$Regis
0000000350: 74 65 72 20 76 61 6C 75	65 20 41 58 20 3D 20 24	ter value AX = \$
0000000360: 69 6E 64 65 66 69 6E 65	64 2E 0D 0A 24 50 65 6E	undefined.)PC\$Pen
0000000370: 75 6C 74 69 6D 61 74 65	20 62 79 74 65 3A 09 09	ultimate byte:oo
0000000380: 0D 0A 24 53 79 73 74 65	6D 20 76 65 72 73 69 6F	)PC\$System versio
0000000390: 6E 3A 20 20 20 2E 20 20	0D 0A 24 53 65 72 69 61	n: . )PC\$Seria
00000003A0: 6C 20 6E 75 6D 62 65 72	20 4F 45 4D 3A 20 20 20	l number OEM:
00000003B0: 20 20 0D 0A 24 53 65 72	69 61 6C 20 6E 75 6D 62	)PC\$Serial numb
00000003C0: 65 72 20 75 73 65 72 3A	20 20 20 20 20 20 20 20	er user:
00000003D0: 20 20 0D 0A 24 24 0F 3C	09 76 02 04 07 04 30 C3	)PC\$PC<ov000000000E0: 51 8A E0 E8 EF FF 86 C4
00000003E0: 09 BA 03 01 E8 E5 FF EB	B1 04 D2 E8 E8 E6 FF 59	QŠaëiÿtĀ±00èæÿY
00000003F0: C3 53 8A FC E8 E9 FF 88	25 4F 88 05 4F 8A C7 E8	ĀSŠüèëÿ`%0`0ŠCè
0000000400: DE FF 88 25 4F 88 05 5B	C3 51 52 32 E4 33 D2 B9	bÿ`%0`+[ĀQR2ā30¹
0000000410: 0A 00 F7 F1 80 CA 30 88	14 4E 33 D2 3D 0A 00 73	š ÷ñëĒ0`ĴN30=š s
0000000420: F1 3C 00 74 04 0C 30 88	04 5A 59 C3 B4 09 CD 21	ñ< t000`0ZYĀ`oí!
0000000430: C3 B8 00 F0 8E C0 26 A0	FE FF EB 01 90 3C FF 75	Ā. 0ŽĀ& bÿë00<ÿu
0000000440: 09 BA 03 01 E8 E5 FF EB	71 90 3C FE 75 09 BA 08	o0♥0èāÿëq0<buo0

Рисунок 5 - загрузочный модуль плохого .EXE



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 01 00 00 00 00 00 00	00 00 00 00 00 00 00 00	@
0000000210:	50 43 0D 0A 24 50 43 2F	58 54 0D 0A 24 41 54 0D	PC.\$PC/XT.\$AT
0000000220:	0A 24 50 53 32 20 6D 6F	64 65 6C 20 33 30 0D 0A	PS2 model 30
0000000230:	24 50 53 32 20 6D 6F 64	65 6C 20 38 30 0D 0A 24	PS2 model 80
0000000240:	50 43 6A 72 0D 0A 24 50	43 20 43 6F 6E 76 65 72	PCjr.\$PC Conver
0000000250:	74 69 62 6C 65 0D 0A 24	52 65 67 69 73 74 65 72	tible.\$Register
0000000260:	20 76 61 6C 75 65 20 41	58 20 3D 20 24 69 6E 64	value AX = \$ind
0000000270:	65 66 69 6E 65 64 2E 0D	0A 24 50 65 6E 75 6C 74	efined.\$Penult
0000000280:	69 6D 61 74 65 20 62 79	74 65 3A 09 09 0D 0A 24	imate byte:oo
0000000290:	53 79 73 74 65 6D 20 76	65 72 73 69 6F 6E 3A 20	System version:
00000002A0:	20 20 2E 20 20 0D 0A 24	53 65 72 69 61 6C 20 6E	. \$Serial n
00000002B0:	75 6D 62 65 72 20 4F 45	4D 3A 2D 20 20 20 20 0D	umber OEM:
00000002C0:	0A 24 53 65 72 69 61 6C	20 6E 75 6D 62 65 72 20	Serial number
00000002D0:	75 73 65 72 3A 20 20 20	20 20 20 20 20 20 20 0D	user:
00000002E0:	0A 24 00 00 00 00 00 00	00 00 00 00 00 00 00 00	\$
00000002F0:	E9 6C 01 24 0F 3C 09 76	02 04 07 04 30 C3 51 8A	él0\$e<ov0♦♦0A05
0000000300:	E0 E8 EF FF 86 C4 B1 04	D2 E8 E8 E6 FF 59 C3 53	æiÿtÄ±0èæÿYÄS
0000000310:	8A FC E8 E9 FF 88 25 4F	88 05 4F 8A C7 E8 DE FF	Šüëéÿ`%0`♦05Càÿÿ
0000000320:	88 25 4F 88 05 5B C3 51	52 32 E4 33 D2 B9 0A 00	`%0`+ [ÄQR2ä30`±
0000000330:	F7 F1 80 CA 30 88 14 4E	33 D2 3D 0A 00 73 F1 3C	÷ñ€É0`9N30= sñ<
0000000340:	00 74 04 0C 30 88 04 5A	59 C3 B4 09 CD 21 C3 B8	t♦90`♦ZYÄ`oíiÄ.
0000000350:	00 F0 8E C0 26 A0 FE FF	EB 01 90 3C FF 75 09 BA	ðŽÄ& þÿë00<ÿuo0
0000000360:	00 00 E8 E5 FF EB 71 90	3C FE 75 09 BA 05 00 E8	èäÿëq0<þuo0+ è
0000000370:	D8 FF EB 64 90 3C FB 75	09 BA 05 00 E8 CB FF EB	øÿëd0<ûuo0+ èÿÿè

Рисунок 6 - загрузочный модуль хорошего .EXE

## Отличия форматов файлов COM и EXE модулей

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

Модуль com-файла содержит данные и код, которые располагаются в одном сегменте, начиная с нулевого адреса. При загрузке происходит смещение в 256 байт (org 100h), генерируется стек, который занимает все оставшуюся память.

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

Модуль плохого exe-файла также содержит данные и код, которые располагаются в одном сегменте, но начиная с трехсотого (в 16 системе счисления) адреса. С 0 адреса до 300h располагаются заголовок - информация для загрузчика, расположенная в начале файла, и таблица настройки адресов, 100h отводится под PSP.

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

Модуль хорошего exe-файла содержит стек, данные и код, расположенные в отдельных сегментах, в отличие от "плохого" EXE. Различается также расположение кода с 200h, так как в "хорошем" EXE нет смещения на 100h.

### Загрузка COM модуля в основную память

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

Во время загрузки COM - программы система выделяет первый свободный сегмент памяти и в его начале (первые 256 байт) размещается PSP. За PSP располагается код программы, то есть с адреса 100h. В IP - счетчик команд, устанавливается значение 100h. В стек записывается адрес 0000h, для возврата по ret.

2. Что располагается с адреса 0?

С адреса 0 располагается PSP, занимающий 256 байт.

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

Все сегментные регистры содержат адрес PSP, то есть значение 48DD.

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

Указатель стека устанавливается на конец сегмента программы (FFFE). Стек, располагающийся с старших адресов, занимает всю доступную память, не зарезервированную PSP и кодом.

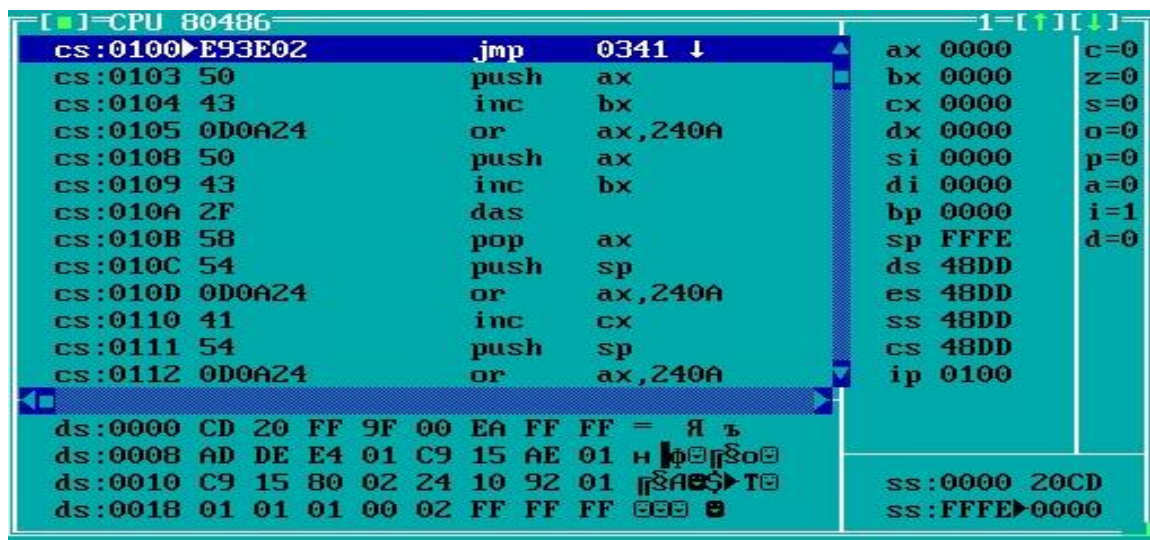


Рисунок 7 - Результат загрузки COM файла в память

## Загрузка "хорошего" EXE модуля в основную память

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

Определяется сегментный адрес свободного участка памяти, размер которого достаточен для размещения программы. DS и ES указывают на начало PSP, CS указывает на начало сегмента кода, SS на начало сегмента стека (значения, указанные в заголовке). Управление передается по адресу, указанному в заголовке.

2. На что указывают регистры DS и ES?

Регистры DS и ES указывают на PSP.

3. Как определяется стек?

Стек определяется с помощью упрощенной директивы `.stack` (или стандартной `SEGMENT STACK ... имя ENDS`), задается размер стека (в соответствии с моделью памяти). SS - указывает на начало сегмента стека (значения, указанные в заголовке), SP - на его смещение.

4. Как определяется точка входа?

Точкой входа в программу может быть метка, объявленная в директиве `END` - директива, которой завершается программа, так как метка не является обязательным параметром этой директивы, то точкой входа, в таком случае, будет начало сегмента кода.

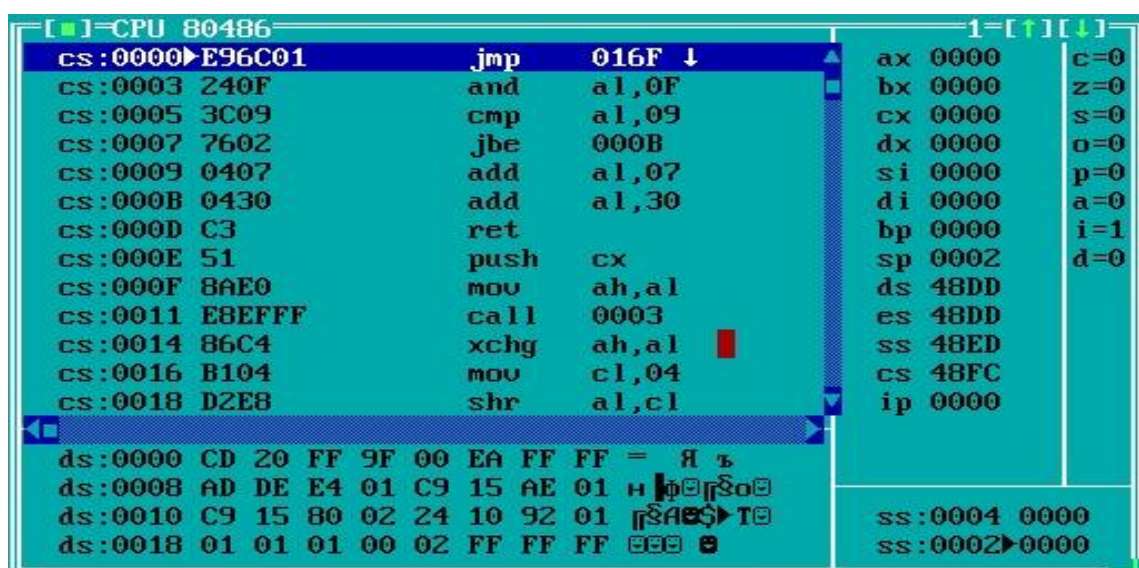


Рисунок 8 - Результат загрузки EXE файла в память



## **Выводы.**

В результате выполнения лабораторной работы были исследованы различия в структурах исходных текстов и модулей COM и EXE файлов и способах их загрузки в основную память.

## ПРИЛОЖЕНИЕ А

COMMENT \*

МАКСИМОВА АНАСТАСИЯ, ГРУППА 8383

\*

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

ORG 100H

START: JMP BEGIN

; ДАННЫЕ

TYPE1 DB 'PC', 0DH, 0AH, '\$'

TYPE2 DB 'PC/XT', 0DH, 0AH, '\$'

TYPE3 DB 'AT', 0DH, 0AH, '\$' ;TYPE5 = TYPE3

TYPE4 DB 'PS2 MODEL 30', 0DH, 0AH, '\$'

TYPE6 DB 'PS2 MODEL 80', 0DH, 0AH, '\$'

TYPE7 DB 'PCJR', 0DH, 0AH, '\$'

TYPE8 DB 'PC CONVERTIBLE', 0DH, 0AH, '\$'

STRING DB 'REGISTER VALUE AX = ', '\$'

VAR0\_STR1 DB 'INDEFINED.', 0DH, 0AH, '\$'

VAR0\_STR2 DB 'PENULTIMATE BYTE: ', 0DH, 0AH, '\$'

SYSTEM\_VERSION DB 'SYSTEM VERSION: ', 0DH, 0AH, '\$'

SERIAL\_NUMB\_OEM DB 'SERIAL NUMBER OEM: ', 0DH, 0AH, '\$'

SERIAL\_NUMB\_USER DB 'SERIAL NUMBER USER: ', 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 ШЕСТН. 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 10 С/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
;-----
PRINTF          PROC  NEAR
                MOV     AH, 09H
                INT     21H
                RET
PRINTF          ENDP
;-----
TYPE_PC        PROC  NEAR
                MOV     AX, 0F000H
                MOV     ES, AX
                MOV     AL,  ES:[0FFFEH]
                JMP     SEARCH_TYPE
SEARCH_TYPE:
                CMP     AL, 0FFH
                JNE     VAR2_1
                MOV     DX, OFFSET TYPE1
                CALL    PRINTF
                JMP     EXIT
VAR2_1:
                CMP     AL, 0FEH
                JNE     VAR2_2
                MOV     DX, OFFSET TYPE2
                CALL    PRINTF
                JMP     EXIT
VAR2_2:
                CMP     AL, 0FBH
                JNE     VAR3
                MOV     DX, OFFSET TYPE2
                CALL    PRINTF
                JMP     EXIT
VAR3:          CMP     AL, 0FCH
                JNE     VAR4
                MOV     DX, OFFSET TYPE3

```



```

                                CALL PRINTF
                                JMP          EXIT

VAR4:                          CMP    AL, 0FAH
                                JNE          VAR6
                                MOV         DX, OFFSET TYPE4
                                CALL PRINTF
                                JMP          EXIT

VAR6:                          CMP    AL, 0F8H
                                JNE          VAR7
                                MOV         DX, OFFSET TYPE6
                                CALL PRINTF
                                JMP          EXIT

VAR7:                          CMP    AL, 0FDH
                                JNE          VAR8
                                MOV         DX, OFFSET TYPE7
                                CALL PRINTF
                                JMP          EXIT

VAR8:                          CMP    AL, 0F9H
                                JNE          VAR0
                                MOV         DX, OFFSET TYPE8
                                CALL PRINTF
                                JMP          EXIT

VAR0:

                                MOV         DX, OFFSET VAR0_STR1
                                CALL PRINTF
                                MOV        DI,  OFFSET VAR0_STR2
                                ADD         DI, 21
                                CALL WRD_TO_HEX
                                MOV         DX, OFFSET VAR0_STR2
                                CALL PRINTF

EXIT:

                                RET

TYPE_PC                        ENDP
; -----
CORRECT                        PROC NEAR

```

```

        PUSH    BX
        MOV     BL, 16          ;BL - ДЕЛИТЕЛЬ
        DIV     BL              ;AH - ОСТАТОК AL - ЦЕЛОЕ
        ADD     AH, '0'
        ADD     AL, '0'
        POP     BX
        RET
CORRECT    ENDP
;-----
VERSION_SYSTEM  PROC  NEAR
        MOV     AH, 30H
        INT     21H

        PUSH    CX
        MOV     CX, AX          ;SAVE

        ;AL
        CALL    CORRECT
        MOV     SI,  OFFSET SYSTEM_VERSION
        ADD     SI, 17
        MOV     [SI], AH

        DEC     SI
        MOV     [SI], AL

        ;AH
        XOR     AX, AX
        MOV     AH, CH
        CALL    CORRECT
        ADD     SI, 4
        MOV     [SI], AH

        DEC     SI
        MOV     [SI], AL

        MOV     DX, OFFSET SYSTEM_VERSION
        CALL    PRINTF

        ;SERIAL_NUMB_OEM
        XOR     AX, AX
        MOV     AL, BH

```

```

CALL    CORRECT
MOV     SI,    OFFSET SERIAL_NUMB_OEM
ADD                     SI, 20
MOV                     [SI], AH

DEC                     SI
MOV                     [SI], AL

MOV                     DX, OFFSET SERIAL_NUMB_OEM
CALL    PRINTF

;SERIAL_NUMB_USER
;BL
POP     CX
XOR     AX, AX
MOV     AL, BL

CALL    CORRECT
MOV     SI,    OFFSET SERIAL_NUMB_USER
ADD                     SI, 21
MOV                     [SI], AH

DEC                     SI
MOV                     [SI], AL

;CH
XOR     AX, AX
MOV     AL, CH

CALL    CORRECT
ADD                     SI, 3
MOV                     [SI], AH

DEC                     SI
MOV                     [SI], AL

;CL
XOR     AX, AX
MOV     AL, CL

CALL    CORRECT
ADD                     SI, 3

```

```

MOV          [SI], AH

DEC          SI
MOV          [SI], AL

MOV          DX, OFFSET SERIAL_NUMB_USER
CALL PRINTF

RET

VERSION_SYSTEM ENDP
; -----
BEGIN:
;ВЫВОД СТРОКИ ТЕКСТА ИЗ ПОЛЯ STRING
MOV          DX, OFFSET STRING
MOV          AH, 09H
INT         21H

;ВЫПОЛНЕНИЕ ЗАДАНИЙ
CALL TYPE_PC
CALL VERSION_SYSTEM

;ВЫХОД В DOS
XOR          AL, AL
MOV          AH, 4CH
INT         21H

TESTPC      ENDS
END        START          ;КОНЕЦ МОДУЛЯ, START - ТОЧКА ВХОДА

```



## ПРИЛОЖЕНИЕ Б

COMMENT \*

МАКСИМОВА АНАСТАСИЯ, ГРУППА 8383

\*

ASTACK SEGMENT STACK

DW 100H DUP(?)

;ОТВОДИТСЯ 100 СЛОВ ПАМЯТИ

ASTACK ENDS

DATA SEGMENT

;ДАННЫЕ

TYPE1 DB 'PC', 0DH, 0AH, '\$'

TYPE2 DB 'PC/XT', 0DH, 0AH, '\$'

TYPE3 DB 'AT', 0DH, 0AH, '\$' ;TYPE5

= TYPE3

TYPE4 DB 'PS2 MODEL 30', 0DH, 0AH, '\$'

TYPE6 DB 'PS2 MODEL 80', 0DH, 0AH, '\$'

TYPE7 DB 'PCJR', 0DH, 0AH, '\$'

TYPE8 DB 'PC CONVERTIBLE', 0DH, 0AH, '\$'

STRING DB 'REGISTER VALUE AX = ', '\$'

VAR0\_STR1 DB 'INDEFINED.', 0DH, 0AH, '\$'

VAR0\_STR2 DB 'PENULTIMATE BYTE: ', 0DH, 0AH, '\$'

SYSTEM\_VERSION DB 'SYSTEM VERSION: ', 0DH, 0AH, '\$'

SERIAL\_NUMB\_OEM DB 'SERIAL NUMBER OEM: ', 0DH, 0AH, '\$'

SERIAL\_NUMB\_USER DB 'SERIAL NUMBER USER: ', 0DH, 0AH, '\$'

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, ES:NOTHING, SS:ASTACK

;START: JMP BEGIN

;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 ШЕСТН. 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 10 С/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
; -----
PRINTF                        PROC NEAR
                                MOV      AH, 09H
                                INT      21H
                                RET
PRINTF                        ENDP
; -----
TYPE_PC                       PROC NEAR
                                MOV      AX, 0F000H
                                MOV      ES, AX
                                MOV      AL, ES:[0FFFEH]
                                JMP      SEARCH_TYPE
SEARCH_TYPE:
                                CMP      AL, 0FFH
                                JNE      VAR2_1
                                MOV      DX, OFFSET TYPE1
                                CALL     PRINTF
                                JMP      EXIT
VAR2_1:
                                CMP      AL, 0FEH
                                JNE      VAR2_2
                                MOV      DX, OFFSET TYPE2
                                CALL     PRINTF
                                JMP      EXIT

```

```

VAR2_2:                CMP    AL, 0FBH
                        JNE     VAR3
                        MOV     DX, OFFSET TYPE2
                        CALL    PRINTF
                        JMP     EXIT

VAR3:                  CMP    AL, 0FCH
                        JNE     VAR4
                        MOV     DX, OFFSET TYPE3
                        CALL    PRINTF
                        JMP     EXIT

VAR4:                  CMP    AL, 0FAH
                        JNE     VAR6
                        MOV     DX, OFFSET TYPE4
                        CALL    PRINTF
                        JMP     EXIT

VAR6:                  CMP    AL, 0F8H
                        JNE     VAR7
                        MOV     DX, OFFSET TYPE6
                        CALL    PRINTF
                        JMP     EXIT

VAR7:                  CMP    AL, 0FDH
                        JNE     VAR8
                        MOV     DX, OFFSET TYPE7
                        CALL    PRINTF
                        JMP     EXIT

VAR8:                  CMP    AL, 0F9H
                        JNE     VAR0
                        MOV     DX, OFFSET TYPE8
                        CALL    PRINTF
                        JMP     EXIT

VAR0:

                        MOV     DX, OFFSET VAR0_STR1
                        CALL    PRINTF
                        MOV     DI,  OFFSET VAR0_STR2
                        ADD     DI, 21

```



```

CALL WRD_TO_HEX
MOV     DX, OFFSET VAR0_STR2
CALL PRINTF

EXIT:

RET

TYPE_PC      ENDP
;-----
CORRECT      PROC NEAR
PUSH BX
MOV BL, 16    ;BL - ДЕЛИТЕЛЬ
DIV     BL    ;AH - ОСТАТОК AL - ЦЕЛОЕ
ADD AH, '0'
ADD AL, '0'
POP BX
RET
CORRECT      ENDP
;-----
VERSION_SYSTEM PROC NEAR
MOV     AH, 30H
INT     21H

PUSH CX
MOV CX, AX    ;SAVE

;AL
CALL CORRECT
MOV SI, OFFSET SYSTEM_VERSION
ADD SI, 17
MOV     [SI], AH

DEC SI
MOV     [SI], AL

;AH
XOR AX, AX
MOV AH, CH
CALL CORRECT
ADD SI, 4
MOV     [SI], AH

DEC SI

```

```

MOV          [SI], AL

MOV          DX, OFFSET SYSTEM_VERSION
CALL PRINTF

; SERIAL_NUMB_OEM
XOR  AX, AX
MOV  AL, BH

CALL CORRECT
MOV  SI,  OFFSET SERIAL_NUMB_OEM
ADD  SI, 20
MOV  [SI], AH

DEC  SI
MOV  [SI], AL

MOV  DX, OFFSET SERIAL_NUMB_OEM
CALL PRINTF

; SERIAL_NUMB_USER
; BL
POP  CX
XOR  AX, AX
MOV  AL, BL

CALL CORRECT
MOV  SI,  OFFSET SERIAL_NUMB_USER
ADD  SI, 21
MOV  [SI], AH

DEC  SI
MOV  [SI], AL

; CH
XOR  AX, AX
MOV  AL, CH

CALL CORRECT
ADD  SI, 3
MOV  [SI], AH

```

```

        DEC         SI
        MOV         [SI], AL

        ;CL
        XOR     AX, AX
        MOV     AL, CL

        CALL     CORRECT
        ADD         SI, 3
        MOV         [SI], AH

        DEC         SI
        MOV         [SI], AL

        MOV         DX, OFFSET SERIAL_NUMB_USER
        CALL     PRINTF

        RET

VERSION_SYSTEM     ENDP
;-----
BEGIN PROC FAR

        PUSH     DS
        PUSH     AX
        XOR     AX, AX
        PUSH     AX

        MOV     AX, DATA
        MOV     DS, AX
        POP     AX

;ВЫВОД СТРОКИ ТЕКСТА ИЗ ПОЛЯ STRING
        MOV         DX, OFFSET STRING
        MOV         AH, 09H
        INT     21H

;ВЫПОЛНЕНИЕ ЗАДАНИЙ
        CALL     TYPE_PC
        CALL     VERSION_SYSTEM

;ВЫХОД В DOS
        XOR         AL, AL
        MOV         AH, 4CH

```

```
                                INT      21H  
BEGIN ENDP  
CODE  ENDS  
      END      BEGIN      ; КОНЕЦ МОДУЛЯ, START - ТОЧКА ВХОДА
```