

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

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

Студент гр. 8381

Преподаватель

Ивлева О.А.

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.

Основные теоретические положения.

Тип IBM PC хранится в байте по адресу 0F000:0FFFFeh, в предпоследнем байте ROM BIOS. Соответствие кода и типа представлены в табл.1.

Таблица 1 – Соответствие кода и типа PC

Тип IBM PC	Код
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

Для определения версии MS DOS следует воспользоваться функцией 30H прерывания 21H. Входным параметром является номер функции в AH. Выходными параметрами являются:

- AL - номер основной версии. Если 0, то < 2.0
- AH - номер модификации
- BH - серийный номер OEM (Original Equipment Manufacturer)
- BL:CH - 24-битовый серийный номер пользователя.

EXE-программа может содержать несколько сегментов. В программе должен содержаться минимум один сегмент данных и один сегмент кода.

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

При запуске COM-программы первые 256 байт необходимо зарезервировать для префикса программного сегмента (PSP). Для этого используется команда `ORG 100h`, которая устанавливает относительный адрес для начала выполнения программы.

Также обязательна директива `ASSUME`, которая устанавливает значения регистров `CS`, `DS` на адрес сегмента программы.

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

Нельзя использовать команды, связанные с адресом сегмента, так как он неизвестен до загрузки этого сегмента в память. Это связано с тем, что в `.COM` файле отсутствует таблица настройки с информацией о типе адресов и их местоположении.

Отличия форматов файлов COM и EXE модулей.

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

Вид файла COM представлен на рис. 5.

D:\OS\ASM\LAB1_C.COM																	
00000000:	E9	10	01	54	79	70	65	20	50	43	3A	20	24	50	43	0D	й→Type PC: \$PC
00000001:	0A	24	50	43	2F	58	54	0D	0A	24	41	54	5F	0D	0A	24	PC/XT)AT_
00000002:	50	53	32	20	6D	6F	64	65	6C	20	33	30	0D	0A	24	50	PS2 model 30)
00000003:	53	32	20	6D	6F	64	65	6C	20	38	30	0D	0A	24	50	43	S2 model 80)
00000004:	6A	72	0D	0A	24	50	43	20	43	6F	6E	76	65	72	74	69	jr)PC Converti
00000005:	62	6C	65	0D	0A	24	45	72	72	6F	72	21	20	54	79	70	ble)Error! Typ
00000006:	65	20	6E	6F	74	20	66	6F	75	6E	64	21	20	59	6F	75	e not found! You
00000007:	72	20	74	79	70	65	3A	20	20	20	0D	0A	24	4F	53		r type:)OS
00000008:	20	76	65	72	73	69	6F	6E	3A	20	30	20	2E	30	20	20	version: 0 .0
00000009:	20	0D	0A	24	4F	45	4D	3A	20	20	20	20	0D	0A	24	53)OEM:)S
0000000A:	65	72	69	61	6C	20	6E	75	6D	62	65	72	3A	20	20	20	erial number:
0000000B:	20	20	20	20	24	50	B4	09	CD	21	58	C3	24	0F	3C	09	\$PrOH!XG\$<o
0000000C:	76	02	04	07	04	30	C3	51	8A	E0	E8	EF	FF	86	C4	B1	v0♦♦0ГQъаипя†д±
0000000D:	04	D2	E8	E8	E6	FF	59	C3	53	8A	FC	E8	E9	FF	88	25	♦ТиижяYGSъыйя€%
0000000E:	4F	88	05	4F	8A	C7	E8	DE	FF	88	25	4F	88	05	5B	C3	О€Оъзиюя€%О€[Г
0000000F:	51	52	32	E4	33	D2	B9	0A	00	F7	F1	80	CA	30	88	14	QR2дЗТNчсѣK0€
00000010:	4E	33	D2	3D	0A	00	73	F1	3C	00	74	04	0C	30	88	04	N3T= sc< t♦90€♦
00000011:	5A	59	C3	B8	00	F0	8E	C0	26	A0	FE	FF	BA	03	01	E8	ZYGë рѣA& юяев♥и
00000012:	93	FF	3C	FF	74	1F	3C	FE	74	21	3C	FB	74	1D	3C	FC	“я<ятv<ют!<ыт+<ь
00000013:	74	1F	3C	FA	74	21	3C	F8	74	23	3C	FD	74	25	3C	F9	tV<ьt!<шт#<эт%<щ
00000014:	74	27	EB	2B	90	BA	0D	01	EB	36	90	BA	12	01	EB	30	t'л+ђе)0л6ђе\$0л0
00000015:	90	BA	1A	01	EB	2A	90	BA	20	01	EB	24	90	BA	2F	01	ђе→0л*ђе 0л\$ђе/0
00000016:	EB	1E	90	BA	3E	01	EB	18	90	BA	45	01	EB	12	90	BF	л▲ђе>0л†ђеE0л†ђi
00000017:	56	01	83	C7	22	E8	4F	FF	89	05	BA	56	01	EB	01	90	V0ѓз"иОя%+ев0л0ђ
00000018:	E8	32	FF	B4	30	CD	21	50	BE	7E	01	83	C6	0D	E8	5F	и2яг0H!Ps~0ѓЖ)и_
00000019:	FF	83	C6	04	58	8A	C4	E8	56	FF	BA	7E	01	E8	15	FF	яѓЖ♦XъДиVяев~0и\$я
0000001A:	BE	94	01	83	C6	06	8A	C7	E8	45	FF	BA	94	01	E8	04	s"0ѓЖ▲ъзиЕяев"0и♦
0000001B:	FF	BF	9F	01	83	C7	14	8B	C1	E8	1C	FF	8A	C3	E8	06	яiц0ѓзђ<Би_яъGi▲
0000001C:	FF	83	EF	02	89	05	BA	9F	01	E8	E9	FE	32	C0	B4	4C	яѓп0%+ев©ийю2AѓL
0000001D:	CD	21															H!

Рисунок 5 – Вид COM файла в шестнадцатеричном виде

COM-файл состоит из одного сегмента, а размер файла не превышает 64 КБ. Код располагается с адреса 0h.

Данные заканчиваются на A0. На B0 начинается сегмент кода, в котором байты отвечают за команды. В строке B0 можно найти коды команд, представленных на рис. 6.

DOSBox 0.74-3, Cpu speed: 3000 cycles, Fram...

AX	F0FC	SI	0000	CS	19F5	IP	01B5	Stack	+0 0222	Flags	7200
BX	0000	DI	0000	DS	19F5				+2 0000		
CX	01D2	BP	0000	ES	F000	HS	19F5		+4 20CD	OF	DF IF SF ZF AF PF CF
DX	0103	SP	FFFC	SS	19F5	FS	19F5		+6 9FFF	0	0 1 0 0 0 0 0 0

CMD >

021F	E893FF	CALL	01B5	DS:0000	CD 20 FF 9F 00 EA F0 FF
01B5	50	PUSH	AX	DS:0008	AD DE 1B 05 C5 06 00 00
01B6	B409	MOV	AH,09	DS:0010	18 01 10 01 18 01 92 01
01B8	CD21	INT	21	DS:0018	01 01 01 00 02 FF FF FF
01BA	58	POP	AX	DS:0020	FF FF FF FF FF FF FF FF
01BB	C3	RET		DS:0028	FF FF FF FF EB 19 C0 11
01BC	240F	AND	AL,0F	DS:0030	A2 01 14 00 18 00 F5 19
01BE	3C09	CMP	AL,09	DS:0038	FF FF FF FF 00 00 00 00
01C0	7602	JNA	01C4	DS:0040	05 00 00 00 00 00 00 00
				DS:0048	00 00 00 00 00 00 00 00

2	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DS:0000	CD	20	FF	9F	00	EA	F0	FE	AD	DE	1B	05	C5	06	00	00
DS:0010	18	01	10	01	18	01	92	01	01	01	01	00	02	FF	FF	FF
DS:0020	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	EB	19	C0	11
DS:0030	A2	01	14	00	18	00	F5	19	FF	FF	FF	FF	00	00	00	00
DS:0040	05	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

1 Step 2ProcStep 3Retrieve 4Help ON 5BRK Menu 6 7 ↑ 8 ↓ 9 ← 10 ⇒

Рисунок 6 – Команды COM модуля

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

Файл «плохого» EXE файла представлен на рис. 7.

D:\OS\ASM\LAB1_C.EXE																
00000000180:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000190:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000001A0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000001B0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000001C0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000001D0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000001E0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000001F0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000200:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000210:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000220:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000230:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000240:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000250:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000260:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000270:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000280:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000290:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000002A0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000002B0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000002C0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000002D0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000002E0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000002F0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000300:	E9	10	01	54	79	70	65	20	50	43	3A	20	24	50	43	0D
00000000310:	0A	24	50	43	2F	58	54	0D	0A	24	41	54	5F	0D	0A	24
00000000320:	50	53	32	20	6D	6F	64	65	6C	20	33	30	0D	0A	24	50
00000000330:	53	32	20	6D	6F	64	65	6C	20	38	30	0D	0A	24	50	43
00000000340:	6A	72	0D	0A	24	50	43	20	43	6F	6E	76	65	72	74	69
00000000350:	62	6C	65	0D	0A	24	45	72	72	6F	72	21	20	54	79	70
00000000360:	65	20	6E	6F	74	20	66	6F	75	6E	64	21	20	59	6F	75
00000000370:	72	20	74	79	70	65	3A	20	20	20	0D	0A	24	4F	53	
00000000380:	20	76	65	72	73	69	6F	6E	3A	20	30	20	2E	30	20	20
00000000390:	20	0D	0A	24	4F	45	4D	3A	20	20	20	20	0D	0A	24	53
000000003A0:	65	72	69	61	6C	20	6E	75	6D	62	65	72	3A	20	20	20
000000003B0:	20	20	20	20	24	50	B4	09	CD	21	58	C3	24	0F	3C	09
000000003C0:	76	02	04	07	04	30	C3	51	8A	E0	E8	EF	FF	86	C4	B1
000000003D0:	04	D2	E8	E8	E6	FF	59	C3	53	8A	FC	E8	E9	FF	88	25
000000003E0:	4F	88	05	4F	8A	C7	E8	DE	FF	88	25	4F	88	05	5B	C3
Type PC: \$PC, \$PC/XT, \$AT, PS2 model 30, S2 model 80, \$PC Convertible, \$Error! Type not found! Your type: \$OS version: 0.0 \$OEM: \$S serial number: \$ProH!XG\$<v0♦♦0GQЪаипя†Д±♦ТиижяYGSъыййя€%O€†Oъзиюя€%O€†Г																

Рисунок 7 – Начало кода «плохого» EXE файла

Код располагается с адреса 300h. С нулевого адреса располагается управляющая информация для загрузчика. В управляющую информацию входит таблица настроек, но также и другая информация. Также 100h резервируются командой ORG 100h.

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

Начало кода в файле «хорошего» EXE представлен на рис. 8.

D:\OS\ASM\LAB1_E.EXE															
0000000028E:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000029E:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000002AE:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000002BE:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000002CE:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000002DE:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000002EE:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000002FE:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000030E:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000031E:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000032E:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000033E:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000034E:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000035E:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000036E:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000037E:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000038E:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000039E:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000003AE:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000003BE:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000003CE:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000003DE:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000003EE:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000003FE:	00	00	54	79	70	65	20	50	43	3A	20	24	50	43	0D 0A
0000000040E:	24	50	43	2F	58	54	0D	0A	24	41	54	5F	0D	0A	24 50
0000000041E:	53	32	20	6D	6F	64	65	6C	20	33	30	0D	0A	24	50 53
0000000042E:	32	20	6D	6F	64	65	6C	20	38	30	0D	0A	24	50	43 6A
0000000043E:	72	0D	0A	24	50	43	20	43	6F	6E	76	65	72	74	69 62
0000000044E:	6C	65	0D	0A	24	45	72	72	6F	72	21	20	54	79	70 65
0000000045E:	20	6E	6F	74	20	66	6F	75	6E	64	21	20	59	6F	75 72
0000000046E:	20	74	79	70	65	3A	20	20	20	20	0D	0A	24	4F	53 20
0000000047E:	76	65	72	73	69	6F	6E	3A	20	30	20	2E	30	20	20 20
0000000048E:	0D	0A	24	4F	45	4D	3A	20	20	20	20	0D	0A	24	53 65
0000000049E:	72	69	61	6C	20	6E	75	6D	62	65	72	3A	20	20	20 20
000000004AE:	20	20	20	24	00	00	00	00	00	00	00	00	00	00	00 00
000000004BE:	00	00	50	B4	09	CD	21	58	C3	24	0F	3C	09	76	02 04
000000004CE:	07	04	30	C3	51	8A	E0	E8	EF	FF	86	C4	B1	04	D2 E8
000000004DE:	E8	E6	FF	59	C3	53	8A	FC	E8	E9	FF	88	25	4F	88 05
000000004EE:	4F	8A	C7	E8	DE	FF	88	25	4F	88	05	5B	C3	51	52 32

```

Type PC: $PC
$PC/XT
S2 model 30
2 model 80
r
le
Error! Type
not found! Your
type:
version: 0 .0
OEM:
Serial number:
$
P
A
Q
S
a
e
i
y
+
A
+
O
e
e
a
y
Y
A
S
S
u
e
e
y
^
%
O
+
O
S
C
e
p
y
^
%
O
+
[
A
Q
R
2

```

Рисунок 8 – Начало кода «хорошего» EXE файла

В «хорошем» EXE с нулевого адреса также располагается управляющая информация для загрузчика. Также перед кодом располагается сегмент стека. Так, при размере стека 200h код располагается с адреса 400h. Отличие от «плохого» EXE в том, что в «хорошем» не резервируется дополнительно 100h, которые в COM файле требовались для PSP.

Загрузка COM модуля в основную память.

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

Запуск файла .COM в отладчике TD.EXE представлен на рис. 9.

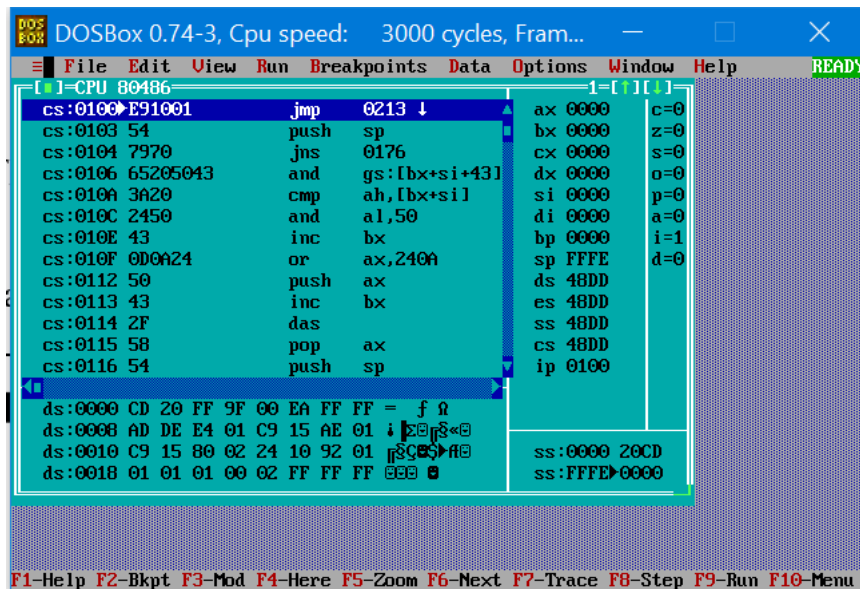


Рисунок 9 – Отладка файла .COM

После загрузки COM-программы в память сегментные регистры указывают на начало PSP. Код располагается с адреса 100h.

2. Что располагается с адреса 0?

С адреса 0 располагается сегмент PSP

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

Сегментные регистры имеют значения 48DDh и указывают на PSP.

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

В COM модуле нельзя объявить стек, он создается автоматически. SP имеет указывает на FFFeh. Стек занимает оставшуюся память, а его адреса изменяются от больших к меньшим, то есть от FFFeh к 0000h.

Загрузка «хорошего» EXE. модуля в основную память.

Запуск «хорошего» EXE модуля в отладчике TD.EXE представлен на рис.

10.

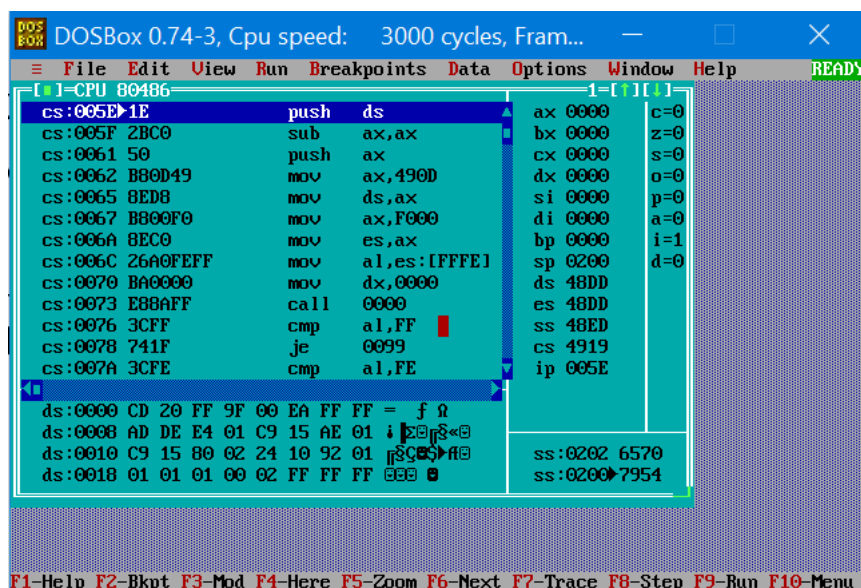


Рисунок 10 – Отладка «хорошего» EXE модуля

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

Для PSP и программы выделяется блок памяти. После запуска программы DS и ES указывают на начало PSP, CS – на начало сегмента команд, а SS – на начало сегмента стека. IP имеет значение отличное от нуля, так как в программе есть дополнительные процедуры, расположенные до основной.

2. На что указывают регистры DS и ES?

Изначально регистры DS и ES указывают на начало сегмента PSP. Именно поэтому в начале программы для корректной работы с данными необходимо загрузить в DS адрес сегмента данных.

3. Как определяется стек?

Стек может быть объявлен при помощи директивы ASSUME, которая устанавливает сегментный регистр SS на начало сегмента стека, а также задает значение SP, указанное в заголовке. Также стек может быть объявлен с помощью директивы STACK. Если стек не объявлять, то он будет создан автоматически.

4. Как определяется точка входа?

Смещение точки входа в программу загружается в указатель команд IP и определяется операндом (функцией или меткой, с которой необходимо начать программу) директивы END.

Выводы.

В ходе выполнения лабораторной работы были изучены COM и EXE модули и их различия. Так же были отлажены и запущены «хорошие» COM и EXE модули и «плохой» EXE.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. LAB1_C.ASM

```

CODE          SEGMENT

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

ORG           100H

START:        JMP     BEGIN

STR           DB "Type PC: $"
PC            DB "PC", 13,10, "$"
PC_XT         DB "PC/XT", 13,10, "$"
AT_           DB "AT_", 13,10, "$"
PS2_30        DB "PS2 model 30", 13,10, "$"
PS2_80        DB "PS2 model 80", 13,10, "$"
PCjr          DB "PCjr", 13,10, "$"
PC_Conv       DB "PC Convertible", 13,10, "$"
ERR           DB "Error! Type not found! Your type:  ", 13,10, "$"
VERS          DB "OS version: 0 .0  ", 13,10, "$"
OEM           DB "OEM:  ", 13,10, "$"
NUM           DB "Serial number:      $"

PRINT        PROC    NEAR

                PUSH    AX
                MOV     AH, 09H
                INT     21H
                POP     AX
                RET

PRINT        ENDP

TETR_TO_HEX  PROC    NEAR

                AND     AL, 0FH
                CMP     AL, 09H
                JBE     NEXT
                ADD     AL, 07H

                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, 4H
        SHR  AL, CL
        CALL TETR_TO_HEX
        POP  CX
        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, 0AH
```

```
LOOP_BD:
```

```
        DIV    CX
```

```
OR      DL, 30H
```

```
MOV     [SI], DL
```

```
        DEC    SI
```

```
XOR     DX, DX
```

```
CMP     AX, 0AH
```

```
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
```

```
BEGIN:
```

```
MOV AX, 0F000H
```

```
MOV    ES, AX
```

```
MOV    AL, ES:[0FFFEH]
```

```
MOV    DX, OFFSET STR
```

```
CALL   PRINT
```

```
CMP     AL, 0FFH
```

```
JE      TYPE1
```

```
CMP     AL, 0FEH
```

```
JE      TYPE2
```

```
CMP     AL, 0FBH
```

```
JE      TYPE2
```

```
CMP     AL, 0FCH
```

```
JE      TYPE3
```

```
CMP     AL, 0FAH
```

```
JE      TYPE4
```

```
CMP     AL, 0F8H
```

```
JE      TYPE5
```

```
CMP     AL, 0FDH
```

```
JE      TYPE6
```

```
CMP     AL, 0F9H
```

```
JE      TYPE7
```

```

        JMP     ELS

TYPE1:
        MOV     DX, OFFSET PC
        JMP     RES

TYPE2:
        MOV     DX, OFFSET PC_XT
        JMP     RES

TYPE3:
        MOV     DX, OFFSET AT_
        JMP     RES

TYPE4:
        MOV     DX, OFFSET PS2_30
        JMP     RES

TYPE5:
        MOV     DX, OFFSET PS2_80
        JMP     RES

TYPE6:
        MOV     DX, OFFSET PCjr
        JMP     RES

TYPE7:
        MOV     DX, OFFSET PC_Conv
        JMP     RES

ELS:
        MOV     DI, OFFSET ERR
        ADD     DI, 34
        CALL    BYTE_TO_HEX
        MOV     [DI], AX
        MOV     DX, OFFSET ERR
        JMP     RES

RES:
        CALL    PRINT

        MOV     AH, 30H
        INT     21H
        PUSH    AX

```

```

MOV     SI, OFFSET VERS
ADD     SI, 13
CALL    BYTE_TO_DEC

ADD     SI, 4
POP     AX
MOV     AL, AH
CALL    BYTE_TO_DEC
MOV     DX, OFFSET VERS
CALL    PRINT

MOV     SI, OFFSET OEM
ADD     SI, 6
MOV     AL, BH
CALL    BYTE_TO_DEC
MOV     DX, OFFSET OEM
CALL    PRINT

MOV     DI, OFFSET NUM
ADD     DI, 20
MOV     AX, CX
CALL    WRD_TO_HEX
MOV     AL, BL
CALL    BYTE_TO_HEX
SUB     DI, 2
MOV     [DI], AX
MOV     DX, OFFSET NUM
CALL    PRINT

XOR     AL, AL
MOV     AH, 4CH
INT     21H

CODE    ENDS
END      START

```


ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. LAB1_E.ASM

```

STACK SEGMENT      STACK

                    DW 100h DUP(0)

STACK              ENDS

DAT_A              SEGMENT

STR                DB "Type PC: $"
PC                 DB "PC", 13,10, "$"
PC_XT              DB "PC/XT", 13,10, "$"
AT_                DB "AT_", 13,10, "$"
PS2_30             DB "PS2 model 30", 13,10, "$"
PS2_80             DB "PS2 model 80", 13,10, "$"
PCjr               DB "PCjr", 13,10, "$"
PC_Conv            DB "PC Convertible", 13,10, "$"
ERR                DB "Error! Type not found! Your type:  ", 13,10, "$"
VERS              DB "OS version: 0 .0  ", 13,10, "$"
OEM                DB "OEM:  ", 13,10, "$"
NUM                DB "Serial number:      $"

DAT_A              ENDS

CODE               SEGMENT

ASSUME             CS:CODE, DS:DAT_A, SS:STACK

PRINT              PROC  NEAR

                    PUSH  AX
                    MOV   AH, 09H
                    INT    21H
                    POP    AX
                    RET

PRINT              ENDP

TETR_TO_HEX        PROC  NEAR

                    AND    AL, 0FH
                    CMP    AL, 09H
                    JBE    NEXT

```

```

        ADD    AL, 07H

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, 4H
    SHR     AL, CL
    CALL    TETR_TO_HEX
    POP     CX
    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, 0AH

LOOP_BD:
    DIV    CX
    OR     DL, 30H
    MOV    [SI], DL
            DEC    SI
    XOR    DX, DX
    CMP    AX, 0AH
    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

MAIN PROC FAR

```
        PUSH  DS
    SUB    AX, AX
    PUSH  AX
    MOV    AX, DAT_A
    MOV    DS, AX

    MOV    AX, 0F000H
    MOV    ES, AX
    MOV    AL, ES:[0FFFEH]

    MOV    DX, OFFSET STR
    CALL  PRINT

    CMP    AL, 0FFH
    JE     TYPE1
```

```

CMP    AL, 0FEH
JE     TYPE2
CMP    AL, 0FBH
JE     TYPE2
CMP    AL, 0FCH
JE     TYPE3
CMP    AL, 0FAH
JE     TYPE4
CMP    AL, 0F8H
JE     TYPE5
CMP    AL, 0FDH
JE     TYPE6
CMP    AL, 0F9H
JE     TYPE7

JMP    ELS

```

TYPE1:

```

MOV    DX, OFFSET PC
JMP    RES

```

TYPE2:

```

MOV    DX, OFFSET PC_XT
JMP    RES

```

TYPE3:

```

MOV    DX, OFFSET AT_
JMP    RES

```

TYPE4:

```

MOV    DX, OFFSET PS2_30
JMP    RES

```

TYPE5:

```

MOV    DX, OFFSET PS2_80
JMP    RES

```

TYPE6:

```

MOV    DX, OFFSET PCjr
JMP    RES

```

TYPE7:

```

MOV    DX, OFFSET PC_Conv
JMP    RES

```

ELS:

```

        MOV     DI, OFFSET ERR
        ADD     DI, 34
        CALL    BYTE_TO_HEX
        MOV     [DI], AX
        MOV     DX, OFFSET ERR
        JMP     RES

RES:

        CALL    PRINT

        MOV     AH, 30H
        INT     21H
        PUSH    AX

MOV     SI, OFFSET VERS
        ADD     SI, 13
        CALL    BYTE_TO_DEC

        ADD     SI, 4H
        POP     AX
        MOV     AL, AH
        CALL    BYTE_TO_DEC
        MOV     DX, OFFSET VERS
        CALL    PRINT

        MOV     SI, OFFSET OEM
        ADD     SI, 6
        MOV     AL, BH
        CALL    BYTE_TO_DEC
        MOV     DX, OFFSET OEM
        CALL    PRINT

        MOV     DI, OFFSET NUM
        ADD     DI, 20
        MOV     AX, CX
        CALL    WRD_TO_HEX
        MOV     AL, BL
        CALL    BYTE_TO_HEX
        SUB     DI, 2
        MOV     [DI], AX
        MOV     DX, OFFSET NUM
        CALL    PRINT

RET

MAIN     ENDP

```

CODE
END

ENDS
MAIN