

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. 7383

Прокопенко Н.

Преподаватель

Губкин А.Ф.

Санкт-Петербург

2019

Постановка задачи.

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

Описание функций:

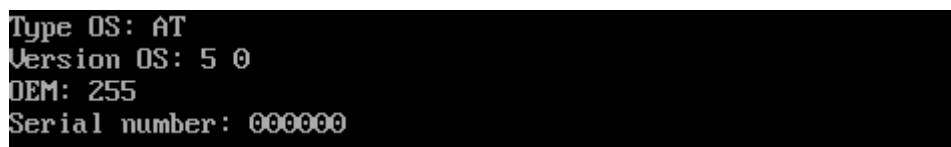
Название функции	Назначение
Write_type	печатает тип ОС
VERSION_OS	печатает версию ОС, серийный номер ОЕМ
Serial_Number	печатает серийный номер пользователя
WRT_MSG	вызывает функцию печати строки
TETR_TO_HEX	вспомогательная функция для работы функции BYTE_TO_HEX
BYTE_TO_HEX	переводит число AL в коды символов 16-ой с/с, записывая получившееся в BL и BH
WRD_TO_HEX	переводит число AX в строку в 16-ой с/с, записывая получившееся в di, начиная с младшей цифры
BYTE_TO_DEC	переводит байт из AL в десятичную с/с и записывает получившееся число по адресу SI, начиная с младшей цифры

Описание структур данных:

Название	Тип	Назначение
OS	db	Тип ОС
OS_VERS	db	Версия ОС
OS_OEM	db	Серийный номер OEM
SER_NUM	db	Серийный номер пользователя
PC	db	PC
PCXT	db	PC/XT
_AT	db	AT
PS2_30	db	PS2 модель 30
PS2_80	db	PS2 модель 80
PCjr	db	PCjr
PC_Cnv	db	PC Convertible

Последовательность действий, выполняемых утилитой:

Программа определяет и выводит на экран следующие значения в заданном порядке: тип ОС, версия ОС, серийный номер OEM, серийный номер пользователя. Результаты работы программы представлены на рисунке 1, рисунке 2, рисунке 3.



```
Type OS: AT
Version OS: 5 0
OEM: 255
Serial number: 000000
```

Рисунок 1 – Результат выполнения программы lab1.com

```

C:\>lab1.exe

Type OS:

Version OS: 5 0

OEM: 255

Serial number: 000000

Type OS:

```

Рисунок 2 – Результат выполнения программы lab1.exe

```

Type OS: AT
Version OS: 5 0
OEM: 255
Serial number: 000000

```

Рисунок 3 – Результат выполнения программы good_exe.exe

Выводы.

В процессе выполнения данной лабораторной работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память. Код программы lab1.asm представлен в приложении А, код программы good_exe.asm представлен в приложении Б.

Ответы на контрольные вопросы.

Отличия исходных текстов COM и EXE программ

1. Сколько сегментов должна содержать COM-программа?

1 сегмент.

2. EXE-программа?

Минимум 1 сегмент.

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

В тексте COM-программы обязательно должна быть директива `ORG 100h`, которая сдвигает адресацию в программе на 256 байт для расположения PSP.

Так же, должна присутствовать директива `ASSUME`, ставящая в соответствие начало программы сегментам кода и данных (при отсутствии директивы `ASSUME`, программа не скомпилируется из-за невозможности обнаружения начала сегмента кода).

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

Нет, в COM-программе нельзя использовать команды вида `mov register, segment` и команды, содержащие дальнюю (`far`) адресацию, т.к. в этих командах используется таблица настройки в которой содержатся адреса сегментов. Такая таблица есть только в EXE-файлах, поэтому COM-программа не может использовать сегментную адресацию.

Отличия форматов файлов COM и EXE модулей

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

HEX-представление .COM файла показано на рисунке 4.

[illegible]

Рисунок 4 – HEX представление .COM файла

СОМ-файл содержит только код и данные. В файле код располагается с нулевого адреса.

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

HEX-представление «плохого» .EXE файла показано на рисунке 5, 6.

```

0000000000: 4D 5A B9 00 03 00 00 00 20 00 00 00 FF FF 00 00 MZ! ♥ yy
0000000010: 00 00 00 00 00 00 01 00 00 3E 00 00 00 01 00 FB 50 0 > 0 uP
0000000020: 6A 72 00 00 00 00 00 00 00 00 00 00 00 00 00 00 jr
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
00000000A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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

```

Рисунок 5 – НЕХ-представление «плохого» .EXE файла(1)

00000002F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	e\$@Type OS: \$Ver
0000000300:	E9 A7 01 54 79 70 65 20	4F 53 3A 20 24 56 65 72	sion OS: . \$
0000000310:	73 69 6F 6E 20 4F 53 3A	20 20 20 2E 20 20 0D 0A	\$OEM: \$Seri
0000000320:	24 4F 45 4D 3A 20 20 20	20 0D 0A 24 53 65 72 69	al number: \$
0000000330:	61 6C 20 6E 75 6D 62 65	72 3A 20 24 20 20 20 20	\$PC/\$PC/XT/
0000000340:	24 0D 0A 24 50 43 0D 0A	24 50 43 2F 58 54 0D 0A	\$AT/\$PS2 model
0000000350:	24 41 54 0D 0A 24 50 53	32 20 6D 6F 64 65 6C 20	30/\$PS2 model 8
0000000360:	33 30 0D 0A 24 50 53 32	20 6D 6F 64 65 6C 20 38	0/\$PCjr/\$PC Co
0000000370:	30 0D 0A 24 50 43 6A 72	0D 0A 24 50 43 20 43 6F	nvertible/\$'oi!
0000000380:	6E 76 65 72 74 69 62 6C	65 0D 0A 24 B4 09 CD 21	Ã\$<ov0♦♦0Ã, õŽ
0000000390:	C3 24 0F 3C 09 76 02 04	07 04 30 C3 B8 00 F0 8E	Å&jþÏÀ°♥œàÿéíÿ<
00000003A0:	C0 26 A1 FE FF C3 BA 03	01 E8 E0 FF E8 ED FF 3C	ÿtL<ptA<ût-<ÿtL<
00000003B0:	FF 74 1C 3C FE 74 1E 3C	FB 74 1A 3C FC 74 1C 3C	ÿtA<øt <ÿt"<ÿt\$°
00000003C0:	FA 74 1E 3C F8 74 20 3C	FD 74 22 3C F9 74 24 BA	D0ë%¤°T0ëv¤°Q0ë!
00000003D0:	44 01 EB 25 90 BA 49 01	EB 1F 90 BA 51 01 EB 19	¤°V0ë¤¤°e0ë¤¤°t0
00000003E0:	90 BA 56 01 EB 13 90 BA	65 01 EB 0D 90 BA 74 01	ë•¤°{0ë0¤ë¤ÿÏÏ,
00000003F0:	EB 07 90 BA 7B 01 EB 01	90 E8 90 FF C3 B8 00 00	°0Ï!%¤of49Pey XŠ
0000000400:	B4 30 CD 21 BE 0D 01 83	C6 0C 50 E8 79 00 58 8A	ÏfÏ°p °¤0ëoÿ%!0
0000000410:	C4 83 C6 03 E8 70 00 BA	0D 01 E8 6F FF BE 21 01	fÏ•ŠÇë_ °!0ë^ÿÏ°
0000000420:	83 C6 07 8A C7 E8 5F 00	BA 21 01 E8 5E FF C3 BA	,0ëwÿŠÏÏ\$ <0Š0 °
0000000430:	2C 01 E8 57 FF 8A C3 E8	24 00 8B D8 8A D3 B4 02	Ï!Š<Ï!;!<0fÇ♥<ÏÏ
0000000440:	CD 21 8A D7 CD 21 BF 3C	01 83 C7 03 8B C1 E8 1E	°<0ë5ÿ°A0ë/ÿÏQŠ
0000000450:	00 BA 3C 01 E8 35 FF BA	41 01 E8 2F FF C3 51 8A	æë-ÿtÏt♦0ëëÿÿÿAS
0000000460:	E0 E8 2D FF 86 C4 B1 04	D2 E8 E8 24 FF 59 C3 53	Šÿëëÿ`%0°♦0ŠÇëÿÿ
0000000470:	8A FC E8 E9 FF 88 25 4F	88 05 4F 8A C7 E8 DE FF	`%0 °[ÏQR2ä30°
0000000480:	88 25 4F 88 05 5B C3 51	52 32 E4 33 D2 B9 0A 00	+ñëÏ0°¤N30= sñ<
0000000490:	F7 F1 80 CA 30 88 14 4E	33 D2 3D 0A 00 73 F1 3C	t♦90°ZYÏëÿëÿÿÿ
00000004A0:	00 74 04 0C 30 88 04 5A	59 C3 E8 F9 FE E8 4D FF	ë ÿ2A`LÍ!
00000004B0:	E8 7C FF 32 C0 B4 4C CD	21	

Рисунок 6 – HEX-представление «плохого» .EXE файла(2)

В «плохом» EXE код и данные не разделены по сегментам, а перемешаны (на скриншоте перед данными видно метку перехода E9 AE 01). Код располагается с адреса 300h, т.к. заголовок занимает 200h байт (байты 8 и 9 указывают, сколько параграфов занимает заголовок) и команда ORG 100h «сдвигает» код на дополнительные 100h. С нулевого адреса располагается заголовок. В первых двух байтах можно увидеть символы MZ, означающие, что формат файла – 16-битный и его следует запускать в соответствии со структурой EXE-файлов. За заголовком следует таблица настройки. Если их убрать, то файл будет загружаться в память как COM-файл.

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

HEX-представление «хорошего» .EXE файла показано на рисунке 7, 8.

00000000:	4D 5A C2 01 03 00 01 00	20 00 00 00 FF FF 00 00	MZB0♥ 0	яя
000000010:	00 02 00 00 1E 01 29 00	3E 00 00 00 01 00 FB 50	0 ▲0) >	0 ыP
000000020:	6A 72 00 00 00 00 00 00	00 00 00 00 00 00 00 00	jr	
000000030:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 1F 01		▼0
000000040:	29 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00)	
000000050:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000060:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000070:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000080:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000090:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000B0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000100:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000110:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000120:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000130:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000140:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000150:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000160:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000170:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000180:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000190:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000001A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000001B0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000001C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000001D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		

Рисунок 7 – НЕХ-представление «хорошего» .EXE файла(1)

0000003F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000400:	54 79 70 65 20 4F 53 3A	20 24 56 65 72 73 69 6F	Type OS: \$Versio
000000410:	6E 20 4F 53 3A 20 20 20	2E 20 20 0D 0A 24 4F 45	n OS: . 0E
000000420:	4D 3A 20 20 20 0D 0A	24 53 65 72 69 61 6C 20	M: 0E
000000430:	6E 75 6D 62 65 72 3A 20	24 20 20 20 20 24 0D 0A	number: \$ 0E
000000440:	24 50 43 0D 0A 24 50 43	2F 58 54 0D 0A 24 41 54	\$PC0E\$PC/XT0E\$AT
000000450:	0D 0A 24 50 53 32 20 6D	6F 64 65 6C 20 33 30 0D	0E\$PS2 model 300E
000000460:	0A 24 50 53 32 20 6D 6F	64 65 6C 20 38 30 0D 0A	0E\$PS2 model 800E
000000470:	24 50 43 6A 72 0D 0A 24	50 43 20 43 6F 6E 76 65	\$PCjr0E\$PC Conve
000000480:	72 74 69 62 6C 65 0D 0A	24 00 00 00 00 00 00 00	rtible0E\$
000000490:	B4 09 CD 21 C3 24 0F 3C	09 76 02 04 07 04 30 C3	roH!Г\$<ov0E♦♦0Г
0000004A0:	B8 00 F0 8E C0 26 A1 FE	FF C3 BA 00 00 E8 E0 FF	ё рhA&УюяГе иая
0000004B0:	E8 ED FF 3C FF 74 1C 3C	FE 74 1E 3C FB 74 1A 3C	иня<ятL<ютA<ыт-<
0000004C0:	FC 74 1C 3C FA 74 1E 3C	F8 74 20 3C FD 74 22 3C	ьтL<ьтA<шт <эт"<
0000004D0:	F9 74 24 BA 41 00 EB 25	90 BA 46 00 EB 1F 90 BA	шт\$еА л?еF л?е
0000004E0:	4E 00 EB 19 90 BA 53 00	EB 13 90 BA 62 00 EB 0D	N л?еS л!еeb л?
0000004F0:	90 BA 71 00 EB 07 90 BA	78 00 EB 01 90 E8 90 FF	еяе л•еex л?иия
000000500:	C3 88 00 00 B4 30 CD 21	BE 0A 00 83 C6 0C 50 E8	Гё р0H!sм рЖ9Ри
000000510:	79 00 58 8A C4 83 C6 03	E8 70 00 BA 0A 00 E8 6F	у ХьдгЖ?ир ем ио
000000520:	FF BE 1E 00 83 C6 07 8A	C7 E8 5F 00 BA 1E 00 E8	я\$A рЖ•ьзи_е▲ и
000000530:	5E FF C3 BA 29 00 E8 57	FF 8A C3 E8 24 00 8B D8	^яГе) ииьльГи\$ <ш
000000540:	8A D3 B4 02 CD 21 8A D7	CD 21 BF 39 00 83 C7 03	ьУг0H!льчH!i9 фЗ▼
000000550:	8B C1 E8 1E 00 BA 39 00	E8 35 FF BA 3E 00 E8 2F	<биA_е9 и\$яе> и/
000000560:	FF C3 51 8A E0 E8 2D FF	86 C4 B1 04 D2 E8 E8 24	яГQьbai-яТд±♦Ти\$
000000570:	F7 59 C3 53 8A FC E8 E9	FF 88 25 4F 88 05 4F 8A	яГGSльийя€%O€+0ь
000000580:	C7 E8 DE FF 88 25 4F 88	05 5B C3 51 52 32 E4 33	Эи0я€%O€+ГГQR2д3
000000590:	D2 B9 0A 00 F7 F1 80 CA	30 88 14 4E 33 D2 3D 0A	ТWм чсЪK0€JN3T=м
0000005A0:	00 73 F1 3C 00 74 04 0C	30 88 04 5A 59 C3 B8 20	sc< т♦90€♦ZYГё
0000005B0:	00 8E D8 E8 F4 FE E8 48	FF E8 77 FF 32 C0 B4 4C	ТшифюиНияи2AгL
0000005C0:	CD 21		H!

Рисунок 8 – НЕХ-представление «хорошего» .EXE файла(2)

В отличие от «плохого» EXE, в «хорошем» код, стек и данные выделены в отдельные сегменты. Код программы начинается с 400h, т.к. дополнительно выделено под стек 200 байт (100 слов). Для «хорошего» EXE в директиве org 100h нет необходимости, т.к. загрузчик автоматически расположит программу после PSP.

Результат загрузки COM модуля в основную память представлен на рисунке 9.

CS	Offset	Instruction	Register	Value
cs:0100	E9A701	jmp	02AA	↓
cs:0103	54	push	sp	
cs:0104	7970	jns	0176	
cs:0106	65204F53	and	gs:[bx+531,c]	
cs:010A	3A20	cmp	ah,[bx+si]	
cs:010C	2456	and	al,56	
cs:010E	657273	jb	gs:0184	
cs:0111	696F6E204F	imul	bp,[bx+6E1,4F]	
cs:0116	53	push	bx	
cs:0117	3A20	cmp	ah,[bx+si]	
cs:0119	2020	and	[bx+si],ah	
cs:011B	2E2020	and	cs:[bx+si],ah	
cs:011E	0D0A24	or	ax,240A	

Register	Value
ax	0000
bx	0000
cx	0000
dx	0000
si	0000
di	0000
bp	0000
sp	FFFE
ds	48DD
es	48DD
ss	48DD
cs	48DD
ip	0100

Segment	Address	Value
ds:0000	CD 20 FF 9F 00 EA FF FF	= f 0
ds:0008	AD DE E4 01 C9 15 AE 01	↓ 20 00
ds:0010	C9 15 80 02 24 10 92 01	↓ 20 00
ds:0018	01 01 01 00 02 FF FF FF	00 00

Рисунок 9 – Результат загрузки .COM в основную память

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

Формат загрузки модуля COM:

1. Выделение сегмента памяти для модуля
2. Установка всех сегментных регистров на начало выделенного сегмента памяти
3. Построение в первых 100h байтах памяти PSP
4. Загрузка содержимого COM-файла и присваивание регистру IP значения 100h.
5. Регистр SP устанавливается в конец сегмента

Код начинается с адреса, содержащимся в CS, в нашем случае это 48DD.

2. Что располагается с адреса 0?

С нулевого адреса располагается PSP.

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

Все сегментные регистры (CS, DS, ES, SS) в данном случае равны 48DD и указывают на начало PSP.

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

Стек занимает весь сегмент COM-программы, его начало находится в конце сегмента. SS указывает на начало сегмента, а SP=FFFEh – на его конец. Стек может дойти до кода/данных программы при достаточном количестве элементов.

Адреса расположены в диапазоне 0000h-FFFFh. Стек растет от больших адресов к меньшим.

Результат загрузки «хорошего» EXE модуля в основную память представлен на рисунке 10.

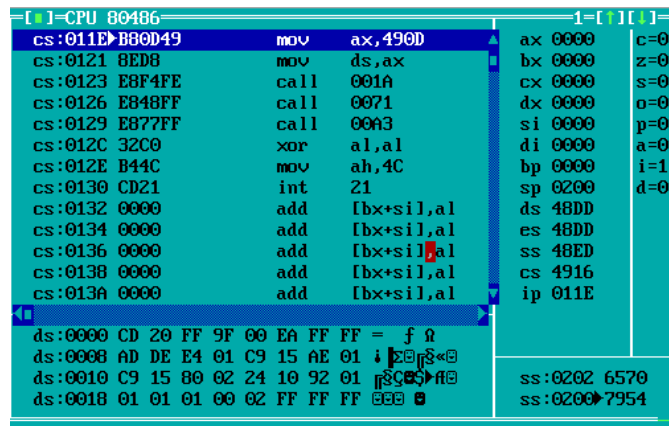


Рисунок 10– Результат загрузки «хорошего» .EXE в основную память

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

SS=48ED – начало сегмента стека, CS=4916 – начало сегмента команд.

2. На что указывают регистры DS и ES?

На начало PSP.

3. Как определяется стек?

В исходном коде модуля стек определяется при помощи директивы STACK, а при исполнении в регистр SS записывается адрес начала сегмента стека, а в SP – его вершины.

4. Как определяется точка входа?

Точка входа в программу определяется с помощью директивы END. После этой директивы указывается метка, куда переходит программа при запуске.

ПРИЛОЖЕНИЕ А

lab1.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN
; ДАННЫЕ
OS db 'Type OS: $'
OS_VERS db 'Version OS: . ',0DH,0AH,'$'
OS_OEM db 'OEM: ',0DH,0AH,'$'
SER_NUM db 'Serial number: ','$'
STRING db ' $'
ENDSTR db 0DH,0AH,'$'

PC db 'PC',0DH,0AH,'$'
PCXT db 'PC/XT',0DH,0AH,'$'
_AT db 'AT',0DH,0AH,'$'
PS2_30 db 'PS2 model 30',0DH,0AH,'$'
PS2_80 db 'PS2 model 80',0DH,0AH,'$'
PCjr db 'PCjr',0DH,0AH,'$'
PC_Cnv db 'PC Convertible',0DH,0AH,'$'

WriteMsg PROC near
    mov AH,09h
    int 21h
    ret
WriteMsg ENDP
;-----
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

System_type PROC near
    mov ax,0F000h
    mov es,ax
    mov ax,es:0FFFEh
    ret
System_type ENDP

Write_type PROC near
```

```

mov dx, OFFSET OS
call WriteMsg
call System_type

; Определяем тип ОС
cmp al,0FFh
je PC_metka
cmp al,0FEh
je PCXT_metka
cmp al,0FBh
je PCXT_metka
cmp al,0FCh
je AT_metka
cmp al,0FAh
je PS2_30_metka
cmp al,0F8h
je PS2_80_metka
cmp al,0FDh
je PCjr_metka
cmp al,0F9h
je PC_Cnv_metka

PC_metka:
    mov dx, OFFSET PC
    jmp konec1
PCXT_metka:
    mov dx, OFFSET PCXT
    jmp konec1
AT_metka:
    mov dx, OFFSET _AT
    jmp konec1
PS2_30_metka:
    mov dx, OFFSET PS2_30
    jmp konec1
PS2_80_metka:
    mov dx, OFFSET PS2_80
    jmp konec1
PCjr_metka:
    mov dx, OFFSET PCjr
    jmp konec1
PC_Cnv_metka:
    mov dx, OFFSET PC_Cnv
    jmp konec1

konec1:
call WriteMsg
ret

```

Write_type ENDP

; Печатает версию системы

VERSION_OS PROC near

; Получаем данные

mov ax,0

mov ah,30h

int 21h

; Пишем в строку OS_VERS номер основной версии ОС

mov si,offset OS_VERS

add si,12

push ax

call BYTE_TO_DEC

; Пишем модификацию ОС

pop ax

mov al,ah

add si,3

call BYTE_TO_DEC

; Пишем версию ОС в консоль

mov dx,offset OS_VERS

call WriteMsg

; Пишем OEM

mov si,offset OS_OEM

add si,7

mov al,bh

call BYTE_TO_DEC

mov dx,offset OS_OEM

call WriteMsg

ret

VERSION_OS ENDP

Serial_Number PROC near

; Пишем серийный номер пользователя

mov dx,offset SER_NUM

call WriteMsg

mov al,bl

call BYTE_TO_HEX

mov bx,ax

mov dl,bl

mov ah,02h

int 21h

mov dl,bh

```

        int 21h
        mov di,offset STRING
        add di,3
        mov ax,cx
        call WRD_TO_HEX
        mov dx,offset STRING
        call WriteMsg

        mov dx,offset ENDSTR
        call WriteMsg

        ret
Serial_Number 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
        pop CX
        ret
BYTE_TO_HEX ENDP
;-----
; перевод в 16с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
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
;-----
; перевод в 10с/с, SI - адрес поля младшей цифры

```

```

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_1
    or AL,30h
    mov [SI],AL
end_1: pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP
;-----
BEGIN:
    call Write_type
    call VERSION_OS
    call Serial_Number
    xor AL,AL
    mov AH,4Ch
    int 21H
TESTPC ENDS
END START

```

ПРИЛОЖЕНИЕ Б

good_exe.asm

```
STACK SEGMENT STACK
    DW 0100H DUP(?)
STACK ENDS

DATA SEGMENT
; ДАННЫЕ
OS db 'Type OS: $'
OS_VERS db 'Version OS:  . ',0DH,0AH,'$'
OS_OEM db 'OEM: ',0DH,0AH,'$'
SER_NUM db 'Serial number: ', '$'
STRING db '    $'
ENDSTR db 0DH,0AH,'$'

PC db 'PC',0DH,0AH,'$'
PCXT db 'PC/XT',0DH,0AH,'$'
_AT db 'AT',0DH,0AH,'$'
PS2_30 db 'PS2 model 30',0DH,0AH,'$'
PS2_80 db 'PS2 model 80',0DH,0AH,'$'
PCjr db 'PCjr',0DH,0AH,'$'
PC_Cnv db 'PC Convertible',0DH,0AH,'$'
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA, ES:NOTHING, SS:STACK
WriteMsg PROC near
    mov AH,09h
    int 21h
    ret
WriteMsg ENDP
;-----
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

System_type PROC near
    mov ax,0F000h
    mov es,ax
    mov ax,es:0FFFEh
    ret
System_type ENDP

Write_type PROC near
    mov dx, OFFSET OS
    call WriteMsg
    call System_type

    ; Определяем тип ОС
    cmp al,0FFh
    je PC_metka
    cmp al,0FEh
    je PCXT_metka
    cmp al,0FBh
    je PCXT_metka
```



```

    cmp al,0FCh
    je AT_metka
    cmp al,0FAh
    je PS2_30_metka
    cmp al,0F8h
    je PS2_80_metka
    cmp al,0FDh
    je PCjr_metka
    cmp al,0F9h
    je PC_Cnv_metka

PC_metka:
    mov dx, OFFSET PC
    jmp konec1
PCXT_metka:
    mov dx, OFFSET PCXT
    jmp konec1
AT_metka:
    mov dx, OFFSET _AT
    jmp konec1
PS2_30_metka:
    mov dx, OFFSET PS2_30
    jmp konec1
PS2_80_metka:
    mov dx, OFFSET PS2_80
    jmp konec1
PCjr_metka:
    mov dx, OFFSET PCjr
    jmp konec1
PC_Cnv_metka:
    mov dx, OFFSET PC_Cnv
    jmp konec1

konec1:
    call WriteMsg
    ret
Write_type ENDP

; Печатает версию системы
VERSION_OS PROC near
    ; Получаем данные
    mov ax,0
    mov ah,30h
    int 21h

    ; Пишем в строку OS_VERS номер основной версии ОС
    mov si,offset OS_VERS
    add si,12
    push ax
    call BYTE_TO_DEC

    ; Пишем модификацию ОС
    pop ax
    mov al,ah
    add si,3
    call BYTE_TO_DEC

    ; Пишем версию ОС в консоль
    mov dx,offset OS_VERS
    call WriteMsg

```

```

        ; Пишем OEM
        mov si,offset OS_OEM
        add si,7
        mov al,bh
        call BYTE_TO_DEC

        mov dx,offset OS_OEM
        call WriteMsg
        ret
VERSION_OS ENDP

Serial_Number PROC near
        ; Пишем серийный номер пользователя
        mov dx,offset SER_NUM
        call WriteMsg
        mov al,bl
        call BYTE_TO_HEX
        mov bx,ax
        mov dl,bl
        mov ah,02h
        int 21h
        mov dl,bh
        int 21h
        mov di,offset STRING
        add di,3
        mov ax,cx
        call WRD_TO_HEX
        mov dx,offset STRING
        call WriteMsg

        mov dx,offset ENDSTR
        call WriteMsg

        ret
Serial_Number 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
        pop CX
        ret
BYTE_TO_HEX ENDP
;-----
; перевод в 16с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
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
;-----
; перевод в 10с/с, SI - адрес поля младшей цифры
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_1
        or AL,30h
        mov [SI],AL
end_1: pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP
;-----
BEGIN:
        mov ax,DATA
        mov ds,ax
        call Write_type
        call VERSION_OS
        call Serial_Number
        xor AL,AL
        mov AH,4Ch
        int 21H
CODE ENDS
END BEGIN

```