

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

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

Студентка гр. 8382

Кузина А.М.

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 из загрузки в основную память.

Ход выполнения работы.

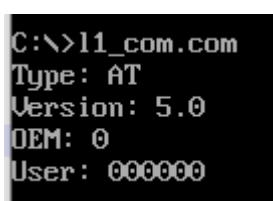
Для определения типа IBM PC и версии системы были написаны .COM и .EXE модули, код которых приведен в приложениях А, В. В таблице 1 указаны соответствия кода и типа PC, который определяется последним байтом ROM BIOS.

Таблица 1 — соответствие кода и типа PC.

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

Версия системы определяется значением регистров AL, AH, BH, BL:CH после выполнения функции 30h прерывания 21h, соответственно номер основной версии, номер модификации, серийный номер OEM и серийный номер пользователя. На рисунках 1 — 3 представлены результаты выполнения .COM модуля, «плохого» .EXE модуля, полученного из исходного кода .COM модуля и «хорошего» .EXE модуля.

Рисунок 1 — результат выполнения .COM модуля.



```
C:\>l1_com.com
Type: AT
Version: 5.0
OEM: 0
User: 000000
```

Рисунок 2 — результат выполнения «плохого» .EXE модуля.

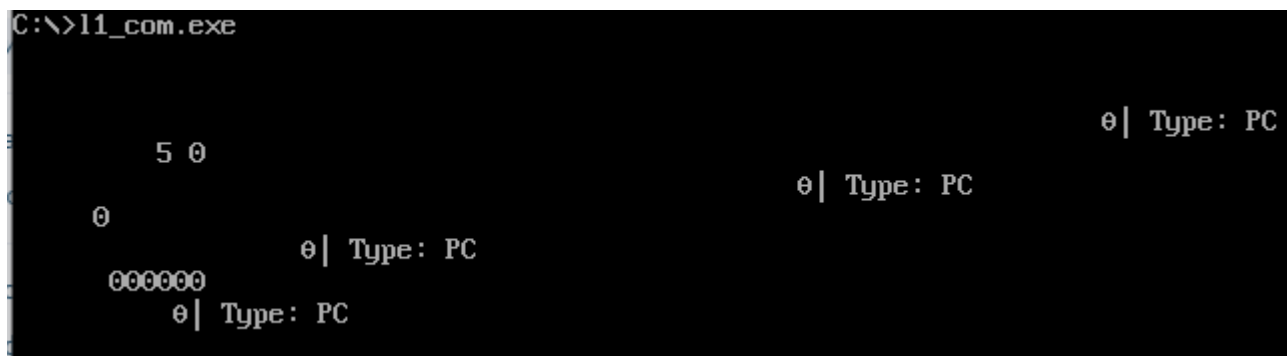
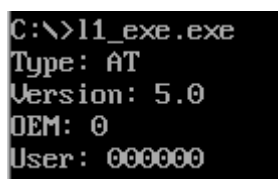


Рисунок 3 — результат выполнения «хорошего» .EXE модуля



Ответы на контрольные вопросы.

Отличия исходных текстов COM и EXE программ.

- 1) Сколько сегментов должна содержать COM-программа?

COM-программа должна содержать всего один сегмент.

- 2) Сколько сегментов должна содержать EXE-программа?

EXE-программа может содержать разное количество сегментов, в зависимости от модели памяти, но не менее одного.

- 3) Какие директивы обязательно должны быть к тексту COM-программы?

Директивы ORG100h и ASSUME. ORG100h выделяет первые 100h байт от начального адреса под PSP, определяя с какого адреса машинный код должен быть помещен в память. ASSUME задает сегменты таким образом, чтобы сегменты данных и кода указывали на один сегмент.

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

В COM-программе нельзя использовать команды, работающие с сегментами т. к. в COM-программе сегментные регистры определяются не при трансляции, а во время запуска программы.

Отличия форматов файлов COM и EXE модулей.

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

COM-файл содержит всего один сегмент, и его размер не превышает 64Кб.

Код программы располагается с адреса 0h, что показано на рисунке 4, но при запуске коду будет предшествовать 100h байт памяти для PSP.

Рисунок 4 — COM-файл в шестнадцатеричном виде.

00000000: E9 B3 00 54 79 70 65 3A	20 50 43 0D 0A 24 54 79	e? Type: PCFS
00000001: 70 65 3A 20 50 43 2F 58	54 0D 0A 24 54 79 70 65	pe: PC/XTFS
00000002: 3A 20 41 54 0D 0A 24 54	79 70 65 3A 20 50 53 32	: ATFS
00000003: 20 6D 6F 64 65 6C 20 33	30 0D 0A 24 54 79 70 65	model 30FS
00000004: 3A 20 50 53 32 20 6D 6F	64 65 6C 20 38 30 0D 0A	: PS2 model 80FS
00000005: 24 54 79 70 65 3A 20 50	43 6A 72 0D 0A 24 54 79	\$Type: PCjrFS
00000006: 70 65 3A 20 50 43 20 43	6F 6E 76 65 72 74 69 62	pe: PC Convertib
00000007: 6C 65 0D 0A 24 20 20 20	20 20 20 20 0D 0A 24 56	leFS
00000008: 65 72 73 69 6F 6E 3A 20	20 2E 20 20 0D 0A 24 56	ersion: . FS
00000009: 65 72 73 69 6F 6E 20 3C	32 2E 30 0D 0A 24 4F 45	ersion <2.0FS
0000000A: 4D 3A 20 0D 0A 24 55 73	65 72 3A 20 0A 20 20 20	M: FSUser: 0
0000000B: 20 20 20 0D 0A 24 B8 00	F0 8E C0 26 A0 FE FF 3C	FS? ?ZA& ?y<
0000000C: FF 74 1C 3C FE 74 1E 3C	FB 74 1A 3C FC 74 1C 3C	yt<?tA<ut><ut<
0000000D: FA 74 1E 3C F8 74 20 3C	FD 74 22 3C F9 74 24 BA	utA<ot <yt"<ut\$?
0000000E: 03 01 EB 25 90 BA 0E 01	EB 1F 90 BA 1C 01 EB 19	♥@e%??FSY??-@e↓
0000000F: 90 BA 27 01 EB 13 90 BA	3C 01 EB 0D 90 BA 51 01	??'@e!!??<@eF??Q@
00000010: EB 07 90 BA 5E 01 EB 01	90 B4 09 CD 21 EB 01 90	e*??^@e@??oi!e@?
00000011: B4 30 CD 21 51 53 B4 30	CD 21 50 3C 00 74 1C BE	?OI!QS?OI!P< t-?
00000012: 7F 01 83 C6 09 E8 92 00	58 8A C4 83 C6 03 E8 89	Δ@??@e' XSA??♥e%
00000013: 00 BA 7F 01 B4 09 CD 21	EB 0C 90 BA 8F 01 B4 09	?Δ@?oi!e???@?@
00000014: CD 21 58 EB 01 90 BE 9E	01 83 C6 05 8A C7 E8 69	I!Xe@??z@??SSCe i
00000015: 00 BA 9E 01 B4 09 CD 21	EB 01 90 BF A6 01 83 C7	?z@?oi!e@?? @?C
00000016: 0B 8B C1 E8 3C 00 8A C3	E8 26 00 83 EF 02 89 05	δ<Ae< S&A& ?i@z&
00000017: BA A6 01 B4 09 CD 21 32	C0 B4 4C CD 21 50 BA 75	? @?oi!2A?LI!P?u
00000018: 01 B4 09 CD 21 58 24 0F	3C 09 76 02 04 07 04 30	@?oi!X\$%<@v@?@
00000019: C3 51 8A E0 E8 EF FF 86	C4 B1 04 D2 E8 E8 E6 FF	AQSAeiy+A+@Oee?y
0000001A: 59 C3 53 8A FC E8 E9 FF	88 25 4F 88 05 4F 8A C7	YASSueey?%O?@OSC
0000001B: E8 DE FF 88 25 4F 88 05	5B C3 51 52 32 E4 33 D2	e?y?%O?@IAQR2a30
0000001C: B9 0A 00 F7 F1 80 CA 30	88 14 4E 33 D2 3D 0A 00	?@ ?n?EO?4N30=@
0000001D: 73 F1 3C 00 74 04 0C 30	88 04 5A 59 C3	sn< t?@0?@ZYA

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

Что располагается с адреса 0?

«Плохой» EXE-файл состоит из одного сегмента, в котором располагаются данные и код, а также не определен сегмент стека, что приводит к неправильной работе файла. На рисунках 5, 6 показано, что код располагается с адреса 300h. С адреса 0h в «плохом» EXE-файле располагается заголовок, а также таблица настроек адресов. MZ в таблице означает, что данный файл является исполняемым.

Рисунок 5 — начало «плохого» EXE-файла в шестнадцатеричном виде.

00000000: 4D 5A EC 01 03 00 01 00	20 00 00 00 FF FF 00 00	MZi
00000010: 00 02 EA 2F 60 00 2C 00	1E 00 00 00 01 00 61 00	Ge/ . ▲ yy a
00000020: 2C 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.
00000030: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000040: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000050: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000060: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000070: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000080: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000090: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000A0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000B0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000C0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000D0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000E0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000F0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000100: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000110: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000120: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000130: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000140: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000150: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000160: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000170: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000180: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	

Рисунок 6 — конец «плохого» EXE-файла в шестнадцатеричном виде.

00000210: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000220: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000230: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000240: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000250: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000260: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000270: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000280: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000290: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000002A0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000002B0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000002C0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000002D0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000002E0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000002F0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000300: E9 B3 00 54 79 70 65 3A	20 50 43 0D 0A 24 54 79	e? Type: PCJ\$Ty
00000310: 70 65 3A 20 50 43 2F 58	54 0D 0A 24 54 79 70 65	pe: PC/XTJ\$Type
00000320: 3A 20 41 54 0D 0A 24 54	79 70 65 3A 20 50 53 32	: ATJ\$Type: PS2
00000330: 20 6D 6F 64 65 6C 20 33	30 0D 0A 24 54 79 70 65	model 30J\$Type
00000340: 3A 20 50 53 32 20 6D 6F	64 65 6C 20 38 30 0D 0A	: PS2 model 80J\$
00000350: 24 54 79 70 65 3A 20 50	43 6A 72 0D 0A 24 54 79	\$Type: PCjrJ\$Ty
00000360: 70 65 3A 20 50 43 20 43	6F 6E 76 65 72 74 69 62	pe: PC Convertib
00000370: 6C 65 0D 0A 24 20 20 20	20 20 20 20 0D 0A 24 56	leJ\$U
00000380: 65 72 73 69 6F 6E 3A 20	20 2E 20 20 0D 0A 24 56	ersion: . J\$U
00000390: 65 72 73 69 6F 6E 20 3C	32 2E 30 0D 0A 24 4F 45	ersion <2.0J\$OE
000003A0: 4D 3A 20 0D 0A 24 55 73	65 72 3A 20 0A 20 20 20	M: J\$User: 0
000003B0: 20 20 20 0D 0A 24 B8 00	F0 8E C0 26 A0 FE FF 3C	J\$? ?ZA& ?y<
000003C0: FF 74 1C 3C FE 74 1E 3C	FB 74 1A 3C FC 74 1C 3C	yt<?t▲<ut-><ut<
000003D0: FA 74 1E 3C F8 74 20 3C	FD 74 22 3C F9 74 24 BA	ut▲<ot <yt"<ut\$?
000003E0: 03 01 EB 25 90 BA 0E 01	EB 1F 90 BA 1C 01 EB 19	▼Qe%??JQe▼??LQe↓
000003F0: 90 BA 27 01 EB 13 90 BA	3C 01 EB 0D 90 BA 51 01	??'Qe!!??<QeJ??Q@
00000400: EB 07 90 BA 5E 01 EB 01	90 B4 09 CD 21 EB 01 90	e*??^QeQ??oI!eQ?
00000410: B4 30 CD 21 51 53 B4 30	CD 21 50 3C 00 74 1C BE	?oI!QS?oI!P< t-?
00000420: 7F 01 83 C6 09 E8 92 00	58 8A C4 83 C6 03 E8 89	ΔQ??Qe' XSA??♥e%
00000430: 00 BA 7F 01 B4 09 CD 21	EB 0C 90 BA 8F 01 B4 09	?ΔQ?oI!eQ??Q?o
00000440: CD 21 58 EB 01 90 BE 9E	01 83 C6 05 8A C7 E8 69	I!XeQ??zQ??sSci
00000450: 00 BA 9E 01 B4 09 CD 21	EB 01 90 BF A6 01 83 C7	?zQ?oI!eQ?? Q?C
00000460: 0B 8B C1 E8 3C 00 8A C3	E8 26 00 83 EF 02 89 05	δ<Ae< S Ae& ?iQ%&
00000470: BA A6 01 B4 09 CD 21 32	C0 B4 4C CD 21 50 BA 75	? Q?oI!2A?LI!P?u
00000480: 01 B4 09 CD 21 58 24 0F	3C 09 76 02 04 07 04 30	Q?oI!X\$*<OvQ♦♦Q
00000490: C3 51 8A E0 E8 EF FF 86	C4 B1 04 D2 E8 E8 E6 FF	AQSaeiy A+♦Qee?y
000004A0: 59 C3 53 8A FC E8 E9 FF	88 25 4F 88 05 4F 8A C7	YASSueey?%?sOSC
000004B0: E8 DE FF 88 25 4F 88 05	5B C3 51 52 32 E4 33 D2	e?y?%?sIAQR2a30
000004C0: B9 0A 00 F7 F1 80 CA 30	88 14 4E 33 D2 3D 0A 00	?Q ?n?E0?Qn30=Q
000004D0: 73 F1 3C 00 74 04 0C 30	88 04 5A 59 C3	sn< t♦Q0?♦ZYA

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

У «хорошего» EXE-файла присутствуют сегменты кода, данных и стека, в то время как у «плохого» только один общий сегмент. В «плохом» файле код всегда начинается с адреса 300h, в «хорошем» файле вначале также идет заголовок и таблица настроек, но т. к. в нем определен сегмент стека, то адресация начинается с 200h + размер стека, 400h в данном случае. На рисунке 7 представлено расположение «хорошего» EXE-файла в памяти.

Рисунок 7 — «хороший» EXE-файл в шестнадцатеричном виде.

00000000370:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000380:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000390:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000003A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000003B0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000003C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000003D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000003E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000003F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000400:	54 79 70 65 3A 20 50 43	0D 0A 24 54 79 70 65 3A	Type: PCJType:
00000000410:	20 50 43 2F 58 54 0D 0A	24 54 79 70 65 3A 20 41	PC/XTJType: A
00000000420:	54 0D 0A 24 54 79 70 65	3A 20 50 53 32 20 6D 6F	TJType: PS2 mo
00000000430:	64 65 6C 20 33 30 0D 0A	24 54 79 70 65 3A 20 50	del 30JType: P
00000000440:	53 32 20 6D 6F 64 65 6C	20 38 30 0D 0A 24 54 79	S2 model 80JType
00000000450:	70 65 3A 20 50 43 6A 72	0D 0A 24 54 79 70 65 3A	pe: PCjrJType:
00000000460:	20 50 43 20 43 6F 6E 76	65 72 74 69 62 6C 65 0D	PC ConvertibleJ
00000000470:	0A 24 20 20 20 20 20 20	20 0D 0A 24 56 65 72 73	JUsers
00000000480:	69 6F 6E 3A 20 20 2E 20	20 0D 0A 24 56 65 72 73	ion: . JUsers
00000000490:	69 6F 6E 20 3C 32 2E 30	0D 0A 24 4F 45 4D 3A 20	ion <2.0JSEM:
000000004A0:	0D 0A 24 55 73 65 72 3A	20 0A 20 20 20 20 20 20	JUser: 0
000000004B0:	0D 0A 24 00 00 00 00 00	00 00 00 00 00 00 00 00	J
000000004C0:	50 BA 72 00 B4 09 CD 21	58 24 0F 3C 09 76 02 04	P?r ?OI!X\$*<0v0
000000004D0:	07 04 30 C3 51 8A E0 E8	EF FF 86 C4 B1 04 D2 E8	*0AQSAeiy A+0Oe
000000004E0:	E8 E6 FF 59 C3 53 8A FC	E8 E9 FF 88 25 4F 88 05	e?yYASSueey?%0?&
000000004F0:	4F 8A C7 E8 DE FF 88 25	4F 88 05 5B C3 51 52 32	OSCe?y?%0?&[AQ2
00000000500:	E4 33 D2 B9 0A 00 F7 F1	80 CA 30 88 14 4E 33 D2	a30?0 ?n?E0?N30
00000000510:	3D 0A 00 73 F1 3C 00 74	04 0C 30 88 04 5A 59 C3	=0 sn< t+00?0ZYA
00000000520:	B8 20 00 8E D8 B8 00 F0	8E C0 26 A0 FE FF 3C FF	? Z0? ?ZA& ?y<y
00000000530:	74 1C 3C FE 74 1E 3C FB	74 1A 3C FC 74 1C 3C FA	t-<?tA<ut->ut-<u
00000000540:	74 1E 3C F8 74 20 3C FD	74 22 3C F9 74 24 BA 00	tA<ot <yt"Cut\$?
00000000550:	00 EB 25 90 BA 0B 00 EB	1F 90 BA 19 00 EB 19 90	e%??0 eV??0 e↓?
00000000560:	BA 24 00 EB 13 90 BA 39	00 EB 0D 90 BA 4E 00 EB	?\$ e!!??9 eJ??N e
00000000570:	07 90 BA 5B 00 EB 01 90	B4 09 CD 21 EB 01 90 B4	*??I e0??OI!e0??
00000000580:	30 CD 21 51 53 B4 30 CD	21 50 3C 00 74 1C BE 7C	OI!QS?OI!P< t-?!
00000000590:	00 83 C6 09 E8 66 FF 58	8A C4 83 C6 03 E8 5D FF	??DefyXSA???vely
000000005A0:	BA 7C 00 B4 09 CD 21 EB	0C 90 BA 8C 00 B4 09 CD	?! ?OI!e0?? ?OI
000000005B0:	21 58 EB 01 90 BE 9B 00	83 C6 05 8A C7 E8 3D FF	?Xe0??> ??&SCe=y
000000005C0:	BA 9B 00 B4 09 CD 21 EB	01 90 BF A3 00 83 C7 0B	?> ?OI!e0?? ?C0
000000005D0:	8B C1 E8 10 FF 8A C3 E8	FA FE 83 EF 02 89 05 BA	<Ac>ySAeu??i0%&?
000000005E0:	A3 00 B4 09 CD 21 32 C0	B4 4C CD 21	? ?OI!2A?LI!

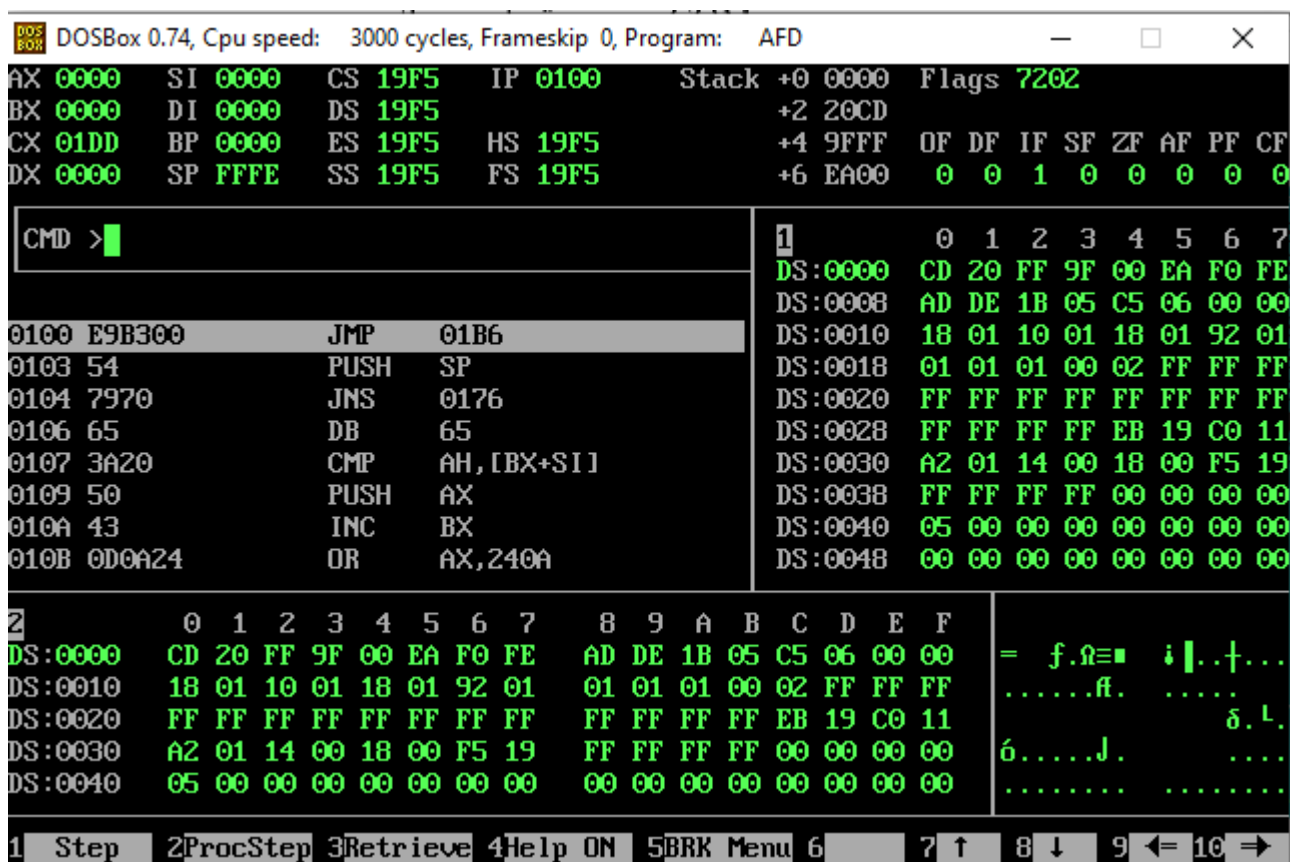
Загрузка COM модуля в основную память.

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

Сначала выделяется свободный сегмент памяти под программу, первые 100h байт выделенной памяти отводятся под PSP, сразу после него в память

загружается файл COM. На рисунке 8 показан COM-файл запущенный в отладчике AFD.

Рисунок 8 — Запуск COM-программы в отладчике.



2) Что располагается с адреса 0?

Начиная с адреса 0 располагается 100h байт PSP.

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

Сегментные регистры — CS, DS, ES, SS – имеют значение 19F5, что показано на рисунке 8, и указывают на начало выделенной памяти — PSP.

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

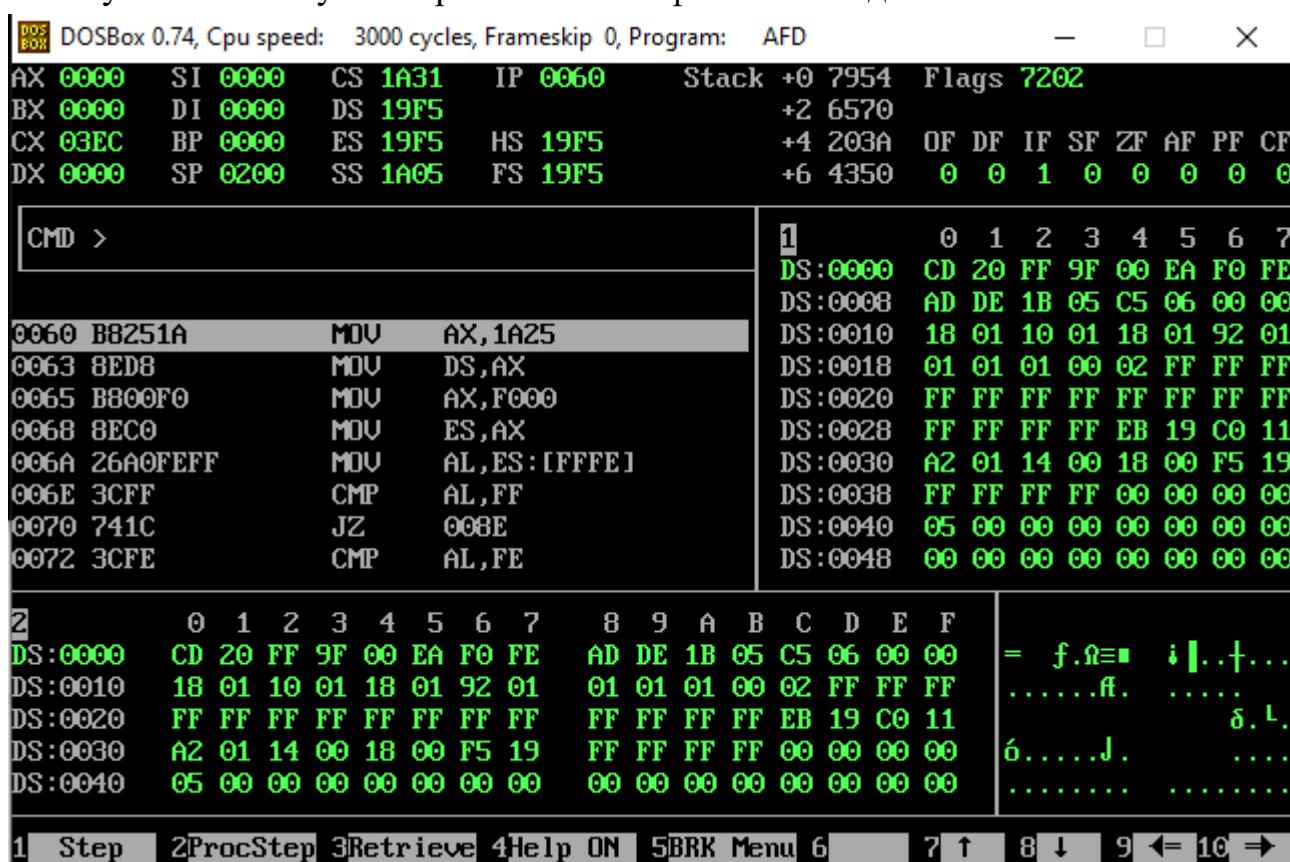
В COM-программе стек генерируется автоматически. Он располагается в выделенной памяти сразу за содержимым COM-файла. Адреса стека могут находиться в памяти от начала выделенного сегмента памяти, до адреса— FFFE, хранящегося в регистре SP , он указывает на последнюю доступную ячейку памяти в сегменте.

Загрузка «хорошего» EXE модуля в основную память.

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

Сначала выделяется свободный сегмент памяти под программу, вначале загружается PSP, затем сама программа. Регистры DS, ES указывают на начало PSP, SS – на начало сегмента стека, CS – на начало сегмента команд. Также IP будет указывать на первую к выполнению команду. На рисунке 9 показан «хороший» EXE-файл запущенный в отладчике AFD.

Рисунок 9 — Запуск «хорошего» EXE-файла в отладчике.



- 2) На что указывают регистры DS и ES?

Данные регистры указывают на начало PSP.

- 3) Как определяется стек?

Имя_сегмента SEGMENT STACK

 DW Количество_выделяемой_памяти DUP(?)

Имя_сегмента ENDS

ASSUME SS:Имя_сегмента

После этого регистр SS будет указывать на начало сегмента стека.

4) Как определяется точка входа?

Точка входа определяется с помощью директивы END — после нее идет название метки или функции с которой будет начато выполнение программы.

Выводы

В ходе выполнения лабораторной работы были написаны два варианта программы, выводящей на экран сведения о типе РС, версии системы, номере OEM и номере пользователя, написанной на ассемблере. Из исходных кодов были получены три различных загрузочных модуля: COM, «хороший» и «плохой» EXE. Были исследованы различия в структурах исходных кодов этих модулей, особенностей работы с каждым из них, структур файлов модулей, а также особенностях их загрузки в память.

ПРИЛОЖЕНИЕ А

Исходный код программы ll_com.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN

typePC db 'Type: PC',0DH,0ah,'$'
typeXT db 'Type: PC/XT',0DH,0ah,'$'
typeAT db 'Type: AT',0DH,0ah,'$'
typePS2_30 db 'Type: PS2 model 30',0DH,0ah,'$'
typePS2_80 db 'Type: PS2 model 80',0DH,0ah,'$'
typePCJR db 'Type: PCjr',0DH,0ah,'$'
typePCConv db 'Type: PC Convertible',0DH,0ah,'$'
Endl db ' ',0DH,0ah,'$'
Version db 'Version: . ',0DH,0ah,'$'
Version2 db 'Version <2.0',0DH,0ah,'$'
OEM db 'OEM: ',0DH,0ah,'$'
UserNumber db 'User: ',0ah,' ',0DH,0ah,'$'

BEGIN:
    mov ax,0F000H
    mov es,ax
    mov al,es:[0FFFEH]

    cmp al,0FFH
        je tPC
    cmp al,0FEH
        je tPC_XT
    cmp al,0Fbh
        je tPC_XT
    cmp al,0FCH
        je tAT
    cmp al,0Fah
        je tPS2_30
    cmp al,0F8H
        je tPS2_80
    cmp al,0FDH
        je tPCjr
    cmp al,0F9H
        je tPCconv

tPC:
    mov dx, offset typePC
    jmp writeType

tPC_XT:
    mov dx, offset typeXT
    jmp writeType

tAT:
    mov dx, offset typeAT
    jmp writeType

tPS2_30:
    mov dx, offset typePS2_30
    jmp writeType
```

```
tPS2_80:
    mov dx, offset typePS2_80
    jmp writeType

tPCjr:
    mov dx, offset typePCJR
    jmp writeType

tPCconv:
    mov dx, offset typePCConv
    jmp writeType

writeType:
    mov ah, 09h
    int 21h
    jmp OSversion

;\\\\"
OSversion:
    mov ah, 30h
    int 21h
    push cx
    push bx

printVer:
    mov ah, 30h
    int 21h
    push ax
    cmp al, 0
        je ver2
    mov si, offset Version
    add si, 9
    call ByteToDec
    pop ax
    mov al, ah
    add si, 3
    call ByteToDec
    mov dx, offset Version
    mov ah, 09h
    int 21h
    jmp OEMnumber

ver2:
    mov dx, offset Version2
    mov ah, 09h
    int 21h
    pop ax
    jmp OEMnumber

OEMnumber:
    mov si, offset OEM
    add si, 5
    mov al, bh
    call ByteToDec
    mov dx, offset OEM
    mov ah, 09h
    int 21h
    jmp User

User:
    mov di, offset UserNumber
    add di, 11
    mov ax, cx
    call WrdToHex
    mov al, bl
    call ByteToHex
```

```
tPCjr:      mov dx, offset typePCJR
            jmp writeType
```

```
tPCconv:
    mov dx, offset typePCConv
    jmp writeType
```

```
writeType:
    mov ah, 09h
    int 21h
    jmp OSversion
```

[illegible]

```
OSversion:
    mov ah, 30h
    int 21h
    push cx
    push bx
```

```
printVer:
    mov ah,30h
    int 21h
    push ax
    cmp al, 0
    je ver2
    mov si,offset Version
    add si,9
    call ByteToDec
    pop ax
    mov al,ah
    add si,3
    call ByteToDec
    mov dx,offset Version
    mov ah,09h
    int 21h
    jmp OEMnumber
```

```
ver2:    mov dx,offset Version2
         mov ah,09h
         int 21h
         pop ax
         jmp OEMnumber
```

```
OEMnumber:
    mov si,offset OEM
    add si,5
    mov al,bh
    call ByteToDec
    mov dx,offset OEM
    mov ah,09h
    int 21h
    jmp User
```

```
User:
    mov di,offset UserNumber
    add di,11
    mov ax,cx
    call WrdToHex
    mov al,bl
    call ByteToHex
```

[illegible]

```

        mov cx,10
bd:      div cx
        or dl,30h
        mov [si],dl
        dec si
        xor dx,dx
        cmp ax,10
            jae bd
        cmp al,00h
            je Endbd
        or al,30h
        mov [si],al
Endbd:   pop dx
        pop cx
        ret
ByteToDec ENDP

TestTPC ENDS
        END START

```

ПРИЛОЖЕНИЕ В

Исходный код программы ll_exe.asm

```
AStack SEGMENT STACK
                                DW 100h DUP(?)
AStack ENDS

DATA SEGMENT
typePC db 'Type: PC',0DH,0ah,'$'
typeXT db 'Type: PC/XT',0DH,0ah,'$'
typeAT db 'Type: AT',0DH,0ah,'$'
typePS2_30 db 'Type: PS2 model 30',0DH,0ah,'$'
typePS2_80 db 'Type: PS2 model 80',0DH,0ah,'$'
typePCJR db 'Type: PCjr',0DH,0ah,'$'
typePCConv db 'Type: PC Convertible',0DH,0ah,'$'
Endl db ' ',0DH,0ah,'$'
Version db 'Version: . ',0DH,0ah,'$'
Version2 db 'Version <2.0',0DH,0ah,'$'
OEM db 'OEM: ',0DH,0ah,'$'
UserNumber db 'User: ',0ah,' ',0DH,0ah,'$'

DATA ENDS

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

PrintEndl PROC near
    push ax
    mov dx, offset Endl
    mov ah, 09h
    int 21h
    pop ax
PrintEndl ENDP

TetrToHex PROC near
    and al,0Fh
    cmp al,09
        jbe next
    add al,07
next:
    add al,30h
    ret
TetrToHex ENDP

ByteToHex PROC near
    push cx
    mov ah,al
    call TetrToHex
    xchg al,ah
    mov cl,4
    shr al,cl
    call TetrToHex
    pop cx
    ret
ByteToHex ENDP

WrdToHex PROC near
    push bx
    mov bh,ah
    call ByteToHex
    mov [di],ah
```

[illegible]

ByteToDec PROC near

bd:

Endbd:

ByteToDec ENDP

;\\ \\

Main PROC FAR

tPC:

tpc_xt:


```

        mov dx, offset typeXT
        jmp writeType
tAT:
        mov dx, offset typeAT
        jmp writeType
tPS2_30:
        mov dx, offset typePS2_30
        jmp writeType
tPS2_80:
        mov dx, offset typePS2_80
        jmp writeType
tPCjr:
        mov dx, offset typePCJR
        jmp writeType
tPCconv:
        mov dx, offset typePCConv
        jmp writeType
writeType:
        mov ah, 09h
        int 21h
        jmp OSversion
;\\\\"
OSversion:
        mov ah, 30h
        int 21h
        push cx
        push bx
printVer:
        mov ah,30h
        int 21h
        push ax
        cmp al, 0
                je ver2
        mov si,offset Version
        add si,9
        call ByteToDec
        pop ax
        mov al,ah
        add si,3
        call ByteToDec
        mov dx,offset Version
        mov ah,09h
        int 21h
        jmp OEMnumber
ver2:
        mov dx,offset Version2
        mov ah,09h
        int 21h
        pop ax
        jmp OEMnumber
OEMnumber:
        mov si,offset OEM
        add si,5
        mov al,bh
        call ByteToDec
        mov dx,offset OEM
        mov ah,09h
        int 21h
        jmp User
User:
        mov di,offset UserNumber

```

