

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 заголовочных модулей”

Студент гр. 7381

Ильясов А.В.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2019

Цель работы.

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

Основные теоретические положения.

Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип PC и выводить строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код переводится в символьную строку, содержащую запись шестнадцатеричного числа и выводится на экран в виде соответствующего сообщения.

Тип IBM PC хранится по адресу 0F000:0FFFEh, в предпоследнем байте ROM BIOS. Соответствие кода и типов в таблице:

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

Таблица 1 – таблица соответствия кода и типа IBM PC

Затем определяется версия системы. Ассемблерная программа должна по значениям регистров AL и AH формировать текстовую строку в формате xx.yy, где xx – номер основной версии, а yy – номер модификации в десятичной системе счисления, формировать строки с серийным номером OEM и серийным номером пользователя. Полученные строки выводятся на экран.

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

```
MOV AH, 30h
```

```
INT 21h
```

Выходными параметрами являются:

AL – номер основной версии. Если 0, то < 2.0

AH – номер модификации

BH – серийный номер OEM

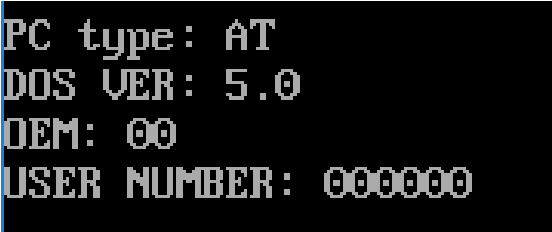
BL:CH – 24-битовый серийный номер пользователя.

Выполнение работы.

Написан текст исходного .COM модуля в соответствии с заданием. Были написаны процедуры получения типа IBM PC, версии системы, OEM и серийного номера пользователя. Также были добавлены процедуры перевода кодов в различные системы счисления из методических указаний. Исходный код программы представлен в приложении А.

Написан текст исходного .EXE модуля с тем же функционалом. Исходный код программы представлен в приложении Б.

Ниже представлены результаты выполнения программы.



```
PC type: AT  
DOS VER: 5.0  
OEM: 00  
USER NUMBER: 000000
```

Рисунок 1 – результат работы .COM модуля

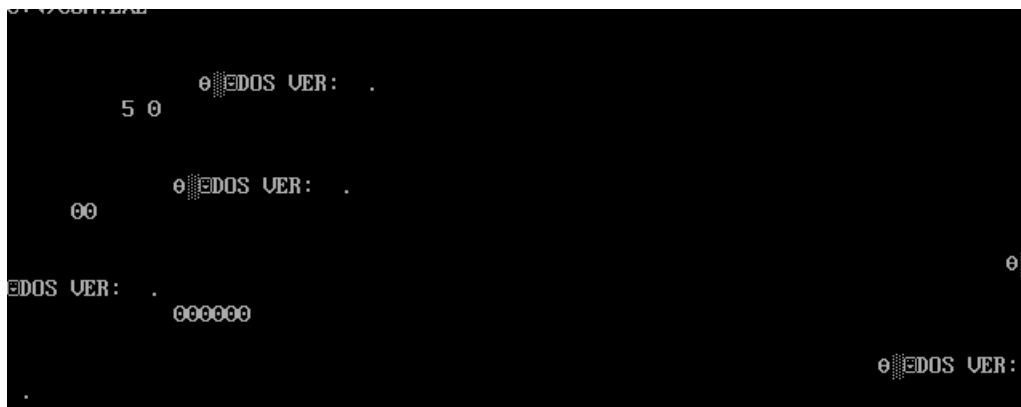


Рисунок 2 – результат работы «плохого» .EXE модуля

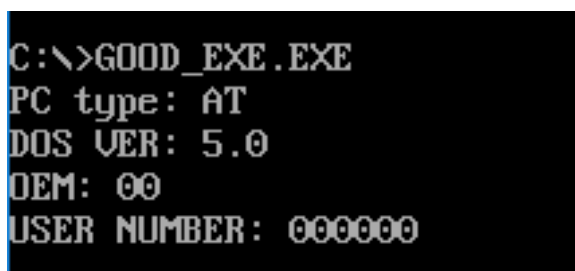


Рисунок 3 – результат работы «хорошего» .EXE модуля

Выводы.

Исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов загрузки в основную память. Реализована программа на языке ассемблера позволяющая определить тип IBM PC и тип системы.

Ответы на контрольные вопросы.

Отличия исходных текстов COM и EXE программ.

1. Сколько сегментов должна содержать COM-программа?

Ответ: один сегмент кода.

2. EXE-программа?

Ответ: «хорошая» .EXE-программа содержит 3 сегмента: сегмент стека, сегмент данных и сегмент кода.

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

Ответ: должна присутствовать директива ORG 100h для выделения памяти под PSP.

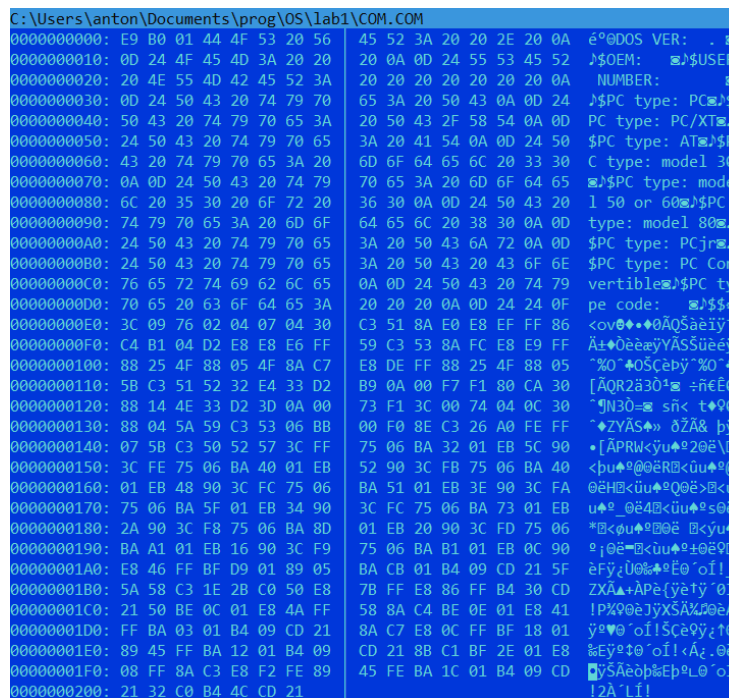
4. Все ли форматы команд можно использовать в COM-программе?

Ответ: т.к. в COM-программе все сегментные регистры определяются в момент запуска программы, а не в момент компиляции, то невозможно использование команды вида `mov <регистр>, seg <имя сегмента>`.

Отличия форматов файлов COM и EXE модулей.

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

Ответ: COM-файл содержит данные и машинные команды. Код начинается с адреса 0h.



Address	Hex Data	Disassembly
00000000	E9 B0 01 44 4F 53 20 56	60000000: JMP 00000001
00000001	0D 24 4F 45 4D 3A 20 20	00000001: JZ 00000004
00000002	20 4E 55 4D 42 45 52 3A	00000002: JZ 00000004
00000003	0D 24 50 43 20 74 79 70	00000003: JZ 00000004
00000004	50 43 20 74 79 70 65 3A	00000004: JZ 00000004
00000005	24 50 43 20 74 79 70 65	00000005: JZ 00000004
00000006	43 20 74 79 70 65 3A 20	00000006: JZ 00000004
00000007	0A 0D 24 50 43 20 74 79	00000007: JZ 00000004
00000008	6C 20 35 30 20 6F 72 20	00000008: JZ 00000004
00000009	74 79 70 65 3A 20 6D 6F	00000009: JZ 00000004
0000000A	24 50 43 20 74 79 70 65	0000000A: JZ 00000004
0000000B	24 50 43 20 74 79 70 65	0000000B: JZ 00000004
0000000C	76 65 72 74 69 62 6C 65	0000000C: JZ 00000004
0000000D	70 65 20 63 6F 64 65 3A	0000000D: JZ 00000004
0000000E	3C 09 76 02 04 07 04 30	0000000E: JZ 00000004
0000000F	C4 B1 04 D2 E8 E8 E6 FF	0000000F: JZ 00000004
00000010	88 25 4F 88 05 4F 8A C7	00000010: JZ 00000004
00000011	5B C3 51 52 32 E4 33 D2	00000011: JZ 00000004
00000012	88 14 4E 33 D2 3D 0A 00	00000012: JZ 00000004
00000013	88 04 5A 59 C3 53 06 BB	00000013: JZ 00000004
00000014	07 58 C3 50 52 57 3C FF	00000014: JZ 00000004
00000015	3C FE 75 06 BA 40 01 EB	00000015: JZ 00000004
00000016	01 EB 48 90 3C FC 75 06	00000016: JZ 00000004
00000017	75 06 BA 5F 01 EB 34 90	00000017: JZ 00000004
00000018	2A 90 3C F8 75 06 BA 8D	00000018: JZ 00000004
00000019	BA A1 01 EB 16 90 3C F9	00000019: JZ 00000004
0000001A	E8 46 FF BF D9 01 89 05	0000001A: JZ 00000004
0000001B	5A 58 C3 1E 2B C0 50 E8	0000001B: JZ 00000004
0000001C	21 50 BE 0C 01 E8 4A FF	0000001C: JZ 00000004
0000001D	FF BA 03 01 B4 09 CD 21	0000001D: JZ 00000004
0000001E	89 45 FF BA 12 01 B4 09	0000001E: JZ 00000004
0000001F	08 FF 8A C3 E8 F2 FE 89	0000001F: JZ 00000004
00000020	21 32 C0 B4 4C CD 21	00000020: JZ 00000004

Рисунок 4 – hex-представление .COM файла

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

Что располагается с адреса 0?

Ответ: в «плохом» файле EXE данные и код содержатся в одном сегменте. Код располагается с адреса 300h, 200h байт занимает заголовок, и еще на 100h сдвигает команда ORG 100h. С адреса 0h идёт таблица настроек.

0000000000:	4D 5A 07 01 03 00 00 00	20 00 00 00 FF FF 00 00	MZ•♥	ÿÿ
0000000010:	00 00 00 00 00 01 00 00	3E 00 00 00 01 00 FB 50	⊙ > ⊙ ůP	
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		
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 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000210:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000220:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000230:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
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 B0 01 44 4F 53 20 56	45 52 3A 20 20 2E 20 0A	é°@DOS VER: . ☒	
0000000310:	0D 24 4F 45 4D 3A 20 20	20 0A 0D 24 55 53 45 52	♪\$OEM: ☒♪\$USER	

Рисунок 5 – hex-представление «плохого» .EXE файла

00000000320:	20 4E 55 4D 42 45 52 3A	20 20 20 20 20 20 20 0A	NUMBER:
00000000330:	0D 24 50 43 20 74 79 70	65 3A 20 50 43 0A 0D 24	PC type: PC
00000000340:	50 43 20 74 79 70 65 3A	20 50 43 2F 58 54 0A 0D	PC type: PC/XT
00000000350:	24 50 43 20 74 79 70 65	3A 20 41 54 0A 0D 24 50	\$PC type: AT
00000000360:	43 20 74 79 70 65 3A 20	6D 6F 64 65 6C 20 33 30	C type: model 30
00000000370:	0A 0D 24 50 43 20 74 79	70 65 3A 20 6D 6F 64 65	PC type: mode
00000000380:	6C 20 35 30 20 6F 72 20	36 30 0A 0D 24 50 43 20	l 50 or 60
00000000390:	74 79 70 65 3A 20 6D 6F	64 65 6C 20 38 30 0A 0D	type: model 80
000000003A0:	24 50 43 20 74 79 70 65	3A 20 50 43 6A 72 0A 0D	\$PC type: PCjr
000000003B0:	24 50 43 20 74 79 70 65	3A 20 50 43 20 43 6F 6E	\$PC type: PC Con
000000003C0:	76 65 72 74 69 62 6C 65	0A 0D 24 50 43 20 74 79	vertible
000000003D0:	70 65 20 63 6F 64 65 3A	20 20 20 0A 0D 24 24 0F	pe code:
000000003E0:	3C 09 76 02 04 07 04 30	C3 51 8A E0 E8 EF FF 86	<ov
000000003F0:	C4 B1 04 D2 E8 E8 E6 FF	59 C3 53 8A FC E8 E9 FF	Ä±
00000000400:	88 25 4F 88 05 4F 8A C7	E8 DE FF 88 25 4F 88 05	%O
00000000410:	5B C3 51 52 32 E4 33 D2	B9 0A 00 F7 F1 80 CA 30	[ÄQR2ä3ð
00000000420:	88 14 4E 33 D2 3D 0A 00	73 F1 3C 00 74 04 0C 30	ñN3ð=s
00000000430:	88 04 5A 59 C3 53 06 BB	00 F0 8E C3 26 A0 FE FF	ñZYÄS»
00000000440:	07 5B C3 50 52 57 3C FF	75 06 BA 32 01 EB 5C 90	•[ÄPRW<ÿu
00000000450:	3C FE 75 06 BA 40 01 EB	52 90 3C FB 75 06 BA 40	<pu
00000000460:	01 EB 48 90 3C FC 75 06	BA 51 01 EB 3E 90 3C FA	ëH
00000000470:	75 06 BA 5F 01 EB 34 90	3C FC 75 06 BA 73 01 EB	u
00000000480:	2A 90 3C F8 75 06 BA 8D	01 EB 20 90 3C FD 75 06	* ë
00000000490:	BA A1 01 EB 16 90 3C F9	75 06 BA B1 01 EB 0C 90	ë
000000004A0:	E8 46 FF BF D9 01 89 05	BA CB 01 B4 09 CD 21 5F	èFÿ
000000004B0:	5A 58 C3 1E 2B C0 50 E8	7B FF E8 86 FF B4 30 CD	ZXA
000000004C0:	21 50 BE 0C 01 E8 4A FF	58 8A C4 BE 0E 01 E8 41	!P%
000000004D0:	FF BA 03 01 B4 09 CD 21	8A C7 E8 0C FF BF 18 01	ÿ
000000004E0:	89 45 FF BA 12 01 B4 09	CD 21 8B C1 BF 2E 01 E8	%Eÿ
000000004F0:	08 FF 8A C3 E8 F2 FE 89	45 FE BA 1C 01 B4 09 CD	ÿSÄ
00000000500:	21 32 C0 B4 4C CD 21		!2Ä

Рисунок 6 – hex-представление «плохого» .EXE файла(продолжение)

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

Ответ: в «хорошем» код, стек и данные выделены в отдельные сегменты. Код программы начинается также с адреса 300h, только в отличие от «плохого» файла, 100h байтов выделено под стек.

[illegible]

Рисунок 7 – hex-представление «хорошего» .EXE файла

00000000310:	45 4D 3A 20 20 20 0A 0D	24 55 53 45 52 20 4E 55	EM: USER NU
00000000320:	4D 42 45 52 3A 20 20 20	20 20 20 20 0A 0D 24 50	MBER: \$P
00000000330:	43 20 74 79 70 65 3A 20	50 43 0A 0D 24 50 43 20	C type: PC \$PC
00000000340:	74 79 70 65 3A 20 50 43	2F 58 54 0A 0D 24 50 43	type: PC/XT \$PC
00000000350:	20 74 79 70 65 3A 20 41	54 0A 0D 24 50 43 20 74	type: AT \$PC t
00000000360:	79 70 65 3A 20 6D 6F 64	65 6C 20 33 30 0A 0D 24	ype: model 30 \$
00000000370:	50 43 20 74 79 70 65 3A	20 6D 6F 64 65 6C 20 35	PC type: model 5
00000000380:	30 20 6F 72 20 36 30 0A	0D 24 50 43 20 74 79 70	0 or 60 \$PC typ
00000000390:	65 3A 20 6D 6F 64 65 6C	20 38 30 0A 0D 24 50 43	e: model 80 \$PC
000000003A0:	20 74 79 70 65 3A 20 50	43 6A 72 0A 0D 24 50 43	type: PCjr \$PC
000000003B0:	20 74 79 70 65 3A 20 50	43 20 43 6F 6E 76 65 72	type: PC Conver
000000003C0:	74 69 62 6C 65 0A 0D 24	50 43 20 74 79 70 65 20	tible \$PC type
000000003D0:	63 6F 64 65 3A 20 20 20	0A 0D 24 00 00 00 00 00	code: \$
000000003E0:	24 0F 3C 09 76 02 04 07	04 30 C3 51 8A E0 E8 EF	\$o<ov ÅQŠřčđ
000000003F0:	FF 86 C4 B1 04 D2 E8 E8	E6 FF 59 C3 53 8A FC E8	'tÄ±Ñčč'YÄSŠüč
00000000400:	E9 FF 88 25 4F 88 05 4F	8A C7 E8 DE FF 88 25 4F	é' %O OŠČčT' %O
00000000410:	88 05 5B C3 51 52 32 E4	33 D2 B9 0A 00 F7 F1 80	[ÄQR2ä3Ňaq ÷ñ€
00000000420:	CA 30 88 14 4E 33 D2 3D	0A 00 73 F1 3C 00 74 04	EQJN3Ň= sñ< t♦
00000000430:	0C 30 88 04 5A 59 C3 53	06 BB 00 F0 8E C3 26 A0	90ZŸÄS» đŽÄ&
00000000440:	FE FF 07 5B C3 50 52 57	3C FF 75 06 BA 2F 00 EB	t'•[ÄPRW<'u\$ / ë
00000000450:	5C 90 3C FE 75 06 BA 3D	00 EB 52 90 3C FB 75 06	\<tu\$= ëR<üu
00000000460:	BA 3D 00 EB 48 90 3C FC	75 06 BA 4E 00 EB 3E 90	\$= ëH<üu\$N ë>
00000000470:	3C FA 75 06 BA 5C 00 EB	34 90 3C FC 75 06 BA 70	<üu\$ \ ë4<üu\$ p
00000000480:	00 EB 2A 90 3C F8 75 06	BA 8A 00 EB 20 90 3C FD	ë* <řu\$Š ë <y
00000000490:	75 06 BA 9E 00 EB 16 90	3C F9 75 06 BA AE 00 EB	u\$ž ë=<üu\$® ë
000000004A0:	0C 90 E8 46 FF BF D6 00	89 05 BA C8 00 B4 09 CD	90čF'žÖ \$Č 'oÍ
000000004B0:	21 5F 5A 58 C3 1E 2B C0	50 BA 10 00 8E DA E8 76	!_ZXÄ+řP\$ ŽÚčv
000000004C0:	FF E8 81 FF B4 30 CD 21	50 BE 09 00 E8 45 FF 58	'č' '0Í!PIo čE'X
000000004D0:	8A C4 BE 0B 00 E8 3C FF	BA 00 00 B4 09 CD 21 8A	ŠÄIđ č<'š 'oÍ!Š
000000004E0:	C7 E8 07 FF BF 15 00 89	45 FF BA 0F 00 B4 09 CD	Čč•'žš \$E'šo 'oÍ
000000004F0:	21 8B C1 BF 2B 00 E8 03	FF 8A C3 E8 ED FE 89 45	!<Äž+ č♥ŠÄčít%E
00000000500:	FE BA 19 00 B4 09 CD 21	32 C0 B4 4C CD 21	tš↓ 'oÍ!2Ř'LI!

Рисунок 8 – hex-представление «плохого» .EXE файла(продолжение)

Загрузка COM модуля в основную память.

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

Ответ:

1. Определяется сегментный адрес свободного участка памяти достаточного для размещения программы размера;
2. Создается блок памяти для PSP и программы (сегмент: 0000h – PSP; сегмент: 0100h – программа). В поля PSP заносятся соответствующие значения;
3. Загружается COM-файл с адреса PSP: 0100h;
4. Значение регистра AX устанавливается в соответствии с параметрами командной строки;

5. Регистры DS, ES и SS устанавливаются на сегмент PSP;
6. Регистр SP устанавливается на конец сегмента;
7. Происходит запуск программы с адреса PSP:0100h.

AX 0000	SI 0000	CS 119C	IP 0100	Stack +0 0000	FLAGS 0200
BX 0000	DI 0000	DS 119C		+2 0000	
CX 0207	BP 0000	ES 119C	HS 119C	+4 0000	OF DF IF SF ZF AF PF CF
DX 0000	SP FFF5	SS 119C	FS 119C	+6 0000	0 0 1 0 0 0 0 0

CMD >					0 1 2 3 4 5 6 7
				DS:0000	CD 20 9C 11 00 EA FD FF
				DS:0008	AD DE ED 04 92 01 00 00
0100 E9B001	JMP	02B3		DS:0010	18 01 10 01 18 01 92 01
0103 44	INC	SP		DS:0018	03 FF FF FF FF FF FF FF
0104 4F	DEC	DI		DS:0020	FF FF FF FF FF FF FF FF
0105 53	PUSH	BX		DS:0028	FF FF FF FF 96 11 C4 FF
0106 205645	AND	[BP+45],DL		DS:0030	92 01 14 00 18 00 9C 11
0109 52	PUSH	DX		DS:0038	FF FF FF FF 00 00 00 00
010A 3A20	CMP	AH,[BX+SI]		DS:0040	05 00 00 00 00 00 00 00
010C 20E200A	AND	[0A20],CH		DS:0048	00 00 00 00 00 00 00 00

	0 1 2 3 4 5 6 7	8 9 A B C D E F	
DS:0000	CD 20 9C 11 00 EA FD FF	AD DE ED 04 92 01 00 00	
DS:0010	18 01 10 01 18 01 92 01	03 FF FF FF FF FF FF FF	
DS:0020	FF FF FF FF FF FF FF FF	FF FF FF FF 96 11 C4 FF	
DS:0030	92 01 14 00 18 00 9C 11	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	

Рисунок 9 – состояние регистров перед запуском COM-программы

2. Что располагается с адреса 0?

Ответ: первые 100h байт занимает PSP.

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

Ответ: все сегментные регистры указывают на начало PSP.

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

Ответ: стек занимает весь сегмент COM-программы, его начало находится в конце сегмента, на него указывает регистр SP = FFF5. SS указывает на начало сегмента.

Загрузка «хорошего» EXE модуля в основную память.

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

Ответ:

1. Определяется сегментный адрес свободного участка памяти, размер которого достаточен для размещения программы;

2. Создается блок памяти для PSP. В поля PSP заносятся соответствующие значения;
3. Определяется начальный сегмент. Загрузочный модуль считывается в начальный сегмент;
4. Таблица настройки считывается в рабочую память;
5. Определяются значения сегментных регистров:

SS = 11AC – сегмент стека;

CS = 11CA – сегмент кода;

DS = 119C – сегмент данных.

2. На что указывают регистры DS и ES?

Ответ: регистры указывают на начало PSP.

3. Как определяется стек?

Ответ: при исполнении в регистр SS записывается адрес начала сегмента стека, а в SP – его вершины.

4. Как определяется точка входа?

Ответ: точка входа в программу определяется с помощью директивы END. После этой директивы указывается адрес, куда переходит программа при запуске.

AX 0000	SI 0000	CS 11CA	IP 00D5	Stack +0 4F44	FLAGS 0200																																		
BX 0000	DI 0000	DS 119C		+2 2053																																			
CX 030E	BP 0000	ES 119C	HS 119C	+4 4556	OF	DF	IF	SF	ZF	AF	PF	CF																											
DX 0000	SP 0100	SS 11AC	FS 119C	+6 3A52	0	0	1	0	0	0	0	0																											
CMD >																																							
				1													0	1	2	3	4	5	6	7															
				DS:0000													CD	20	9C	11	00	EA	FD	FF															
				DS:0008													AD	DE	ED	04	92	01	00	00															
00D5 1E PUSH DS				DS:0010													18	01	10	01	18	01	92	01															
00D6 2BC0 SUB AX,AX				DS:0018													03	FF	FF	FF	FF	FF	FF	FF															
00D8 50 PUSH AX				DS:0020													FF	FF	FF	FF	FF	FF	FF	FF															
00D9 BABC11 MOV DX,11BC				DS:0028													FF	FF	FF	FF	96	11	C4	FF															
00DC 8EDA MOV DS,DX				DS:0030													92	01	14	00	18	00	9C	11															
00DE E876FF CALL 0057				DS:0038													FF	FF	FF	FF	00	00	00	00															
00E1 E881FF CALL 0065				DS:0040													05	00	00	00	00	00	00	00															
00E4 B430 MOV AH,30				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	9C	11	00	EA	FD	FF	AD	DE	ED	04	92	01	00	00	11							
DS:0010																18	01	10	01	18	01	92	01	03	FF	FF	FF	FF	FF	FF	FF	11							
DS:0020																FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	96	11	C4	FF	11							
DS:0030																92	01	14	00	18	00	9C	11	FF	FF	FF	FF	00	00	00	00	11							
DS:0040																05	00	00	00	00	00	00	00	00	00	00	00	00	00	00	11								
1 Step				2StepProc				3Retrieve				4 Help				5Set BRK				6				7 up				8 dn				9 le				0 ri			

Рисунок 10 - состояние регистров перед запуском EXE-программы

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД .COM МОДУЛЯ

TESTPC SEGMENT

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

ORG 100H

START: JMP BEGIN

DOS_V	db	'DOS VER: '
DOS_F	db	'.'
END_DOS_V	db	' ', 0AH, 0DH, '\$'
OEM	db	'OEM: '
ENDOEM	db	' ', 0AH, 0DH, '\$'
USERN	db	'USER NUMBER: '
USERNEND	db	' ', 0AH, 0DH, '\$'
TYPE_PC	db	'PC type: PC', 0AH, 0DH, '\$'
TYPE_PC_XT	db	'PC type: PC/XT', 0AH, 0DH, '\$'
TYPE_AT	db	'PC type: AT', 0AH, 0DH, '\$'
TYPE_PS2_M30	db	'PC type: model 30', 0AH, 0DH, '\$'
TYPE_PS2_M50_60	db	'PC type: model 50 or 60', 0AH, 0DH, '\$'
TYPE_PS2_M80	db	'PC type: model 80', 0AH, 0DH, '\$'
TYPE_PSjr	db	'PC type: PCjr', 0AH, 0DH, '\$'
TYPE_PC_CONV	db	'PC type: PC Convertible', 0AH, 0DH, '\$'
TYPE_UNKNOWN	db	'PC type code: '
TYPE_CODE	db	' ', 0AH, 0DH, '\$'

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 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_1
    or AL,30h
    mov [SI],AL
end_1: pop DX
    pop CX
    ret
```

BYTE_TO_DEC ENDP

;-----

GET_PC_CODE PROC near

```
    push BX
    push ES
    mov     BX,0F000H
    mov     ES,BX
    mov     AL,ES:[0FFFEH]
    pop     ES
    pop     BX
    ret
```

GET_PC_CODE ENDP

```

PRINT_PC_TYPE PROC near
    push AX
    push DX
    push DI

    T0: cmp AL, 0FFh
    jne T1
    mov DX, offset TYPE_PC;
    jmp print

    T1: cmp AL, 0FEh
    jne T2
    mov DX, offset TYPE_PC_XT;
    jmp print

    T2: cmp AL, 0FBh

    jne T3
    mov DX, offset TYPE_PC_XT;
    jmp print

    T3: cmp AL, 0FCh
    jne T4
    mov DX, offset TYPE_AT;
    jmp print

    T4: cmp AL, 0FAh
    jne T5
    mov DX, offset TYPE_PS2_M30;
    jmp print

    T5: cmp AL, 0FCh
    jne T6
    mov DX, offset TYPE_PS2_M50_60;
    jmp print

```



```
T6: cmp AL, 0F8h
jne T7
mov DX, offset TYPE_PS2_M80;
jmp print
```

```
T7: cmp AL, 0FDh
jne T8
mov DX, offset TYPE_PSjr
jmp print
```

```
T8: cmp AL, 0F9h
jne T9
mov DX, offset TYPE_PC_CONV
jmp print
```

```
T9:
call BYTE_TO_HEX
mov DI, OFFSET TYPE_CODE
mov [DI], AX
mov DX, offset TYPE_UNKNOWN;
```

```
print:
mov AH, 09h
int 21h
pop DI
pop DX
pop AX
ret
```

```
PRINT_PC_TYPE ENDP
```

```
BEGIN:
push DS
sub AX, AX
```

```

push AX

; PC TYPE
call GET_PC_CODE
call PRINT_PC_TYPE

mov  AH,30H
INT  21H

; DOS
push AX
mov SI, offset DOS_F
call BYTE_TO_DEC
pop AX
mov AL, AH
mov SI, offset END_DOS_V
call BYTE_TO_DEC
mov DX, offset DOS_V;
mov AH, 09h
int 21h

; OEM
mov AL, BH
call BYTE_TO_HEX
mov DI, offset ENDOEM
mov [DI-1], AX
mov DX, offset OEM;
mov AH, 09h
int 21h

; USER NUMBER
mov AX, CX
mov DI, offset USERNEND
call WRD_TO_HEX

```

```
mov AL, BL
call BYTE_TO_HEX
mov [DI-2], AX

mov DX, offset USERN
mov AH, 09h
int 21h

; ВЫХОД В DOS
xor AL,AL
mov AH,4Ch
int 21H
TESTPC ENDS
END START
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД .EXE МОДУЛЯ

SSEG SEGMENT stack

db 100 dup(?)

SSEG ENDS

DATA SEGMENT

DOS_V	db	'DOS VER: '
DOS_F	db	'.'
END_DOS_V	db	' ', 0AH, 0DH, '\$'
OEM	db	'OEM: '
ENDOEM	db	' ', 0AH, 0DH, '\$'
USERN	db	'USER NUMBER: '
USERNEND	db	' ', 0AH, 0DH, '\$'
TYPE_PC	db	'PC type: PC', 0AH, 0DH, '\$'
TYPE_PC_XT	db	'PC type: PC/XT', 0AH, 0DH, '\$'
TYPE_AT	db	'PC type: AT', 0AH, 0DH, '\$'
TYPE_PS2_M30	db	'PC type: model 30', 0AH, 0DH, '\$'
TYPE_PS2_M50_60	db	'PC type: model 50 or 60', 0AH, 0DH, '\$'
TYPE_PS2_M80	db	'PC type: model 80', 0AH, 0DH, '\$'
TYPE_PSjr	db	'PC type: PCjr', 0AH, 0DH, '\$'
TYPE_PC_CONV	db	'PC type: PC Convertible', 0AH, 0DH, '\$'
TYPE_UNKNOWN	db	'PC type code: '
TYPE_CODE	db	' ', 0AH, 0DH, '\$'

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:CODE, ES:DATA, SS:SSEG

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 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_1
    or AL,30h
    mov [SI],AL
end_1: pop DX
    pop CX
    ret
```

BYTE_TO_DEC ENDP

;-----

GET_PC_CODE PROC near

```
    push BX
    push ES
    mov  BX,0F000H
    mov  ES,BX
    mov  AL,ES:[0FFFEH]
    pop  ES
    pop  BX
```

```

        ret
GET_PC_CODE      ENDP

PRINT_PC_TYPE PROC near
    push AX
    push DX
    push DI

    T0: cmp AL, 0FFh
    jne T1
    mov DX, offset TYPE_PC;
    jmp print

    T1: cmp AL, 0FEh
    jne T2
    mov DX, offset TYPE_PC_XT;
    jmp print

    T2: cmp AL, 0FBh
    jne T3
    mov DX, offset TYPE_PC_XT;
    jmp print

    T3: cmp AL, 0FCh
    jne T4
    mov DX, offset TYPE_AT;
    jmp print

    T4: cmp AL, 0FAh
    jne T5
    mov DX, offset TYPE_PS2_M30;
    jmp print

    T5: cmp AL, 0FCh

```

```
jne T6
mov DX, offset TYPE_PS2_M50_60;
jmp print
```

```
T6: cmp AL, 0F8h
jne T7
mov DX, offset TYPE_PS2_M80;
jmp print
```

```
T7: cmp AL, 0FDh
jne T8
mov DX, offset TYPE_PSjr
jmp print
```

```
T8: cmp AL, 0F9h
jne T9
mov DX, offset TYPE_PC_CONV
jmp print
```

```
T9:
call BYTE_TO_HEX
mov DI, OFFSET TYPE_CODE
mov [DI], AX
mov DX, offset TYPE_UNKNOWN;
```

```
print:
mov AH, 09h
int 21h
pop DI
pop DX
pop AX
ret
```

```
PRINT_PC_TYPE ENDP
```


START:

```
    push DS
    sub AX, AX
    push AX

    mov DX, DATA
    mov DS, DX

    ; PC TYPE
    call GET_PC_CODE
    call PRINT_PC_TYPE

    mov  AH,30H
    INT  21H

    ; DOS
    push AX
    mov SI, offset DOS_F
    call BYTE_TO_DEC
    pop AX
    mov AL, AH
    mov SI, offset END_DOS_V
    call BYTE_TO_DEC
    mov DX, offset DOS_V;
    mov AH, 09h
    int 21h

    ; OEM
    mov AL, BH
    call BYTE_TO_HEX
    mov DI, offset ENDOEM
    mov [DI-1], AX
    mov DX, offset OEM;
```

```

mov AH, 09h
int 21h

; USER NUMBER
mov AX, CX
mov DI, offset USERNEND
call WRD_TO_HEX
mov AL, BL
call BYTE_TO_HEX
mov [DI-2], AX

mov DX, offset USERN
mov AH, 09h
int 21h

; ВЫХОД В DOS
xor AL,AL
mov AH,4Ch
int 21H
CODE ENDS
END START

```