

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.

Студент гр. 9383

Гладких А.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

ПОСТАНОВКА ЗАДАЧИ

Цель работы.

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

Сведения о функциях и структурах данных управляющей программы.

Процедура	Описание
TETR_TO_HEX	Перевод десятичной цифры в код символа ASCII
BYTE_TO_HEX	Перевод байта в AL в два символа шестн. числа AX
WRD_TO_HEX	Перевод в 16 с/с 16-ти разрядного числа
BYTE_TO_DEC	Перевод в 10 с/с, SI – адрес поля младшей цифры
TASK_1	Вывод типа ПК на экран
TASK_2	Вывод версии DOS на экран

Задание.

Шаг 1. Написать текст исходно .COM модуля, который определяет тип РС и версию системы. Построить «плохой» .EXE модуль, полученный из исходного текста для .COM модуля.

Шаг 2. Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1, построить и отладить его. Таким образом будет получен «хороший» .EXE.

Шаг 3. Сравнить исходные тексты для .COM и .EXE модулей. Ответить на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

Шаг 4. Запустить FAR и открыть файлы загрузочного модуля .COM и файл «плохого» .EXE в шестнадцатеричном виде. Затем открыть файл

загрузочного модуля «хорошего» .EXE и сравнить его с предыдущими файлами. Ответить на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

Шаг 5. Открыть отладчик TD.EXE и загрузить .COM. Ответить на контрольные вопросы «Загрузка COM модуля в основную память». Представить в отчете план загрузки модуля .COM в основную память.

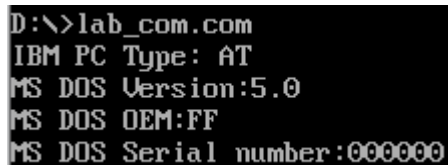
Шаг 6. Открыть отладчик TD.EXE и загрузить «хороший» .EXE. Ответить на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

Исходный код.

Исходный код представлен в приложении А.

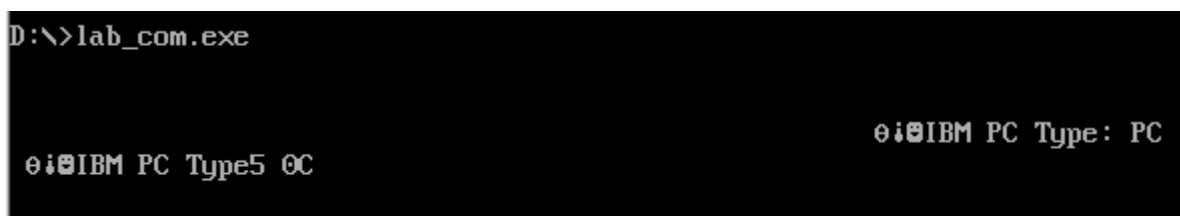
РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ ПРОБЛЕМ

Шаг 1. Был написан текст исходного .COM модуля, который определяет тип PC и версию системы. Был построен «плохой» .EXE модуль, полученный из исходного текста для .COM модуля.



```
D:\>lab_com.com
IBM PC Type: AT
MS DOS Version: 5.0
MS DOS OEM: FF
MS DOS Serial number: 000000
```

Рисунок 1. Демонстрация работы .COM модуля

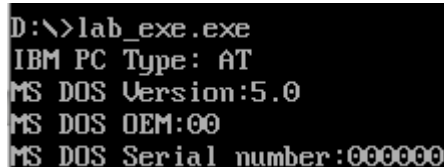


```
D:\>lab_com.exe

?i?IBM PC Type5 0C
?i?IBM PC Type: PC
```

Рисунок 2. Демонстрация работы «плохого» .EXE модуля

Шаг 2. Был написан текст «хорошего» .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1.



```
D:\>lab_exe.exe
IBM PC Type: AT
MS DOS Version: 5.0
MS DOS OEM: 00
MS DOS Serial number: 000000
```

Рисунок 3. Демонстрация работы «хорошего» .EXE модуля

Шаг 3. Произведено сравнение исходных текстов для .COM и .EXE модулей. Исходя из результатов сравнения были отвечены контрольные вопросы «Отличия исходных текстов COM и EXE программ»:

1. Сколько сегментов должна содержать COM-программа?

Ответ: COM-программа содержит лишь один сегмент, в котором хранится и код, и данные, а стек располагается начиная с конца этого сегмента.

2. EXE-программа?

Ответ: EXE-программа должна содержать один и более сегментов, обычно сегменты кода, данных и стека отделены друг от друга.

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

Ответ: директива `org 100h`, чтобы сместить адресацию на 256 байт (на размер PSP) от нулевого адреса. Также необходимо прописать директиву `ASSUME`, так как в противном случае компилятор Турбо Ассемблера выдаст ошибку, потому что не сможет «установить» в память выполняемую программу из-за того, что не будет знать о сегменте кода.

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

Ответ: нельзя использовать команды, которые непосредственно берут адрес сегмента. Например `mov ax, @data` или `mov ax, seg vec1` (`vec1` – часть `data segment`). Это происходит из-за того, что в `.COM`-файле нет таблицы настроек. Программа изначально не знает, где находятся `@data` и `vec1` в памяти, поэтому требуется создать таблицу настроек, в которой будет храниться их относительный адрес. Эта таблица подключается на этапе линковки. В `COM`-программе такое не происходит.

Шаг 4. С помощью программы Notepad++ было произведено сравнение файлов загрузочного модуля `.COM`, «плохого» `.EXE` и «хорошего» `.EXE`. Исходя из результатов сравнения были отвечены контрольные вопросы «Отличия форматов файлов `COM` и `EXE` модулей»:

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

Ответ: В `COM`-файле все данные размещаются в одного сегменте, адресация которого начинается с `0h`.

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	e9	ad	02	49	42	4d	20	50	43	20	54	79	70	65	3a	20	й-. IBM PC Type:
00000010	50	43	0d	0a	24	49	42	4d	20	50	43	20	54	79	70	65	PC..\$IBM PC Type
00000020	3a	20	50	43	2f	58	54	0d	0a	24	49	42	4d	20	50	43	: PC/XT..\$IBM PC
00000030	20	54	79	70	65	3a	20	41	54	0d	0a	24	49	42	4d	20	Type: AT..\$IBM
00000040	50	43	20	54	79	70	65	3a	20	50	53	32	20	4d	6f	64	PC Type: PS2 Mod
00000050	65	6c	20	33	30	0d	0a	24	49	42	4d	20	50	43	20	54	e1 30..\$IBM PC T
00000060	79	70	65	3a	20	50	53	32	20	4d	6f	64	65	6c	20	35	ype: PS2 Model 5
00000070	30	20	6f	72	20	36	30	0d	0a	24	49	42	4d	20	50	43	0 or 60..\$IBM PC
00000080	20	54	79	70	65	3a	20	50	53	32	20	4d	6f	64	65	6c	Type: PS2 Model
00000090	20	38	30	0d	0a	24	49	42	4d	20	50	43	20	54	79	70	80..\$IBM PC Typ
000000a0	65	3a	20	50	43	6a	72	0d	0a	24	49	42	4d	20	50	43	e: PCjr..\$IBM PC
000000b0	20	54	79	70	65	3a	20	50	43	20	43	6f	6e	76	65	72	Type: PC Conver
000000c0	74	69	62	6c	65	0d	0a	24	49	42	4d	20	50	43	20	54	tible..\$IBM PC T
000000d0	79	70	65	20	55	6e	6b	6e	6f	77	6e	2c	20	63	6f	64	ype Unknown, cod
000000e0	65	3a	20	20	0d	0a	24	4d	53	20	44	4f	53	20	56	65	e: ..\$MS DOS Ve
000000f0	72	73	69	6f	6e	3a	20	3c	20	32	2e	30	0d	0a	24	4d	rsion: < 2.0..\$M
00000100	53	20	44	4f	53	20	56	65	72	73	69	6f	6e	3a	20	2e	S DOS Version: .
00000110	20	0d	0a	24	4d	53	20	44	4f	53	20	4f	45	4d	3a	20	..\$MS DOS OEM:
00000120	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00000130	20	20	20	20	20	0d	0a	24	4d	53	20	44	4f	53	20		..\$MS DOS
00000140	53	65	72	69	61	6c	20	6e	75	6d	62	65	72	3a	20	20	Serial number:
00000150	20	20	20	0d	0a	24	24	0f	3c	09	76	02	04	07	04		..\$\$.<.v....
00000160	30	c3	51	8a	e0	e8	ef	ff	86	c4	b1	04	d2	e8	e8	e6	0ГQJамиятД±.Тюж
00000170	ff	59	c3	53	8a	fc	e8	e9	ff	88	25	4f	88	05	4f	8a	яYGSJьийя€%0€.0M
00000180	c7	e8	de	ff	88	25	4f	88	05	5b	c3	51	52	32	e4	33	Эи0я€%0€. [ГQR2д3
00000190	d2	b9	0a	00	f7	f1	80	ca	30	88	14	4e	33	d2	3d	0a	ТW..чсYK0€.N3T=.
000001a0	00	73	f1	3c	00	74	04	0c	30	88	04	5a	59	c3	50	b4	.sc<.t..0€.ZYTPr
000001b0	09	cd	21	58	c3	50	53	52	06	57	b8	00	f0	8e	c0	bf	.H!XГPSPR.We.pRAi
000001c0	fe	ff	26	8a	05	3c	ff	74	34	3c	fe	74	39	3c	fb	74	мя«J..<яt4<яt9<яt
000001d0	35	3c	fc	74	3a	3c	fa	74	3f	3c	fc	74	44	3c	f8	74	5<яt:<яt?<яtD<яt
000001e0	49	3c	fd	74	4e	3c	f9	74	53	e8	76	ff	bf	c8	01	88	I<яtN<яtSивяяИ.€
000001f0	45	1a	88	65	1b	8b	d7	e8	b4	ff	eb	49	90	ba	03	01	E.€e.<Чиг'ялI.e..
00000200	e8	ab	ff	eb	40	90	ba	15	01	e8	a2	ff	eb	37	90	ba	и«ял@.e..и'ял7.e
00000210	2a	01	e8	99	ff	eb	2e	90	ba	3c	01	e8	90	ff	eb	25	*.и"ял..e<.и.ял%
00000220	90	ba	58	01	e8	87	ff	eb	1c	90	ba	7a	01	e8	7e	ff	.eX.и'ял..eз.и-я
00000230	eb	13	90	ba	96	01	e8	75	ff	eb	0a	90	ba	aa	01	e8	л..e-.ииял..e€e.и
00000240	6c	ff	eb	01	90	5f	07	5a	5b	58	c3	50	53	52	06	57	лял.._Z[XГPSPR.W
00000250	b4	30	cd	21	3c	00	74	1d	50	e8	2f	ff	ad	bf	ff	01	гOH!<.t.Ри/я-ia.
00000260	88	65	0f	58	86	e0	e8	22	ff	ad	88	65	11	8b	d7	e8	€e.X'ял"я-€e.<Чи
00000270	3c	ff	eb	07	90	ba	e7	01	e8	33	ff	8a	c7	e8	e2	fe	<ял..eз.и3яMЗивю
00000280	bf	14	02	88	45	0b	88	65	0c	8b	d7	e8	20	ff	8a	c3	i...€E.€e.<Чи ялГ
00000290	e8	cf	fe	bf	39	02	88	45	15	88	65	16	8b	c1	83	c7	иПюi9.€E.€e.<Бi3
000002a0	1a	e8	cf	fe	ba	39	02	e8	04	ff	5f	07	5a	5b	58	c3	.иПюe9.и.я_Z[XГ
000002b0	e8	02	ff	e8	95	ff	32	c0	b4	4c	cd	21					и.яи"я2ArLH!

Рисунок 4. Структура COM-файла

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

Ответ: файл некорректно работает, так файл состоит из одного сегмента — данные и код хранятся в одном месте, а сегмента стека нет. Адресация этого сегмента начинается с 300h. Начиная с адреса 0h располагается PSP (до 100h) и таблица настройки и заголовков (со 100h до 300h).

```

00000000 4d 5a bc 01 03 00 00 00 20 00 00 00 ff ff 00 00 MZj.....ня..
00000010 00 00 00 00 00 01 00 00 3e 00 00 00 01 00 fb 50 .....>.....мР
00000020 6a 72 00 00 00 00 00 00 00 00 00 00 00 00 00 00 jr.....
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 .....

```

Рисунок 5. Структура «плохого» EXE-файла

```

00000300 e9 ad 02 49 42 4d 20 50 43 20 54 79 70 65 3a 20 й-.IBM PC Type:
00000310 50 43 0d 0a 24 49 42 4d 20 50 43 20 54 79 70 65 PC..$IBM PC Type
00000320 3a 20 50 43 2f 58 54 0d 0a 24 49 42 4d 20 50 43 : PC/XT..$IBM PC
00000330 20 54 79 70 65 3a 20 41 54 0d 0a 24 49 42 4d 20 Type: AT..$IBM
00000340 50 43 20 54 79 70 65 3a 20 50 53 32 20 4d 6f 64 PC Type: PS2 Mod
00000350 65 6c 20 33 30 0d 0a 24 49 42 4d 20 50 43 20 54 e1 30..$IBM PC T
00000360 79 70 65 3a 20 50 53 32 20 4d 6f 64 65 6c 20 35 ype: PS2 Model 5
00000370 30 20 6f 72 20 36 30 0d 0a 24 49 42 4d 20 50 43 0 or 60..$IBM PC
00000380 20 54 79 70 65 3a 20 50 53 32 20 4d 6f 64 65 6c Type: PS2 Model
00000390 20 38 30 0d 0a 24 49 42 4d 20 50 43 20 54 79 70 80..$IBM PC Typ
000003a0 65 3a 20 50 43 6a 72 0d 0a 24 49 42 4d 20 50 43 e: PCjr..$IBM PC
000003b0 20 54 79 70 65 3a 20 50 43 20 43 6f 6e 76 65 72 Type: PC Conver
000003c0 74 69 62 6c 65 0d 0a 24 49 42 4d 20 50 43 20 54 tible..$IBM PC T
000003d0 79 70 65 20 55 6e 6b 6e 6f 77 6e 2c 20 63 6f 64 ype Unknown, cod
000003e0 65 3a 20 20 0d 0a 24 4d 53 20 44 4f 53 20 56 65 e: ..$MS DOS Ve
000003f0 72 73 69 6f 6e 3a 20 3c 20 32 2e 30 0d 0a 24 4d rsion: < 2.0..$M
00000400 53 20 44 4f 53 20 56 65 72 73 69 6f 6e 3a 20 2e S DOS Version: .
00000410 20 0d 0a 24 4d 53 20 44 4f 53 20 4f 45 4d 3a 20 ..$MS DOS OEM:
00000420 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000430 20 20 20 20 20 20 0d 0a 24 4d 53 20 44 4f 53 20 ..$MS DOS
00000440 53 65 72 69 61 6c 20 6e 75 6d 62 65 72 3a 20 20 Serial number:
00000450 20 20 20 20 0d 0a 24 24 0f 3c 09 76 02 04 07 04 ..$.<.v....
00000460 30 c3 51 8a e0 e8 ef ff 86 c4 b1 04 d2 e8 e8 e6 ОГQЛамп+Д+.Тижк
00000470 ff 59 c3 53 8a fc e8 e9 ff 88 25 4f 88 05 4f 8a яУТSLьыия€%0€.0Л
00000480 c7 e8 de ff 88 25 4f 88 05 5b c3 51 52 32 e4 33 Эи0я€%0€. [ГQR2д3
00000490 d2 b9 0a 00 f7 f1 80 ca 30 88 14 4e 33 d2 3d 0a ТМ...чсЪКО€.N3T=.
000004a0 00 73 f1 3c 00 74 04 0c 30 88 04 5a 59 c3 50 b4 .sc<.t...0€.ZYTPr
000004b0 09 cd 21 58 c3 50 53 52 06 57 b8 00 f0 8e c0 bf .H!XTPSR.Wë.pHAI
000004c0 fe ff 26 8a 05 3c ff 74 34 3c fe 74 39 3c fb 74 ян4Л.<ят4<ят9<ят
000004d0 35 3c fc 74 3a 3c fa 74 3f 3c fc 74 44 3c f8 74 5<ят:<ят?<ятD<ят
000004e0 49 3c fd 74 4e 3c f9 74 53 e8 76 ff bf c8 01 88 I<ятN<ятSiвнiИ.€
000004f0 45 1a 88 65 1b 8b d7 e8 b4 ff eb 49 90 ba 03 01 Е.€е.<ЧигялI.е..
00000500 e8 ab ff eb 40 90 ba 15 01 e8 a2 ff eb 37 90 ба и«ял@.е..ифял7.е
00000510 2a 01 e8 99 ff eb 2e 90 ba 3c 01 e8 90 ff eb 25 *.и³ял..е<.и.ял*
00000520 90 ba 58 01 e8 87 ff eb 1c 90 ba 7a 01 e8 7e ff .еХ.ифял..ез.и~я
00000530 eb 13 90 ba 96 01 e8 75 ff eb 0a 90 ba aa 01 e8 л..е-.ииял..е€.и
00000540 6c ff eb 01 90 5f 07 5a 5b 58 c3 50 53 52 06 57 лял.._..Z[XTPSR.W
00000550 b4 30 cd 21 3c 00 74 1d 50 e8 2f ff ad bf ff 01 гОН!<.t.Ри/я-iя.
00000560 88 65 0f 58 86 e0 e8 22 ff ad 88 65 11 8b d7 e8 €е.X+ам"я-€е.<Чи
00000570 3c ff eb 07 90 ba e7 01 e8 33 ff 8a c7 e8 e2 fe <ял..ез.иЗяЛЗивю
00000580 bf 14 02 88 45 0b 88 65 0c 8b d7 e8 20 ff 8a c3 i...€Е.€е.<Чи яЛГ
00000590 e8 cf fe bf 39 02 88 45 15 88 65 16 8b c1 83 c7 иПюi9.€Е.€е.<Бr3
000005a0 1a e8 cf fe ba 39 02 e8 04 ff 5f 07 5a 5b 58 c3 .иПюe9.и.я_.Z[XГ
000005b0 e8 02 ff e8 95 ff 32 c0 b4 4c cd 21 и.ии*я2&rLH!

```

Рисунок 6. Структура «плохого» EXE-файла

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

Ответ: файл хранится в виде трех сегментов. Код начинается с адреса 3e0h, следуя за сегментом данных. В начале также содержится таблица настроек и заголовок.

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	4d	5a	4a	01	03	00	01	00	20	00	00	00	ff	ff	00	00	MZJ..... ..яя..
00000010	80	00	d3	05	59	01	1e	00	1e	00	00	00	01	00	5a	01	Ъ.У.Ү.....Z.
00000020	1e	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

Рисунок 7. Структура «хорошего» EXE-файла

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000320	50	43	6a	72	0d	0a	24	49	42	4d	20	50	43	20	54	79	PCjr..\$IBM PC Ty
00000330	70	65	3a	20	50	43	20	43	6f	6e	76	65	72	74	69	62	pe: PC Convertib
00000340	6c	65	0d	0a	24	49	42	4d	20	50	43	20	54	79	70	65	le..\$IBM PC Type
00000350	20	55	6e	6b	6e	6f	77	6e	2c	20	63	6f	64	65	3a	20	Unknown, code:
00000360	20	0d	0a	24	4d	53	20	44	4f	53	20	56	65	72	73	69	..\$MS DOS Versi
00000370	6f	6e	3a	20	3c	20	32	2e	30	0d	0a	24	4d	53	20	44	on: < 2.0..\$MS D
00000380	4f	53	20	56	65	72	73	69	6f	6e	3a	20	2e	20	0d	0a	OS Version: . . .
00000390	24	4d	53	20	44	4f	53	20	4f	45	4d	3a	20	20	20	20	\$MS DOS OEM:
000003a0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
000003b0	20	20	20	0d	0a	24	4d	53	20	44	4f	53	20	53	65	72	..\$MS DOS Ser
000003c0	69	61	6c	20	6e	75	6d	62	65	72	3a	20	20	20	20	20	ial number:
000003d0	20	0d	0a	24	00	00	00	00	00	00	00	00	00	00	00	00	..\$.
000003e0	24	0f	3c	09	76	02	04	07	04	30	c3	51	8a	e0	e8	ef	\$.<.v....0ГQЪаип
000003f0	ff	86	c4	b1	04	d2	e8	e8	e6	ff	59	c3	53	8a	fc	e8	ятД±.ТиижЯУГ\$Ъьи
00000400	e9	ff	88	25	4f	88	05	4f	8a	c7	e8	de	ff	88	25	4f	йя€%0€.ользиЮя€%0
00000410	88	05	5b	c3	51	52	32	e4	33	d2	b9	0a	00	f7	f1	80	€. [ГQR2д3ТМ#.чсВ
00000420	ca	30	88	14	4e	33	d2	3d	0a	00	73	f1	3c	00	74	04	КО€.N3Т=.ssc<.t.
00000430	0c	30	88	04	5a	59	c3	50	b4	09	cd	21	58	c3	50	53	.0€.ZYTPPг.Н!ХГPS
00000440	52	06	57	b8	00	f0	8e	c0	bf	fe	ff	26	8a	05	3c	ff	R.Wë.рґАіюя&Ъ.<я
00000450	74	34	3c	fe	74	39	3c	fb	74	35	3c	fc	74	3a	3c	fa	t4<юt9<ыt5<ыt:<ъ
00000460	74	3f	3c	fc	74	44	3c	f8	74	49	3c	fd	74	4e	3c	f9	t?<ыtD<штI<этN<щ
00000470	74	53	e8	76	ff	bf	c5	00	88	45	1a	88	65	1b	8b	d7	tСивяіЕ.€Е.€е.<Ч
00000480	e8	b4	ff	eb	49	90	ba	00	00	e8	ab	ff	eb	40	90	ba	игялI.е...и<ял@.е
00000490	12	00	e8	a2	ff	eb	37	90	ba	27	00	e8	99	ff	eb	2e	..иўял7.е'.и™ял.
000004a0	90	ba	39	00	e8	90	ff	eb	25	90	ba	55	00	e8	87	ff	.е9.и.ял%.еU.и†я
000004b0	eb	1c	90	ba	77	00	e8	7e	ff	eb	13	90	ba	93	00	e8	л..ew.и~ял..е".и
000004c0	75	ff	eb	0a	90	ba	a7	00	e8	6c	ff	eb	01	90	5f	07	уял..е\$.илял.._.
000004d0	5a	5b	58	c3	50	53	52	06	57	b4	30	cd	21	3c	00	74	Z [ХГPSR.WгOH!<.t
000004e0	1d	50	e8	2f	ff	ad	bf	fc	00	88	65	0f	58	86	e0	e8	.Ри/я-іь.€е.Xтаи
000004f0	22	ff	ad	88	65	11	8b	d7	e8	3c	ff	eb	07	90	ba	e4	"я-€е.<Чи<ял..ед
00000500	00	e8	33	ff	8a	c7	e8	e2	fe	bf	11	01	88	45	0b	88	.изяЪзивюі...€Е.€
00000510	65	0c	8b	d7	e8	20	ff	8a	c3	e8	cf	fe	bf	36	01	88	е.<Чи яьГиПоі6.€
00000520	45	15	88	65	16	8b	c1	83	c7	1a	e8	cf	fe	ba	36	01	Е.€е.<ВѓЗ.иПоєб.
00000530	e8	04	ff	5f	07	5a	5b	58	c3	b8	08	00	8e	d8	e8	fd	и.я_.Z [ХГё...тшиэ
00000540	fe	e8	90	ff	32	c0	b4	4c	cd	21							юи.я2АГЛH!

Рисунок 8. Структура «хорошего» EXE-файла

Шаг 5. С помощью отладчика TD.EXE были отвечены контрольные вопросы «Загрузка СОМ модуля в основную память»:

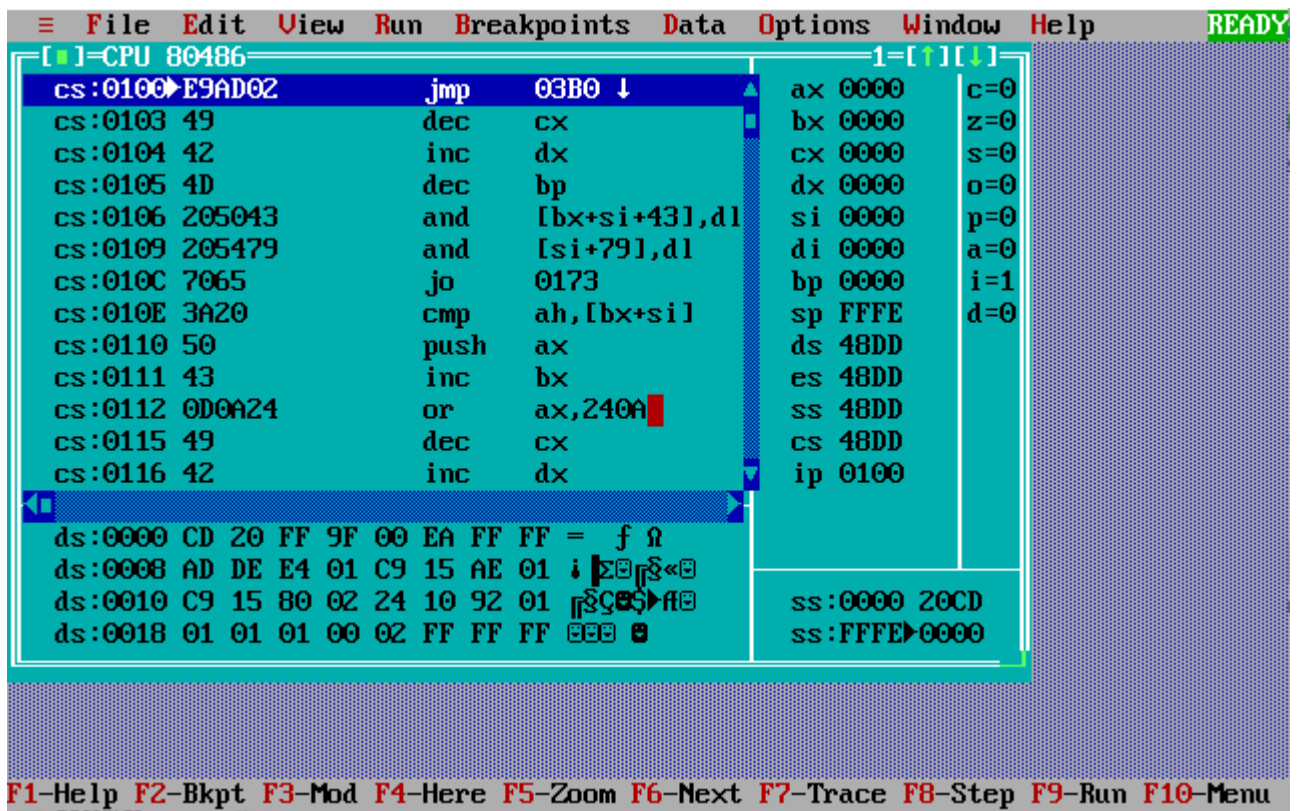


Рисунок 9. Отладчик TD.EXE с открытым COM файлом

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

Ответ: Начиная с 0 адреса в оперативную память загружается PSP. Сам код располагается, начиная с адреса 100h, сразу после PSP.

2. *Что располагается с адреса 0?*

Ответ: PSP.

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

Ответ: Все сегментные регистры имеют одинаковое значение – 48DD. Сегментные регистры указывают на начало PSP.

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

Ответ: COM-программа генерирует стек автоматически. В начале указатель стека (SP) расположен по адресу FFFEh, то есть в самом верху сегмента памяти. Соответственно растет стек в сторону 0h.

```

File Edit View Run Breakpoints Data Options Window Help
[ ] CPU 80486 1=[↑][↓]
cs:0159 B8F548 mov ax,48F5 ax 0000 c=0
cs:015C 8ED8 mov ds,ax bx 0000 z=0
cs:015E E8FDFE call 005E cx 0000 s=0
cs:0161 E890FF call 00F4 dx 0000 o=0
cs:0164 32C0 xor al,al si 0000 p=0
cs:0166 B44C mov ah,4C di 0000 a=0
cs:0168 CD21 int 21 bp 0000 i=1
cs:016A 0000 add [bx+si],al sp 0080 d=0
cs:016C 0000 add [bx+si],al ds 48DD
cs:016E 0000 add [bx+si],al es 48DD
cs:0170 0000 add [bx+si],al ss 48ED
cs:0172 0000 add [bx+si],al cs 490B
cs:0174 0000 add [bx+si],al ip 0159

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

ss:0082 204D
ss:0080 4249

```

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

2. На что указывают регистры DS и ES?

3. Как определяется стек?

4. Как определяется точка входа?

11

Выводы.

Были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab_com.asm

```
TESTPC  SEGMENT
        ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
org 100h

start:  jmp begin

PC_STRING db 'IBM PC Type: PC', 0DH, 0AH, '$'
PCXT_STRING db 'IBM PC Type: PC/XT', 0DH, 0AH, '$'
AT_STRING db 'IBM PC Type: AT', 0DH, 0AH, '$'
PS2_MODEL_30_STRING db 'IBM PC Type: PS2 Model 30', 0DH, 0AH, '$'
PS2_MODEL_50_or_60_STRING db 'IBM PC Type: PS2 Model 50 or 60', 0DH, 0AH,
'$'
PS2_MODEL_80_STRING db 'IBM PC Type: PS2 Model 80', 0DH, 0AH, '$'
PCjr_STRING db 'IBM PC Type: PCjr', 0DH, 0AH, '$'
PC_Convertible_STRING db 'IBM PC Type: PC Convertible', 0DH, 0AH, '$'
PC_TYPE_UNKNOWN db 'IBM PC Type Unknown, code: ', 0DH, 0AH, '$'

VERSION_LESS_2_NUM_STRING db 'MS DOS Version: < 2.0', 0DH, 0AH, '$'
VERSION_NUM_STRING db 'MS DOS Version: . ', 0DH, 0AH, '$'
OEM_NUM_STRING db 'MS DOS OEM: ', 0DH, 0AH, '$'
SERIAL_NUM_STRING db 'MS DOS Serial number: ', 0DH, 0AH, '$'

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
    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

```

```

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

```

```

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_l
        or al, 30h
        mov [si], al

end_l:
        pop dx
        pop cx
        ret

BYTE_TO_DEC endp

WRITEMSG PROC NEAR
        push ax
        mov ah, 9
        int 21h
        pop ax
        ret
WRITEMSG ENDP

TASK_1 PROC NEAR

        push ax
        push bx
        push dx
        push es
        push di

        mov ax, 0F000h
        mov es, ax
        mov di, 0FFFEh
        mov al, es:[di]

        cmp al, 0FFh
        je pc

        cmp al, 0FEh
        je pc_xt

```

```

    cmp al, 0FBh
    je pc_xt

    cmp al, 0FCh
    je at

    cmp al, 0FAh
    je ps2_model_30

    cmp al, 0FCh
    je ps2_model_50_or_60

    cmp al, 0F8h
    je ps2_model_80

    cmp al, 0FDh
    je pcjr

    cmp al, 0F9h
    je pc_convertible

    call BYTE_TO_HEX
    mov di, offset PC_TYPE_UNKNOWN
    mov [di + 26], al
    mov [di + 27], ah
    mov dx, di
    call WRITEMSG
    jmp end_task1

pc:
    mov dx, offset PC_STRING
    call WRITEMSG
    jmp end_task1

pc_xt:
    mov dx, offset PCXT_STRING
    call WRITEMSG
    jmp end_task1

at:

```



```

        mov dx, offset AT_STRING
        call WRITEMSG
        jmp end_task1

ps2_model_30:
        mov dx, offset PS2_MODEL_30_STRING
        call WRITEMSG
        jmp end_task1

ps2_model_50_or_60:
        mov dx, offset PS2_MODEL_50_or_60_STRING
        call WRITEMSG
        jmp end_task1

ps2_model_80:
        mov dx, offset PS2_MODEL_80_STRING
        call WRITEMSG
        jmp end_task1

pcjr:
        mov dx, offset PCjr_STRING
        call WRITEMSG
        jmp end_task1

pc_convertible:
        mov dx, offset PC_Convertible_STRING
        call WRITEMSG
        jmp end_task1

end_task1:
        pop di
        pop es
        pop dx
        pop bx
        pop ax

        ret

TASK_1 ENDP

TASK_2 PROC NEAR

```

```

push ax
push bx
push dx
push es
push di

mov ah, 30h
int 21h

;al - version number
;ah - mod number
;bh - OEM number
;bl:cx - serial number

cmp al, 0h
je less_than_2

push ax
call BYTE_TO_DEC
lodsw

mov di, offset VERSION_NUM_STRING
mov [di + 15], ah

pop ax

xchg ah, al
call BYTE_TO_DEC
lodsw
mov [di + 17], ah
mov dx, di
call WRITEMSG
jmp after_version

less_than_2:

mov dx, offset VERSION_LESS_2_NUM_STRING
call WRITEMSG

after_version:

mov al, bh

```

```

    call BYTE_TO_HEX
    mov di, offset OEM_NUM_STRING
    mov [di + 11], al
    mov [di + 12], ah

    mov dx, di
    call WRITEMSG

    mov al, bl
    call BYTE_TO_HEX
    mov di, offset SERIAL_NUM_STRING
    mov [di + 21], al
    mov [di + 22], ah

    mov ax, cx
    add di, 26
    call WRD_TO_HEX

    mov dx, offset SERIAL_NUM_STRING
    call WRITEMSG

    pop di
    pop es
    pop dx
    pop bx
    pop ax

    ret

TASK_2 ENDP

begin:

    call TASK_1

    call TASK_2

    ; Выход в DOS
    xor al, al
    mov ah, 4ch
    int 21h

```

```

TESTPC  ENDS
        END start

```

Название файла: lab_exe.asm

```

MYSTACK SEGMENT STACK
        dw 64 DUP(?)
MYSTACK ENDS

```

```

DATA SEGMENT

```

```

PC_STRING db 'IBM PC Type: PC', 0DH, 0AH, '$'
PCXT_STRING db 'IBM PC Type: PC/XT', 0DH, 0AH, '$'
AT_STRING db 'IBM PC Type: AT', 0DH, 0AH, '$'
PS2_MODEL_30_STRING db 'IBM PC Type: PS2 Model 30', 0DH, 0AH, '$'
PS2_MODEL_50_or_60_STRING db 'IBM PC Type: PS2 Model 50 or 60', 0DH, 0AH,
'$'
PS2_MODEL_80_STRING db 'IBM PC Type: PS2 Model 80', 0DH, 0AH, '$'
PCjr_STRING db 'IBM PC Type: PCjr', 0DH, 0AH, '$'
PC_Convertible_STRING db 'IBM PC Type: PC Convertible', 0DH, 0AH, '$'
PC_TYPE_UNKNOWN db 'IBM PC Type Unknown, code: ', 0DH, 0AH, '$'

VERSION_LESS_2_NUM_STRING db 'MS DOS Version: < 2.0', 0DH, 0AH, '$'
VERSION_NUM_STRING db 'MS DOS Version: . ', 0DH, 0AH, '$'
OEM_NUM_STRING db 'MS DOS OEM: ', 0DH, 0AH, '$'
SERIAL_NUM_STRING db 'MS DOS Serial number: ', 0DH, 0AH, '$'

```

```

DATA ENDS

```

```

CODE SEGMENT

```

```

        ASSUME CS:CODE, DS:DATA, SS:MYSTACK

```

```

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
    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

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

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

WRITEMSG PROC NEAR
    push ax
    mov ah, 9
    int 21h
    pop ax
    ret
WRITEMSG ENDP

TASK_1 PROC NEAR

    push ax
    push bx
    push dx
    push es
    push di

    mov ax, 0F000h
    mov es, ax
    mov di, 0FFFFh
    mov al, es:[di]

```

```

cmp al, 0FFh
je pc

cmp al, 0FEh
je pc_xt

cmp al, 0FBh
je pc_xt

cmp al, 0FCh
je at

cmp al, 0FAh
je ps2_model_30

cmp al, 0FCh
je ps2_model_50_or_60

cmp al, 0F8h
je ps2_model_80

cmp al, 0FDh
je pcjr

cmp al, 0F9h
je pc_convertible

```

```

call BYTE_TO_HEX
mov di, offset PC_TYPE_UNKNOWN
mov [di + 26], al
mov [di + 27], ah
mov dx, di
call WRITEMSG
jmp end_task1

```

```

pc:
mov dx, offset PC_STRING
call WRITEMSG
jmp end_task1

```

```

pc_xt:
    mov dx, offset PCXT_STRING
    call WRITEMSG
    jmp end_task1

at:
    mov dx, offset AT_STRING
    call WRITEMSG
    jmp end_task1

ps2_model_30:
    mov dx, offset PS2_MODEL_30_STRING
    call WRITEMSG
    jmp end_task1

ps2_model_50_or_60:
    mov dx, offset PS2_MODEL_50_or_60_STRING
    call WRITEMSG
    jmp end_task1

ps2_model_80:
    mov dx, offset PS2_MODEL_80_STRING
    call WRITEMSG
    jmp end_task1

pcjr:
    mov dx, offset PCjr_STRING
    call WRITEMSG
    jmp end_task1

pc_convertible:
    mov dx, offset PC_Convertible_STRING
    call WRITEMSG
    jmp end_task1

end_task1:
    pop di
    pop es
    pop dx
    pop bx
    pop ax

```



```

        ret

TASK_1 ENDP

TASK_2 PROC NEAR

        push ax
        push bx
        push dx
        push es
        push di

        mov ah, 30h
        int 21h

        ;al - version number
        ;ah - mod number
        ;bh - OEM number
        ;bl:cx - serial number

        cmp al, 0h
        je less_than_2

        push ax
        call BYTE_TO_DEC
        lodsw

        mov di, offset VERSION_NUM_STRING
        mov [di + 15], ah

        pop ax

        xchg ah, al
        call BYTE_TO_DEC
        lodsw
        mov [di + 17], ah
        mov dx, di
        call WRITEMSG
        jmp after_version

less_than_2:

```

```

        mov dx, offset VERSION_LESS_2_NUM_STRING
        call WRITEMSG

after_version:

        mov al, bh
        call BYTE_TO_HEX
        mov di, offset OEM_NUM_STRING
        mov [di + 11], al
        mov [di + 12], ah

        mov dx, di
        call WRITEMSG

        mov al, bl
        call BYTE_TO_HEX
        mov di, offset SERIAL_NUM_STRING
        mov [di + 21], al
        mov [di + 22], ah

        mov ax, cx
        add di, 26
        call WRD_TO_HEX

        mov dx, offset SERIAL_NUM_STRING
        call WRITEMSG

        pop di
        pop es
        pop dx
        pop bx
        pop ax

        ret

TASK_2 ENDP

MAIN PROC FAR
        mov ax, DATA
        mov ds, ax

```

```
call TASK_1

call TASK_2

; 室 DOS
xor al, al
mov ah, 4ch
int 21h

MAIN ENDP

CODE ENDS

END Main
```