

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

Камзолов Н.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.

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».

Функции.

TETR_TO_HEX – переводит 4 последних бита регистра AL в символ шестнадцатеричного числа. Ответ помещается в регистр AL.

BYTE_TO_HEX – переводит значение регистра AL(1 байта) в два символа шестнадцатеричного числа. Ответ помещается в регистр AX.

WRD_TO_HEX – переводит значение регистра AX(2 байта) в символы шестнадцатеричной системы счисления. Ответ кладется по адресу, указанному в DI.

BYTE_TO_DEC – переводит значение регистра AL(1 байт) в символы десятичной системы счисления. Ответ кладется по адресу, указанному в DI.

PC_TYPE – определяет тип PC и выводит строку с названием модели.

PRINT_BUF – печатает в консоль текущее содержимое регистра DX.

MSDOS_VER_PRINT – определяет версию систему, серийный номер OEM и серийный номер пользователя и выводит их на экран.

Последовательность действий.

1. Сразу попадаем на метку `begin`(для модуля `.COM`) и в функцию `MAIN`(для модуля `.EXE`).
2. Здесь поочерёдно вызываются функции `PC_TYPE` и `MSDOS_VER_PRINT`.
3. Завершаем программу с помощью прерывания `4CH`.

Результаты исследования проблем.

Шаг 1. Был написан текст и построен .COM модуль, а также построен «плохой» .EXE, полученный из исходного текста для .COM модуля.

```

ТИП модели IBM PC:FC
Версия MSDOS:5.0
Серийный номер OEM:FF
Серийный номер пользователя:0000000

```

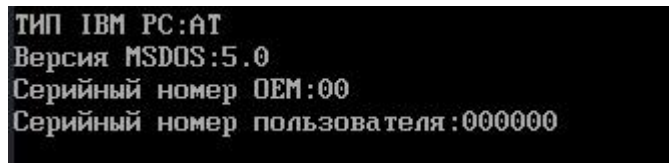
Рисунок 1. Демонстрация корректной работы .COM модуля.

```
D:\>lab1.exe

                цкЗначение регистра AX=
                цкЗначение регистра AX=
T                цкЗначение регистра AX=
I AX=                цкЗначение регистра AX=
```

Рисунок 2. Демонстрация некорректной работы «плохого» .EXE модуля.

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



```
ТИП IBM PC:AT
Версия MSDOS:5.0
Серийный номер OEM:00
Серийный номер пользователя:0000000
```

Рисунок 3. Демонстрация корректной работы «хорошего» .EXE модуля.

Шаг 3. Произведено сравнения исходных текстов для .COM и .EXE модулей. По результатам сравнения можно ответить на вопросы «Отличия исходных текстов COM и EXE программ»:

1. Сколько сегментов должна содержать COM-программа?

Ответ: только один(PSP). Когда COM-программа начинает работать, все сегментные регистры содержат адрес префикса программного сегмента PSP.

2. EXE-программа?

Ответ: файл типа .EXE может содержать ряд сегментов, которые динамически перемещаются в пределах программной области. При этом не менее одного.

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

Ответ: Директива org 100h. Она нужна по причине того, что в начале COM программы определяется 256-байтовый PSP.

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

Ответ: Запрещаются команды, использующие seg, так как отсутствует таблица настроек.

Шаг 4. Произведено сравнение загрузочных модулей .COM и «плохого» .EXE с загрузочным модулем «хорошего» .EXE в шестнадцатеричном виде с помощью FAR. По итогам сравнения можно ответить на 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 хранятся в одном сегменте, при этом размер программы не может превышать 64Кб. Код начинается с адреса

0h, но это происходит, потому что мы устанавливаем смещение для PSP в 100h.

0000000000: E9 68 02 87 AD A0 E7 A5	AD A8 A5 20 E0 A5 A3 A8	щф0Значение реги
0000000010: E1 E2 E0 A0 20 41 58 3D	20 0D 0A 24 92 88 8F 20	стра AX= Л0\$ТИП
0000000020: 49 42 4D 20 50 43 3A 50	43 0D 0A 24 92 88 8F 20	IBM PC:PCЛ0\$ТИП
0000000030: 49 42 4D 20 50 43 3A 50	43 2F 58 54 0D 0A 24 92	IBM PC:PC/XTЛ0\$T
0000000040: 88 8F 20 49 42 4D 20 50	43 3A 41 54 0D 0A 24 92	ИП IBM PC:ATЛ0\$T
0000000050: 88 8F 20 49 42 4D 20 50	43 3A 50 53 32 20 AC AE	ИП IBM PC:PS2 мо
0000000060: A4 A5 AB EC 20 33 30 0D	0A 24 92 88 8F 20 49 42	дель 30Л0\$ТИП IB
0000000070: 4D 20 50 43 3A 50 53 32	20 AC AE A4 A5 AB EC 20	М PC:PS2 модель
0000000080: 38 30 0D 0A 24 92 88 8F	20 49 42 4D 20 50 43 3A	80Л0\$ТИП IBM PC:
0000000090: 50 43 6A 72 0D 0A 24 92	88 8F 20 49 42 4D 20 50	PCjrЛ0\$ТИП IBM P
00000000A0: 43 3A 50 43 20 43 6F 6E	76 65 72 74 69 62 6C 65	C:PC Convertible
00000000B0: 0D 0A 24 92 88 8F 20 AC	AE A4 A5 AB A8 20 49 42	Л0\$ТИП модели IB
00000000C0: 4D 20 50 43 3A 20 20 0D	0A 24 82 A5 E0 E1 A8 EF	М PC: Л0\$Версия
00000000D0: 20 4D 53 44 4F 53 3A 20	2E 20 0D 0A 24 82 A5 E0	MSDOS: . Л0\$Вер
00000000E0: E1 A8 EF 20 4D 53 44 4F	53 20 3C 20 32 2E 30 2E	сия MSDOS < 2.0.
00000000F0: 0D 0A 24 91 A5 E0 A8 A9	AD EB A9 20 AD AE AC A5	Л0\$Серийный номе
0000000100: E0 20 4F 45 4D 3A 20 20	0D 0A 24 91 A5 E0 A8 A9	р OEM: Л0\$Серий
0000000110: AD EB A9 20 AD AE AC A5	E0 20 AF AE AB EC A7 AE	ный номер пользо
0000000120: A2 A0 E2 A5 AB EF 3A 20	20 20 20 20 24 0D 0A 24	вателя: Л0\$
0000000130: 24 0F 3C 09 76 02 04 07	04 30 C3 51 8A E0 E8 EF	\$*<ov0♦♦♦0 QКршя
0000000140: FF 86 C4 B1 04 D2 E8 E8	E6 FF 59 C3 53 8A FC E8	Ж—♦тшщц Y SKNш
0000000150: E9 FF 88 25 4F 88 05 4F	8A C7 E8 DE FF 88 25 4F	щ И%OI*OK ш И%O
0000000160: 88 05 5B C3 50 51 52 32	E4 33 D2 B9 0A 00 F7 F1	И* PQR2ф3т ю ёё
0000000170: 80 CA 30 88 14 4E 33 D2	3D 0A 00 73 F1 3C 00 74	Aл0ИгN3т= sē< t
0000000180: 04 0C 30 88 04 5A 59 58	C3 50 06 57 52 B8 00 F0	♦Q0И♦ZYX P*WRт Ё
0000000190: 8E C0 BF FE FF 26 8A 05	32 E4 3C FF 74 2E 3C FE	0Lт■ &K*2ф< t.<■
00000001A0: 74 30 3C FB 74 2C 3C FC	74 2E 3C FA 74 30 3C F8	t0<v/t,<N*t.<·t0<°
00000001B0: 74 32 3C FD 74 34 3C F9	74 36 BF B3 01 E8 7B FF	t2<xt4<·t6т Ош{
00000001C0: 88 45 12 88 65 13 BA B3	01 EB 2B 90 BA 1C 01 EB	ИЕ†Ие!! Оы+P LОы
00000001D0: 25 90 BA 2C 01 EB 1F 90	BA 3F 01 EB 19 90 BA 4F	%P ,Оы▼P ?Оы↓P О
00000001E0: 01 EB 13 90 BA 6A 01 EB	0D 90 BA 85 01 EB 07 90	Оы!!P jОы↓P ЕОы•P
00000001F0: BA 97 01 EB 01 90 B4 09	CD 21 5A 5F 07 58 C3 50	ЧОыӨP о=!Z_·X P
0000000200: B4 09 CD 21 58 C3 50 53	51 B4 30 CD 21 3C 00 75	о=!X PSQ Ө=!< u
0000000210: 09 BA DD 01 E8 E8 FF EB	1E 90 50 BF CA 01 E8 43	о Ошш ы▲PPт ОшС
0000000220: FF 8B 04 88 65 0D 58 86	E0 E8 38 FF 8B 04 88 65	Л♦Ие,ЛХЖрш8 Л♦Ие
0000000230: 0F BA CA 01 E8 C8 FF 8A	C7 BF F3 01 E8 FC FE 88	* ОшL K еОшN■И
0000000240: 45 13 88 65 14 BA F3 01	E8 B4 FF 8A C3 BF 0B 02	Е!!Иеg еОш K т сӨ
0000000250: E8 E8 FE 88 45 1C 88 65	1D 8B C1 83 C7 21 E8 EB	шш■ИЕLИе↔ЛLГ !шы
0000000260: FE BA 0B 02 E8 98 FF 59	5B 58 C3 E8 1B FF E8 95	■ сӨшш Y[X ш← шX
0000000270: FF 32 C0 B4 4C CD 21		2L L=!

Рисунок 4. Загрузочный модуль .COM в шестнадцатеричном виде.

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

Ответ: Данные и код располагаются в одном сегменте, так как файл в принципе состоит из одного сегмента, что мешает корректно работать программе. Код располагается, начиная с адреса 300h. Начиная с адреса 0h располагается таблица настройки и заголовок.

0000000000:	4D 5A 77 01 03 00 00 00	20 00 00 00 FF FF 00 00	MZw0♥
0000000010:	00 00 00 00 00 01 00 00	3E 00 00 00 01 00 FB 50	0 > 0 VP
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	
0000000300:	E9 68 02 87 AD A0 E7 A5	AD A8 A5 20 E0 A5 A3 A8	щh0Значение реги
0000000310:	E1 E2 E0 A0 20 41 58 3D	20 0D 0A 24 92 88 8F 20	стра AX= J0\$ТИП
0000000320:	49 42 4D 20 50 43 3A 50	43 0D 0A 24 92 88 8F 20	IBM PC:PCJ0\$ТИП
0000000330:	49 42 4D 20 50 43 3A 50	43 2F 58 54 0D 0A 24 92	IBM PC:PC/XTJ0\$T
0000000340:	88 8F 20 49 42 4D 20 50	43 3A 41 54 0D 0A 24 92	ИП IBM PC:ATJ0\$T
0000000350:	88 8F 20 49 42 4D 20 50	43 3A 50 53 32 20 AC AE	ИП IBM PC:PS2 мо
0000000360:	A4 A5 AB EC 20 33 30 0D	0A 24 92 88 8F 20 49 42	дель 30J0\$ТИП IB
0000000370:	4D 20 50 43 3A 50 53 32	20 AC AE A4 A5 AB EC 20	M PC:PS2 модель
0000000380:	38 30 0D 0A 24 92 88 8F	20 49 42 4D 20 50 43 3A	80J0\$ТИП IBM PC:
0000000390:	50 43 6A 72 0D 0A 24 92	88 8F 20 49 42 4D 20 50	PCjrJ0\$ТИП IBM P
00000003A0:	43 3A 50 43 20 43 6F 6E	76 65 72 74 69 62 6C 65	C:PC Convertible
00000003B0:	0D 0A 24 92 88 8F 20 AC	AE A4 A5 AB A8 20 49 42	J0\$ТИП модели IB
00000003C0:	4D 20 50 43 3A 20 20 0D	0A 24 82 A5 E0 E1 A8 EF	M PC: J0\$Версия
00000003D0:	20 4D 53 44 4F 53 3A 20	2E 20 0D 0A 24 82 A5 E0	MSDOS: . J0\$Вер
00000003E0:	E1 A8 EF 20 4D 53 44 4F	53 20 3C 20 32 2E 30 2E	сия MSDOS < 2.0.
00000003F0:	0D 0A 24 91 A5 E0 A8 A9	AD EB A9 20 AD AE AC A5	J0\$Серийный номе
0000000400:	E0 20 4F 45 4D 3A 20 20	0D 0A 24 91 A5 E0 A8 A9	р OEM: J0\$Серий
0000000410:	AD EB A9 20 AD AE AC A5	E0 20 AF AE AB EC A7 AE	ный номер пользо
0000000420:	A2 A0 E2 A5 AB EF 3A 20	20 20 20 20 24 0D 0A 24	вателя: J0\$
0000000430:	24 0F 3C 09 76 02 04 07	04 30 C3 51 8A E0 E8 EF	\$0<ov0♦♦♦0 QКршя
0000000440:	FF 86 C4 B1 04 D2 E8 E8	E6 FF 59 C3 53 8A FC E8	Ж-тшщц Y SKNш
0000000450:	E9 FF 88 25 4F 88 05 4F	8A C7 E8 DE FF 88 25 4F	щ И%OI♦OK ш И%O
0000000460:	88 05 5B C3 50 51 52 32	E4 33 D2 B9 0A 00 F7 F1	И+[]PQR2ф3т 0 ё
0000000470:	80 CA 30 88 14 4E 33 D2	3D 0A 00 73 F1 3C 00 74	A-0ИJN3т=0 së< t
0000000480:	04 0C 30 88 04 5A 59 58	C3 50 06 57 52 B8 00 F0	♦0И♦ZYX P♦WR7 Ё
0000000490:	8E C0 BF FE FF 26 8A 05	32 E4 3C FF 74 2E 3C FE	0 _■ &K♦2ф< t.<■
00000004A0:	74 30 3C FB 74 2C 3C FC	74 2E 3C FA 74 30 3C F8	t0<vt,<N#t.<·t0<°
00000004B0:	74 32 3C FD 74 34 3C F9	74 36 BF B3 01 E8 7B FF	t2<xt4<·t6_1 0ш{
00000004C0:	88 45 12 88 65 13 BA B3	01 EB 2B 90 BA 1C 01 EB	ИЕ\$Ие!! 0ы+P L0ы
00000004D0:	25 90 BA 2C 01 EB 1F 90	BA 3F 01 EB 19 90 BA 4F	%P ,0ы▼P ?0ыJP O
00000004E0:	01 EB 13 90 BA 6A 01 EB	0D 90 BA 85 01 EB 07 90	0ы!!P j0ыJP E0ы•P
00000004F0:	BA 97 01 EB 01 90 B4 09	CD 21 5A 5F 07 58 C3 50	Ч0ы0P о=!Z•X P
0000000500:	B4 09 CD 21 58 C3 50 53	51 B4 30 CD 21 3C 00 75	о=!X PSQ 0=!< u
0000000510:	09 BA DD 01 E8 E8 FF EB	1E 90 50 BF CA 01 E8 43	о 0шш ы▲PP_ 0шC
0000000520:	FF 8B 04 88 65 0D 58 86	E0 E8 38 FF 8B 04 88 65	Л♦ИеJXЖрш8 Л♦Ие
0000000530:	0F BA CA 01 E8 C8 FF 8A	C7 BF F3 01 E8 FC FE 88	0 L0шL K _е0шN■И
0000000540:	45 13 88 65 14 BA F3 01	E8 B4 FF 8A C3 BF 0B 02	E!!ИеJ е0ш K _ё0
0000000550:	E8 E8 FE 88 45 1C 88 65	1D 8B C1 83 C7 21 E8 EB	шш■ИE_LИе↔Л-Г !шы
0000000560:	FE BA 0B 02 E8 98 FF 59	5B 58 C3 E8 1B FF E8 95	■ ё0шш Y[X ш< шX
0000000570:	FF 32 C0 B4 4C CD 21	2 _L=!	

Рисунок 5. Загрузочный модуль «плохого» .EXE в шестнадцатеричном виде.

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

Ответ: Файл состоит из трех сегментов(стека/данных/кода). Код начинается с адреса 290h после таблицы настроек и стека..

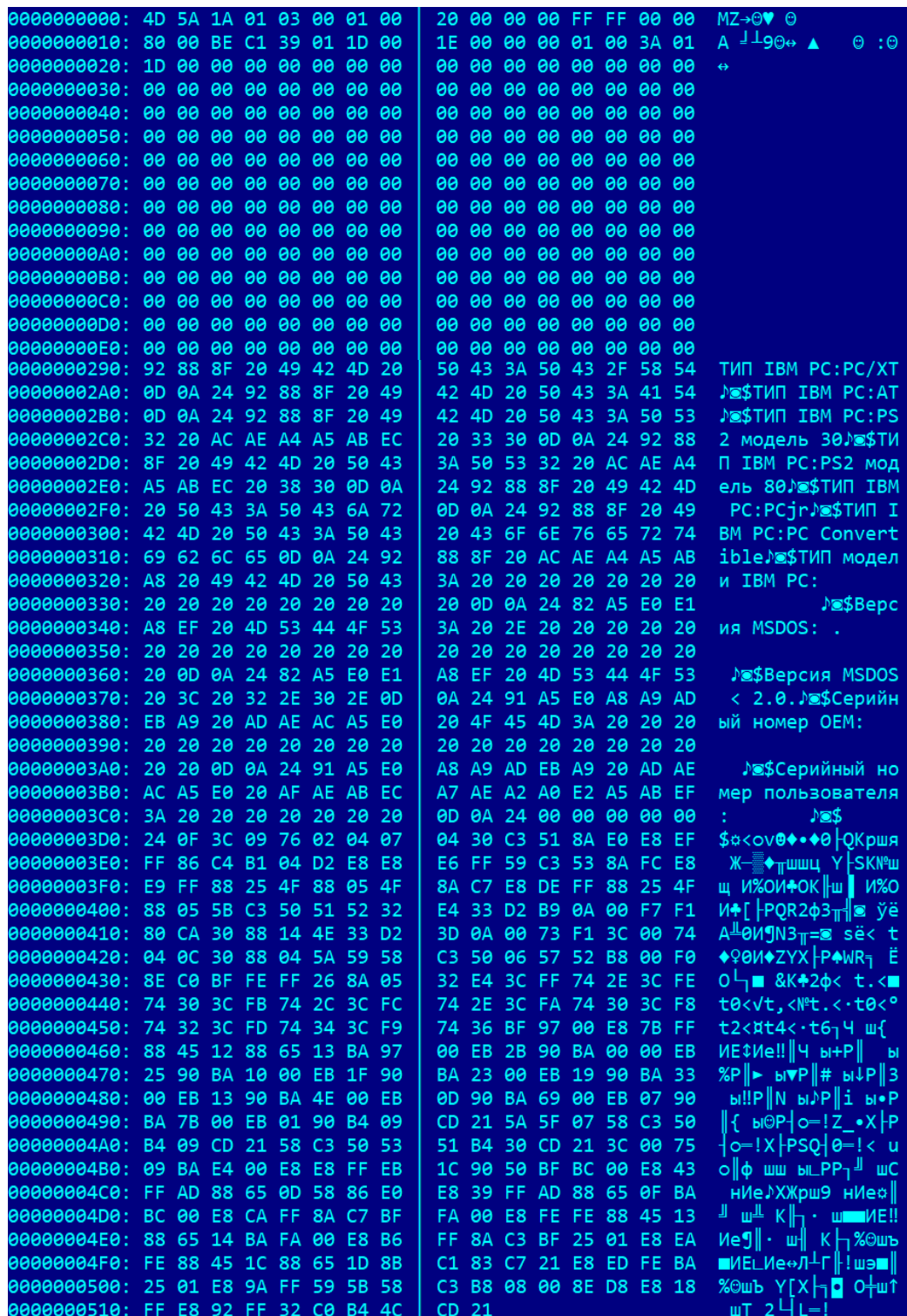


Рисунок 6. Загрузочный модуль «хорошего» .EXE в шестнадцатеричном виде.

Шаг 5. Модуль .COM был открыт в отладчике TD.EXE. По итогам открытия файла в отладчике можно ответить на контрольные вопросы по теме «Загрузка .COM модуля в основную память».

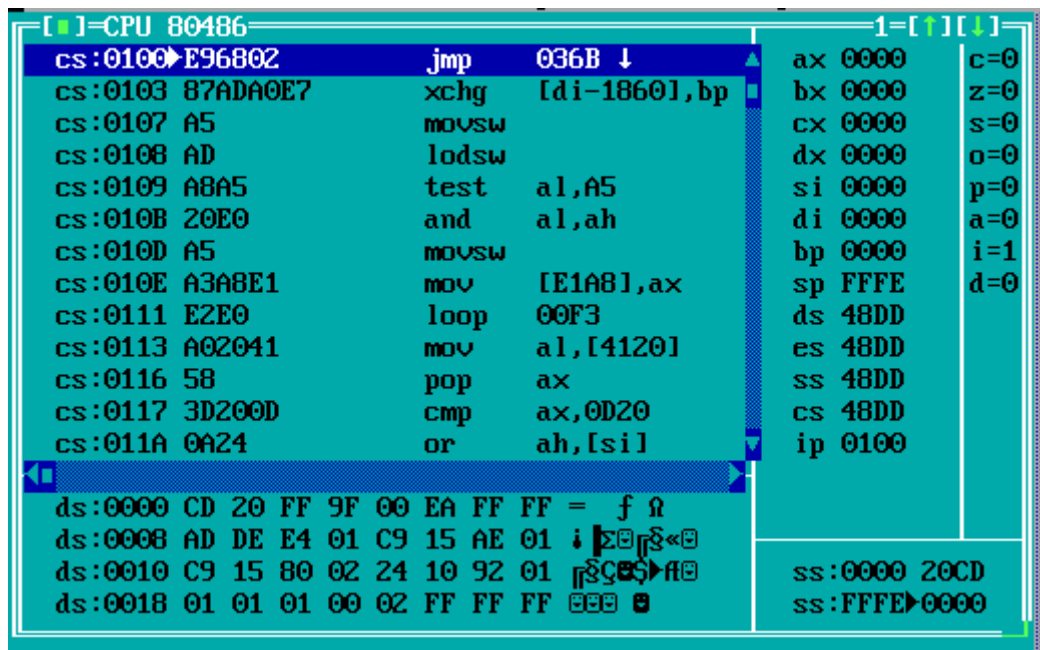


Рисунок 7. Модуль .COM, открытый в отладчике TD.EXE.

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

Ответ: В начале находится свободный участок оперативной памяти, у которого достаточно места для работы программы. Затем по нулевому адресу этого участка помещается PSP. Код располагается, начиная с адреса 100h, сразу после PSP.

2. Что располагается с адреса 0?

Ответ: с нулевого адреса располагается PSP(префикс программного сегмента).

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

Ответ: CS, ES, DS, SS имеют значение 48DD, они указывают на начало PSP.

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

Ответ: COM-программа генерирует стек автоматически при создании исполняемого файла(ассемблировании и линковки). Регистр SP указывает на конец стека(FFFE). Стек может увеличиваться и доходить до сегмента кода, что может сломать программу, именно поэтому размер COM файла ограничен.

Шаг 6. «Хороший» модуль .EXE был открыт в отладчике TD.EXE. По итогам открытия файла в отладчике можно ответить на контрольные вопросы по теме «Загрузка «хорошего» .EXE модуля в основную память».

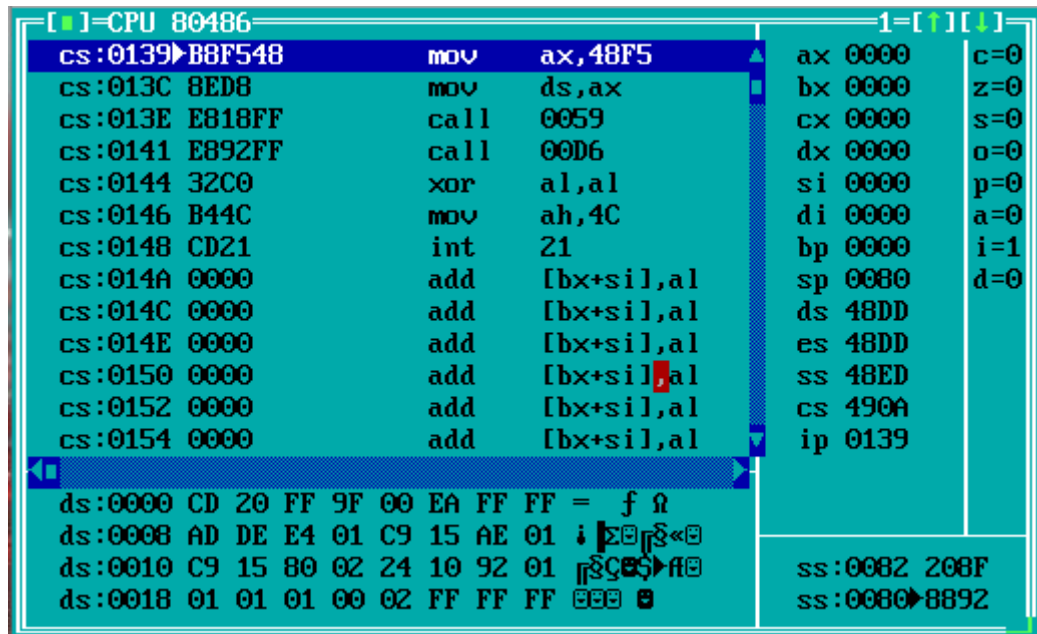


Рисунок 8. «Хороший» модуль .EXE, открытый в отладчике TD.EXE.

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

Ответ: Аналогично .COM модулю .EXE загружается со смещением относительно PSP в 100h. Исходя из значений в заголовке, происходит перемещение адресов сегментов.

DS и ES имеют значения 48DD, CS – 490A, SS – 48ED.

2. На что указывают регистры DS и ES?

Ответ: Они указывают на начало сегмента PSP.

3. Как определяется стек?

Ответ: С помощью директивы .STACK, в которой указывается размер стека. При этом SP указывает на конец стека, а SS на начало.

4. Как определяется точка входа?

Ответ: С помощью директивы .END.

Выводы.

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

lab1_com.asm:

```
TESTPC  SEGMENT

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

org 100h


start: jmp begin


STRING db 'Значение регистра AX= ', 0DH, 0AH, '$'
IBM_TYPE_PC db 'ТИП IBM PC:PC', 0DH, 0AH, '$'
IBM_TYPE_PC_XT db 'ТИП IBM PC:PC/XT', 0DH, 0AH, '$'
IBM_TYPE_AT db 'ТИП IBM PC:AT', 0DH, 0AH, '$'
IBM_TYPE_PS2_30 db 'ТИП IBM PC:PS2 модель 30', 0DH, 0AH, '$'
IBM_TYPE_PS2_80 db 'ТИП IBM PC:PS2 модель 80', 0DH, 0AH, '$'
IBM_TYPE_PC_JR db 'ТИП IBM PC:PCjr', 0DH, 0AH, '$'
IBM_TYPE_PC_CONV db 'ТИП IBM PC:PC Convertible', 0DH, 0AH, '$'
IBM_TYPE_UNKNOWN db 'ТИП модели IBM PC: ', 0DH, 0AH, '$'
SYSTEM_VER db 'Версия MSDOS: . ', 0DH, 0AH, '$'
SYSTEM_VER_LOWER_2_0 db 'Версия MSDOS < 2.0.', 0DH, 0AH, '$'
OEM_NUMBER db 'Серийный номер OEM: ', 0DH, 0AH, '$'
SERIAL_NUMBER db 'Серийный номер пользователя:          $', 0DH, 0AH,
'$'


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 ax
    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
    pop ax
    ret

BYTE_TO_DEC endp

PC_TYPE proc near

    push ax
    push es
    push di
    push dx

    mov ax, 0f000h
    mov es, ax
    mov di, 0ffffh
    mov al, es:[di]

    xor ah, ah
    cmp al, 0ffh
    je pc_print

    cmp al, 0feh

```

```

        je pcxt_print

        cmp al, 0fbh
        je pcxt_print

        cmp al, 0fch
        je at_print

        cmp al, 0fah
        je ps2_model_30_print

        cmp al, 0f8h
        je ps2_model_80_print

        cmp al, 0fdh
        je pcjr_print

        cmp al, 0f9h
        je pc_conv_print

        mov di, offset IBM_TYPE_UNKNOWN
        call BYTE_TO_HEX
        mov [di+18], al
        mov [di+19], ah
        mov dx, offset IBM_TYPE_UNKNOWN
        jmp print_dx

pc_print:
        mov dx, offset IBM_TYPE_PC
        jmp print_dx
pcxt_print:
        mov dx, offset IBM_TYPE_PC_XT
        jmp print_dx
at_print:
        mov dx, offset IBM_TYPE_AT
        jmp print_dx
ps2_model_30_print:
        mov dx, offset IBM_TYPE_PS2_30

```

```

        jmp print_dx
ps2_model_80_print:
        mov dx, offset IBM_TYPE_PS2_80
        jmp print_dx
pcjr_print:
        mov dx, offset IBM_TYPE_PC_JR
        jmp print_dx
pc_conv_print:
        mov dx, offset IBM_TYPE_PC_CONV
        jmp print_dx
print_dx:
        mov ah, 9h
        int 21h

        pop dx
        pop di
        pop es
        pop ax
ret
PC_TYPE endp

PRINT_BUF proc near
        push ax
        mov ah, 9h
        int 21h
        pop ax
        ret
PRINT_BUF endp

MSDOS_VER_PRINT proc near
        push ax
        push bx
        push cx

        mov ah, 30h
        int 21h

        ;xor ax, ax

```



```

    cmp al, 0h
    jne greater_0
    mov dx, offset SYSTEM_VER_LOWER_2_0
    call PRINT_BUF
    jmp oem_print

greater_0:
    push ax
    mov di, offset SYSTEM_VER
    call BYTE_TO_DEC
    mov ax, ds:si
    mov [di+13], ah
    pop ax
    xchg ah, al
    call BYTE_TO_DEC
    mov ax, ds:si
    mov [di+15], ah
    mov dx, offset SYSTEM_VER
    call PRINT_BUF

oem_print:
    mov al, bh
    mov di, offset OEM_NUMBER
    call BYTE_TO_HEX
    mov [di+19], al
    mov [di+20], ah
    mov dx, offset OEM_NUMBER
    call PRINT_BUF

    mov al, bl
    mov di, offset SERIAL_NUMBER
    call BYTE_TO_HEX
    mov [di+28], al
    mov [di+29], ah
    mov ax, cx
    add di, 33
    call WRD_TO_HEX

```

```

        mov dx, offset SERIAL_NUMBER
        call PRINT_BUF

        pop cx
        pop bx
        pop ax
        ret
MSDOS_VER_PRINT endp

```

begin:

```

        call PC_TYPE

        call MSDOS_VER_PRINT

        xor al, al
        mov ah, 4ch
        int 21h

TESTPC  ENDS
        END start

```

lab1_exe.asm

```

ASTACK  SEGMENT  STACK
        DW 64 DUP(?)
ASTACK  ENDS

```

```

DATA  SEGMENT
IBM_TYPE_PC DB 'ТИП IBM PC:PC', 0DH, 0AH, '$'
IBM_TYPE_PC_XT DB 'ТИП IBM PC:PC/XT', 0DH, 0AH, '$'
IBM_TYPE_AT DB 'ТИП IBM PC:AT', 0DH, 0AH, '$'
IBM_TYPE_PS2_30 DB 'ТИП IBM PC:PS2 МОДЕЛЬ 30', 0DH, 0AH, '$'
IBM_TYPE_PS2_80 DB 'ТИП IBM PC:PS2 МОДЕЛЬ 80', 0DH, 0AH, '$'
IBM_TYPE_PC_JR DB 'ТИП IBM PC:PCJR', 0DH, 0AH, '$'
IBM_TYPE_PC_CONV DB 'ТИП IBM PC:PC CONVERTIBLE', 0DH, 0AH, '$'
IBM_TYPE_UNKNOWN DB 'ТИП МОДЕЛИ IBM PC:
0AH, '$'

```

```

SYSTEM_VER DB 'ВЕРСИЯ MSDOS: .', 0DH, 0AH,
'$'
SYSTEM_VER_LOWER_2_0 DB 'ВЕРСИЯ MSDOS < 2.0.', 0DH, 0AH, '$'
OEM_NUMBER DB 'СЕРИЙНЫЙ НОМЕР OEM:', 0DH,
0AH, '$'
SERIAL_NUMBER DB 'СЕРИЙНЫЙ НОМЕР ПОЛЬЗОВАТЕЛЯ:', 0DH, 0AH,
'$'
DATA ENDS

```

```

CODE SEGMENT

```

```

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

```

```

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 AX
        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
        POP AX
        RET

BYTE_TO_DEC ENDP

PC_TYPE PROC NEAR

        PUSH AX
        PUSH ES
        PUSH DI
        PUSH DX

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

        XOR AH, AH
        CMP AL, 0FFH
        JE PC_PRINT

        CMP AL, 0FEH
        JE PCXT_PRINT

        CMP AL, 0FBH
        JE PCXT_PRINT

        CMP AL, 0FCH
        JE AT_PRINT

        CMP AL, 0FAH
        JE PS2_MODEL_30_PRINT

        CMP AL, 0F8H
        JE PS2_MODEL_80_PRINT

```

```

    CMP AL, 0FDH
    JE PCJR_PRINT

    CMP AL, 0F9H
    JE PC_CONV_PRINT

    MOV DI, OFFSET IBM_TYPE_UNKNOWN
    CALL BYTE_TO_HEX
    MOV [DI+18], AL
    MOV [DI+19], AH
    MOV DX, OFFSET IBM_TYPE_UNKNOWN
    JMP PRINT_DX

PC_PRINT:
    MOV DX, OFFSET IBM_TYPE_PC
    JMP PRINT_DX

PCXT_PRINT:
    MOV DX, OFFSET IBM_TYPE_PC_XT
    JMP PRINT_DX

AT_PRINT:
    MOV DX, OFFSET IBM_TYPE_AT
    JMP PRINT_DX

PS2_MODEL_30_PRINT:
    MOV DX, OFFSET IBM_TYPE_PS2_30
    JMP PRINT_DX

PS2_MODEL_80_PRINT:
    MOV DX, OFFSET IBM_TYPE_PS2_80
    JMP PRINT_DX

PCJR_PRINT:
    MOV DX, OFFSET IBM_TYPE_PC_JR
    JMP PRINT_DX

PC_CONV_PRINT:
    MOV DX, OFFSET IBM_TYPE_PC_CONV
    JMP PRINT_DX

PRINT_DX:
    MOV AH, 9H
    INT 21H

```

```

        POP DX
        POP DI
        POP ES
        POP AX
RET
PC_TYPE ENDP

PRINT_BUF PROC NEAR
        PUSH AX
        MOV AH, 9H
        INT 21H
        POP AX
        RET
PRINT_BUF ENDP

MSDOS_VER_PRINT PROC NEAR
        PUSH AX
        PUSH BX
        PUSH CX

        MOV AH, 30H
        INT 21H

        ;XOR AX, AX

        CMP AL, 0H
        JNE GREATER_0
        MOV DX, OFFSET SYSTEM_VER_LOWER_2_0
        CALL PRINT_BUF
        JMP OEM_PRINT

GREATER_0:
        PUSH AX
        MOV DI, OFFSET SYSTEM_VER
        CALL BYTE_TO_DEC
        LODSW
        MOV [DI+13], AH
        POP AX

```



```

        XCHG AH, AL
        CALL BYTE_TO_DEC
        LODSW
        MOV [DI+15], AH
        MOV DX, OFFSET SYSTEM_VER
        CALL PRINT_BUF

OEM_PRINT:
        MOV AL, BH
        MOV DI, OFFSET OEM_NUMBER
        CALL BYTE_TO_HEX
        MOV [DI+19], AL
        MOV [DI+20], AH
        MOV DX, OFFSET OEM_NUMBER
        CALL PRINT_BUF

        MOV AL, BL
        MOV DI, OFFSET SERIAL_NUMBER
        CALL BYTE_TO_HEX
        MOV [DI+28], AL
        MOV [DI+29], AH
        MOV AX, CX
        ADD DI, 33
        CALL WRD_TO_HEX
        MOV DX, OFFSET SERIAL_NUMBER
        CALL PRINT_BUF

        POP CX
        POP BX
        POP AX
        RET

MSDOS_VER_PRINT ENDP

MAIN PROC FAR
        MOV  AX, DATA
        MOV  DS, AX
        CALL PC_TYPE

```

```
CALL MSDOS_VER_PRINT
```

```
XOR AL, AL
```

```
MOV AH, 4CH
```

```
INT 21H
```

```
MAIN ENDP
```

```
CODE ENDS
```

```
END MAIN
```