

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

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

Студент гр.0382

Бочаров Г.С.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

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

Задание.

1. Написать текст исходного .COM модуля, который определяет тип РС и версию системы. Результатом будет “хороший” .COM модуль, а также “плохой” .EXE, полученный из исходного текста для .COM модуля.

2. Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1. Результатом будет “хороший” .EXE.

3. Сравнить исходные тексты для .COM и .EXE модулей. Ответить на контрольные вопросы “Отличия исходных текстов COM и EXE программ”.

4. Запустить FAR и открыть файл 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”.

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

6. Открыть отладчик TD.EXE и загрузить “хороший” .EXE. Ответить на контрольные вопросы “Загрузка “хорошего” EXE модуля в основную память”.

7. Оформить отчет в соответствии с требованиями. В отчете необходимо привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей – в отладчике.

Выполнение работы:

1. При выполнении работы был использован шаблон и описанные в нем функции из методического пособия.

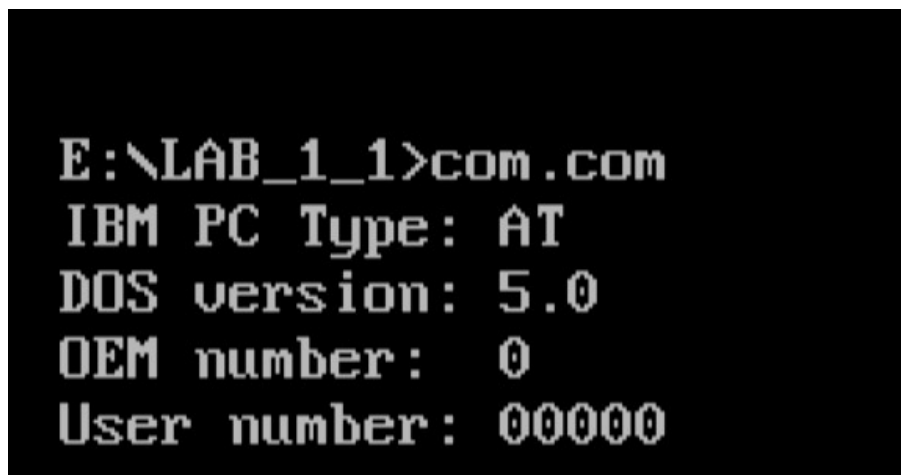
Для получения информации о типе PC и версии системы были написаны следующие процедуры:

`print_pc_type` — определяет тип PC, сравнивая значение по адресу 0F000:0FFFFh с возможными вариантами типа PC.

`print_dos_version` — определяет версию системы с помощью функции 30h и прерывания 21h. На выходе в регистре AL лежит номер основной версии, в AH – номер модификации, в BH – серийный номер OEM, в BL:CH – 24-битовый серийный номер пользователя.

Функции `print_oem_serial_number` , `print_user_serial_number` пользуются значениями регистров, полученными в результате работы процедуры `print_dos_version` и преобразуют их для последующего вывода.

В результате шага имеем “хороший” .COM модуль и “плохой” .EXE модуль. Выводы, полученные при их запуске, представлены на рисунке 1 и рисунке 2 соответственно.

A screenshot of a DOS command prompt window with a black background and white text. The prompt is 'E:\LAB_1_1>'. The command entered is 'com.com'. The output displayed is: 'IBM PC Type: AT', 'DOS version: 5.0', 'OEM number: 0', and 'User number: 00000'.

```
E:\LAB_1_1>com.com
IBM PC Type: AT
DOS version: 5.0
OEM number: 0
User number: 00000
```

Рисунок 1 – результат запуска модуля com.com

```
E:\LAB_1_1>com.exe

5 0
0
00000  IBM PC Type: PC
                                IBM PC Type: PC
                                IBM PC Type: PC
```

Рисунок 2 – результат запуска модуля com.exe

2. Для написания “хорошего” .EXE модуля программа была разбита на сегменты кода, данных и стека.

```
E:\LAB_1_1>exe.exe
IBM PC Type: AT
DOS version: 5.0
OEM number: 0
User number: 00000
```

Рисунок 3 – результат запуска модуля exe.exe

3. Сравнивая исходные тексты для .COM и .EXE модулей, ответим на контрольные вопросы “Отличия исходных текстов COM и EXE программ”:

1) Сколько сегментов должна содержать COM-программа?

Один сегмент (содержит и код, и данные; стек же генерируется сам).

2) EXE-программа?

Сегмент кода является обязательным. Можно добавить сегмент данных и стека..

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

Поскольку первые 256 байт занимает блок данных PSP, нужно, чтобы адресация имела смещение в 256 байт от нулевого адреса. Это достигается директивой ORG 100h.

Также директива ASSUME указывает, с каким сегментом ассоциировать регистры CS и DS.

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

Т.к. в COM-программе все сегментные регистры определяются в момент запуска программы, а не в момент компиляции (ассемблирования), то мы не можем использовать команды с указанием сегментов. Например: `mov ax, DATA`

4. Рассматривая шестнадцатеричные виды модулей, ответим на контрольные вопросы “Отличия форматов файлов COM и EXE модулей”:

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

.COM модуль состоит из одного сегмента, содержащего и код, и данные. Код начинается с адреса 0h (но при загрузке модуля устанавливается смещение в 100h).

Рисунок 4 – “хороший” .COM в 16-м виде

2) Какова структура файла “плохого” EXE? С какого адреса располагается код? Что располагается с адреса 0?

«Плохой» .EXE модуль, состоит из одного сегмента. Код располагается с адреса 300h (512 байт занимает заголовок и relocation table, 256 - смещение). Модуль в шестнадцатеричном виде представлен на рисунке 5.

00000000: 4D 5A 0E 01 03 00 00 00	20 00 00 00 FF FF 00 00	MZD0	yy
000000010: 00 00 68 C3 00 01 00 00	1E 00 00 00 01 00 00 00	hA 0	0
000000020: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000030: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000040: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000050: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000060: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000070: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000080: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
000000090: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000A0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
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 F9 01 49 42 4D 20 50	43 20 54 79 70 65 3A 20	éù0IBM PC Type:	
0000000310: 50 43 0D 0A 24 49 42 4D	20 50 43 20 54 79 70 65	PC:IBM PC Type	
0000000320: 3A 20 50 43 2F 58 54 0D	0A 24 49 42 4D 20 50 43	: PC/XTIBM PC	
0000000330: 20 54 79 70 65 3A 20 41	54 0D 0A 24 49 42 4D 20	Type: ATIBM	
0000000340: 50 43 20 54 79 70 65 3A	20 50 53 32 20 6D 6F 64	PC Type: PS2 mod	
0000000350: 65 6C 20 33 30 0D 0A 24	49 42 4D 20 50 43 20 54	el 30IBM PC T	
0000000360: 79 70 65 3A 20 50 53 32	20 6D 6F 64 65 6C 20 35	ype: PS2 model 5	
0000000370: 30 20 6F 72 20 36 30 0D	0A 24 49 42 4D 20 50 43	0 or 60IBM PC	
0000000380: 20 54 79 70 65 3A 20 50	53 32 20 6D 6F 64 65 6C	Type: PS2 model	
0000000390: 20 38 30 0D 0A 24 49 42	4D 20 50 43 20 54 79 70	80IBM PC Typ	
00000003A0: 65 3A 20 50 3F 6A 72 0D	0A 24 49 42 4D 20 50 43	e: P?jrIBM PC	
00000003B0: 20 54 79 70 65 3A 20 50	43 20 43 6F 6E 76 65 72	Type: PC Conver	
00000003C0: 74 69 62 6C 65 0D 0A 24	44 4F 53 20 76 65 72 73	tibleDOS vers	
00000003D0: 69 6F 6E 3A 20 20 2E 20	20 0D 0A 24 4F 45 4D 20	ion: . OEM	
00000003E0: 6E 75 6D 62 65 72 3A 20	20 20 0D 0A 24 55 73 65	number: Use	
00000003F0: 72 20 6E 75 6D 62 65 72	3A 20 20 20 20 20 20 20	r number:	
0000000400: 0D 0A 24 B4 09 CD 21 C3	24 0F 3C 09 76 02 04 07	!oI!A\$ocov0	
0000000410: 04 30 C3 51 8A E0 E8 EF	FF 86 C4 B1 04 D2 E8 E8	0AQ\$aeiÿtA±0èè	
0000000420: E6 FF 59 C3 53 8A FC E8	E9 FF 88 25 4F 88 05 4F	æÿYAS\$üeeÿ"%0*0	
0000000430: 8A C7 E8 DE FF 88 25 4F	88 05 5B C3 51 52 50 32	\$Cèbÿ"%0*[AQRP2	
0000000440: E4 33 D2 B9 0A 00 F7 F1	80 CA 30 88 14 4E 33 D2	äz0¹ ÷ñ€E0`9N30	
0000000450: 3D 0A 00 73 F1 3C 00 74	04 0C 30 88 04 58 5A 59	= sñ< t00`XZY	
0000000460: C3 B8 00 F0 8E C0 26 A0	FE FF 3C FF 74 1C 3C FE	Ä. ðZÄ& bÿ<ÿtL<b	
0000000470: 74 1E 3C FB 74 1A 3C FC	74 1C 3C FA 74 1E 3C F8	t▲<ÿt><ÿtL<ÿt▲<ø	
0000000480: 74 20 3C FD 74 22 3C F9	74 24 BA 03 01 EB 25 90	t <ÿt"<ÿt\$0♥0%0	
0000000490: BA 90 02 EB 1F 90 BA 2A	01 EB 19 90 BA 3C 01 EB	0000♥00*000000<00	
00000004A0: 13 90 BA 7A 01 EB 0D 90	BA 96 01 EB 07 90 BA AA	!!00z00000-0000000	
00000004B0: 01 EB 01 90 E8 4C FF C3	B4 30 CD 21 BE D5 01 E8	000000LÿÄ`0Í!%000	
00000004C0: 7A FF 8A C4 83 C6 03 E8	72 FF BA C8 01 E8 33 FF	zÿ\$Äf4♥erÿ00003ÿ	
00000004D0: C3 BE E9 01 8A C7 E8 63	FF BA DC 01 E8 24 FF C3	Ä%00\$Cècÿ0000ÿÿÄ	
00000004E0: BF ED 01 83 C7 11 8B C1	E8 39 FF 8A C3 E8 23 FF	zi0fC<Ä0ÿ\$Ä0#ÿ	

Рисунок 5 – “плохой” .EXE в 16-м виде (начало и конец)

3) Какова структура файла “хорошего” EXE? Чем он отличается от файла “плохого” EXE?

Имеет несколько сегментов, в начале модуля располагается заголовок и relocation table (200h байт), затем располагаются стэк, данные и код.

00000000: 4D 5A 10 00 07 00 01 00	20 00 00 00 FF FF 00 00	MZ	• 0	пл
00000010: 00 08 FE 8D F9 00 90 00	1E 00 00 00 01 00 FA 00	иКш	h	▲ 0 ь
00000020: 90 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	h		
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			
000000940: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
000000950: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
000000960: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
000000970: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
000000980: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
000000990: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
0000009A0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
0000009B0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
0000009C0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
0000009D0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
0000009E0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
0000009F0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
000000A00: 49 42 4D 20 50 43 20 54	79 70 65 3A 20 50 43 0D	IBM PC Type: PC		
000000A10: 0A 24 49 42 4D 20 50 43	20 54 79 70 65 3A 20 50	IBM PC Type: P		
000000A20: 43 2F 58 54 0D 0A 24 49	42 4D 20 50 43 20 54 79	C/XT IBM PC Ty		
000000A30: 70 65 3A 20 41 54 0D 0A	24 49 42 4D 20 50 43 20	pe: AT IBM PC		
000000A40: 54 79 70 65 3A 20 50 53	32 20 6D 6F 64 65 6C 20	Type: PS2 model		
000000A50: 33 30 0D 0A 24 49 42 4D	20 50 43 20 54 79 70 65	30 IBM PC Type		
000000A60: 3A 20 50 53 32 20 6D 6F	64 65 6C 20 35 30 20 6F	: PS2 model 50 o		
000000A70: 72 20 36 30 0D 0A 24 49	42 4D 20 50 43 20 54 79	r 60 IBM PC Ty		
000000A80: 70 65 3A 20 50 53 32 20	6D 6F 64 65 6C 20 38 30	pe: PS2 model 80		
000000A90: 0D 0A 24 49 42 4D 20 50	43 20 54 79 70 65 3A 20	IBM PC Type:		
000000AA0: 50 3F 6A 72 0D 0A 24 49	42 4D 20 50 43 20 54 79	P?jr IBM PC Ty		
000000AB0: 70 65 3A 20 50 43 20 43	6F 6E 76 65 72 74 69 62	pe: PC Convertib		
000000AC0: 6C 65 0D 0A 24 44 4F 53	20 76 65 72 73 69 6F 6E	le DOS version		
000000AD0: 3A 20 20 2E 20 20 0D 0A	24 4F 45 4D 20 6E 75 6D	: . OEM num		
000000AE0: 62 65 72 3A 20 20 20 0D	0A 24 55 73 65 72 20 6E	ber: User n		
000000AF0: 75 6D 62 65 72 3A 20 20	20 20 20 20 20 0D 0A 24	umber: OEM		
000000B00: B4 09 CD 21 C3 24 0F 3C	09 76 02 04 07 04 30 C3	roHIG\$ecov0++0G		
000000B10: 51 8A E0 E8 EF FF 86 C4	B1 04 D2 E8 E8 E6 FF 59	QьaипяTД±+ТиижаY		
000000B20: C3 53 8A FC E8 E9 FF 88	25 4F 88 05 4F 8A C7 E8	ГSльийл€%O€+Oл3и		
000000B30: DE FF 88 25 4F 88 05 5B	C3 51 52 50 32 E4 33 D2	Юя€%O€+[ГQRP2д3Т		
000000B40: B9 0A 00 F7 F1 80 CA 30	88 14 4E 33 D2 3D 0A 00	иW чсbK0€JN3T-ш		
000000B50: 73 F1 3C 00 74 04 0C 30	88 04 58 5A 59 C3 B8 00	sc< t+0€+XZYГё		
000000B60: F0 8E C0 26 A0 FE FF 3C	FF 74 1C 3C FE 74 1E 3C	pTbA& юя<яtL<юtA<		
000000B70: FB 74 1A 3C FC 74 1C 3C	FA 74 1E 3C F8 74 20 3C	yt-<ьtL<ьtA<wt <		
000000B80: FD 74 22 3C F9 74 24 BA	00 00 FB 25 90 BA 8D 00	at"<wt\$e л%HeK		
000000B90: EB 1F 90 BA 27 00 EB 19	90 BA 39 00 EB 13 90 BA	лvHe' л4He9 л!!He		
000000BA0: 77 00 EB 0D 90 BA 93 00	EB 07 90 BA A7 00 EB 01	w л4He" л•Heб л@		
000000BB0: 90 E8 4C FF C3 B4 30 CD	21 BE D2 00 E8 7A FF 8A	ђиLяГr0H!sT изяь		
000000BC0: C4 83 C6 03 E8 72 FF BA	C5 00 E8 33 FF C3 BE E6	ДfЖVирлE и3лГсж		
000000BD0: 00 8A C7 E8 63 FF BA D9	00 E8 24 FF C3 BF EA 00	л3исясш и3яГiк		
000000BE0: 83 C7 11 8B C1 E8 39 FF	8A C3 E8 23 FF BF F7 00	f3+«би9яьђи#яiч		
000000BF0: 89 05 BA EA 00 E8 08 FF	C3 B8 80 00 8E D8 E8 5D	ш+«к иQаГёb Tllm]		
000000C00: FF E8 B1 FF E8 C7 FF E8	D3 FF 32 C0 B4 4C CD 21	яи±ия3лиYя2ArLH!		

Рисунок 6 – “хороший” .EXE в 16-м виде

5. Загружая .COM модуль в отладчике TD.EXE, ответим на контрольные вопросы “Загрузка COM модуля в основную память”. Результат загрузки программы в отладчик представлены на рисунке 7.

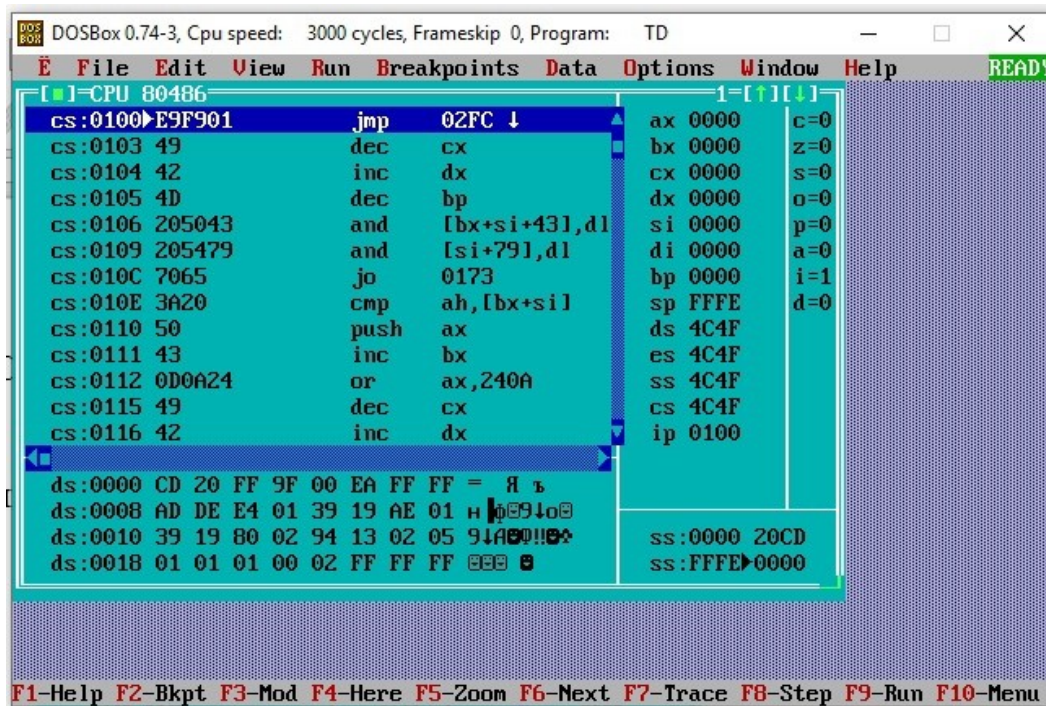


Рисунок 7 – загруженный в отладчик “хороший” .COM модуль

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

Находится адрес участка свободной памяти, достаточного для загрузки программы. Создаётся блок памяти для PSP и программы. Происходит загрузка файла начиная с адреса 100h. Код же располагается с адреса CS:0100h.

2) Что располагается с адреса 0?

Сегмент PSP.

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

Сегментные регистры CS, DS, ES, SS указывают на PSP (4C4Fh)

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

Стек занимает весь программный сегмент. SS указывает на начало сегмента 4C4Fh; SP – последний адрес FFFh. Таким образом, адреса 4C4F:0000h – 4C4F:FFFh.

6. Загружая “хороший” .EXE модуль в отладчике TD.EXE, ответим на контрольные вопросы “Загрузка “хорошего” EXE модуля в основную память”.

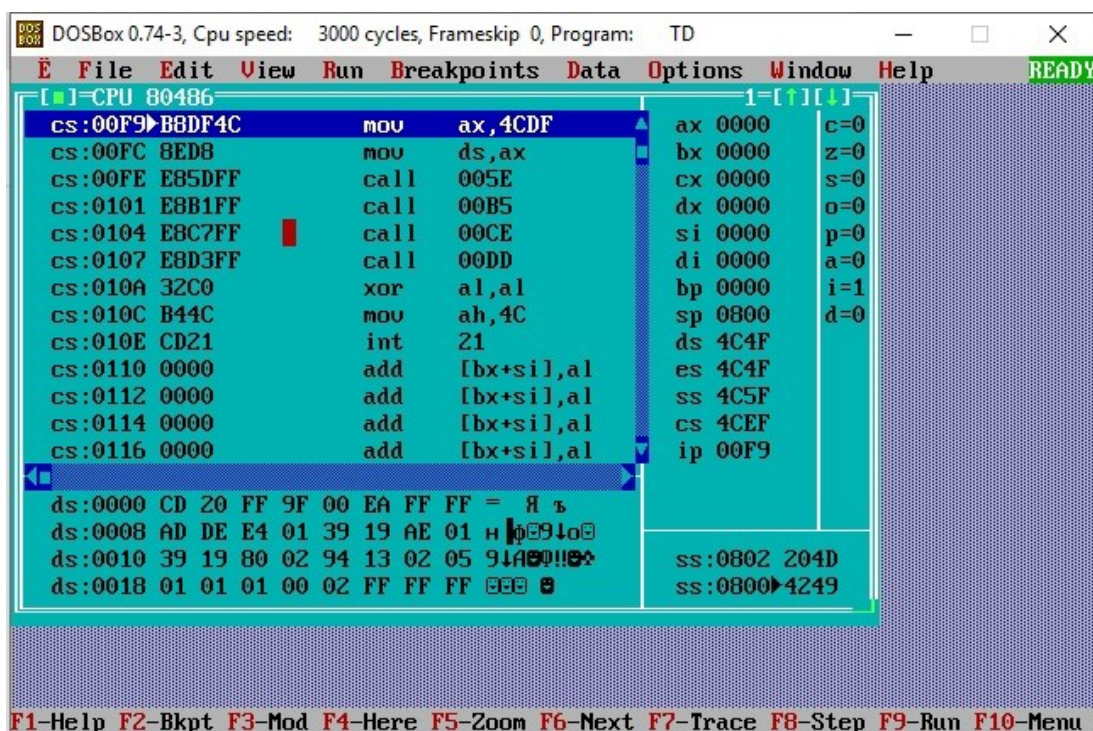


Рисунок 8 – загруженный в отладчик “хороший” .EXE модуль

1) Как загружается “хороший” EXE? Какие значения имеют сегментные регистры?

Загружается PSP. DS и ES устанавливаются на начало сегмента PSP(4C4F), SS – на начало сегмента стека, CS – на начало сегмента команд. В IP загружается смещение точки входа в программу.

2) На что указывают регистры DS и ES?

На начало PSP.

3) Как определяется стек?

В программе выделяется память для сегмент стека. При помощи директивы `assume` регистр `SS` устанавливает на начало сегмента стека. `SP` – указывает на конец стека.

4) Как определяется точка входа?

Точка входа определяется меткой после директивы `END`.

Исходный код программы см. в приложении А.

Выводы.

В ходе работы были исследованы различия в структурах исходных текстов модулей типов `.COM` и `.EXE`, в структурах файлов загрузочных моделей и способов их загрузки в основную память; была написана программа, выводящую информацию о типе `PC` и версии системы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: com.asm

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

start:
    jmp begin

data:
    PC db          'IBM PC Type: PC',0DH, 0AH,'$'
    PCXT db 'IBM PC Type: PC/XT', 0DH, 0AH,'$'
    AT db          'IBM PC Type: AT', 0DH, 0AH,'$'
    PS230 db       'IBM PC Type: PS2 model 30', 0DH, 0AH,'$'
    PS25060 db     'IBM PC Type: PS2 model 50 or 60', 0DH, 0AH,'$'
    PS280 db       'IBM PC Type: PS2 model 80', 0DH, 0AH,'$'
    PCjr db 'IBM PC Type: P?jr', 0DH, 0AH,'$'
    PCC db 'IBM PC Type: PC Convertible', 0DH, 0AH,'$'
    DOSV db 'DOS version: . ', 0DH, 0AH,'$'
    OEM db 'OEM number: ', 0DH, 0AH,'$'
    USERNAME db 'User number: ', 0DH, 0AH,'$'

print PROC NEAR
    mov ah, 09h
    int 21h
    ret
print ENDP

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

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
    push ax
    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 ax
    pop DX
    pop CX

```

```

    ret
byte_to_dec ENDP

print_pc_type PROC NEAR
    mov ax, 0F000h
    mov es, ax
    mov al, es:[0FFFFh]

    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_30

    cmp al, 0f8h
    je ps2_80

    cmp al, 0fdh
    je pc_jr

    cmp al, 0f9h
    je pc_conv
_pc:
    mov dx, offset PC
    jmp _out
pc_xt:
    mov dx, offset PC_XT
    jmp _out
_at:
    mov dx, offset AT
    jmp _out
ps2_30:
    mov dx, offset PS230
    jmp _out
ps2_80:
    mov dx, offset PS280
    jmp _out
pc_jr:
    mov dx, offset PCjr

```

```

        jmp _out
pc_conv:
        mov dx, offset PCC
        jmp _out
_out:
        call print
        ret
print_pc_type ENDP

```

```

print_dos_version PROC NEAR
        mov ah, 30h
        int 21h

        mov si, offset DOSV + 13
        call byte_to_dec

        mov al, ah
        add si, 3
        call byte_to_dec
        mov dx, offset DOSV
        call print
        ret
print_dos_version ENDP

```

```

print_oem_serial_number PROC NEAR
        mov si, offset OEM + 13
        mov al, bh
        call byte_to_dec
        mov dx, offset OEM
        call print
        ret
print_oem_serial_number ENDP

```

```

print_user_serial_number PROC NEAR
        mov di, offset USERNAME
        add di, 17
        mov ax, cx
        call wrd_to_hex
        mov al, bl
        call byte_to_hex
        mov di, offset USERNAME + 13
        mov [di], ax
        mov dx, offset USERNAME
        call print
        ret
print_user_serial_number ENDP

```



```

begin:
    call print_pc_type
    call print_dos_version
    call print_oem_serial_number
    call print_user_serial_number
    xor al, al
    mov ah, 4Ch
    int 21h

```

TESTPC ENDS

END start

Название файла: exe.asm

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

ASTACK SEGMENT STACK

DW 1024 DUP(?)

ASTACK ENDS

DATA SEGMENT

```

    PC db      'IBM PC Type: PC', 0DH, 0AH, '$'
    PCXT db    'IBM PC Type: PC/XT', 0DH, 0AH, '$'
    AT db      'IBM PC Type: AT', 0DH, 0AH, '$'
    PS230 db   'IBM PC Type: PS2 model 30', 0DH, 0AH, '$'
    PS25060 db 'IBM PC Type: PS2 model 50 or 60', 0DH,
0AH, '$'
    PS280 db   'IBM PC Type: PS2 model 80', 0DH, 0AH, '$'
    PCjr db    'IBM PC Type: P?jr', 0DH, 0AH, '$'
    PCC db     'IBM PC Type: PC Convertible', 0DH,
0AH, '$'
    DOSV db    'DOS version:  . ', 0DH, 0AH, '$'
    OEM db     'OEM number:   ', 0DH, 0AH, '$'
    USERNAME db 'User number:      ', 0DH, 0AH, '$'

```

DATA ENDS

CODE SEGMENT

```

    print PROC NEAR
        mov ah, 09h
        int 21h
        ret
    print ENDP

```

```

tetr_to_hex PROC near
    and AL, 0Fh
    cmp AL, 09
    jbe next
    add AL, 07

```

```

next:
    add AL,30h
    ret
tetr_to_hex ENDP

```

```

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

```

```

wrд_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
wrд_to_hex ENDP

```

```

byte_to_dec PROC near
    push CX
    push DX
    push ax
    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 ax
        pop DX
        pop CX
        ret
byte_to_dec ENDP

```

```

print_pc_type PROC NEAR
    mov ax, 0F000h
    mov es, ax
    mov al, es:[0FFFEh]

```

```

    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_30

```

```

    cmp al, 0f8h
    je ps2_80

```

```

    cmp al, 0fdh
    je pc_jr

```

```

    cmp al, 0f9h
    je pc_conv

```

```

_pc:
    mov dx, offset PC
    jmp _out

```

```

pc_xt:
    mov dx, offset PC_XT
    jmp _out

```

```

_at:
    mov dx, offset AT

```



```

        jmp _out
ps2_30:
        mov dx, offset PS230
        jmp _out
ps2_80:
        mov dx, offset PS280
        jmp _out
pc_jr:
        mov dx, offset PCjr
        jmp _out
pc_conv:
        mov dx, offset PCC
        jmp _out
_out:
        call print
        ret
print_pc_type ENDP

```

```

print_dos_version PROC NEAR
        mov ah, 30h
        int 21h

        mov si, offset DOSV + 13
        call byte_to_dec

        mov al, ah
        add si, 3
        call byte_to_dec
        mov dx, offset DOSV
        call print
        ret
print_dos_version ENDP

```

```

print_oem_serial_number PROC NEAR
        mov si, offset OEM + 13
        mov al, bh
        call byte_to_dec
        mov dx, offset OEM
        call print
        ret
print_oem_serial_number ENDP

```

```

print_user_serial_number PROC NEAR
        mov di, offset USERNAME
        add di, 17
        mov ax, cx
        call wrd_to_hex

```

```

    mov al, bl
    call byte_to_hex
    mov di, offset USERNAME + 13
    mov [di], ax
    mov dx, offset USERNAME
    call print
    ret
print_user_serial_number ENDP

main PROC FAR
    mov ax, data
    mov ds, ax
    call print_pc_type
    call print_dos_version
    call print_oem_serial_number
    call print_user_serial_number
    xor al, al
    mov ah, 4Ch
    int 21h
main ENDP
CODE ENDS
END main

```