

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 МОДУЛЕЙ.

Студент гр. 0382

Санников В.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Исследовать различие в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей 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 необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля.

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

3) Сравнить исходные тексты для .COM и .EXE модулей. Ответить на контрольные вопросы.

4) С помощью FAR ответить на контрольные вопросы.

5) С помощью отладчика TD.exe дать ответы на контрольные вопросы с помощью .COM и .EXE модулей.

6) Оформление отчета.

Ход работы.

В ходе данной лабораторной работы были выполнены следующие пункты:

1) Для выполнения первого пункта был написан .COM модуль, в котором присутствуют функции преобразования двоичных кодов в символы шестнадцатеричных и десятичных чисел, функции определения типа компьютера и версии системы DOS. Следовательно мы имеем «хороший» .COM модуль и «плохой» .EXE модуль. Результаты первого шага приведенные на рисунках 1 и 2.

```
C:\>COM_FILE.COM
IBM type: AT
MS DOS version: 5.0
OEM number: 255
User's number: 000000h
```

Рисунок 1 — Выполнение «хорошего» .COM модуля

```
C:\>COM.exe

5 0                                ц#IBM type:                                ц#IBM type:
255                                ц#IBM type:                                ц#IBM type:
000000                                ц#IBM type:                                ц#IBM type:
```

Рисунок 2 — Выполнение «плохого» .EXE модуля

2) Чтобы написать «хороший» .EXE модуль мы разбиваем программу на сегмент данных, кода, стека и добавляем главную процедуру MAIN. Результат запуска данного модуля представлен на рисунке 3.

```
C:\>EXE.EXE
IBM type: AT
MS DOS version: 5.0
OEM number: 255
User's number: 000000h
```

Рисунок 3 — Выполнение «хорошего» .EXE модуля

3) Теперь сравним данные модули и ответим на контрольные вопросы:

- Сколько сегментов должна содержать COM-программа?

Программа должна содержать один сегмент (код, данные, стек, который генерируется сам)

- EXE-программа?

Обязательно только сегмент кода, сегмент стека и данных описываются отдельно друг от друга.

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

Директива `ORG 100h` нужна обязательно, так как первые 256 байт занимает PSP. Нам необходимо смещение адресации в 256 байт от нулевого адреса. И директива `ASSUME`, которая указывает, с каким сегментом нужно связать регистры сегмента кода и сегмента данных.

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

Нельзя использовать команды с указанием сегментов, так как сегментные регистры в .COM-программе определяются в момент ее запуска, а не при линковке, поскольку не имеет relocation table.

4) На основе шестнадцатеричных модулей ответим на следующие контрольные вопросы:

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

.COM модуль состоит из одного сегмента, который содержит код и данные. Код располагается с нулевого адреса. PSP занимает 100h байт, поэтому устанавливается смещение 100h. (см Рисунок 4)

000000000000:	E9 23 02 49 42 4D 20 74	79 70 65 3A 20 20 0D 0A	Й#IBM type: 1
000000000010:	24 49 42 4D 20 74 79 70	65 3A 20 50 43 0D 0A 24	\$IBM type: PC#
000000000020:	49 42 4D 20 74 79 70 65	3A 20 50 43 2F 58 54 0D	IBM type: PC/XT
000000000030:	0A 24 49 42 4D 20 74 79	70 65 3A 20 41 54 0D 0A	\$IBM type: AT
000000000040:	24 49 42 4D 20 74 79 70	65 3A 20 50 53 32 20 6D	\$IBM type: PS2 m
000000000050:	6F 64 65 6C 20 33 30 0D	0A 24 49 42 4D 20 74 79	odel 30
000000000060:	70 65 3A 20 50 53 32 20	6D 6F 64 65 6C 20 35 30	pe: PS2 model 50
000000000070:	20 6F 72 20 36 30 0D 0A	24 49 42 4D 20 74 79 70	or 60
000000000080:	65 3A 20 50 53 32 20 6D	6F 64 65 6C 20 38 30 0D	e: PS2 model 80
000000000090:	0A 24 49 42 4D 20 74 79	70 65 3A 20 50 43 6A 72	\$IBM type: PCjr
0000000000A0:	0D 0A 24 49 42 4D 20 74	79 70 65 3A 20 50 43 20	\$IBM type: PC
0000000000B0:	43 6F 6E 76 65 72 74 69	62 6C 65 0D 0A 24 4D 53	Convertible
0000000000C0:	20 44 4F 53 20 76 65 72	73 69 6F 6E 3A 20 20 2E	DOS version: .
0000000000D0:	20 20 0D 0A 24 4F 45 4D	20 6E 75 6D 62 65 72 3A	\$OEM number:
0000000000E0:	20 20 20 0D 0A 24 55 73	65 72 27 73 20 6E 75 6D	\$User's num
0000000000F0:	62 65 72 3A 20 20 20 20	20 20 20 68 0D 0A 24 24	ber: h
00000000100:	0F 3C 09 76 02 04 07 04	30 C3 51 8A E0 E8 EF FF	0<ov000GQBaипя
00000000110:	86 C4 B1 04 D2 E8 E8 E6	FF 59 C3 53 8A FC E8 E9	тДтТимияYTSБий
00000000120:	FF 88 25 4F 88 05 4F 8A	C7 E8 DE FF 88 25 4F 88	е#е#е#е#е#е#е#е#е#
00000000130:	05 5B C3 51 52 32 E4 33	D2 B9 0A 00 F7 F1 80 CA	!{QR2n3TN# чсРК
00000000140:	30 88 14 4E 33 D2 3D 0A	00 73 F1 3C 00 74 04 0C	0е#N3T= sc< t+:
00000000150:	30 88 04 5A 59 C3 50 B4	09 CD 21 58 C3 B8 00 F0	е#ZYTPron!XTe p
00000000160:	8E C0 26 A0 FE FF 3C FF	74 2D 3C F8 74 2F 3C FC	тА# яя<ят-<ят/<я
00000000170:	74 31 3C FA 74 33 3C FC	74 35 3C F8 74 37 3C FD	tl<т3<т5<т7<э
00000000180:	74 39 3C F9 74 3B BF 03	01 83 C7 0A E8 7B FF 89	т9<тут; i♥GrSи(ят
00000000190:	05 BA 03 01 EB 31 90 BA	11 01 EB 2B 90 BA 20 01	е#e#e#e#e#e#e#e#e#
000000001A0:	EB 25 90 BA 32 01 EB 1F	90 BA 41 01 EB 19 90 BA	л#е2On#е#e#e#e#e#e#
000000001B0:	5A 01 EB 13 90 BA 79 01	EB 0D 90 BA 92 01 EB 07	ZOn!!heyOn#е#e#e#e#e#e#
000000001C0:	90 BA A3 01 EB 01 90 E8	8C FF C3 33 C0 B4 30 CD	е#e#e#e#e#e#e#e#e#
000000001D0:	21 BE BE 01 83 C6 10 E8	59 FF 8A C4 83 C6 03 E8	!ssGr#и#я#я#я#я#я#я#
000000001E0:	51 FF BA BE 01 E8 6E FF	C3 33 C0 B4 30 CD 21 BE	Qeс#и#я#я#я#я#я#я#
000000001F0:	D5 01 83 C6 0E 8A C7 E8	39 FF BA D5 01 E8 56 FF	МOr#я#я#я#я#я#я#я#я#я#
00000000200:	C3 33 C0 B4 30 CD 21 BF	E6 01 83 C7 14 8B C1 E8	ГзАр0Н!и#OnЗя<Би
00000000210:	09 FF 8A C3 E8 F3 FE BF	E6 01 83 C7 0F 89 05 BA	о#я#я#я#я#я#я#я#я#я#
00000000220:	E6 01 E8 31 FF C3 E8 34	FF E8 9F FF E8 BA FF E8	ж#и#я#я#я#я#я#я#я#я#я#
00000000230:	CF FF 32 C0 B4 4C CD 21		Пя2ArLH!

Рисунок 4 - «хороший» .COM-модуль в 16-м виде (первая строка — начало сегмента, последняя строка — конец сегмента)

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

.EXE состоит из одного сегмента. Код располагается с адреса 300h (200h — заголовок и relocation table, 100h — смещение). С адреса 0h располагается заголовок .EXE файла. Модуль в 16-м виде представлен на рисунке 5.

```
0000000000: 4D 5A 38 01 03 00 00 00 20 00 00 00 FF FF 00 00 MZ80♥ ЯЯ
0000000010: 00 00 3D A7 00 01 00 00 1E 00 00 00 01 00 00 00 =$ 0 ▲ 0
0000000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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 23 02 49 42 4D 20 74 79 70 65 3A 20 20 0D 0A Й#IBM type:  IBM
0000000310: 24 49 42 4D 20 74 79 70 65 3A 20 50 43 0D 0A 24 $IBM type: PC$
0000000320: 49 42 4D 20 74 79 70 65 3A 20 50 43 2F 58 54 0D IBM type: PC/XT$
0000000330: 0A 24 49 42 4D 20 74 79 70 65 3A 20 41 54 0D 0A $IBM type: AT$
0000000340: 24 49 42 4D 20 74 79 70 65 3A 20 50 53 32 20 6D $IBM type: PS2 m
0000000350: 6F 64 65 6C 20 33 30 0D 0A 24 49 42 4D 20 74 79 odel 30$IBM ty
0000000360: 70 65 3A 20 50 53 32 20 6D 6F 64 65 6C 20 35 30 pe: PS2 model 50
0000000370: 20 6F 72 20 36 30 0D 0A 24 49 42 4D 20 74 79 70 or 60$IBM typ
0000000380: 65 3A 20 50 53 32 20 6D 6F 64 65 6C 20 38 30 0D e: PS2 model 80$
0000000390: 0A 24 49 42 4D 20 74 79 70 65 3A 20 50 43 6A 72 $IBM type: PCjr
00000003A0: 0D 0A 24 49 42 4D 20 74 79 70 65 3A 20 50 43 20 $IBM type: PC
00000003B0: 43 6F 6E 76 65 72 74 69 62 6C 65 0D 0A 24 4D 53 Convertible$SMS
00000003C0: 20 44 4F 53 20 76 65 72 73 69 6F 6E 3A 20 20 2E DOS version: .
```

Рисунок 5 - «плохой» .EXE в 16-м виде (строка 0000000300 — начало кода, последняя строка конец сегмента)

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

«Хороший» .EXE имеет следующие сегмента: стека, данных и кода, также в начале модуля располагается заголовок и relocation table (200h байт).

Модуль «хорошего» .EXE см на рисунке 6.

000000001F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000200:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000210:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000220:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000230:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000240:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000250:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000260:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000270:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000280:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000290:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000002A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000002B0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000002C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000002D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000002E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000002F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000300:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000310:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000320:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000330:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000340:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000350:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000360:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000370:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000380:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000390:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000003A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000003B0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000003C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000003D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000003E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000003F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000400:	49 42 4D 20 74 79 70 65	3A 20 20 0D 0A 24 49 42	0000 >000
00000000410:	4D 20 74 79 70 65 3A 20	50 43 0D 0A 24 49 42 4D	000 >0000
00000000420:	20 74 79 70 65 3A 20 50	43 2F 58 54 0D 0A 24 49	00000000
00000000430:	42 4D 20 74 79 70 65 3A	20 41 54 0D 0A 24 49 42	00000000
00000000440:	4D 20 74 79 70 65 3A 20	50 53 32 20 6D 6F 64 65	000 >0'00
00000000450:	6C 20 33 30 0D 0A 24 49	42 4D 20 74 79 70 65 3A	00000000
00000000460:	20 50 53 32 20 6D 6F 64	65 6C 20 35 30 20 6F 72	00000000
00000000470:	20 36 30 0D 0A 24 49 42	4D 20 74 79 70 65 3A 20	00000000 >
00000000480:	50 53 32 20 6D 6F 64 65	6C 20 38 30 0D 0A 24 49	0'00 000
00000000490:	42 4D 20 74 79 70 65 3A	20 50 43 6A 72 0D 0A 24	00000000
000000004A0:	49 42 4D 20 74 79 70 65	3A 20 50 43 20 43 6F 6E	00000 >000
000000004B0:	76 65 72 74 69 62 6C 65	0D 0A 24 4D 53 20 44 4F	00000000
000000004C0:	53 20 76 65 72 73 69 6F	6E 3A 20 20 2E 20 20 0D	000000+00
000000004D0:	0A 24 4F 45 4D 20 6E 75	6D 62 65 72 3A 20 20 20	0000000 >+
000000004E0:	0D 0A 24 55 73 65 72 27	73 20 6E 75 6D 62 65 72	00000000
000000004F0:	3A 20 20 20 20 20 20 20	68 0D 0A 24 00 00 00 00	>++++00
00000000500:	24 0F 3C 09 76 02 04 07	04 30 C3 51 8A E0 E8 EF	00000000
00000000510:	FF 86 C4 B1 04 D2 E8 E8	E6 FF 59 C3 53 8A FC E8	00000000
00000000520:	E9 FF 88 25 4F 88 05 4F	8A C7 E8 DE FF 88 25 4F	00000000
00000000530:	88 05 5B C3 51 52 32 E4	33 D2 B9 0A 00 F7 F1 80	00000000
00000000540:	CA 30 88 14 4E 33 D2 3D	0A 00 73 F1 3C 00 74 04	00000000 <0
00000000550:	0C 30 88 04 5A 59 C3 50	B4 09 CD 21 58 C3 B8 00	00000000 ,
00000000560:	F0 8E C0 26 A0 FE FF 3C	FF 74 2D 3C FB 74 2F 3C	00>000000
00000000570:	FC 74 31 3C FA 74 33 3C	FC 74 35 3C F8 74 37 3C	00000000
00000000580:	FD 74 39 3C F9 74 3B BF	00 00 83 C7 0A E8 7B FF	0000 000

Рисунок 6 - «хороший» .EXE модуль в 16-м виде (0200 — начало сегмента стека, 0400 — начало сегмента данных, 0500 — начало сегмента кода

5) Ответим на контрольные вопросы по отладчику TD.exe. Загрузку в отладчик «хорошего» .COM модуля см на рисунке 7.

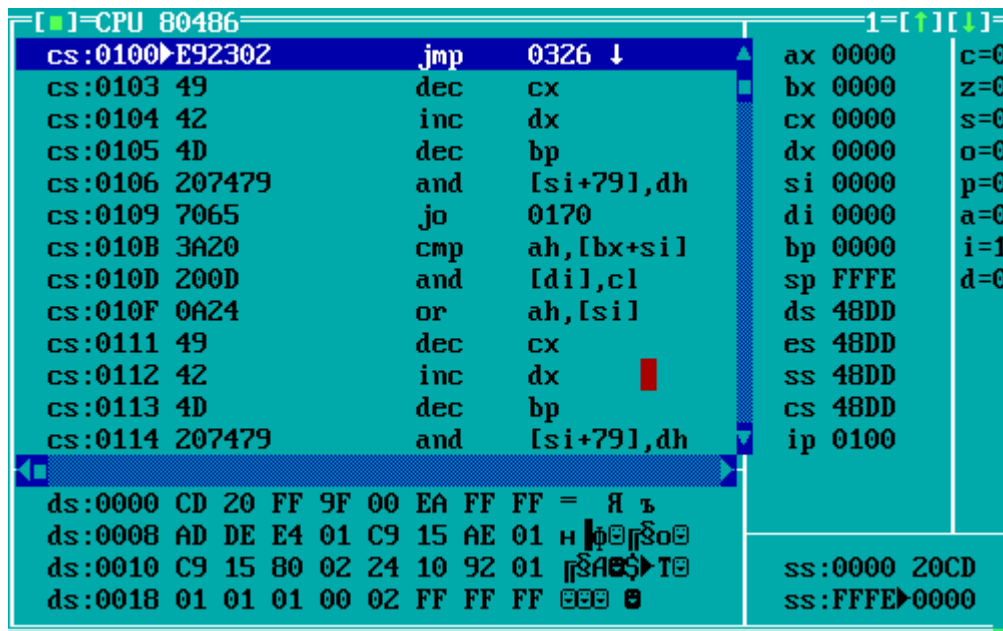


Рисунок 7 - «хороший» .COM модуль в отладчике

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

Сначала определяется сегментный адрес участка ОП, у которого есть место для загрузки программы. Затем создается блок памяти для PSP и программы. Начиная с PSP:0100h помещаются считанные образы COM-файла. Код располагается с адреса CS:0100h.

- Что располагается с адреса 0?

Сегмент PSP.

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

Регистры CS, DS, ES, SS указывают на PSP и имеют значение в данном случае 48DDh (см Рисунок 7)

- Как определяется стек? Какую область памяти он занимает? Какие адреса?

Для .COM файлов стек определяется с FFFh до 0000h. SS указывается на начало сегмента = 48DDh; SP указывается на последний адрес кратный двум = FFFh. Значит адреса стека 48DD:0000h — 48DD:FFh.

6) Загрузим «хороший» .EXE модуль в отладчик TD.exe и ответим на контрольные вопросы. См рисунок 8.

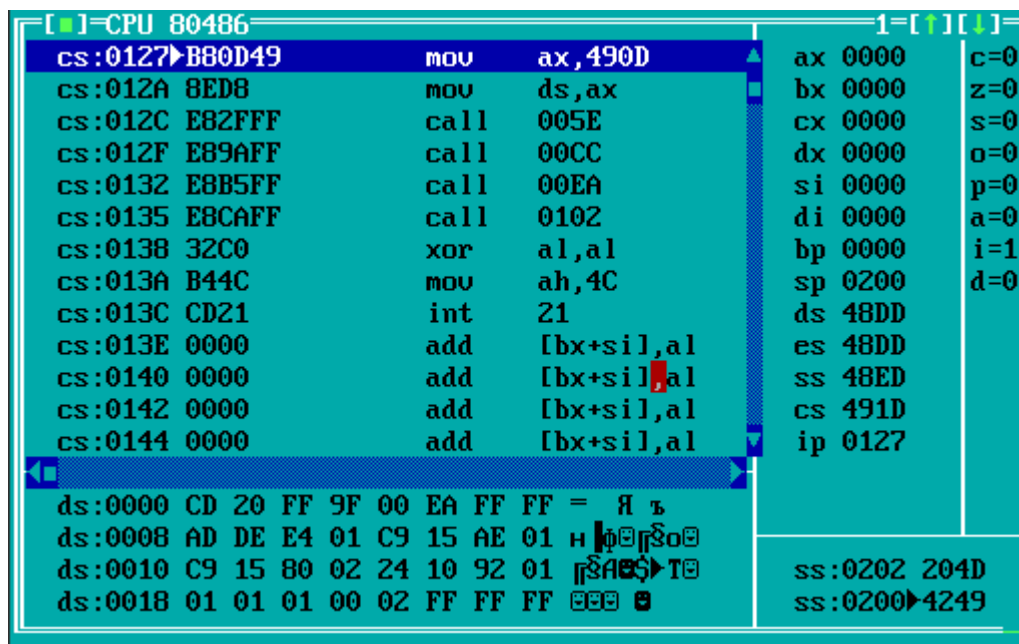


Рисунок 8 - «хороший» .EXE модуль в отладчике

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

EXE-файл загружается, начиная с адреса PSP:0100h. В процессе загрузки считывается информация заголовка (PSP) .EXE в начале файла и выполняется перемещение адресов сегментов, то есть DS и ES устанавливаются на начало сегмента PSP (DS=ES=48DD), SS (SS=48ED) – на начало сегмента стека, CS (CS=491D) – на начало сегмента команд. В IP загружается смещение точки входа в программу, которая берётся из метки после директивы END.

- На что указывают регистры DS и ES?

На начало PSP.

- Как определяется стек?

С помощью директивы .stack либо вручную через segment.

- Как определяется точка входа?

Директивой END. Она определяет адрес, с которого начинается выполнение программы.

Вывод.

В ходе данной лабораторной работы был написан текст .COM и .EXE модулей, были изучены способы загрузки данных модулей в основную память и различия между двумя типами модулей. Данные файлы отвечали за вывод на экран информации о типе PC и версии системы DOS.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММ

Файл COM.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START:    jmp BEGIN
;-----
NOTYPE db 'IBM type: ', 0DH, 0AH, '$'
PC db 'IBM type: PC', 0DH, 0AH, '$'
XT db 'IBM type: PC/XT', 0DH, 0AH, '$'
AT db 'IBM type: AT', 0DH, 0AH, '$'
PS230 db 'IBM type: PS2 model 30', 0DH, 0AH, '$'
PS250_60 db 'IBM type: PS2 model 50 or 60', 0DH, 0AH, '$'
PS280 db 'IBM type: PS2 model 80', 0DH, 0AH, '$'
PCjr db 'IBM type: PCjr', 0DH, 0AH, '$'
PC_Convertible db 'IBM type: PC Convertible', 0DH, 0AH, '$'
DOSV db 'MS DOS version: . ', 0DH, 0AH, '$'
OEM db 'OEM number: ', 0DH, 0AH, '$'
USR db "User's number:      h", 0DH, 0AH, "$"
;-----
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
; байт в AL переводится в два символа 16-го 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 16-ти разрядного числа
; в AX - число, в 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, в 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
;-----
PRINT PROC near
        push AX
        mov AH, 09h
        int 21h
        pop AX
        ret
PRINT ENDP
;-----
; Определение одной из систем
PC_VERSION PROC near
        mov AX, 0F000h
        mov ES, AX
        mov AL, ES:[0FFFEh]

        cmp AL, 0FFh
        je print_pc

```

```

    cmp AL, 0FBh
    je print_XT

    cmp AL, 0FCh
    je print_AT

    cmp AL, 0FAh
    je print_ps230

    cmp AL, 0FCh
    je print_ps250_60

    cmp AL, 0F8h
    je print_ps280

    cmp AL, 0FDh
    je print_PCjr

    cmp AL, 0F9h
    je print_pc_convertible

    mov DI, offset NOTYPE
    add DI, 10
    call BYTE_TO_HEX
    mov [DI], AX
    mov DX, offset NOTYPE
    jmp final

print_pc:
    mov DX, offset PC
    jmp final

print_XT:
    mov DX, offset XT
    jmp final

print_AT:
    mov DX, offset AT
    jmp final

print_ps230:
    mov DX, offset PS230
    jmp final

print_ps250_60:
    mov DX, offset ps250_60
    jmp final

print_ps280:
    mov DX, offset ps280
    jmp final

```

```

print_PCjr:
    mov DX, offset PCjr
    jmp final

print_pc_convertible:
    mov DX, offset PC_Convertible
    jmp final

final:
    call PRINT
    ret
PC_VERSION ENDP
;-----
DOS_VERSION PROC near
    xor AX, AX
    mov AH, 30h
    int 21h

    mov SI, offset DOSV
    add SI, 16
    call BYTE_TO_DEC
    mov AL, AH
    add SI, 3
    call BYTE_TO_DEC
    mov DX, offset DOSV
    call PRINT
    ret
DOS_VERSION ENDP
;-----
OEM_NUMBER PROC near
    xor AX, AX
    mov AH, 30h
    int 21h

    mov SI, offset OEM
    add SI, 14
    mov AL, BH
    call BYTE_TO_DEC
    mov DX, offset OEM
    call PRINT
    ret
OEM_NUMBER ENDP
;-----
USER_NUMBER PROC near
    xor AX, AX
    mov AH, 30h
    int 21h

    mov DI, offset USR
    add DI, 20
    mov AX, CX

```

```

        call WRD_TO_HEX
        mov AL, BL
        call BYTE_TO_HEX
        mov di, offset USR
        add DI, 15
        mov [DI], AX
        mov DX, offset USR
        call PRINT
        ret
USER_NUMBER ENDP
;-----
BEGIN:
        call PC_VERSION
        call DOS_VERSION
        call OEM_NUMBER
        call USER_NUMBER
        xor AL, AL
        mov AH, 4Ch
        int 21h
TESTPC ENDS
        END START

```

Файл EXE.asm

```

AStack SEGMENT STACK
        DW 256 DUP(?)
AStack ENDS
;-----
DATA SEGMENT
        NOTYPE db 'IBM type:  ', 0DH, 0AH, '$'
        PC db 'IBM type: PC', 0DH, 0AH, '$'
        XT db 'IBM type: PC/XT', 0DH, 0AH, '$'
        AT db 'IBM type: AT', 0DH, 0AH, '$'
        PS230 db 'IBM type: PS2 model 30', 0DH, 0AH, '$'
        PS250_60 db 'IBM type: PS2 model 50 or 60', 0DH, 0AH, '$'
        PS280 db 'IBM type: PS2 model 80', 0DH, 0AH, '$'
        PCjr db 'IBM type: PCjr', 0DH, 0AH, '$'
        PC_Convertible db 'IBM type: PC Convertible', 0DH, 0AH, '$'
        DOSV db 'MS DOS version:  . ', 0DH, 0AH, '$'
        OEM db 'OEM number:  ', 0DH, 0AH, '$'
        USR db "User's number:      h", 0DH, 0AH, "$"
DATA ENDS
;-----
TESTPC SEGMENT
        ASSUME CS:TESTPC, 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
; байт в AL переводится в два символа 16-го 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 16-ти разрядного числа
; в AX - число, в 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, в 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
;-----
PRINT PROC near
        push AX
        mov AH, 09h
        int 21h
        pop AX
        ret
PRINT ENDP
;-----
; Определение одной из систем
PC_VERSION PROC near
        mov AX, 0F000h
        mov ES, AX
        mov AL, ES:[0FFFEh]

        cmp AL, 0FFh
        je print_pc

        cmp AL, 0FBh
        je print_XT

        cmp AL, 0FCh
        je print_AT

        cmp AL, 0FAh
        je print_ps230

        cmp AL, 0FCh
        je print_ps250_60

        cmp AL, 0F8h
        je print_ps280

        cmp AL, 0FDh
        je print_PCjr

        cmp AL, 0F9h
        je print_pc_convertible

        mov DI, offset NOTYPE
        add DI, 10
        call BYTE_TO_HEX
        mov [DI], AX
        mov DX, offset NOTYPE
        jmp final

```

```

print_pc:
    mov DX, offset PC
    jmp final

print_XT:
    mov DX, offset XT
    jmp final

print_AT:
    mov DX, offset AT
    jmp final

print_ps230:
    mov DX, offset PS230
    jmp final

print_ps250_60:
    mov DX, offset ps250_60
    jmp final

print_ps280:
    mov DX, offset ps280
    jmp final

print_PCjr:
    mov DX, offset PCjr
    jmp final

print_pc_convertible:
    mov DX, offset PC_Convertible
    jmp final

final:
    call PRINT
    ret
PC_VERSION ENDP
;-----
DOS_VERSION PROC near
    xor AX, AX
    mov AH, 30h
    int 21h

    mov SI, offset DOSV
    add SI, 16
    call BYTE_TO_DEC
    mov AL, AH
    add SI, 3
    call BYTE_TO_DEC
    mov DX, offset DOSV
    call PRINT
    ret
DOS_VERSION ENDP

```

```

;-----
OEM_NUMBER PROC near
    xor AX, AX
    mov AH, 30h
    int 21h

    mov SI, offset OEM
    add SI, 14
    mov AL, BH
    call BYTE_TO_DEC
    mov DX, offset OEM
    call PRINT
    ret
OEM_NUMBER ENDP
;-----
USER_NUMBER PROC near
    xor AX, AX
    mov AH, 30h
    int 21h

    mov DI, offset USR
    add DI, 20
    mov AX, CX
    call WRD_TO_HEX
    mov AL, BL
    call BYTE_TO_HEX
    mov di, offset USR
    add DI, 15
    mov [DI], AX
    mov DX, offset USR
    call PRINT
    ret
USER_NUMBER ENDP
;-----
MAIN PROC far
    mov ax, data
    mov ds, ax
    call PC_VERSION
    call DOS_VERSION
    call OEM_NUMBER
    call USER_NUMBER

    xor AL, AL
    mov AH, 4Ch
    int 21h
MAIN ENDP
TESTPC ENDS
    END MAIN

```