

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра математического обеспечения и применения ЭВМ**

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

Студент гр. 7383

\_\_\_\_\_

Васильев А.И.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2019

### **Цель работы.**

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

### **Основные теоретические положения.**

В лабораторной работе проводится исследование в одной системе двух различных форматов загрузочных модулей, их сравнение и понимание того, как система программирования и управляющая программа обращаются с ними. Система программирования включает компилятор с языка ассемблер, который изготавливает объектные модули. Компоновщик (Linker) по совокупности объектных модулей изготавливает загрузочный модуль, а также, функция ядра – загрузчик, которая помещает программу в основную память и запускает ее на выполнение. Все эти компоненты согласованно работают для изготовления и выполнения загрузочных модулей разного типа.

### **Выполнение работы.**

Был написан текст исходного .COM модуля, который определяет тип РС и версию системы. Описание функций, использованных в программе, представлено в табл. 1. Описание структур данных представлено в табл. 2.

Таблица 1 – Описание функций.

<b>Название функции</b>	<b>Описание</b>
OutOnDisplay	Вызывает функцию печати строки, выводит информацию на экран
TETR_TO_HEX	Вспомогательная функция для работы функции BYTE_TO_HEX
Type_of_system	Выводит на экран информацию о типе ОС
Write_type	
Write_os_version	Выводит на экран информацию о версии ОС
Write_OEM	Выводит на экран серийный номер OEM
Serial_Number	Выводит на экран серийный номер пользователя

Окончание таблицы 1.

BYTE_TO_HEX	Переводит число AL в коды символов 16-0й с/с, записывая результат в BL и BH
WRD_TO_HEX	Переводит число AX в строку 16-ой с/с, записывая результат в DI, начиная с младшей цифры
BYTE_TO_DEC	Переводит байт из AL в десятичную с/с и записывает получившееся число по адресу SI, начиная с младшей цифры

Таблица 2 – Описание структур данных.

Название	Тип	Назначение
OS	db	Тип ОС
OS_VERSION	db	Версия ОС
OS_OEM	db	Серийный номер OEM
SER_NUMBER	db	Серийный номер пользователя
PC	db	PC
PCXT	db	PC/XT
_AT	db	AT
PS2_30	db	PS2 модель 30
PS2_80	db	PS2 модель 80
PCjr	db	PCjr
PC__Cnv	db	PC Convertible

Последовательность действий, выполняемых утилитой:

Программа определяет и выводит на экран следующие значения в заданном порядке: тип ОС, серийный номер OEM, версия ОС, серийный номер пользователя. Результаты работы программы представлены на рис. 1 – 3.

```
Type OS: AT
OEM: 255
Version OS: 5 0
Serial number: 000000
```

Рисунок 1 – Результат выполнения программы good\_com.com

```
C:\>GOOD_COM.EXE

Type OS:

Type OS: 255

Version OS: 5 0 255

Type OS:

Type OS: 000000

Type OS:

Type OS:
```

Рисунок 2 – Результат выполнения программы good\_com.exe

```
C:\>GOOD_EXE.EXE
Type OS: AT
OEM: 255
Version OS: 5 0
Serial number: 000000
```

Рисунок 3 – Результат выполнения программы good\_exe.exe

## Выводы.

В процессе выполнения лабораторной работы были исследованы различия структур исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память. Код программы good\_com.asm представлен в приложении А, код программы good\_exe.asm представлен в приложении Б.

## Ответы на контрольные вопросы.

### Отличия исходных текстов COM и EXE программ

1. Сколько сегментов должны содержать COM-программа?

1 сегмент.

2. EXE-программа?

Не менее 1 сегмента.

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

В тексте COM-программы обязательно должна быть директива ORG 100h, которая сдвигает адресацию в программе на 256 байт для расположения PSP.

Помимо этого, должна присутствовать директива ASSUME, ставящая в соответствие начало программы сегментам кода и данных (при отсутствии директивы ASSUME, программа не скомпилируется из-за невозможности обнаружения начала сегмента кода).

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

Нет, в COM-программе нельзя использовать команды вида mov register, segment и команды, содержащие дальнюю (far) адресацию, так как в этих командах используется таблица настройки в которой содержатся адреса сегментов. Такая таблица есть только в EXE-файлах, поэтому COM-программа не может использовать сегментную адресацию.

### Отличия форматов файлов COM и EXE модулей

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

COM-файл содержит только код и данные. В файле код располагается с нулевого адреса. 16-ричное представление COM файла представлено на рис.4.

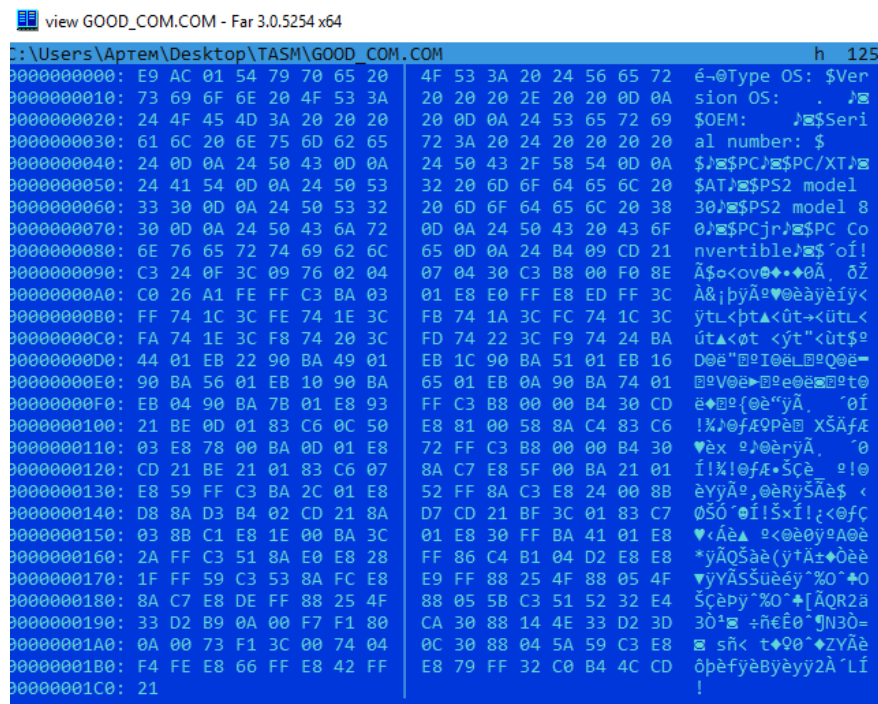


Рисунок 4 – HEX-представление COM файла

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

HEX-представление «плохого» EXE файла представлено на рис. 5-6.

```
view GOOD_COM.EXE - Far 3.0.5254 x64
C:\Users\Арте\\Desktop\TASM\GOOD_COM.EXE h 125
00000000: 4D 5A C1 00 03 00 00 00 20 00 00 00 FF FF 00 00 MZÁ ▼ yy
00000001: 00 00 00 00 00 00 01 00 3E 00 00 00 01 00 FB 50  @ >  @ ūP
00000002: 6A 72 00 00 00 00 00 00 00 00 00 00 00 00 00 00 jr
00000003: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000004: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000005: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000006: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000007: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000008: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000009: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000011: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000012: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000013: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000014: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000015: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000017: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000018: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000019: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Рисунок 5 – HEX-представление «плохого» EXE файла (начало)

C:\Users\Aprem\Desktop\TASM\GOOD_COM.EXE		
000000230:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000240:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000250:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000260:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000270:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000280:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000290:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000002A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000002B0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000002C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000002D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000002E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000002F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000300:	E9 AC 01 54 79 70 65 20	4F 53 3A 20 24 56 65 72
000000310:	73 69 6F 6E 20 4F 53 3A	20 20 20 2E 20 20 0D 0A
000000320:	24 4F 45 4D 3A 20 20 20	20 0D 0A 24 53 65 72 69
000000330:	61 6C 20 6E 75 6D 62 65	72 3A 20 24 20 20 20 20
000000340:	24 0D 0A 24 50 43 0D 0A	24 50 43 2F 58 54 0D 0A
000000350:	24 41 54 0D 0A 24 50 53	32 20 6D 6F 64 65 6C 20
000000360:	33 30 0D 0A 24 50 53 32	20 6D 6F 64 65 6C 20 38
000000370:	30 0D 0A 24 50 43 6A 72	0D 0A 24 50 43 20 43 6F
000000380:	6E 76 65 72 74 69 62 6C	65 0D 0A 24 B4 09 CD 21
000000390:	C3 24 0F 3C 09 76 02 04	07 04 30 C3 B8 00 F0 8E
0000003A0:	C0 26 A1 FE FF C3 BA 03	01 E8 E0 FF E8 ED FF 3C
0000003B0:	FF 74 1C 3C FE 74 1E 3C	FB 74 1A 3C FC 74 1C 3C
0000003C0:	FA 74 1E 3C F8 74 20 3C	FD 74 22 3C F9 74 24 BA
0000003D0:	44 01 EB 22 90 BA 49 01	EB 1C 90 BA 51 01 EB 16
0000003E0:	90 BA 56 01 EB 10 90 BA	65 01 EB 0A 90 BA 74 01
0000003F0:	EB 04 90 BA 7B 01 E8 93	FF C3 B8 00 00 B4 30 CD
000000400:	21 BE 0D 01 83 C6 0C 50	E8 81 00 58 8A C4 B3 C6
000000410:	03 E8 78 00 BA 0D 01 E8	72 FF C3 B8 00 00 B4 30
000000420:	CD 21 BE 21 01 83 C6 07	8A C7 E8 5F 00 BA 21 01
000000430:	E8 59 FF C3 BA 2C 01 E8	52 FF 8A C3 E8 24 00 8B
000000440:	D8 8A D3 B4 02 CD 21 8A	D7 CD 21 BF 3C 01 83 C7
000000450:	03 8B C1 E8 1E 00 BA 3C	01 E8 30 FF BA 41 01 E8
000000460:	2A FF C3 51 8A E0 E8 28	FF 86 C4 B1 04 D2 E8 E8
000000470:	1F FF 59 C3 53 8A FC E8	E9 FF 88 25 4F 88 05 4F
000000480:	8A C7 E8 DE FF 88 25 4F	88 05 58 C3 51 52 32 E4
000000490:	33 D2 B9 0A 00 F7 F1 80	CA 30 88 14 4E 33 D2 3D
0000004A0:	0A 00 73 F1 3C 00 74 04	0C 30 88 04 5A 59 C3 E8
0000004B0:	F4 FE E8 66 FF E8 42 FF	E8 79 FF 32 C0 B4 4C CD

Рисунок 6 – HEX-представление «плохого EXE файла (окончание)

В «плохом» EXE код и данные не разделены по сегментам, а перемешаны (на скриншоте перед данными видно метку перехода E9 AE 01). Код располагается с адреса 300h, так как заголовок занимает 200h байт (байты 8 и 9 указывают, сколько параграфов занимает заголовок) и команда ORG 100h «сдвигает» код на дополнительные 100h. С нулевого адреса располагается заголовок. В первых двух байтах можно увидеть символы MZ, означающие, что формат файла – 16-битный и его следует запускать в соответствии со структурой EXE-файлов. За заголовком следует таблица настройки. Если их убрать, то файл будет загружаться в память как COM-файл.

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

HEX-представление «хорошего» EXE файла представлено на рис. 7-8.

```

view GOOD_EXE.EXE - Far 3.0.5254 x64
C:\Users\Aprem\Desktop\TASM\GOOD_EXE.EXE h 1251
00000000: 4D 5A CA 01 03 00 01 00 20 00 00 00 FF FF 00 00 MZK000 > 0 ыP
000000010: 00 02 00 00 23 01 29 00 3E 00 00 00 01 00 FB 50
000000020: 6A 72 00 00 00 00 00 00 00 00 00 00 00 00 00 00 jr
000000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 24 01 $0
000000040: 29 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
0000000B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

Рисунок 7 – HEX-представление «хорошего» EXE файла (начало)

```

view GOOD_EXE.EXE - Far 3.0.5254 x64
C:\Users\Aprem\Desktop\TASM\GOOD_EXE.EXE
000000330: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000340: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000350: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000360: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000370: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000380: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000390: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000003A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000003B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000003C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000003D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000003E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000003F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000400: 54 79 70 65 20 4F 53 3A 20 24 56 65 72 73 69 6F Type OS: $Versio
000000410: 6E 20 4F 53 3A 20 20 20 2E 20 20 0D 0A 24 4F 45 n OS: . $OE
000000420: 4D 3A 20 20 20 20 20 0A 24 53 65 72 69 61 6C 20 M: $Serial
000000430: 6E 75 6D 62 65 72 3A 20 24 20 20 20 20 24 0D 0A number: $ $
000000440: 24 50 43 0D 0A 24 50 43 2F 58 54 0D 0A 24 41 54 $PC$PC/XT$AT
000000450: 0D 0A 24 50 53 32 20 6D 6F 64 65 6C 20 33 30 0D $PS2 model 30$
000000460: 0A 24 50 53 32 20 6D 6F 64 65 6C 20 38 30 0D 0A $PS2 model 80$
000000470: 24 50 43 6A 72 0D 0A 24 50 43 20 43 6F 6E 76 65 $PCjr$PC Conve
000000480: 72 74 69 62 6C 65 0D 0A 24 00 00 00 00 00 00 00 rtible)$
000000490: B4 09 CD 21 C3 24 0F 3C 09 76 02 04 07 04 30 C3 roH!Г$ocov0000Г
0000004A0: B8 00 F0 8E C0 26 A1 FE FF C3 BA 00 00 E8 E0 FF e p$A&YюяГе иая
0000004B0: E8 ED FF 3C FF 74 1C 3C FE 74 1E 3C FB 74 1A 3C иня<ятL<ютA<ыт-<
0000004C0: FC 74 1C 3C FA 74 1E 3C F8 74 20 3C FD 74 22 3C ьтL<ьтA<wt <эт"<
0000004D0: F9 74 24 BA 41 00 EB 22 90 BA 46 00 EB 1C 90 BA щт$eA л"ГеF лLГе
0000004E0: 4E 00 EB 16 90 BA 53 00 EB 10 90 BA 62 00 EB 0A N л-ГеS л-Геb л
0000004F0: 90 BA 71 00 EB 04 90 BA 78 00 E8 93 FF C3 B8 00 Геq л-Геx и"яГе
000000500: 00 B4 30 CD 21 BE 0A 00 83 C6 0C 50 E8 81 00 58 r0H!s fЖQРиГ X
000000510: 8A C4 83 C6 03 E8 78 00 BA 0A 00 E8 72 FF C3 B8 лДfЖix eи ияГе
000000520: 00 00 B4 30 CD 21 BE 1E 00 83 C6 07 8A C7 E8 5F r0H!s fЖ•лзи_
000000530: 00 BA 1E 00 E8 59 FF C3 BA 29 00 E8 52 FF 8A C3 eA иYяГе) иРялГ
000000540: E8 24 00 8B D8 8A D3 B4 02 CD 21 8A D7 CD 21 BF и$ <ШьYrГ0H!лЧH!i
000000550: 39 00 83 C7 03 8B C1 E8 1E 00 BA 39 00 E8 30 FF 9 fЗ▼<БиA e9 иоя
000000560: BA 3E 00 E8 2A FF C3 51 8A E0 E8 28 FF 86 C4 B1 e> и"яQлai(я†Д±
000000570: 04 D2 E8 E8 1F FF 59 C3 53 8A FC E8 E9 FF 88 25 ♦ТиияYГSльийя€%
000000580: 4F 88 05 4F 8A C7 E8 DE FF 88 25 4F 88 05 5B C3 O€40лЗиюя€%O€4[Г
000000590: 51 52 32 E4 33 D2 B9 0A 00 F7 F1 80 CA 30 88 14 QR2дЗTNи чсбK0€9
0000005A0: 4E 33 D2 3D 0A 00 73 F1 3C 00 74 04 0C 30 88 04 N3T=и sc< т♦Q0€♦
0000005B0: 5A 59 C3 B8 20 00 8E D8 E8 EF FE E8 61 FF E8 3D ZYГе fшиюияи=
0000005C0: FF E8 74 FF 32 C0 B4 4C CD 21 итя2ArLH!

```

Рисунок 8 – HEX-представление «хорошего» EXE файла (окончание)

В отличие от «плохого» EXE, в «хорошем» код, стек и данные выделены в отдельные сегменты. Код программы начинается с 400h, т.к. дополнительно выделено под стек 200 байт (100 слов). Для «хорошего» EXE в директиве org



100h нет необходимости, т.к. загрузчик автоматически расположит программу после PSP.

**Результат загрузки COM модуля в основную память представлен на рис.9.**

The screenshot shows a DOS debugger window with the following content:

[CPU 80486]				1=[↑][↓]	
cs:0100	E9AC01	jmp	02AF ↓	ax	0000 c=0
cs:0103	54	push	sp	bx	0000 z=0
cs:0104	7970	jns	0176	cx	0000 s=0
cs:0106	65204F53	and	gs:[bx+53],cl	dx	0000 o=0
cs:010A	3A20	cmp	ah,[bx+si]	si	0000 p=0
cs:010C	2456	and	al,56	di	0000 a=0
cs:010E	657273	jb	gs:0184	bp	0000 i=1
cs:0111	696F6E204F	imul	bp,[bx+6E],4F	sp	0000 d=0
cs:0116	53	push	bx	ds	48DD
cs:0117	3A20	cmp	ah,[bx+si]	es	48DD
cs:0119	2020	and	[bx+si],ah	ss	48EC
cs:011B	2E2020	and	cs:[bx+si],ah	cs	48ED
cs:011E	0D0A24	or	ax,240A	ip	0100

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

ss:0002 6474  
ss:0000 0000

Рисунок 9 – Результат загрузки COM в основную память

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

Формат загрузки модуля COM:

1. Выделение сегмента памяти для модуля
2. Установка всех сегментных регистров на начало выделенного сегмента памяти
3. Построение в первых 100h байтах памяти PSP
4. Загрузка содержимого COM-файла и присваивание регистру IP значения 100h.
5. Регистр SP устанавливается в конец сегмента

Код начинается с адреса, содержащимся в CS, в нашем случае это 48DD.

2. Что располагается с адреса 0?

С нулевого адреса располагается PSP.

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

Все сегментные регистры (CS, DS, ES, SS) в данном случае равны 48DD и указывают на начало PSP.

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

Стек занимает весь сегмент COM-программы, его начало находится в конце сегмента. SS указывает на начало сегмента, а SP=FFFEh – на его конец. Стек может дойти до кода/данных программы при достаточном количестве элементов.

Адреса расположены в диапазоне 0000h-FFFFh. Стек растет от больших адресов к меньшим.

**Результат загрузки «хорошего» EXE модуля в основную память представлен на рис. 10.**

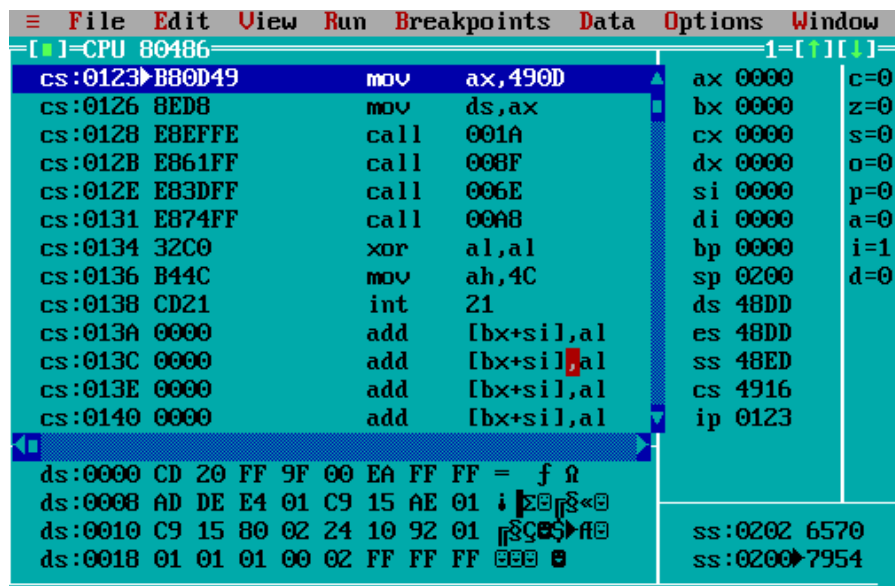


Рисунок 10 – Результат загрузки «хорошего» EXE в основную память

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

SS=48ED – начало сегмента стека, CS=4916 – начало сегмента команд.

**2. На что указывают регистры DS и ES?**

На начало PSP.

**3. Как определяется стек?**

В исходном коде модуля стек определяется при помощи директивы `STACK`, а при исполнении в регистр `SS` записывается адрес начала сегмента стека, а в `SP` – его вершины.

#### **4. Как определяется точка входа?**

Точка входа в программу определяется с помощью директивы `END`. После этой директивы указывается метка, куда переходит программа при запуске.

## ПРИЛОЖЕНИЕ А

### GOOD\_COM.ASM

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

START: JMP BEGIN
; ДАННЫЕ
OS           db 'Type OS: $'
OS_VERSION   db 'Version OS:  .  ',0DH,0AH,'$'
OS_OEM       db 'OEM:      ',0DH,0AH,'$'
SER_NUMBER   db 'Serial number: ', '$'
STRING       db '      $'
ENDSTR       db 0DH,0AH,'$'

PC           db 'PC',0DH,0AH,'$'
PCXT         db 'PC/XT',0DH,0AH,'$'
_AT         db 'AT',0DH,0AH,'$'
PS2_30       db 'PS2 model 30',0DH,0AH,'$'
PS2_80       db 'PS2 model 80',0DH,0AH,'$'
PCjr         db 'PCjr',0DH,0AH,'$'
PC_Cnv       db 'PC Convertible',0DH,0AH,'$'

OutOnDisplay PROC near
    mov AH,09h
    int 21h
    ret
OutOnDisplay 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

Type_of_system PROC near
    mov ax,0F000h
    mov es,ax
    mov ax,es:0FFFEh
    ret
```

Type\_of\_system ENDP

Write\_type PROC near ;вывод сообщения о типе системы

```
mov dx, OFFSET OS
call OutOnDisplay
call Type_of_system
```

; Определяем тип ОС

```
cmp al,0FFh
je PC_
cmp al,0FEh
je PCXT_
cmp al,0FBh
je PCXT_
cmp al,0FCh
je AT_
cmp al,0FAh
je PS2_30_
cmp al,0F8h
je PS2_80_
cmp al,0FDh
je PCjr_
cmp al,0F9h
je PC_Cnv_
```

PC\_:

```
mov dx, OFFSET PC
jmp OutMsg
```

PCXT\_:

```
mov dx, OFFSET PCXT
jmp OutMsg
```

AT\_:

```
mov dx, OFFSET _AT
jmp OutMsg
```

PS2\_30\_:

```
mov dx, OFFSET PS2_30
jmp OutMsg
```

PS2\_80\_:

```
mov dx, OFFSET PS2_80
jmp OutMsg
```

PCjr\_:

```
mov dx, OFFSET PCjr
jmp OutMsg
```

PC\_Cnv\_:

```

        mov dx, OFFSET PC_Cnv
;      jmp OutMsg

OutMsg:
    call OutOnDisplay
    ret
Write_type ENDP

; Печатает версию системы
Write_os_version PROC near
    ; Получение данных
    mov ax,0
    mov ah,30h
    int 21h

    ; Пишем в строку OS_VERS номер основной версии ОС
    mov si,offset OS_VERSION
    add si,12
    push ax
    call BYTE_TO_DEC

    ; Пишем модификацию ОС
    pop ax
    mov al,ah
    add si,3
    call BYTE_TO_DEC

    ; Пишем версию ОС в консоль
    mov dx,offset OS_VERSION
    call OutOnDisplay
    ret
Write_os_version ENDP

; ВЫВОД OEM
Write_OEM PROC near
    mov ax,0
    mov ah,30h
    int 21h

    mov si,offset OS_OEM
    add si,7
    mov al,bh
    call BYTE_TO_DEC

```

```

        mov dx,offset OS_OEM
        call OutOnDisplay
        ret
Write_OEM ENDP

Serial_Number PROC near
    ; Пишем серийный номер пользователя
    mov dx,offset SER_NUMBER
    call OutOnDisplay
    mov al,bl
    call BYTE_TO_HEX
    mov bx,ax
    mov dl,bl
    mov ah,02h
    int 21h
    mov dl,bh
    int 21h
    mov di,offset STRING
    add di,3
    mov ax,cx
    call WRD_TO_HEX
    mov dx,offset STRING
    call OutOnDisplay

    mov dx,offset ENDSTR
    call OutOnDisplay

    ret
Serial_Number ENDP
;-----
BYTE_TO_HEX PROC near
;байт в AL переводится в два символа шестн. числа в AX
    push CX
    mov AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX
    pop CX
    ret
BYTE_TO_HEX ENDP
;-----
; перевод в 16с/с 16-ти разрядного числа

```

```

; в AX - число, DI - адрес последнего символа
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
;-----
; перевод в 10с/с, SI - адрес поля младшей цифры
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
;-----
BEGIN:
    call Write_type
    call Write_OEM

```



```
        call Write_os_version
        call Serial_Number
        xor AL,AL
        mov AH,4Ch
        int 21H
TESTPC ENDS
END START
}
```

## ПРИЛОЖЕНИЕ Б

### GOOD\_EXE.ASM

STACK SEGMENT STACK

DW 0100H DUP(?)

STACK ENDS

DATA SEGMENT

; ДАННЫЕ

OS db 'Type OS: \$'  
OS\_VERSION db 'Version OS: . ',0DH,0AH,'\$'  
OS\_OEM db 'OEM: ',0DH,0AH,'\$'  
SER\_NUMBER db 'Serial number: ', '\$'  
STRING db ' \$'  
ENDSTR db 0DH,0AH,'\$'

PC db 'PC',0DH,0AH,'\$'  
PCXT db 'PC/XT',0DH,0AH,'\$'  
\_AT db 'AT',0DH,0AH,'\$'  
PS2\_30 db 'PS2 model 30',0DH,0AH,'\$'  
PS2\_80 db 'PS2 model 80',0DH,0AH,'\$'  
PCjr db 'PCjr',0DH,0AH,'\$'  
PC\_Cnv db 'PC Convertible',0DH,0AH,'\$'  
DATA ENDS

CODE SEGMENT

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

OutOnDisplay PROC near

mov AH,09h

int 21h

ret

OutOnDisplay 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

Type\_of\_system PROC near

mov ax,0F000h

mov es,ax

mov ax,es:0FFFEh

ret

Type\_of\_system ENDP

Write\_type PROC near ;вывод сообщения о типе системы

mov dx, OFFSET OS

call OutOnDisplay

call Type\_of\_system

; Определяем тип ОС

cmp al,0FFh

je PC\_

cmp al,0FEh

je PCXT\_

cmp al,0FBh

je PCXT\_

cmp al,0FCh

je AT\_

cmp al,0FAh

je PS2\_30\_

cmp al,0F8h

je PS2\_80\_

cmp al,0FDh

je PCjr\_

cmp al,0F9h

je PC\_Cnv\_

PC\_:

mov dx, OFFSET PC

jmp OutMsg

PCXT\_:

mov dx, OFFSET PCXT

jmp OutMsg

AT\_:

```

        mov dx, OFFSET _AT
        jmp OutMsg
PS2_30_:
        mov dx, OFFSET PS2_30
        jmp OutMsg
PS2_80_:
        mov dx, OFFSET PS2_80
        jmp OutMsg
PCjr_:
        mov dx, OFFSET PCjr
        jmp OutMsg
PC_Cnv_:
        mov dx, OFFSET PC_Cnv
;      jmp OutMsg

OutMsg:
call OutOnDisplay
ret
Write_type ENDP

; Печатает версию системы
Write_os_version PROC near
; Получение данных
mov ax,0
mov ah,30h
int 21h

; Пишем в строку OS_VERS номер основной версии ОС
mov si,offset OS_VERSION
add si,12
push ax
call BYTE_TO_DEC

; Пишем модификацию ОС
pop ax
mov al,ah
add si,3
call BYTE_TO_DEC

```

```

; Пишем версию ОС в консоль
mov dx,offset OS_VERSION
call OutOnDisplay
ret
Write_os_version ENDP

; ВЫВОД OEM
Write_OEM PROC near
    mov ax,0
    mov ah,30h
    int 21h

    mov si,offset OS_OEM
    add si,7
    mov al,bh
    call BYTE_TO_DEC

    mov dx,offset OS_OEM
    call OutOnDisplay
    ret
Write_OEM ENDP

Serial_Number PROC near
; Пишем серийный номер пользователя
mov dx,offset SER_NUMBER
call OutOnDisplay
mov al,bl
call BYTE_TO_HEX
mov bx,ax
mov dl,bl
mov ah,02h
int 21h
mov dl,bh
int 21h
mov di,offset STRING
add di,3
mov ax,cx
call WRD_TO_HEX
mov dx,offset STRING

```

```

call OutOnDisplay

mov dx,offset ENDSTR
call OutOnDisplay

ret
Serial_Number ENDP
;-----
BYTE_TO_HEX PROC near
;байт в AL переводится в два символа шестн. числа в AX
push CX
mov AH,AL
call TETR_TO_HEX
xchg AL,AH
mov CL,4
shr AL,CL
call TETR_TO_HEX
pop CX
ret
BYTE_TO_HEX ENDP
;-----
; перевод в 16с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
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

```

```

;-----
; перевод в 10с/с, SI - адрес поля младшей цифры
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
;-----
BEGIN:
    mov ax, DATA
    mov ds, ax
    call Write_type
    call Write_OEM
    call Write_os_version
    call Serial_Number
    xor AL,AL
    mov AH,4Ch
    int 21H
CODE ENDS
END BEGIN

```