

**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**

Студентка гр. 7383

Маркова А.В.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2019

### **Постановка задачи.**

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

Описание процедур, используемых в работе:

- TETR\_TO\_HEX – вспомогательная функция, которая переводит из двоичной в шестнадцатеричную систему счисления, используется для работы функции BYTE\_TO\_HEX;
- BYTE\_TO\_HEX – переводит байтовое число из регистра AL в шестнадцатеричную систему счисления;
- WRD\_TO\_HEX – переводит число из регистра AX в шестнадцатеричную систему;
- BYTE\_TO\_DEC – переводит байтовое число в десятичную систему счисления;
- LINE\_OUTPUT – выводит сообщение на экран;
- GET\_PC\_TYPE – определение типа IBP PC из предпоследнего байта ROM BIOS;
- Write\_PC\_TYPE – вывод типа IBM PC на экран;
- Write\_PC\_VERS – определяет номер версии PC и выводит его на экран;
- Write\_PC\_OEM – определяет серийный номер OEM и выводит его на экран;
- Write\_SER\_NUM – вывод на экран серийного номера пользователя;

Таблица 1 – Описание структур данных:

Название	Тип	Назначение
PC_TYPE	db	Тип IBM PC
PC_VERS	db	Версия PC
PC_OEM	db	Серийный номер OEM
SER_NUM	db	Серийный номер пользователя

TYPE_PC	db	PC (код: FF)
TYPE_PCXT	db	PC/XT (код: FE, FB)
TYPE_AT	db	AT (код: FC)
TYPE_PS2_30	db	PS2 модель 30 (код: FA)
TYPE_PS2_50_60	db	PS2 модель 50/60 (код: FC)
TYPE_PS2_80	db	PS2 модель 80 (код: F8)
TYPE_PCjr	db	PCjr (код: FD)
TYPE_PC_Con	db	PC Convertible (код: F9)

### Ход работы.

1. Был написан текст исходного .COM модуля, который определяет тип IBM PC, версию системы, номер OEM и серийный номер пользователя.
2. Смонтирован виртуальный диск k с каталогом tasm с помощью команды, представленной на рис. 1.

```
Z:\>mount k d:/7383/lr1/tasm
Drive K is mounted as local directory d:/7383/lr1/tasm\
Z:\>k:\
```

Рисунок 1 – Создание виртуального диска

3. Было произведено транслирование программы с помощью строки tasm «имя файла», в последствии чего создался объектный файл, результат вызова команды показан на рис. 2.

```
K:\>tasm bad
Turbo Assembler Version 3.1 Copyright (c) 1988, 1992 Borland International

Assembling file:   bad.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  471k
```

Рисунок 2 – Транслирование программы

4. Линковка загрузочного модуля с помощью команды tlink и tlink/t.
5. Получены «хороший» .COM модуль и «плохой» .EXE из исходного текста для .COM модуля.
6. Был написан текст исходного .EXE модуля, который выполняет те же функции, получен «хороший» .EXE.
7. Результаты работы программы показаны на рис. 3 – 5.

```
K:\>bad.com
Type PC: AT
System version: 5.0
Original Equipment Manufacturer: 255
User serial number: 000000
```

Рисунок 3 – Результат выполнения программы bad.com

```
K:\>bad.exe

    Type PC:
Type PC:          5 0
    Type PC:          255
    Type PC:          000000
    Type PC:
```

Рисунок 4 – Результат выполнения программы bad.exe

```
K:\>good.exe
Type PC: AT
System version: 5.0
Original Equipment Manufacturer: 255
User serial number: 000000
```

Рисунок 5 – Результат выполнения программы good.exe

## **Выводы.**

В ходе данной лабораторной работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память. Код программы bad\_exe.asm представлен в приложении А, код программы good\_exe.asm представлен в приложении Б. Было замечено, что структура EXE программ сложнее структуры COM программ.

## **Ответы на контрольные вопросы.**

### **Отличия исходных текстов COM и EXE программ**

1. Сколько сегментов должна содержать COM-программа?

Один сегмент.

2. EXE-программа?

Минимум один сегмент, может состоять и из нескольких.

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

Обязательными директивами являются ORG 100h, которая сдвигает адресацию в программе на 256 байт для расположения PSP, и ASSUME, которая ставит в соответствие сегментам CS и DS начало программы. Если же не будет директивы ASSUME, то программа не скомпилируется, так как не обнаружит начало сегмента кода.

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

Не все, в COM-программе нельзя использовать команды с дальнейшей адресацией, так как в этих командах используется таблица настройки, в которой содержатся адреса сегментов, однако такая таблица существует только в EXE-файлах, в COM-программах нельзя использовать сегментную адресацию.

То есть в COM-программе нельзя использовать команды вида mov регистр или сегмент.

## Отличия форматов файлов COM и EXE модулей

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

В файлах данного типа содержатся только машинный код и данные программы. В файле код начинается с нулевого адреса. Структура COM-файла представлена на рис. 6.

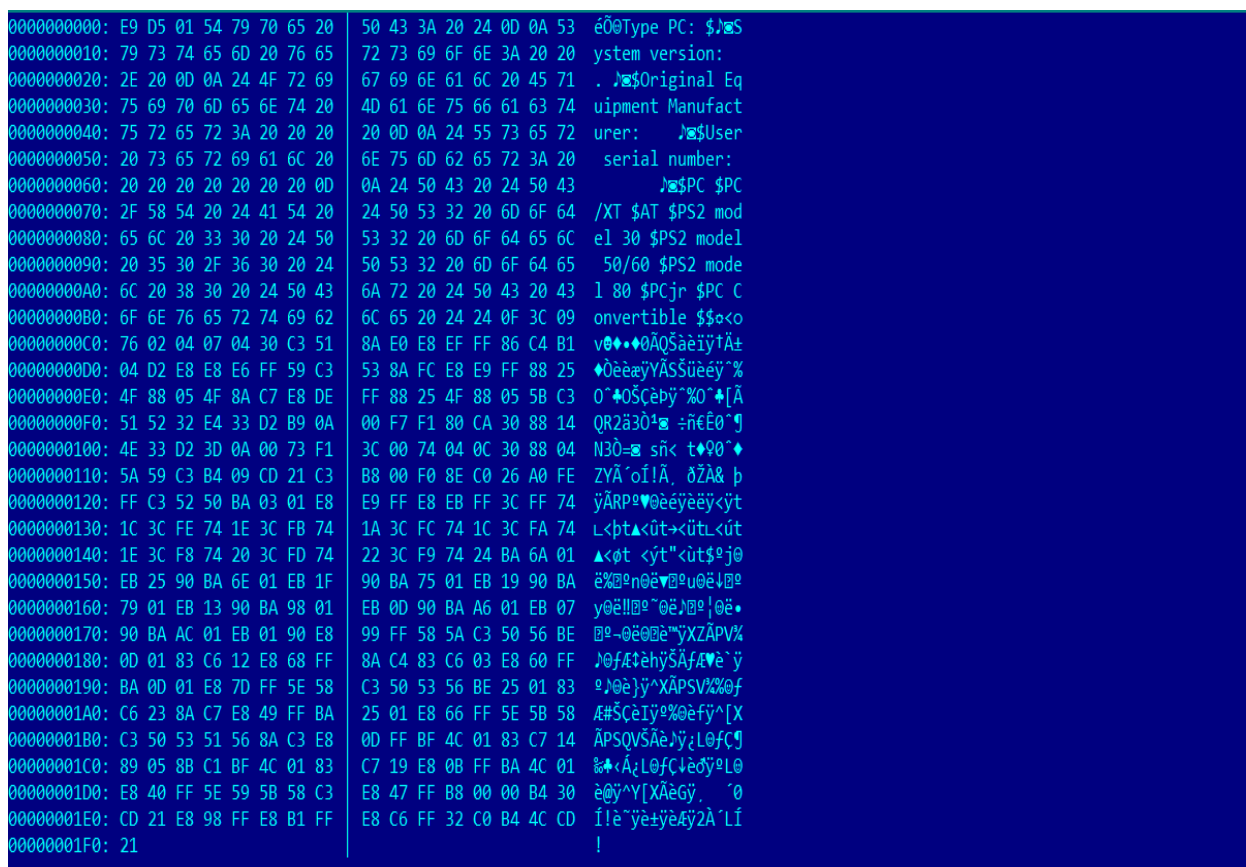


Рисунок 6 – HEX представление .COM файла

HEX – представление было получено в FarManager с помощью нажатия комбинации клавиш F3 и F4.

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

По рис. 7 видно, что структура «плохого» EXE-файла больше, чем у COM-файла. Код располагается с адреса 300h, так как заголовок занимает 200h байт и команда ORG 100h сдвигает код на дополнительные 100h. Следовательно с нулевого адреса располагается заголовок. В первых двух

байтах видим символы MZ, которые указывают на то, что формат файла 16-битный и его следует запускать в соответствии со структурой EXE-файлов. Затем идёт таблица настроек, при помощи которых строится данный EXE-файл и зарезервированные директивой ORG 100h байт.

00000000: 4D 5A F1 00 03 00 00 00	20 00 00 00 FF FF 00 00	MZ	000000200: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000010: 00 00 00 00 00 01 00 00	3E 00 00 00 01 00 FB 50	>	000000210: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000020: 6A 72 00 00 00 00 00 00	00 00 00 00 00 00 00 00	jr	000000220: 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		000000230: 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		000000240: 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		000000250: 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		000000260: 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		000000270: 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		000000280: 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		000000290: 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		0000002A0: 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		0000002B0: 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		0000002C0: 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		0000002D0: 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		0000002E0: 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		0000002F0: 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		000000300: E9 D5 01 54 79 70 65 20	50 43 3A 20 24 0D 0A 53
000000110: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000310: 79 73 74 65 6D 20 76 65	72 73 69 6F 6E 3A 20 20
000000120: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000320: 2E 20 0D 0A 24 4F 72 69	67 69 6E 61 6C 20 45 71
000000130: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000330: 75 69 70 6D 65 6E 74 20	4D 61 6E 75 66 61 63 74
000000140: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000340: 75 72 65 72 3A 20 20 20	20 0D 0A 24 55 73 65 72
000000150: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000350: 20 73 65 72 69 61 6C 20	6E 75 6D 62 65 72 3A 20
000000160: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000360: 20 20 20 20 20 20 20 00	0A 24 50 43 20 24 50 43
000000170: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000370: 2F 58 54 20 24 41 54 20	24 50 53 32 20 6D 6F 64
000000180: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000380: 65 6C 20 33 30 20 24 50	53 32 20 6D 6F 64 65 6C
000000190: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000390: 20 35 30 2F 36 30 20 24	50 53 32 20 6D 6F 64 65
0000001A0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		0000003A0: 6C 20 38 30 20 24 50 43	6A 72 20 24 50 43 20 43
0000001B0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		0000003B0: 6F 6E 76 65 72 74 69 62	6C 65 20 24 24 0F 3C 09
0000001C0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		0000003C0: 76 02 04 07 04 30 C3 51	8A 0E E8 EF FF 86 C4 B1
0000001D0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		0000003D0: 04 D2 E8 E8 E6 FF 59 C3	53 8A FC E8 E9 FF 88 25
0000001E0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		0000003E0: 4F 88 05 4F 8A C7 E8 DE	FF 88 25 4F 88 05 5B C3
0000001F0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		0000003F0: 51 52 32 E4 33 D2 B9 0A	00 F7 F1 80 CA 30 88 14
000000200: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000400: 4E 33 D2 3D 0A 00 73 F1	3C 00 74 04 0C 30 88 04
000000210: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000410: 5A 59 C3 B4 09 CD 21 C3	B8 00 F0 8E C0 26 A0 FE
000000220: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000420: FF C3 52 50 BA 03 01 E8	E9 FF E8 EF FF 3C FF 74
000000230: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000430: 1C 3C FE 74 1E 3C FE 74	1A 3C FC 74 1C 3C FA 74
000000240: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000440: 1E 3C F8 74 20 3C FD 74	22 3C F9 74 24 BA 6A 01
000000250: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000450: EB 25 90 BA 6E 01 EB 1F	90 BA 75 01 EB 19 90 BA
000000260: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000460: 79 01 EB 13 90 BA 98 01	EB 0D 90 BA A6 01 EB 07
000000270: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000470: 90 BA AC 01 EB 01 90 E8	99 FF 58 5A C3 50 56 BE
000000280: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000480: 0D 01 83 C6 12 E8 68 FF	8A C4 83 C6 03 E8 60 FF
000000290: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		000000490: BA 0D 01 E8 7D FF 5E 58	C3 50 53 56 BE 25 01 83
0000002A0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		0000004A0: C6 23 8A C7 E8 49 FF BA	25 01 E8 66 FF 5E 58 58
0000002B0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		0000004B0: C3 50 53 51 56 BA C3 E8	0D FF BF 4C 01 83 C7 14
0000002C0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		0000004C0: 89 05 8B C1 BF 4C 01 83	C7 19 E8 0B FF BA 4C 01
0000002D0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		0000004D0: E8 40 FF 5E 59 58 58 C3	E8 47 FF B8 00 00 B4 30
0000002E0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		0000004E0: CD 21 E8 98 FF E8 B1 FF	E8 C6 FF 32 C0 B4 4C CD
0000002F0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		0000004F0: 21	!

Рисунок 7 – HEX-представление «плохого» .EXE-файла

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

В отличие от «плохого» EXE, в «хорошем» код, стек и данные выделены в отдельные сегменты. В структуре «хорошего» файла отсутствует директива ORG 100h, которая резервировала пространство, так как в EXE нет необходимости её использовать, потому что загрузчик автоматически ставит программу после PSP. С нулевого адреса все так же находится заголовок с таблицей настроек. С адреса 200h в «хорошем» EXE-файле идет сегмент стека,

под который дополнительно выделено 20 байт, а с адреса 220h располагается код программы. Иллюстрация «хорошего» EXE-файла представлена на рис. 8.

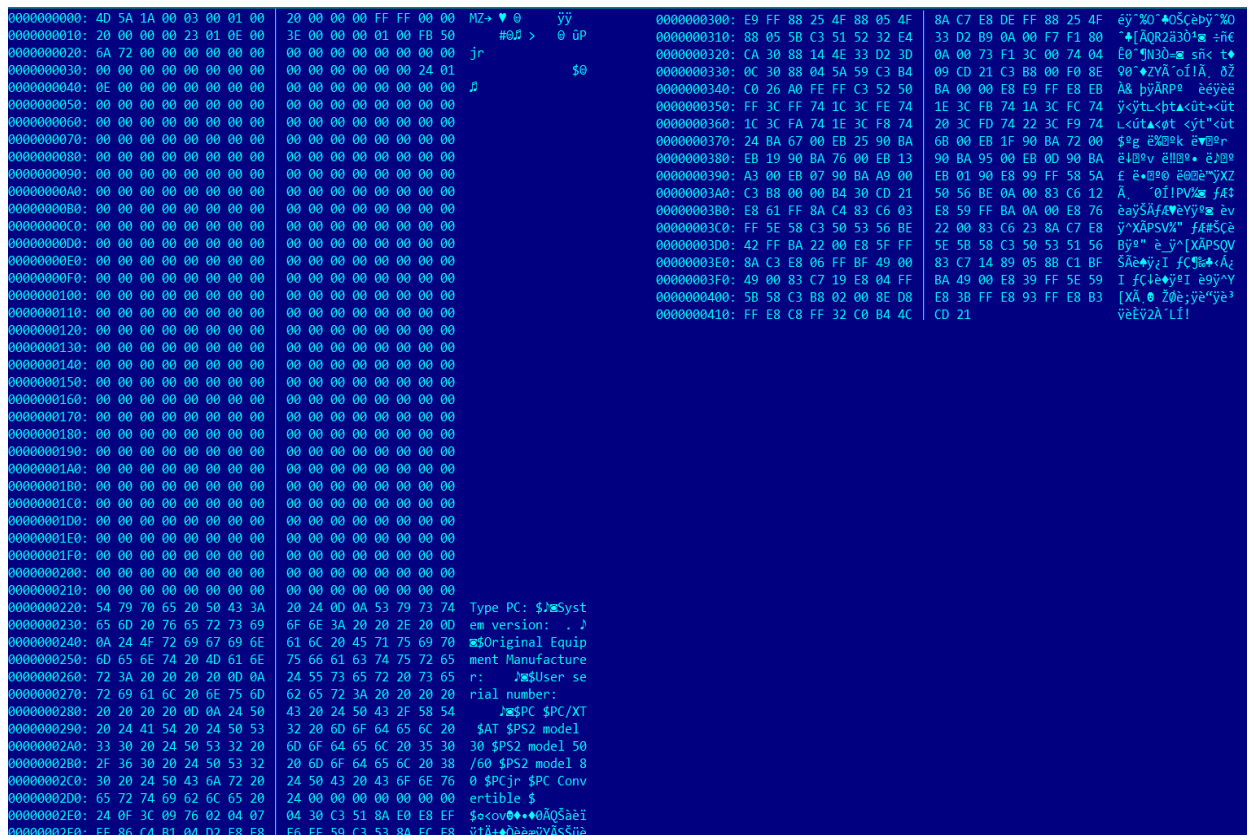


Рисунок 8 – HEX – представление «хорошего» EXE – файла

### Загрузка COM модуля в основную память

Для ответа на следующие вопросы открыли отладчик с помощью команды td «имя файла», появившееся окно представлено на рис. 9.

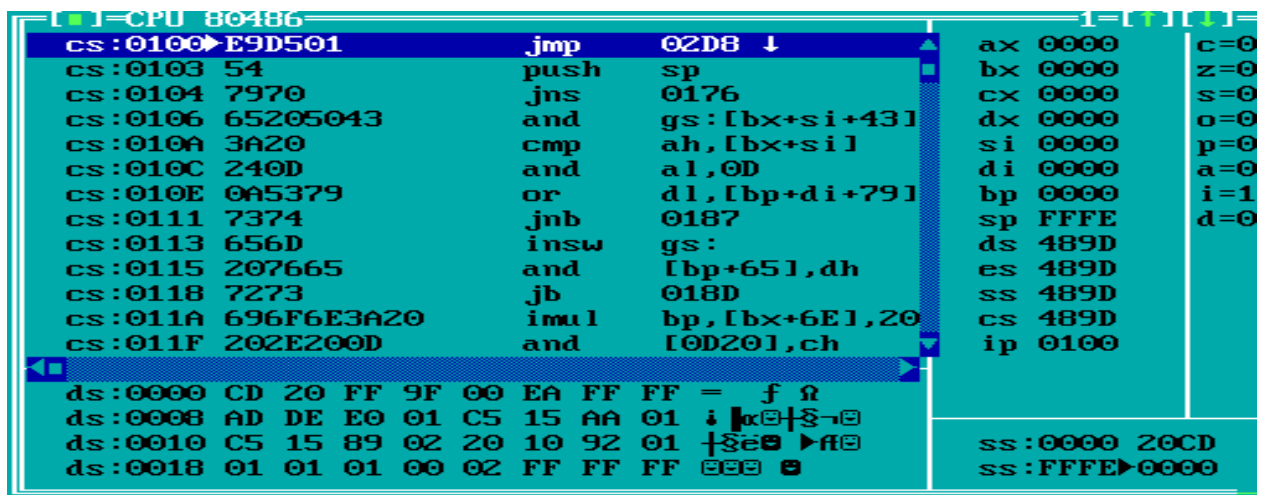


Рисунок 9 – Результат загрузки .COM в отладчик



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

Сначала система выделяет свободный сегмент памяти, затем заносит его адрес во все сегментные регистры. Потом происходит построение PSP в первых 100h байтах памяти. Загружается содержимое COM-файла и присваивается регистру IP значение 100h. Указатель на стек SP указывает на конец выделенной памяти, поэтому запись данных стека осуществляется снизу вверх.

Начало кода определяется директивой ORG 100h от начала выделенного фрагмента, в данном случае код будет располагаться с адреса 489D:0100h.

2. Что располагается с адреса 0?

PSP.

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

Все сегментные регистры (CS, SS, ES, DS) имеют одно и то же значение равное 489D (см. рис. 9). И все они указывают на начало сегмента программы, то есть на начало PSP.

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

DOS автоматически определяет стек. Если для программы размер сегмента в 64К достаточен, то DOS устанавливает  $SP = FFFE$ . Стек занимает весь сегмент COM – программы, а его начало находится в конце сегмента. SS указывает на начало, а SP – на конец. При добавлении данных в стек, адрес на который указывает SP уменьшается и данные записываются в конце выделенной области памяти, то есть заполняется снизу вверх. Стек может дойти до кода/данных программы при большом количестве элементов и тем самым переписать их. Адреса могут быть расположены в диапазоне 0000h-FFFFh.

## Загрузка «хорошего» EXE модуля в основную память

Результат запуска «хорошего» EXE модуля в отладчике представлен на рис. 10.

The screenshot shows a debugger window with the following content:

CPU 80486	
cs:0123 B8AF48	mov ax, 48AF
cs:0126 BED8	mov ds, ax
cs:0128 E83BFF	call 0066
cs:012B E893FF	call 00C1
cs:012E E8B3FF	call 00E4
cs:0131 E8C8FF	call 00FC
cs:0134 32C0	xor al, al
cs:0136 B44C	mov ah, 4C
cs:0138 CD21	int 21
cs:013A 0000	add [bx+si], al
cs:013C 0000	add [bx+si], al
cs:013E 0000	add [bx+si], al
cs:0140 0000	add [bx+si], al

Register	Value
ax	0000
bx	0000
cx	0000
dx	0000
si	0000
di	0000
bp	0000
sp	0020
ds	489D
es	489D
ss	48AD
cs	48BB
ip	0123

Address	Disassembly
ds:0000	CD 20 FF 9F 00 EA FF FF = f 0
ds:0008	AD DE E0 01 C5 15 AA 01 i 0 0 0 0 0 0
ds:0010	C5 15 89 02 20 10 92 01 0 0 0 0 0 0
ds:0018	01 01 01 00 02 FF FF FF 0 0 0 0

Register	Value
c	=0
z	=0
s	=0
o	=0
p	=0
a	=0
i	=1
d	=0

Register	Value
ss	0022 6570
ss	0020 7954

Рисунок 10 – Результат загрузки «хорошего» .EXE в отладчик

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

Сначала помещается PSP, а дальше устанавливаются сегментные регистры. DS и ES устанавливаются на начало PSP (DS = ES = 489D). SS – на начало сегмента стека (SS = 48AD), CS – на начало сегмента кода (CS = 48BB).

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. При запуске программы в SS заносится адрес сегмента стека, его начало, а в SP – адрес его вершины. (см. рис. 10).

4. Как определяется точка входа?

Точка входа в программу определяется с помощью директивы END, после которой указывается метка, куда переходит программа при запуске.

## ПРИЛОЖЕНИЕ А

### BAD\_EXE.ASM

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN
PC_TYPE    db 'Type PC: $'
PC_VERS    db 0DH,0AH,'System version:  . ',0DH,0AH,'$'
PC_OEM     db 'Original Equipment Manufacturer:      ',0DH,0AH,'$'
SER_NUM    db 'User serial number:                ',0DH,0AH,'$'
TYPE_PC     db 'PC $'
TYPE_PCXT  db 'PC/XT $'
TYPE_AT     db 'AT $'
TYPE_PS2_30 db 'PS2 model 30 $'
TYPE_PS2_50_60 db 'PS2 model 50/60 $'
TYPE_PS2_80 db 'PS2 model 80 $'
TYPE_PCjr   db 'PCjr $'
TYPE_PC_Con db 'PC Convertible $'
TETR_TO_HEX PROC near
    and AL,0Fh
    cmp AL,09
    jbe NEXT
    add AL,07
NEXT:      add AL,30h
    ret
TETR_TO_HEX ENDP

BYTE_TO_HEX PROC near
    push CX
    mov  AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov  CL,4
    shr  AL,CL
    call TETR_TO_HEX
    pop  CX
    ret
BYTE_TO_HEX ENDP

WRD_TO_HEX PROC near
    push BX
    mov  BH,AH
```

```

        call BYTE_TO_HEX
        mov  [DI],AH
        dec  DI
        mov  [DI],AL
        dec  DI
        mov  AL,BH
        call BYTE_TO_HEX
        mov  [DI],AH
        dec  DI
        mov  [DI],AL
        pop  BX
        ret

```

WRD\_TO\_HEX ENDP

BYTE\_TO\_DEC PROC near

```

        push CX
        push DX
        xor  AH,AH
        xor  DX,DX
        mov  CX,10
loop_bd: div CX
        or   DL,30h
        mov  [SI],DL
        dec  SI
        xor  DX,DX
        cmp  AX,10
        jae  loop_bd
        cmp  AL,00h
        je   end_1
        or   AL,30h
        mov  [SI],AL
end_1:   pop  DX
        pop  CX
        ret

```

BYTE\_TO\_DEC ENDP

LINE\_OUTPUT PROC near

```

        mov  AH,09H
        int  21H
        ret

```

LINE\_OUTPUT ENDP

GET\_PC\_TYPE PROC near

```

        mov  AX,0F000H

```

```

        mov ES,AX
        mov AL,ES:[0FFFEH]
        ret
GET_PC_TYPE ENDP

```

Write\_PC\_TYPE PROC near

```

    push DX
    push AX
    mov DX, OFFSET PC_TYPE
    call LINE_OUTPUT
    call GET_PC_TYPE
    cmp AL,0FFH
    je PC

    cmp AL,0FEH
    je PCXT

    cmp AL,0FBH
    je PCXT

    cmp AL,0FCH
    je PC_AT

    cmp AL,0FAH
    je PS2_30

    cmp AL,0F8H
    je PS2_80

    cmp AL,0FDH
    je PCjr

    cmp AL,0F9H
    je PC_Convertible

PC:
    mov DX, OFFSET TYPE_PC
    jmp print
PCXT:
    mov DX, OFFSET TYPE_PCXT
    jmp print
PC_AT:
    mov DX, OFFSET TYPE_AT

```

```

        jmp print
PS2_30:
        mov DX, OFFSET TYPE_PS2_30
        jmp print
PS2_80:
        mov DX, OFFSET TYPE_PS2_80
        jmp print
PCjr:
        mov DX, OFFSET TYPE_PCjr
        jmp print
PC_Convertible:
        mov DX, OFFSET TYPE_PC_Con
        jmp print
print:
        call LINE_OUTPUT
        pop AX
        pop DX
        ret
Write_PC_TYPE ENDP

```

```

Write_PC_VERS PROC near
        push AX
        push SI
        mov SI,OFFSET PC_VERS
        add SI,18
        call BYTE_TO_DEC
        mov AL,AH
        add SI,3
        call BYTE_TO_DEC
        mov DX,OFFSET PC_VERS
        call LINE_OUTPUT
        pop SI
        pop AX
        ret
Write_PC_VERS ENDP

```

```

Write_PC_OEM PROC near
        push AX
        push BX
        push SI
        mov SI,OFFSET PC_OEM
        add SI,35
        mov AL,BH

```

```

        call BYTE_TO_DEC
        mov  DX,OFFSET PC_OEM
        call LINE_OUTPUT
        pop  SI
        pop  BX
        pop  AX
        ret
Write_PC_OEM ENDP
Write_SER_NUM PROC near
        push AX
        push BX
        push CX
        push SI
        mov  AL,BL
        call BYTE_TO_HEX
        mov  DI,OFFSET SER_NUM
        add  DI,20
        mov  [DI],AX
        mov  AX,CX
        mov  DI,OFFSET SER_NUM
        add  DI,25
        call WRD_TO_HEX
        mov  DX,OFFSET SER_NUM
        call LINE_OUTPUT
        pop  SI
        pop  CX
        pop  BX
        pop  AX
        ret
Write_SER_NUM ENDP
BEGIN:
        call Write_PC_TYPE
        mov  AX,0
        mov  AH,30H
        int  21H
        call Write_PC_VERS
        call Write_PC_OEM
        call Write_SER_NUM
        xor  AL,AL
        mov  AH,4CH
        int  21H
TESTPC ENDS
        END START

```

## ПРИЛОЖЕНИЕ Б

### GOOD\_EXE.ASM

```
ASTACK SEGMENT STACK
    DW 010H DUP(?)
ASTACK ENDS

DATA SEGMENT
PC_TYPE      db 'Type PC: $'
PC_VERS      db 0DH,0AH,'System version:  . ',0DH,0AH,'$'
PC_OEM       db 'Original Equipment Manufacturer:      ',0DH,0AH,'$'
SER_NUM      db 'User serial number:          ',0DH,0AH,'$'
TYPE_PC      db 'PC $'
TYPE_PCXT    db 'PC/XT $'
TYPE_AT      db 'AT $'
TYPE_PS2_30   db 'PS2 model 30 $'
TYPE_PS2_50_60 db 'PS2 model 50/60 $'
TYPE_PS2_80   db 'PS2 model 80 $'
TYPE_PCjr     db 'PCjr $'
TYPE_PC_Con   db 'PC Convertible $'
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA, ES:NOTHING, 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 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

```
LINE_OUTPUT PROC near
    mov  AH,09H
    int  21H
```

```

        ret
LINE_OUTPUT ENDP

GET_PC_TYPE PROC near
    mov AX,0F000H
    mov ES,AX
    mov AL,ES:[0FFFEH]
    ret
GET_PC_TYPE ENDP

Write_PC_TYPE PROC near
    push DX
    push AX
    mov DX, OFFSET PC_TYPE
    call LINE_OUTPUT
    call GET_PC_TYPE

    cmp AL,0FFH
    je PC

    cmp AL,0FEH
    je PCXT

    cmp AL,0FBH
    je PCXT

    cmp AL,0FCH
    je PC_AT

    cmp AL,0FAH
    je PS2_30

    cmp AL,0F8H
    je PS2_80

    cmp AL,0FDH
    je PCjr

    cmp AL,0F9H
    je PC_Convertible

PC:
    mov DX, OFFSET TYPE_PC

```

```

        jmp print
PCXT:
        mov DX, OFFSET TYPE_PCXT
        jmp print
PC_AT:
        mov DX, OFFSET TYPE_AT
        jmp print
PS2_30:
        mov DX, OFFSET TYPE_PS2_30
        jmp print
PS2_80:
        mov DX, OFFSET TYPE_PS2_80
        jmp print
PCjr:
        mov DX, OFFSET TYPE_PCjr
        jmp print
PC_Convertible:
        mov DX, OFFSET TYPE_PC_Con
        jmp print
print:
        call LINE_OUTPUT
        pop AX
        pop DX
        ret
Write_PC_TYPE ENDP

```

```

Write_PC_VERS PROC near
        mov AX,0
        mov AH,30H
        int 21H
        push AX
        push SI
        mov SI,OFFSET PC_VERS
        add SI,18
        call BYTE_TO_DEC
        mov AL,AH
        add SI,3
        call BYTE_TO_DEC
        mov DX,OFFSET PC_VERS
        call LINE_OUTPUT
        pop SI
        pop AX
        ret

```

```
Write_PC_VERS  ENDP
```

```
Write_PC_OEM PROC near
```

```
    push AX
    push BX
    push SI
    mov  SI,OFFSET PC_OEM
    add  SI,35
    mov  AL,BH
    call BYTE_TO_DEC
    mov  DX,OFFSET PC_OEM
    call LINE_OUTPUT
    pop  SI
    pop  BX
    pop  AX
    ret
```

```
Write_PC_OEM ENDP
```

```
Write_SER_NUM PROC near
```

```
    push AX
    push BX
    push CX
    push SI
    mov  AL,BL
    call BYTE_TO_HEX
    mov  DI,OFFSET SER_NUM
    add  DI,20
    mov  [DI],AX
    mov  AX,CX
    mov  DI,OFFSET SER_NUM
    add  DI,25
    call WRD_TO_HEX
    mov  DX,OFFSET SER_NUM
    call LINE_OUTPUT
    pop  SI
    pop  CX
    pop  BX
    pop  AX
    ret
```

```
Write_SER_NUM ENDP
```

```
BEGIN:
```

```
    mov  AX,DATA
```

```
    mov  DS,AX
    call Write_PC_TYPE
    call Write_PC_VERS
    call Write_PC_OEM
    call Write_SER_NUM
    xor  AL,AL
    mov  AH,4CH
    int  21H
CODE  ENDS
END  BEGIN
```