

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

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

Студентка гр. 8381

\_\_\_\_\_

Муковский Д.В.

Преподаватель

\_\_\_\_\_

Губкин А.Ф.

Санкт-Петербург

2020

### **Цель работы.**

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

### **Необходимые сведения для составления программы.**

Тип IBM PC хранится в байте по адресу 0F000:0FFFE, в предпоследнем байте ROM BIOS. Соответствие кода и типа компьютера представлено в таблице 1:

Таблица 1 – Идентификация типа компьютера

Код	Тип компьютера
FF	Оригинальный IBM PC
FE	XT, Portable PC
FD	PCjr
FC	AT
FB	XT с памятью 640 К на мат. плате
FA	PS/2 модель 25 или 30
F9	Convertible PC
F8	PS/2 модели 55SX, 70, 80
9A	Compaq XT, Compaq Plus
30	Sperry PC
2D	Compaq PC, Compaq Deskpro

Для определения версии MS DOS следует воспользоваться функцией 30H прерывания 21H. Входным параметром является номер функции в AH:

MOV AH, 30h

INT 21h

Выходными параметрами являются:

AL – номер основной версии. Если 0, то <2.0;

AH – номер модификации;

ВН – серийный номер OEM (Original Equipment Manufacturer);

BL:СХ – 24-битовый серийный номер пользователя.

### **Постановка задачи.**

Требуется реализовать текст исходного .COM модуля, который определяет тип РС и версию системы. Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип РС и выводить строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код переводиться в символьную строку, содержащую запись шестнадцатеричного числа и выводиться на экран в виде соответствующего сообщения. Затем определяется версия системы. Ассемблерная программа должна по значениям регистров AL и AH формировать текстовую строку в формате xx.yy, где xx - номер основной версии, а yy - номер модификации в десятичной системе счисления, формировать строки с серийным номером OEM (Original Equipment Manufacturer) и серийным номером пользователя. Полученные строки выводятся на экран.

Далее необходимо отладить полученный исходный модуль и получить «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля.

Затем нужно написать текст «хорошего» .EXE модуля, который выполняет те же функции, что и модуль .COM, далее его построить, отладить и сравнить исходные тексты для .COM и .EXE модулей.

### **Выполнение работы.**

Первым делом был написан исходный модуль .COM, который определяет тип РС и версию системы. Полученный модуль был скомпилирован, в результате чего был получен “плохой” .EXE модуль, а с помощью программы exe2bin получен “хороший” .COM модуль. После

линковки было выведено предупреждение, что сегмент стека отсутствует, что представлено на рис. 1.

```
Run File [LR1_COM.EXE]:  
List File [NUL.MAP]:  
Libraries [.LIB]:  
LINK : warning L4021: no stack segment
```

Рисунок 1 – Предупреждение об отсутствие стека

Результат выполнения «плохого» .EXE модуля представлен на рис. 2.

S:\>LR1\_COM.exe



Рисунок 2 – Выполнение «плохого» .EXE модуля

Результат выполнения «хорошего» .COM модуля представлен на рис. 3.

```
S:\>LR1_COM.com  
IBM PC TYPE: AT  
MSDOS VERSION: 5.0  
OEM NUMBER: 0  
SERIAL NUMBER: 00 0000
```

Рисунок 3 - Выполнение «хорошего» .COM модуля

Далее был написан исходный .EXE модуль. В результате компилирования был получен «хороший» .EXE модуль. Результат его выполнения представлен на рис. 4.

```
S:\>lr1_exe.exe  
IBM PC TYPE: AT  
MSDOS VERSION: 5.0  
OEM NUMBER: 0  
SERIAL NUMBER: 00 0000
```

Рисунок 4 - Выполнение «хорошего» .EXE модуля

## Отличия исходных текстов COM и EXE программ.

### 1. Сколько сегментов должна содержать COM-программа?

Один сегмент, в котором размещаются программные коды, данные и стек.

### 2. EXE программа?

Для программного кода, данных и стека предусмотрены отдельные сегменты.

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

Директива ORG 100h (смещение 100h), так как при загрузке COM-файла в память, DOS занимает первые 256 байт (100h) блоком данных PSP и располагает код программы только после этого блока.

Директива ASSUME, ставящая в соответствие адрес сегмента программы сегментам кода и данных. Ошибки программы без данной директивы представлены на рис. 5.

```
LR1_COM.asm(71): error A2062: Missing or unreachable CS
LR1_COM.asm(79): error A2062: Missing or unreachable CS
LR1_COM.asm(91): error A2062: Missing or unreachable CS
LR1_COM.asm(100): error A2062: Missing or unreachable CS
LR1_COM.asm(111): error A2062: Missing or unreachable CS
LR1_COM.asm(115): error A2062: Missing or unreachable CS
LR1_COM.asm(119): error A2062: Missing or unreachable CS
LR1_COM.asm(123): error A2062: Missing or unreachable CS
LR1_COM.asm(127): error A2062: Missing or unreachable CS
LR1_COM.asm(131): error A2062: Missing or unreachable CS
LR1_COM.asm(135): error A2062: Missing or unreachable CS
LR1_COM.asm(139): error A2062: Missing or unreachable CS
LR1_COM.asm(143): error A2062: Missing or unreachable CS
LR1_COM.asm(147): error A2062: Missing or unreachable CS
LR1_COM.asm(151): error A2062: Missing or unreachable CS
LR1_COM.asm(184): error A2062: Missing or unreachable CS
LR1_COM.asm(196): error A2062: Missing or unreachable CS
LR1_COM.asm(204): error A2062: Missing or unreachable CS

49960 + 453205 Bytes symbol space free

0 Warning Errors
23 Severe Errors
```

Рисунок 5 - .COM без директивы ASSUME

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

Нет, так как в отличие от EXE-программы, в которой существует таблица настроек, хранящая информацию о типе адресов и их местоположении в коде, COM-программа ею не располагает. Адреса сегментов определяются загрузчиком в момент запуска программы, а не в момент компиляции. Поэтому в COM-программах невозможно использовать команды вида: mov [регистр], seg [сегмент].

## Отличия форматов файлов .COM и .EXE модулей

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

Вид файла COM в шестнадцатеричном формате представлен на рис. 6.

0000000000: E9 57 01 50 43 0D 0A 24	50 43 2F 58 54 0D 0A 24	щW0PC.0\$PC/XT0\$
0000000010: 41 54 0D 0A 24 50 53 32	20 4D 4F 44 45 4C 20 33	AT0\$PS2 MODEL 3
0000000020: 30 0D 0A 24 50 53 32 20	4D 4F 44 45 4C 20 35 30	00\$PS2 MODEL 50
0000000030: 2F 36 30 0D 0A 24 50 53	32 20 4D 4F 44 45 4C 20	/600\$PS2 MODEL
0000000040: 38 30 0D 0A 24 50 73 6A	72 0D 0A 24 50 43 20 43	800\$P\$jr0\$PC C
0000000050: 4F 4E 56 45 52 54 49 42	4C 45 0D 0A 24 41 4E 4F	ONVERTIBLE0\$ANO
0000000060: 54 48 45 52 20 54 59 50	45 3A 20 0D 0A 24 49 42	THER TYPE: 0\$IB
0000000070: 4D 20 50 43 20 54 59 50	45 3A 20 24 4D 53 44 4F	M PC TYPE: \$MSDO
0000000080: 53 20 56 45 52 53 49 4F	4E 3A 20 20 2E 20 0D 0A	S VERSION: . 0
0000000090: 24 4F 45 4D 20 4E 55 4D	42 45 52 3A 20 20 20 20	\$OEM NUMBER:
00000000A0: 20 0D 0A 24 53 45 52 49	41 4C 20 4E 55 4D 42 45	0\$SERIAL NUMBE
00000000B0: 52 3A 20 20 20 20 20 20	0D 0A 24 24 0F 3C 09 76	R: 0\$<ov
00000000C0: 02 04 07 04 30 C3 51 8A	E0 E8 EF FF 86 C4 B1 04	0♦♦♦0 QKршя Ж—♦
00000000D0: D2 E8 E8 E6 FF 59 C3 53	8A FC E8 E9 FF 88 25 4F	тшщц Y SK№щц И%O
00000000E0: 88 05 4F 8A C7 32 E4 E8	DC FF 88 25 4F 88 05 5B	И♦OK 2фш И%OI♦[
00000000F0: C3 51 52 50 32 E4 33 D2	B9 0A 00 F7 F1 80 CA 30	QRP2ф3т  üÄ±0
0000000100: 88 14 4E 33 D2 3D 0A 00	73 F1 3D 00 00 76 04 0C	ИJN3т= sē= v♦♀
0000000110: 30 88 04 58 5A 59 C3 50	B4 09 CD 21 58 C3 BA 03	0И♦XZY P o=IX ♥
0000000120: 01 EB 31 90 BA 08 01 EB	2B 90 BA 10 01 EB 25 90	0ы1P 0ы+P 0ы%P
0000000130: BA 15 01 EB 1F 90 BA 24	01 EB 19 90 BA 36 01 EB	\$0ы▼P  \$0ы↓P  60ы
0000000140: 13 90 BA 45 01 EB 0D 90	BA 4C 01 EB 07 90 BA 5D	!!P  E0ы↓P  L0ы•P  ]
0000000150: 01 EB 01 90 E8 C0 FF EB	38 90 52 50 BA 6E 01 E8	0ы0PшL ы8PRP  п0ш
0000000160: B5 FF B8 00 F0 8E C0 26	A0 FE FF 3C FF 74 AF 3C	‡ ‡ EOL&a■ < тп<
0000000170: FE 74 B1 3C FB 74 AD 3C	FC 74 AF 3C FA 74 B1 3C	■t<vтн<№тп<·t<
0000000180: FC 74 B3 3C F8 74 B5 3C	FD 74 B7 3C F9 74 B9 EB	№t <°t‡<тt‡<·t‡ы
0000000190: BD B4 30 CD 21 8D 36 7C	01 83 C6 0F E8 52 FF 83	J‡0= H6 0Г†шR Г
00000001A0: C6 03 8A C4 E8 4A FF BA	7C 01 E8 6A FF 8A C7 8D	†♥K—шJ    0шj K H
00000001B0: 36 91 01 83 C6 0C E8 38	FF BA 91 01 E8 58 FF 8A	6C0Г†фш8   C0шX K
00000001C0: C3 8D 36 A4 01 83 C6 0F	E8 FB FE 89 04 83 C6 06	H6д0Г†шv■й♦Г†♦
00000001D0: 8B FE 8B C1 E8 00 FF BA	A4 01 E8 3A FF 58 5A 32	л■лLш   д0ш: XZ2
00000001E0: C0 B4 4C CD 21 C3		L=L= †

Рисунок 6 - .COM модуль в шестнадцатеричном виде

Данный файл состоит из одного сегмента, размер файла не может превышать 64 Кб. Код начинается с адреса 0h, но при запуске модуля устанавливается смещение в 100h.

Сначала идут байты данных, на рис.6 первым выделено слово «PC». А байты, отвечающие за код, начинаются с номера ВС и на приведенном рисунке выделена первая команда процедуры TETR\_TO\_HEX (см. Приложение А).

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

Вид «плохого» EXE файла в шестнадцатеричном виде представлен на рис. 7.

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	
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	
0000001C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000001D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000001E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000001F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000200:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000210:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000220:	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	
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 57 01 50 43 00 0A 24	50 43 2F 58 54 00 0A 24	éW0PCJw\$PC/XTJw\$
000000310:	41 54 00 0A 24 50 53 32	20 40 4F 44 45 4C 20 33	ATJw\$PS2 MODEL 3
000000320:	30 00 0A 24 50 53 32 20	40 4F 44 45 4C 20 35 30	0Jw\$PS2 MODEL 50
000000330:	2F 36 30 00 0A 24 50 53	32 20 40 4F 44 45 4C 20	/60Jw\$PS2 MODEL
000000340:	38 30 00 0A 24 50 73 6A	72 00 0A 24 50 43 20 43	80Jw\$P\$jrJw\$PC C
000000350:	4F 4E 56 45 52 54 49 42	4C 45 00 0A 24 41 4E 4F	ONVERTIBLEJw\$ANO
000000360:	54 48 45 52 20 54 59 50	45 3A 20 00 0A 24 49 42	THER TYPE: Jw\$IB
000000370:	40 20 50 43 20 54 59 50	45 3A 20 24 40 53 44 4F	M PC TYPE: \$MSDO
000000380:	53 20 56 45 52 53 49 4F	4E 3A 20 20 2E 20 00 0A	S VERSION: . Jw\$
000000390:	24 4F 45 40 20 4E 55 40	42 45 52 3A 20 20 20 20	\$OEM NUMBER:
0000003A0:	20 00 0A 24 53 45 52 49	41 4C 20 4E 55 40 42 45	Jw\$SERIAL NUMBE
0000003B0:	52 3A 20 20 20 20 20 20	00 0A 24 24 0F 3C 09 76	R: Jw\$S0<ov
0000003C0:	02 04 07 04 30 C3 51 8A	E0 E8 EF FF 86 C4 B1 04	0000003D0: D2 E8 E8 E6 FF 59 C3 53
0000003D0:	D2 E8 E8 E6 FF 59 C3 53	8A FC E8 E9 FF 88 25 4F	0000003E0: 88 05 4F 8A C7 32 E4 E8
0000003E0:	88 05 4F 8A C7 32 E4 E8	DC FF 88 25 4F 88 05 5B	0000003F0: C3 51 52 50 32 E4 33 D2
0000003F0:	C3 51 52 50 32 E4 33 D2	B9 0A 00 F7 F1 80 CA 30	000000400: 88 14 4E 33 D2 30 0A 00
000000400:	88 14 4E 33 D2 30 0A 00	73 F1 3D 00 00 76 04 0C	000000410: 30 88 04 58 5A 59 C3 50
000000410:	30 88 04 58 5A 59 C3 50	B4 09 CD 21 58 C3 BA 03	000000420: 01 E8 31 90 8A 08 01 E8
000000420:	01 E8 31 90 8A 08 01 E8	2B 90 BA 10 01 E8 25 90	000000430: BA 15 01 E8 1F 90 BA 24
000000430:	BA 15 01 E8 1F 90 BA 24	01 E8 19 90 BA 36 01 E8	000000440: 13 90 BA 45 01 E8 0D 90
000000440:	13 90 BA 45 01 E8 0D 90	BA 4C 01 E8 07 90 BA 5D	000000450: 01 E8 01 90 E8 C0 FF E8
000000450:	01 E8 01 90 E8 C0 FF E8	38 90 52 50 BA 6E 01 E8	000000460: 85 FF B8 00 F0 8E C0 26
000000460:	85 FF B8 00 F0 8E C0 26	A0 FE FF 3C FF 74 AF 3C	000000470: FE 74 B1 3C FB 74 AD 3C
000000470:	FE 74 B1 3C FB 74 AD 3C	FC 74 AF 3C FA 74 B1 3C	000000480: FC 74 B3 3C F8 74 B5 3C
000000480:	FC 74 B3 3C F8 74 B5 3C	FD 74 B7 3C F9 74 B9 E8	000000490: 8D B4 3C CD 21 8D 36 7C
000000490:	8D B4 3C CD 21 8D 36 7C	01 83 C6 0F E8 52 FF 83	0000004A0: C6 03 8A C4 E8 4A FF BA
0000004A0:	C6 03 8A C4 E8 4A FF BA	7C 01 E8 6A FF 8A C7 8D	0000004B0: 36 91 01 83 C6 0C E8 38
0000004B0:	36 91 01 83 C6 0C E8 38	FF BA 91 01 E8 58 FF 8A	0000004C0: C3 8D 36 A4 01 83 C6 0F
0000004C0:	C3 8D 36 A4 01 83 C6 0F	E8 FB FE 89 04 83 C6 06	0000004D0: 8B FE 8B C1 E8 00 FF BA
0000004D0:	8B FE 8B C1 E8 00 FF BA	A4 01 E8 3A FF 58 5A 32	0000004E0: C0 B4 4C CD 21 C3
0000004E0:	C0 B4 4C CD 21 C3		

Рисунок 7 – «плохой» .EXE модуль в шестнадцатеричном виде

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





В «хорошем» EXE с нулевого адреса располагается таблица настроек. Также перед кодом выделена память под стек, в данном случае при размере стека 200h код располагается с адреса 400h. Еще одним отличием «хорошего» модуля является то, что в нем не резервируются дополнительно 100h, требуемые для PSP в .COM файле.

## Загрузка COM модуля в основную память

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

Загрузка файла .COM в отладчике Turbo Debugger представлен на рис. 9.

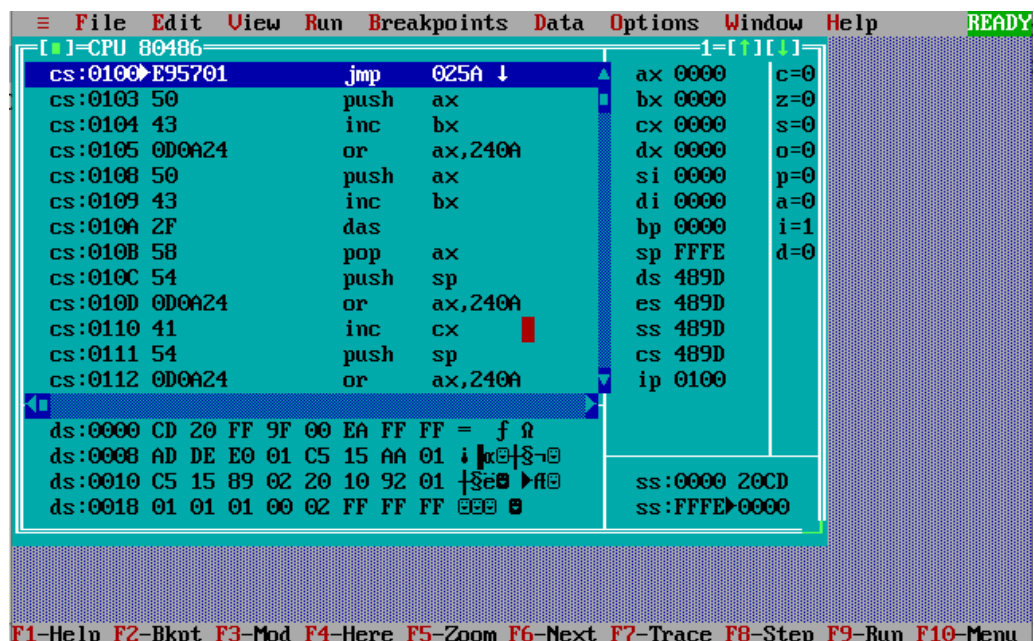


Рисунок 9 – Отладка .COM файла

После загрузки COM-программы в память сегментные регистры указывают на начало PSP. Код располагается с адреса 100h, IP = 0100h.

**2. Что располагается с 0 адреса?**

С нулевого адреса располагается адрес начала PSP.

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

Сегментные регистры имеют значение 489Dh и указывают на PSP.

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

Стек определяется автоматически, указатель стека устанавливается на конец сегмента. Если для программы размер сегмента 64 кб достаточен, то DOS устанавливает в регистре SP-адрес конца сегмента – FFFEh, что можно увидеть на рис. 9. Адреса расположены в диапазоне от FFFEh до 0000h.

### Загрузка «хорошего» EXE модуля в основную память

Загрузка файла .EXE в отладчике Turbo Debugger представлен на рис. 10.

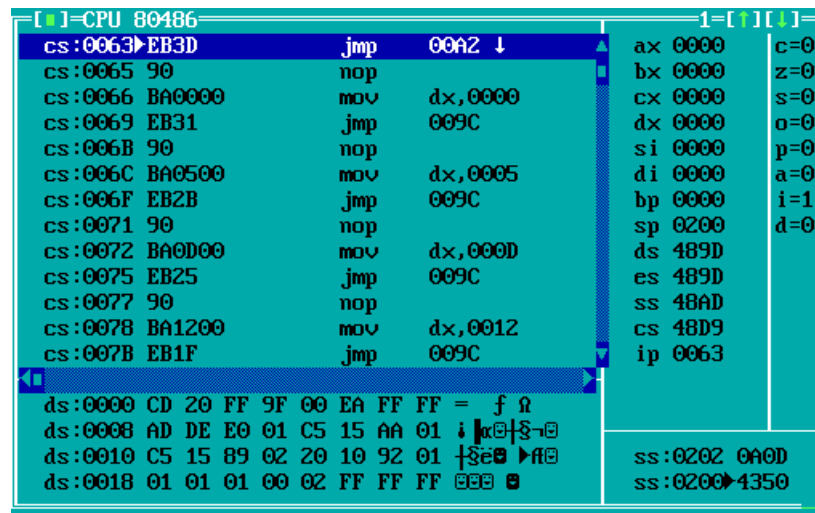


Рисунок 10 – Отладка .EXE файла

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

Для PSP и программы выделяется блок памяти. После запуска программы DS и ES указывают на начало PSP (489Dh), CS – на начало сегмента команд (48D9), а SS – на начало сегмента стека (48AD). IP имеет ненулевое значение, так как в программе есть дополнительные процедуры, которые расположены до основной.

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

Изначально регистры указывают на начало PSP. Поэтому в начале программы для корректной работы регистр DS инициализируется адресом начала сегмента данных.

### 3. Как определяется стек?

Стек определяется в исходном коде при помощи директивы. STACK, а при исполнении в регистр SS записывается адрес начала сегмента стека, а в указатель стека SP- указатель на конец стека.

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

Смещение точки входа в программу загружается в указатель команд IP и определяется операндом директивы END, который называется точкой входа.

#### **Вывод.**

В ходе работы было проведено исследование различий в структурах исходных текстов модулей .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

# ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ LR1\_COM.ASM

```
LAB SEGMENT

                                ASSUME CS:LAB, DS:LAB, ES:NOTHING, SS:NOTHING
                                ORG 100H

MAIN: JMP BEGIN

; данные
PC                                DB    'PC', 0DH, 0AH, '$'
PC_XT                            DB    'PC/XT', 0DH, 0AH, '$'
AT_                              DB    'AT', 0DH, 0AH, '$'
PS2_30                          DB    'PS2 MODEL 30', 0DH, 0AH, '$'
PS2_5060                        DB    'PS2 MODEL 50/60', 0DH, 0AH, '$'
PS2_80                          DB    'PS2 MODEL 80', 0DH, 0AH, '$'
PCJR                            DB    'Psjr', 0DH, 0AH, '$'
PC_CONVERTIBLE                  DB    'PC CONVERTIBLE', 0DH, 0AH, '$'
TYPE_ANOTHER DB 'ANOTHER TYPE: ', 0DH, 0AH, '$'

IBM_PC_NAME                      DB    'IBM PC TYPE: ', '$'
OS_NAME                         DB    'MSDOS VERSION:  . ', 0DH, 0AH,
'$'
OEM_NAME                        DB    'OEM NUMBER:      ', 0DH, 0AH, '$'
SERIAL_NAME                     DB    'SERIAL NUMBER:      ', 0DH, 0AH, '$'

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 шест. 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 цифра
RET
```

BYTE\_TO\_HEX ENDP

-----  
-----

WRD\_TO\_HEX PROC NEAR  
;перевод в 16 с/с 16 разрядного числа  
;в ax - число, 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
XOR  AH, AH
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, si - адрес поля младшей цифры

```
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 AX, 00H
JBE END_L
OR AL, 30H
MOV [SI], AL
```

END\_L:

```
POP AX
POP DX
POP CX
RET
```

BYTE\_TO\_DEC

ENDP

-----  
-----

PRINT PROC NEAR

```
PUSH AX
MOV AH, 09H
INT 21H
POP AX
RET
```

PRINT ENDP

-----  
-----

PC\_WRITE:

MOV DX, OFFSET PC  
JMP WRITE

PC\_XT\_WRITE:

MOV DX, OFFSET PC\_XT  
JMP WRITE

AT\_WRITE:

MOV DX, OFFSET AT\_  
JMP WRITE

PS2\_30\_WRITE:

MOV DX, OFFSET PS2\_30  
JMP WRITE

PS2\_5060\_WRITE:

MOV DX, OFFSET PS2\_5060  
JMP WRITE

PS2\_80\_WRITE:

MOV DX, OFFSET PS2\_80  
JMP WRITE

PCJR\_WRITE:

MOV DX, OFFSET PCJR  
JMP WRITE

PC\_CONVERTIBLE\_WRITE:

MOV DX, OFFSET PC\_CONVERTIBLE  
JMP WRITE

PC\_ANOTHER\_WRITE:

MOV DX, OFFSET TYPE\_ANOTHER  
JMP WRITE

WRITE:

CALL PRINT

JMP OS

BEGIN:

PUSH DX

PUSH AX

MOV DX, OFFSET IBM\_PC\_NAME

CALL PRINT

MOV AX, 0F000H

MOV ES, AX

MOV AL, ES:[0FFFEH]

CMP AL, 0FFH

JE PC\_WRITE

CMP AL, 0FEH

JE PC\_XT\_WRITE

CMP AL, 0FBH

JE PC\_XT\_WRITE

CMP AL, 0FCH

JE AT\_WRITE

CMP AL, 0FAH

JE PS2\_30\_WRITE

CMP AL, 0FCH

JE PS2\_5060\_WRITE

CMP AL, 0F8H

JE PS2\_80\_WRITE

CMP AL, 0FDH

JE PCJR\_WRITE

CMP AL, 0F9H

JE PC\_CONVERTIBLE\_WRITE

JMP PC\_ANOTHER\_WRITE

OS:

MOV AH, 30H ;дает номер версии dos al -  
основная версия, ah - номер модификации



```

INT 21H
LEA      SI, OS_NAME
ADD      SI, 15
CALL BYTE_TO_DEC
ADD      SI, 3
MOV  AL, AH
CALL      BYTE_TO_DEC
MOV DX, OFFSET OS_NAME
CALL PRINT

```

OEM:

```

MOV  AL, BH ;bh - серийный номер
LEA  SI, OEM_NAME
ADD  SI, 12
CALL BYTE_TO_DEC
MOV  DX, OFFSET OEM_NAME
CALL PRINT

```

SERIAL:

```

MOV      AL, BL
LEA      SI, SERIAL_NAME
ADD      SI, 15
CALL BYTE_TO_HEX
MOV      [SI], AX
ADD      SI, 6
MOV      DI, SI
MOV  AX, CX
CALL WRD_TO_HEX
MOV  DX, OFFSET SERIAL_NAME
CALL PRINT

```

```

POP      AX
POP  DX
XOR      AL, AL
MOV  AH, 4CH
INT      21H
RET

```

LAB

ENDS

END MAIN

## ПРИЛОЖЕНИЕ В

### ИСХОДНЫЙ КОД ПРОГРАММЫ LR1\_EXE.ASM

```
ASTACK SEGMENT STACK
                                DW 100h DUP(?)

ASTACK ENDS

; данные
DATA SEGMENT

PC DB 'PC', 0DH, 0AH, '$'
PC_XT DB 'PC/XT', 0DH, 0AH, '$'
AT_ DB 'AT', 0DH, 0AH, '$'
PS2_30 DB 'PS2 MODEL 30', 0DH, 0AH, '$'
PS2_5060 DB 'PS2 MODEL 50/60', 0DH, 0AH, '$'
PS2_80 DB 'PS2 MODEL 80', 0DH, 0AH, '$'
PCJR DB 'Psjr', 0DH, 0AH, '$'
PC_CONVERTIBLE DB 'PC CONVERTIBLE', 0DH, 0AH,
'$'

TYPE_ANOTHER DB 'ANOTHER TYPE: ', 0DH, 0AH, '$'
IBM_PC_NAME DB 'IBM PC TYPE: ', '$'
OS_NAME DB 'MSDOS VERSION: . ',
0DH, 0AH, '$'

OEM_NAME DB 'OEM NUMBER: ', 0DH, 0AH,
'$'

SERIAL_NAME DB 'SERIAL NUMBER: ',
0DH, 0AH, '$'
DATA ENDS

CODE SEGMENT

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

TETR_TO_HEX PROC NEAR
    AND AL, 0FH
    CMP AL, 09
    JBE NEXT
    ADD AL, 07
    NEXT: ADD AL, 30H
    RET
TETR_TO_HEX ENDP
```

-----  
-----

```
BYTE_TO_HEX          PROC NEAR
;байт в al переводится в два символа шест. 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 цифра
        RET
BYTE_TO_HEX          ENDP
```

-----  
-----

```
WRD_TO_HEX           PROC NEAR
;перевод в 16 с/с 16 разрядного числа
;в ax - число, 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
        XOR  AH, AH
        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, si - адрес поля младшей цифры
    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  AX, 00H
    JBE  END_L
    OR  AL, 30H
    MOV  [SI], AL

END_L:
    POP  AX
    POP  DX
    POP  CX
    RET

BYTE_TO_DEC      ENDP
```

-----  
-----

```
PRINT PROC NEAR
    PUSH AX
    MOV  AH, 09H
    INT  21H
    POP  AX
```

RET

PRINT ENDP

;-  
-----

MAIN PROC FAR

JMP BEGIN

PC\_WRITE:

MOV DX, OFFSET PC

JMP WRITE

PC\_XT\_WRITE:

MOV DX, OFFSET PC\_XT

JMP WRITE

AT\_WRITE:

MOV DX, OFFSET AT\_

JMP WRITE

PS2\_30\_WRITE:

MOV DX, OFFSET PS2\_30

JMP WRITE

PS2\_5060\_WRITE:

MOV DX, OFFSET PS2\_5060

JMP WRITE

PS2\_80\_WRITE:

MOV DX, OFFSET PS2\_80

JMP WRITE

PCJR\_WRITE:

MOV DX, OFFSET PCJR

JMP WRITE

PC\_CONVERTIBLE\_WRITE:

```
MOV DX, OFFSET PC_CONVERTIBLE
JMP WRITE
```

```
PC_ANOTHER_WRITE:
MOV DX, OFFSET TYPE_ANOTHER
JMP WRITE
```

```
WRITE:
CALL PRINT
JMP OS
```

```
BEGIN:
PUSH DX
PUSH AX

MOV AX, DATA
MOV DS, AX

MOV DX, OFFSET IBM_PC_NAME
CALL PRINT

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

CMP AL, 0FFH
JE PC_WRITE
CMP AL, 0FEH
JE PC_XT_WRITE
CMP AL, 0FBH
JE PC_XT_WRITE
CMP AL, 0FCH
JE AT_WRITE
CMP AL, 0FAH
JE PS2_30_WRITE
CMP AL, 0FCH
JE PS2_5060_WRITE
CMP AL, 0F8H
```

```

JE PS2_80_WRITE
CMP AL, 0FDH
JE PCJR_WRITE
CMP AL, 0F9H
JE PC_CONVERTIBLE_WRITE

```

```

JMP PC_ANOTHER_WRITE

```

OS:

MOV AH, 30H ;дает номер версии dos al -  
основная версия, ah - номер модификации

```

INT 21H
LEA SI, OS_NAME
ADD SI, 15
CALL BYTE_TO_DEC
ADD SI, 3
MOV AL, AH
CALL BYTE_TO_DEC
MOV DX, OFFSET OS_NAME
CALL PRINT

```

OEM:

```

MOV AL, BH ;bh - серийный номер
LEA SI, OEM_NAME
ADD SI, 12
CALL BYTE_TO_DEC
MOV DX, OFFSET OEM_NAME
CALL PRINT

```

SERIAL:

```

MOV AL, BL
LEA SI, SERIAL_NAME
ADD SI, 15
CALL BYTE_TO_HEX
MOV [SI], AX
ADD SI, 6
MOV DI, SI

MOV AX, CX

```

```
CALL WRD_TO_HEX
MOV  DX, OFFSET SERIAL_NAME
CALL PRINT
```

```
POP      AX
POP  DX
XOR      AL, AL
MOV  AH, 4CH
INT      21H
RET
```

```
MAIN      ENDP
CODE ENDS

END  MAIN
```