

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

**отчет**  
**по лабораторной работе №1**  
**по дисциплине «Операционные системы»**  
**ТЕМА: Исследование структур загрузочных моделей**

Студент гр.0382

Злобин А.С.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

### **Цель работы.**

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

### **Задание.**

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

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

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

4. Запустить FAR и открыть файл загрузочного модуля .COM и файл “плохого” .EXE в шестнадцатеричном виде. Затем открыть файл загрузочного модуля “хорошего” .EXE и сравнить его с предыдущими файлами. Ответить на контрольные вопросы “Отличия форматов файлов COM и EXE модулей”.

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

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

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

### **Выполнение работы:**

1. При написании .COM модуля был использован шаблон из методический указаний, в котором реализованы процедуры преобразования двоичных кодов в символы шестнадцатеричных и десятичных чисел. Для определения типа PC и версии системы были написаны процедуры: PC\_TYPE, OS\_VER.

PC\_TYPE – процедура для вывода типа ПК, получаемого считыванием предпоследнего байта ROM BIOS (по адресу 0F000:0FFFFh). Далее процедура соотносит численное значение типа известным названиям, а если такого нет, то конвертирует число в строку. Наконец, строковое значение типа выводится с помощью процедуры PRINT.

OS\_VER – процедура для вывода версии MS DOS, а также серийных номеров OEM и пользователя. Вся необходимая информация получается при вызове функции 30h прерывания 21h. После вызова функции в AL содержится номер основной версии (если версия раньше 2.0 то AL=0), а в AH – номер модификации. В BH хранится серийный номер OEM, однако по неизвестным причинам это значение напрямую зависит от значения в AL до вызова функции (если AL = 0, то значение серийного номера будет 255 (FF), если 1 – то 16, иначе 0). В BL и CX хранится 3-хбайтовый серийный номер 3 пользователя. Все полученные значения конвертируются в строки и выводятся на экран.

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

```
C:\>lb1_com
IBM type: AT
MS DOS version: 5.0
OEM number:255
User_s number: 000000h
```

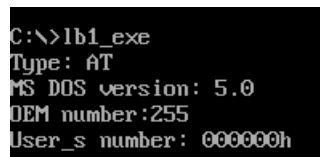
Рисунок 1 – результат запуска модуля lb1\_com.com

```
C:\>lb1_com.exe

          5 0          0 IBM type:          0 IBM typ
e:          255          0 IBM type:
          000000 0 IBM type:
C:\>_
```

Рисунок 2 – результат запуска модуля lb1\_com.exe

2. Для написания “хорошего” .EXE модуля разобьем программу на сегменты кода, данных и стека, также добавим главную процедуру. В .COM модуле имелась директива `org 100h`, что нужно, поскольку при загрузке COM модуля в память DOS первые 256 байт блоком данных занимает PSP, код программы располагается лишь после этого блока. В .EXE модуле же мы в этом не нуждаемся, поскольку блок PSP расположен вне сегмента кода. На рисунке 3 представлены результаты запуска данного модуля.



```
C:\>lb1_exe
Type: AT
MS DOS version: 5.0
OEM number:255
User_s number: 000000h
```

Рисунок 3 – результат запуска модуля lb1\_exe.exe

## Вопросы.

### Отличия исходных текстов COM и EXE программ

1) *Сколько сегментов должна содержать COM-программа?*

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

2) *EXE-программа?*

Обязательно один – сегмент кода. Также можно добавить сегмент данных и стека, они описываются отдельно друг от друга.

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

Обязательно нужна директива `ORG 100h`, поскольку первые 256 байт занимает блок данных PSP. Поэтому нужно, чтобы `CS:IP` имел смещение в 256 байт от нулевого адреса. Так же необходима директива `ASSUME` для указания сегмента кода и данных на один сегмент.

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

Нет, так как в отличие от EXE, адреса сегментных регистров определяются при запуске программы и отсутствует таблица relocation table, соответственно мы не можем использовать команды с указанием сегментов

## Отличия форматов файлов COM и EXE модулей

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

.COM модуль состоит из одного сегмента, в котором содержится и код, и данные. Код располагается с нулевого адреса, но устанавливается смещение 100h на конец PSP

начало сегмента →	00000000: E9 09 02 49 42 4D 20 74 79 70 65 3A 20 20 0D 0A йоIBM type: ь	79 70 65 3A 20 20 0D 0A йоIBM type: ь
	000000010: 24 49 42 4D 20 74 79 70 65 3A 20 50 43 0D 0A 24 \$IBM type: PCь	65 3A 20 50 43 0D 0A 24 \$IBM type: PCь
	000000020: 49 42 4D 20 74 79 70 65 3A 20 50 43 2F 58 54 0D IBM type: PC/XTь	3A 20 50 43 2F 58 54 0D IBM type: PC/XTь
	000000030: 0A 24 49 42 4D 20 74 79 70 65 3A 20 41 54 0D 0A \$IBM type: ATь	70 65 3A 20 41 54 0D 0A \$IBM type: ATь
	000000040: 24 49 42 4D 20 74 79 70 65 3A 20 50 53 32 20 6D \$IBM type: PS2 m	65 3A 20 50 53 32 20 6D \$IBM type: PS2 m
	000000050: 6F 64 65 6C 20 33 30 0D 0A 24 49 42 4D 20 74 79 ode1 30ь\$IBM ty	0A 24 49 42 4D 20 74 79 ode1 30ь\$IBM ty
	000000060: 70 65 3A 20 50 53 32 20 6D 6F 64 65 6C 20 35 30 pe: PS2 model 50	6D 6F 64 65 6C 20 35 30 pe: PS2 model 50
	000000070: 20 6F 72 20 36 30 0D 0A 24 49 42 4D 20 74 79 70 or 60ь\$IBM typ	24 49 42 4D 20 74 79 70 or 60ь\$IBM typ
	000000080: 65 3A 20 50 53 32 20 6D 6F 64 65 6C 20 38 30 0D e: PS2 model 80ь	6F 64 65 6C 20 38 30 0D e: PS2 model 80ь
	000000090: 0A 24 49 42 4D 20 74 79 70 65 3A 20 50 43 6A 72 \$IBM type: PCjr	70 65 3A 20 50 43 6A 72 \$IBM type: PCjr
	0000000A0: 0D 0A 24 49 42 4D 20 74 79 70 65 3A 20 50 43 20 ь\$IBM type: PC	79 70 65 3A 20 50 43 20 ь\$IBM type: PC
	0000000B0: 43 6F 6E 76 65 72 74 69 62 6C 65 0D 0A 24 4D 53 Convertibleь\$MS	62 6C 65 0D 0A 24 4D 53 Convertibleь\$MS
	0000000C0: 20 44 4F 53 20 76 65 72 73 69 6F 6E 3A 20 20 2E DOS version: .	73 69 6F 6E 3A 20 20 2E DOS version: .
	0000000D0: 20 20 0D 0A 24 4F 45 4D 20 6E 75 6D 62 65 72 3A ь\$OEM number:	20 6E 75 6D 62 65 72 3A ь\$OEM number:
	0000000E0: 20 20 20 0D 0A 24 55 73 65 72 5F 73 20 6E 75 6D ь\$User_s num	65 72 5F 73 20 6E 75 6D ь\$User_s num
	0000000F0: 62 65 72 3A 20 20 20 20 20 20 20 68 0D 0A 24 24 ber: ь\$	20 20 20 68 0D 0A 24 24 ber: ь\$
	000000100: 0F 3C 09 76 02 04 07 04 30 C3 51 8A E0 E8 EF FF о<ov♦♦♦0ГQ/ьаипя	30 C3 51 8A E0 E8 EF FF о<ov♦♦♦0ГQ/ьаипя
	000000110: 86 C4 B1 04 D2 E8 E8 E6 FF 59 C3 53 8A FC E8 E9 †Д±♦ТиижяYGSльбий	FF 59 C3 53 8A FC E8 E9 †Д±♦ТиижяYGSльбий
	000000120: FF 88 25 4F 88 05 4F 8A C7 E8 DE FF 88 25 4F 88 я€%0€♦0/ь3и0я€%0€	C7 E8 DE FF 88 25 4F 88 я€%0€♦0/ь3и0я€%0€
	000000130: 05 5B C3 51 52 32 E4 33 D2 B9 0A 00 F7 F1 80 CA †[ГQR2д3Tи€ чсбК	D2 B9 0A 00 F7 F1 80 CA †[ГQR2д3Tи€ чсбК
	000000140: 30 88 14 4E 33 D2 3D 0A 00 73 F1 3C 00 74 04 0C 0€JN3T=ь sc< т♦	00 73 F1 3C 00 74 04 0C 0€JN3T=ь sc< т♦
	000000150: 30 88 04 5A 59 C3 50 B4 09 CD 21 58 C3 53 52 06 0€♦ZYГProH!XГSR♦	09 CD 21 58 C3 53 52 06 0€♦ZYГProH!XГSR♦
	000000160: BB 00 F0 8E C3 26 A0 FE FF 3C FF 74 2A 3C FE 74 » рбГ& юя<ят*<ют	FF 3C FF 74 2A 3C FE 74 » рбГ& юя<ят*<ют
	000000170: 2C 3C FB 74 28 3C FC 74 2A 3C FA 74 2C 3C F8 74 ,<ыт(<ьт*<ьт,<шт	2A 3C FA 74 2C 3C F8 74 ,<ыт(<ьт*<ьт,<шт
	000000180: 2E 3C FD 74 30 3C F9 74 32 BF 0D 01 E8 7B FF 89 .<эт0<шт2iJи{я%	32 BF 0D 01 E8 7B FF 89 .<эт0<шт2iJи{я%
	000000190: 05 BA 03 01 EB 28 90 BA 11 01 EB 22 90 BA 20 01 †еVол(ђе<ол"ђе 0	11 01 EB 22 90 BA 20 01 †еVол(ђе<ол"ђе 0
	0000001A0: EB 1C 90 BA 32 01 EB 16 90 BA 41 01 EB 10 90 BA лLђе2ол=ђеAол>ђе	90 BA 41 01 EB 10 90 BA лLђе2ол=ђеAол>ђе
	0000001B0: 79 01 EB 0A 90 BA 92 01 EB 04 90 BA A3 01 E8 95 у0лџђе'0л>ђеJи♦	EB 04 90 BA A3 01 E8 95 у0лџђе'0л>ђеJи♦
	0000001C0: FF 07 5A 58 C3 50 53 51 33 C0 B4 30 CD 21 BE CE я•ZXГPSQ3Ar0H!s0	33 C0 B4 30 CD 21 BE CE я•ZXГPSQ3Ar0H!s0
	0000001D0: 01 E8 5F FF 8A C4 83 C6 03 E8 57 FF BA BE 01 E8 0и_ялДfЖиWяес0и	03 E8 57 FF BA BE 01 E8 0и_ялДfЖиWяес0и
	0000001E0: 74 FF BE E2 01 8A C7 E8 49 FF BA D5 01 E8 66 FF тясв0/ь3иIяеX0ифя	49 FF BA D5 01 E8 66 FF тясв0/ь3иIяеX0ифя
	0000001F0: BF FA 01 8B C1 E8 23 FF 8A C3 E8 0D FF BF F5 01 ьь0<Би#ялГиJяix0	8A C3 E8 0D FF BF F5 01 ьь0<Би#ялГиJяix0
	000000200: 89 05 BA E6 01 E8 4E FF 59 5B 58 C3 B8 00 F0 8E %†еж0иNяY[XГё рћ	59 5B 58 C3 B8 00 F0 8E %†еж0иNяY[XГё рћ
	000000210: C0 26 A0 FF FF E8 45 FF E8 AA FF 32 C0 B4 4C CD A& яяиEяиЄя2ArLH	E8 AA FF 32 C0 B4 4C CD A& яяиEяиЄя2ArLH
конец сегмента →	000000220: 21 !	!

Рисунок 4 – .COM в 16-м виде

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

Состоит из одного сегмента. Код, стек и данные располагается с адреса 300h (200h занимает заголовок и relocation table, 100h - смещение).

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 09 02 49 42 4D 20 74	79 70 65 3A 20 20 0D 0A	й00IBM type: 00
000000310:	24 49 42 4D 20 74 79 70	65 3A 20 50 43 0D 0A 24	\$IBM type: PC00\$
000000320:	49 42 4D 20 74 79 70 65	3A 20 50 43 2F 58 54 0D	IBM type: PC/XT0
000000330:	0A 24 49 42 4D 20 74 79	70 65 3A 20 41 54 0D 0A	00\$IBM type: AT00
000000340:	24 49 42 4D 20 74 79 70	65 3A 20 50 53 32 20 6D	\$IBM type: PS2 m
000000350:	6F 64 65 6C 20 33 30 0D	0A 24 49 42 4D 20 74 79	odel 3000\$IBM ty
000000360:	70 65 3A 20 50 53 32 20	6D 6F 64 65 6C 20 35 30	pe: PS2 model 50
000000370:	20 6F 72 20 36 30 0D 0A	24 49 42 4D 20 74 79 70	or 6000\$IBM type:
000000380:	65 3A 20 50 53 32 20 6D	6F 64 65 6C 20 38 30 0D	e: PS2 model 800
000000390:	0A 24 49 42 4D 20 74 79	70 65 3A 20 50 43 6A 72	00\$IBM type: PCjr
0000003A0:	0D 0A 24 49 42 4D 20 74	79 70 65 3A 20 50 43 20	00\$IBM type: PC
0000003B0:	43 6F 6E 76 65 72 74 69	62 6C 65 0D 0A 24 4D 53	Convertible00\$MS
0000003C0:	20 44 4F 53 20 76 65 72	73 69 6F 6E 3A 20 20 2E	DOS version: .
0000003D0:	20 20 0D 0A 24 4F 45 4D	20 6E 75 6D 62 65 72 3A	00\$OEM number:
0000003E0:	20 20 20 0D 0A 24 55 73	65 72 5F 73 20 6E 75 6D	00\$User_s num
0000003F0:	62 65 72 3A 20 20 20 20	20 20 20 68 0D 0A 24 24	ber: h00\$
000000400:	0F 3C 09 76 02 04 07 04	30 C3 51 8A E0 E8 EF FF	0<ov000000Q0лаппя
000000410:	86 C4 B1 04 D2 E8 E8 E6	FF 59 C3 53 8A FC E8 E9	тДт0ТиижяYGS/ьий
000000420:	FF 88 25 4F 88 05 4F 8A	C7 E8 DE FF 88 25 4F 88	я€%0€+0/ь3и0я€%0€
000000430:	05 5B C3 51 52 32 E4 33	D2 B9 0A 00 F7 F1 80 CA	+[[ГQR2д3ТН0 чсбK
000000440:	30 88 14 4E 33 D2 3D 0A	00 73 F1 3C 00 74 04 0C	0€JN3T= sc< t0
000000450:	30 88 04 5A 59 C3 50 B4	09 CD 21 58 C3 53 52 06	0€0ZYГProH!XGSR0
000000460:	BB 00 F0 8E C3 26 A0 FE	FF 3C FF 74 2A 3C FE 74	» ртГ& юя<ят*<yt
000000470:	2C 3C FB 74 28 3C FC 74	2A 3C FA 74 2C 3C F8 74	,<yt(<ьt*<ьt,<шт
000000480:	2E 3C FD 74 30 3C F9 74	32 BF 0D 01 E8 7B FF 89	.<эт0<шт2iJ0и{я%
000000490:	05 BA 03 01 EB 28 90 BA	11 01 EB 22 90 BA 20 01	+€▼0л(ђе<0л"ђе 0
0000004A0:	EB 1C 90 BA 32 01 EB 16	90 BA 41 01 EB 10 90 BA	лJђе20л=ђеA0л>ђе
0000004B0:	79 01 EB 0A 90 BA 92 01	EB 04 90 BA A3 01 E8 95	у0лJђе'0лJђеJ0и•
0000004C0:	FF 07 5A 58 C3 50 53 51	33 C0 B4 30 CD 21 BE CE	я•ZXГPSQ3Ar0H!s0
0000004D0:	01 E8 5F FF 8A C4 83 C6	03 E8 57 FF BA BE 01 E8	0и_яьДГЖ▼иWяес0и
0000004E0:	74 FF BE E2 01 8A C7 E8	49 FF BA D5 01 E8 66 FF	тясв0/ь3иIяеX0ифя
0000004F0:	BF FA 01 8B C1 E8 23 FF	8A C3 E8 0D FF BF F5 01	йь0<Би#я/ьГиJяix0
000000500:	89 05 BA E6 01 E8 4E FF	59 5B 58 C3 B8 00 F0 8E	%0еж0иNяY[XГё рт
000000510:	C0 26 A0 FF FF E8 45 FF	E8 AA FF 32 C0 B4 4C CD	A& яяиЕяиЕя2ArLH
000000520:	21	!	

Рисунок 5 – “плохой” .EXE в 16-м виде

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

В начале модуля располагается заголовок и relocation table (200h байт), затем сегменты в порядке их определения в коде, т.е. сначала сегмент стека, потом сегмент данных, а затем – сегмент кода.

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:	49 42 4D 20 74 79 70 65	3A 20 20 0D 0A 24 54 79	IBM type: 00\$Type:
0000000250:	70 65 3A 20 50 43 0D 0A	24 54 79 70 65 3A 20 50	pe: PC00\$Type: P
0000000260:	43 2F 58 54 0D 0A 24 54	79 70 65 3A 20 41 54 0D	C/XT00\$Type: AT0
0000000270:	0A 24 54 79 70 65 3A 20	50 53 32 20 6D 6F 64 65	00\$Type: PS2 mode
0000000280:	6C 20 33 30 0D 0A 24 54	79 70 65 3A 20 50 53 32	l 3000\$Type: PS2
0000000290:	20 6D 6F 64 65 6C 20 35	30 20 6F 72 20 36 30 0D	model 50 or 600
00000002A0:	0A 24 54 79 70 65 3A 20	50 53 32 20 6D 6F 64 65	00\$Type: PS2 mode
00000002B0:	6C 20 38 30 0D 0A 24 54	79 70 65 3A 20 50 43 6A	l 8000\$Type: PCj
00000002C0:	72 0D 0A 24 54 79 70 65	3A 20 50 43 20 43 6F 6E	r00\$Type: PC Con
00000002D0:	76 65 72 74 69 62 6C 65	0D 0A 24 4D 53 20 44 4F	vertible00\$MS DO
00000002E0:	53 20 76 65 72 73 69 6F	6E 3A 20 20 2E 20 00 0D	S version: . 0
00000002F0:	0A 24 4F 45 4D 20 6E 75	6D 62 65 72 3A 20 20 20	00\$OEM number:
0000000300:	0D 0A 24 55 73 65 72 5F	73 20 6E 75 6D 62 65 72	00\$User_s number
0000000310:	3A 20 20 20 20 20 20 20	68 0D 0A 24 00 00 00 00	: h00\$
0000000320:	24 0F 3C 09 76 02 04 07	04 30 C3 51 8A E0 E8 EF	\$0<ov00000Q00\$
0000000330:	FF 86 C4 B1 04 D2 E8 E8	E6 FF 59 C3 53 8A FC E8	ятД±0ТиижяYGSльи
0000000340:	E9 FF 88 25 4F 88 05 4F	8A C7 E8 DE FF 88 25 4F	ия€%O€0003и0я€%0
0000000350:	88 05 5B C3 51 52 32 E4	33 D2 B9 0A 00 F7 F1 80	€0[ГQR2д3Т00 чс0
0000000360:	CA 30 88 14 4E 33 D2 3D	0A 00 73 F1 3C 00 74 04	K0€0N3T=0 sc< t0
0000000370:	0C 30 88 04 5A 59 C3 50	B4 09 CD 21 58 C3 50 52	90€0ZYГProH!XГPR
0000000380:	06 B8 00 F0 8E C0 26 A0	FE FF 3C FF 74 2A 3C FE	0€ рhA& юя<ят*<ю
0000000390:	74 2C 3C FB 74 28 3C FC	74 2A 3C FA 74 2C 3C F8	t,<ыt(<ыt*<ыt,<ш
00000003A0:	74 2E 3C FD 74 30 3C F9	74 32 BF 0A 00 E8 7B FF	t.<эт0<шт2и ифя
00000003B0:	89 05 BA 00 00 EB 28 90	BA 0E 00 EB 22 90 BA 19	0€ л(йед л"йед
00000003C0:	00 EB 1C 90 BA 27 00 EB	16 90 BA 32 00 EB 10 90	л_йе' л_йе2 л_й
00000003D0:	BA 62 00 EB 0A 90 BA 77	00 EB 04 90 BA 84 00 E8	eb лйеew лйе,, и
00000003E0:	95 FF 07 5A 58 C3 50 53	51 33 C0 B4 30 CD 21 BE	•я•ZXГPSQ3Ar0H!s
00000003F0:	AB 00 E8 5F 8A C4 83	C6 03 E8 57 FF BA 9B 00	« и_ялбдГЖиИяе>
0000000400:	E8 74 FF BE BF 00 8A C7	E8 49 FF BA B2 00 E8 66	итяsi л3иIаеI иф
0000000410:	FF BF D7 00 8B C1 E8 23	FF 8A C3 E8 0D FF BF D2	яiЧ <Би#ялбГи_яiТ
0000000420:	00 89 05 BA C3 00 E8 4E	FF 59 5B 58 C3 B8 04 00	0€еГ ил_яY[ХГё0
0000000430:	8E D8 E8 49 FF E8 AE FF	32 C0 B4 4C CD 21	ишиIяи_я2ArLH!

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

Загрузка COM модуля в основную память”.

Результат загрузки программы в отладчик представлены на рисунке 7.

[CPU 80486]		[1-1111]	
cs:0100 E90902	jmp 030C ↓	ax 0000	c=0
cs:0103 49	dec cx	bx 0000	z=0
cs:0104 42	inc dx	cx 0000	s=0
cs:0105 4D	dec bp	dx 0000	o=0
cs:0106 207479	and [si+791,dh	si 0000	p=0
cs:0109 7065	jo 0170	di 0000	a=0
cs:010B 3A20	cmp ah,[bx+sil	bp 0000	i=1
cs:010D 200D	and [dil,cl	sp FFFE	d=0
cs:010F 0A24	or ah,[sil	ds 48DD	
cs:0111 49	dec cx	es 48DD	
cs:0112 42	inc dx	ss 48DD	
cs:0113 4D	dec bp	cs 48DD	
cs:0114 207479	and [si+791,dh	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 20 15 00			
ds:0010 C9 15 80 02 24 10 92 01 15 00 00			
ds:0018 01 01 01 00 02 FF FF FF 00 00			
		ss:0000 20CD	
		ss:FFFE 0000	

Рисунок 7 – .COM модуль

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





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=490D) – на начало сегмента команд. В IP загружается смещение точки входа в программу, которая берётся из метки после директивы END. Причём дополнительный программный сегмент (PSP) присутствует в каждом EXE-файле.

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

Эти регистры указывают на начало PSP.

3) *Как определяется стек?*

Стек определяется с помощью директивы .stack, после которой задаётся размер стека. При исполнении регистр SS указывает на начало сегмента стека, а SP на конца стека(его смещение)

4) *Как определяется точка входа?*

Точка входа определяется директивой END. Данная метка определяет адрес, с которого начинается выполнение программы, т.е. – точка входа в программу.

Исходный код программы см. в приложении А.

## **Выводы.**

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

# ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab1\_com.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN
; Данные
TYPE_PC db 'Type: PC',0DH,0AH,'$'
TYPE_PC_XT db 'Type: PC/XT',0DH,0AH,'$'
TYPE_AT db 'Type: AT',0DH,0AH,'$'
TYPE_PS2_M30 db 'Type: PS2 модель 30',0DH,0AH,'$'
TYPE_PS2_M50_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'$'
TYPE_PS2_M80 db 'Type: PS2 модель 80',0DH,0AH,'$'
TYPE_PC_JR db 'Type: PCjr',0DH,0AH,'$'
TYPE_PC_CONV db 'Type: PC Convertible',0DH,0AH,'$'

VERSIONS db 'Version MS-DOS:  . ',0DH,0AH,'$'
SERIAL_NUMBER db 'Serial number OEM: ',0DH,0AH,'$'
USER_NUMBER db 'User serial number:      H $'

; Процедуры
;-----
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
    ret
BYTE_TO_HEX ENDP
;-----
WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти 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
    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 10с/с, 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_l
    or AL,30h
    mov [SI],AL
end_l:
    pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP
;-----
WRITESTRING PROC near
    mov AH,09h
    int 21h
    ret
WRITESTRING ENDP

PC_TYPE PROC near
    mov ax, 0f000h ; получаем номер модели
    mov es, ax
    mov al, es:[0ffffh]

    cmp al, 0ffh ; начинаем сравнивать
    je pc

```

```

    cmp al, 0feh
    je pc_xt
    cmp al, 0fbh
    je pc_xt
    cmp al, 0fch
    je pc_at
    cmp al, 0fah
    je pc_ps2_m30
    cmp al, 0f8h
    je pc_ps2_m80
    cmp al, 0fdh
    je pc_jr
    cmp al, 0f9h
    je pc_conv
pc:
    mov dx, offset TYPE_PC
    jmp writetype
pc_xt:
    mov dx, offset TYPE_PC_XT
    jmp writetype
pc_at:
    mov dx, offset TYPE_AT
    jmp writetype
pc_ps2_m30:
    mov dx, offset TYPE_PS2_M30
    jmp writetype
pc_ps2_m50_60:
    mov dx, offset TYPE_PS2_M50_60
    jmp writetype
pc_ps2_m80:
    mov dx, offset TYPE_PS2_M80
    jmp writetype
pc_jr:
    mov dx, offset TYPE_PC_JR
    jmp writetype
pc_conv:
    mov dx, offset TYPE_PC_CONV
    jmp writetype
writetype:
    call WRITESTRING
    ret
PC_TYPE ENDP

OS_VER PROC near
    mov ah, 30h
    int 21h
    push ax

    mov si, offset VERSIONS
    add si, 16
    call BYTE_TO_DEC
    pop ax
    mov al, ah

```

```

    add si, 3
    call BYTE_TO_DEC
    mov dx, offset VERSIONS
    call WRITESTRING

    mov si, offset SERIAL_NUMBER
    add si, 19
    mov al, bh
    call BYTE_TO_DEC
    mov dx, offset SERIAL_NUMBER
    call WRITESTRING

    mov di, offset USER_NUMBER
    add di, 25
    mov ax, cx
    call WRD_TO_HEX
    mov al, bl
    call BYTE_TO_HEX
    sub di, 2
    mov [di], ax
    mov dx, offset USER_NUMBER
    call WRITESTRING
    ret
OS_VER ENDP

```

```

; Код
BEGIN:
    call PC_TYPE
    call OS_VER

    xor AL,AL
    mov AH,4Ch
    int 21H
TESTPC ENDS
END START

```

**Название файла: lab1\_exe.asm**

```

AStack    SEGMENT    STACK
           DW 128 DUP(?)
AStack    ENDS

```

```

DATA      SEGMENT
    TYPE_PC db  'Type: PC',0DH,0AH,'$'
    TYPE_PC_XT db 'Type: PC/XT',0DH,0AH,'$'
    TYPE_AT db  'Type: AT',0DH,0AH,'$'
    TYPE_PS2_M30 db 'Type: PS2 модель 30',0DH,0AH,'$'
    TYPE_PS2_M50_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'$'
    TYPE_PS2_M80 db 'Type: PS2 модель 80',0DH,0AH,'$'
    TYPE_PC_JR db 'Type: PCjr',0DH,0AH,'$'
    TYPE_PC_CONV db 'Type: PC Convertible',0DH,0AH,'$'

    VERSIONS db 'Version MS-DOS:  . ',0DH,0AH,'$'

```

```

        SERIAL_NUMBER db  'Serial number OEM:  ',0DH,0AH,'$'
        USER_NUMBER db  'User serial number:      H $'
DATA ENDS

```

```

CODE SEGMENT

```

```

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

```

```

    ; Процедуры

```

```

;-----

```

```

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

```

```

    ret

```

```

BYTE_TO_HEX ENDP

```

```

;-----

```

```

WRD_TO_HEX PROC near

```

```

;перевод в 16 с/с 16-ти 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

```

```

    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 10с/с, 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_l
    or AL,30h
    mov [SI],AL
end_l:
    pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP
;-----
WRITESTRING PROC near
    mov AH,09h
    int 21h
    ret
WRITESTRING ENDP

PC_TYPE PROC near
    mov ax, 0f000h ; получаем номер модели
    mov es, ax
    mov al, es:[0ffffh]

    cmp al, 0ffh ; начинаем сравнивать
    je pc
    cmp al, 0feh
    je pc_xt
    cmp al, 0fbh
    je pc_xt
    cmp al, 0fch
    je pc_at
    cmp al, 0fah
    je pc_ps2_m30
    cmp al, 0f8h
    je pc_ps2_m80
    cmp al, 0fdh
    je pc_jr
    cmp al, 0f9h
    je pc_conv
pc:
    mov dx, offset TYPE_PC
    jmp writetype
pc_xt:

```

```

        mov dx, offset TYPE_PC_XT
        jmp writetype
pc_at:
        mov dx, offset TYPE_AT
        jmp writetype
pc_ps2_m30:
        mov dx, offset TYPE_PS2_M30
        jmp writetype
pc_ps2_m50_60:
        mov dx, offset TYPE_PS2_M50_60
        jmp writetype
pc_ps2_m80:
        mov dx, offset TYPE_PS2_M80
        jmp writetype
pc_jr:
        mov dx, offset TYPE_PC_JR
        jmp writetype
pc_conv:
        mov dx, offset TYPE_PC_CONV
        jmp writetype
writetype:
        call WRITESTRING
        ret
PC_TYPE ENDP

```

```

OS_VER PROC near

```

```

        mov ah, 30h
        int 21h
        push ax

```

```

        mov si, offset VERSIONS
        add si, 16
        call BYTE_TO_DEC
        pop ax
        mov al, ah
        add si, 3
        call BYTE_TO_DEC
        mov dx, offset VERSIONS
        call WRITESTRING

```

```

        mov si, offset SERIAL_NUMBER
        add si, 19
        mov al, bh
        call BYTE_TO_DEC
        mov dx, offset SERIAL_NUMBER
        call WRITESTRING

```

```

        mov di, offset USER_NUMBER
        add di, 25
        mov ax, cx
        call WRD_TO_HEX
        mov al, bl

```



```
    call BYTE_TO_HEX
    sub di, 2
    mov [di], ax
    mov dx, offset USER_NUMBER
    call WRITESTRING
    ret
OS_VER ENDP
```

```
Main PROC FAR
    sub     AX,AX
    push    AX
    mov     AX,DATA
    mov     DS,AX
    call    PC_TYPE
    call    OS_VER
    xor     AL,AL
    mov     AH,4Ch
    int     21H
    ;ret
Main ENDP
CODE ENDS
        END Main
```