

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

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

Студент гр. 9382

\_\_\_\_\_

Субботин М.О.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2021

### **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.**

«Истина познается в сравнении», как говорили древние. К счастью, у нас есть возможность исследовать в одной системе два различных формата загрузочных модулей, сравнить их и лучше понять как система программирования и управляющая программа обращаются с ними. Система программирования включает компилятор с языка ассемблер (часто называется, просто, 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 на выполнение. Все эти компоненты согласованно работают для изготовления и выполнения загрузочных модулей разного типа. Для выполнения лабораторной работы сначала нужно изготовить загрузочные модули. 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 и серийным номером пользователя. Полученные строки выводятся на экран.

Были созданы методы TYPE\_IBM\_PC и OS\_VERSION. Первый метод выводит на экран информацию о типе ПК, вторая о системе. В зависимости от предпоследнего байта ROM BIOS, первый метод выводит соответствующий байту тип модели. На рисунке ниже представлены соответствия тип – байт.

PC	FF
PC/XT	FE, FB
AT	FC
PS2 модель 30	FA
PS2 модель 50 или 60	FC
PS2 модель 80	F8
PCjr	FD
PC Convertible	F9

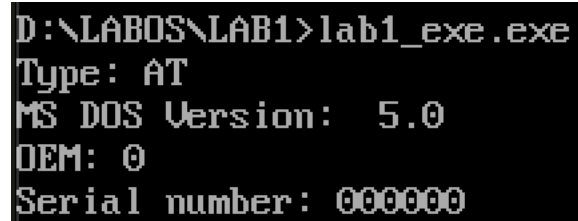
Метод OS\_VERSION выводит следующую информацию: номер основной версии, номер модификации в десятичной системе счисления, серийный номер OEM, серийный номер пользователя.

```
D:\LABOS\LAB1>lab1.com
Type: AT
MS DOS Version: 5.0
OEM: 0
Serial number: 000000
```

Diagram illustrating the layout of a 64-bit floating-point number (IEEE 754 standard):

- Sign: 1 bit (0), labeled  $\theta$  [Type: PC]
- Exponent: 11 bits (5 0), labeled  $\theta$  [Type: PC]
- Mantissa: 52 bits (0000000), labeled  $\theta$  [Type: PC]

3



```
D:\LABOS\LAB1>lab1_exe.exe
Type: AT
MS DOS Version: 5.0
OEM: 0
Serial number: 000000
```

Рисунок 4. Результат выполнения “хорошего” .EXE файла.

### **Выводы.**

Были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

## **ПРИЛОЖЕНИЕ А**

### **ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ**

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

1. Сколько сегментов должна содержать COM-программа?

COM-программа должна содержать один сегмент, в этом одном сегменте хранятся код и данные, стек же генерируется автоматически.

2. EXE-программа?

EXE-программа может содержать  $\geq 1$  сегментов. EXE-программа может содержать произвольный объем основной памяти, поэтому кол-во сегментов ограничено размером основной памяти.

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

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING и

ORG 100H. Первая директива указывает на то, что сегменты данных и кода находятся в одном сегменте. Вторая директива нужна для смещения адресов на 256 байт, т.к. первые 256 байт указывают на PSP.

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

Нельзя использовать команды типа: "MOV <register>, SEG <segment name>", т.к. в COM-программе нет таблицы настроек.

#### **Отличия форматов файлов COM и EXE модулей**

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

COM-программа состоит из одного сегмента (сегмент кода и сегмент данных), сегмент стека получается автоматически. COM-программа не превышает 64 Кб.

Как будет видно, код начинается с адреса 0h, но из-за директивы org 100h происходит смещение на 100h.

00000000	E9 B4 01 54 79 70 65 3A 20 50 43 0D 0A 24 54 79	01.Type: PC..\$Type
00000010	70 65 3A 20 54 50 43 2F 58 54 0D 0A 24 54 79 70 65	pe: PC/XT..\$Type
00000020	3A 20 41 54 0D 0A 24 54 79 70 65 3A 20 50 53 32	: AT..\$Type: PS2
00000030	20 6D 6F 64 65 6C 20 33 30 0D 0A 24 54 79 70 65	model 30..\$Type
00000040	3A 20 50 53 32 20 6D 6F 64 65 6C 20 38 30 0D 0A	: PS2 model 80..
00000050	24 54 79 70 65 3A 20 50 53 6A 72 0D 0A 24 54 79	\$Type: PSjr..\$Ty
00000060	70 65 3A 20 50 43 20 43 6F 6E 76 65 72 74 69 62	pe: PC Convertib
00000070	6C 65 0D 0A 24 4D 53 20 44 4F 53 20 56 65 72 73	le..\$MS DOS Vers
00000080	69 6F 6E 3A 20 20 20 2E 20 20 0D 0A 24 4F 45 4D	ion: . ..\$OEM
00000090	3A 20 20 20 0D 0A 24 53 65 72 69 61 6C 20 6E 75	: ..\$Serial nu
000000A0	6D 62 65 72 3A 20 20 20 20 20 20 20 20 20 20 20	mber:
000000B0	20 20 20 20 20 20 20 20 20 20 20 20 24 24 0F	\$\$.
000000C0	3C 09 76 02 04 07 04 30 C3 51 8A E0 E8 EF FF 86	<.v....0 QèαΦη à
000000D0	C4 B1 04 D2 E8 E8 E6 FF 59 C3 53 8A FC E8 E9 FF	—∕.πΦΦμ Υ Sè^nΦθ
000000E0	88 25 4F 88 05 4F 8A C7 E8 DE FF 88 25 4F 88 05	ê%0ê.0è  Φ ■ ê%0ê.
000000F0	5B C3 51 52 32 E4 33 D2 B9 0A 00 F7 F1 80 CA 30	[ QR2Σ3T  ..≈±çL=0
00000100	88 14 4E 33 D2 3D 0A 00 73 F1 3C 00 74 04 0C 30	ê.N3T=.s±<.t..0
00000110	88 04 5A 59 C3 B8 00 F0 8E C0 26 A0 FE FF 3C FF	ê.ZY ¶.≡Ä L&á. <
00000120	74 1C 3C FE 74 1E 3C FB 74 1A 3C FC 74 1C 3C FA	t.<°t.<√t.<^nt.<·
00000130	74 1E 3C F8 74 20 3C FD 74 22 3C F9 74 24 BA 03	t.<°t <²t"<°t\$  .
00000140	01 EB 22 90 BA 0E 01 EB 1C 90 BA 1C 01 EB 16 90	.δ"É  ..δ.É  ..δ.É
00000150	BA 27 01 EB 10 90 BA 3C 01 EB 0A 90 BA 51 01 EB	'.δ.É  <.δ.É  Q.δ
00000160	04 90 BA 5E 01 B4 09 CD 21 C3 B4 30 CD 21 50 BE	.É  ^.& .!=! H =P
00000170	75 01 83 C6 11 E8 7A FF 58 8A C4 83 C6 03 E8 71	u.â _.φz Xè-â _.φq
00000180	FF BA 75 01 B4 09 CD 21 BE 8D 01 83 C6 05 8A C7	u.& .!=! î.â _.è
00000190	E8 5F FF BA 8D 01 B4 09 CD 21 BF 97 01 83 C7 14	Φ_   î.& .!=! γù.â _.
000001A0	8B C1 E8 35 FF 83 EF 02 8A C3 E8 1C FF 89 05 BA	ï _Φ5 ân.è Φ. ë.
000001B0	97 01 B4 09 CD 21 C3 E8 5B FF E8 AD FF 32 C0 B4	ù.& .!=! _Φ[ Φ; 2
000001C0	4C CD 21 +	L=!

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

В “плохом” EXE-файле сегменты данных и кода расположены в одном сегменте, что не очень хорошо, т.к. в EXE-файле сегменты кода и данных должны быть разделены. Код начинается с адреса 300h(PSP -100h, заголовок– 100h, org 100h), с адреса 0h располагается заголовок, а конкретно сигнатура MZ формата 2 байта (4D 5A).

```

00000000 4D 5A C3 00 03 00 00 00 20 00 00 00 FF FF 00 00 MZ|..... ..
00000010 00 00 CB 6F 00 01 00 00 1E 00 00 00 01 00 00 00 ..т.....
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000210 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000220 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000230 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000240 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000250 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000260 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000270 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000280 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000290 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000002A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000002B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000002C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000002D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000002E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000002F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000300 E9 B4 01 54 79 70 65 3A 20 50 43 0D 0A 24 54 79 0- .Type: PC..$Ty
00000310 70 65 3A 20 50 43 2F 58 54 0D 0A 24 54 79 70 65 pe: PC/XT..$Type
00000320 3A 20 41 54 0D 0A 24 54 79 70 65 3A 20 50 53 32 : AT..$Type: PS2
00000330 20 6D 6F 64 65 6C 20 33 30 0D 0A 24 54 79 70 65 model 30..$Type
00000340 3A 20 50 53 32 20 6D 6F 64 65 6C 20 38 30 0D 0A : PS2 model 80..
00000350 24 54 79 70 65 3A 20 50 53 6A 72 0D 0A 24 54 79 $Type: PSjr..$Ty
00000360 70 65 3A 20 50 43 20 43 6F 6E 76 65 72 74 69 62 pe: PC Convertib
00000370 6C 65 0D 0A 24 4D 53 20 44 4F 53 20 56 65 72 73 le..$MS DOS Vers
00000380 69 6F 6E 3A 20 20 20 2E 20 20 0D 0A 24 4F 45 4D ion: . .$.OEM
00000390 3A 20 20 20 0D 0A 24 53 65 72 69 61 6C 20 6E 75 : ..$Serial nu
000003A0 6D 62 65 72 3A 20 20 20 20 20 20 20 20 20 20 mber:
000003B0 20 20 20 20 20 20 20 20 20 20 20 20 24 24 0F $.
000003C0 3C 09 76 02 04 07 04 30 C3 51 8A E0 E8 EF FF 86 <.v....0|Qeæñ ä
000003D0 C4 B1 04 D2 E8 E8 E6 FF 59 C3 53 8A FC E8 E9 FF -тфм. Y|Sè"фö
000003E0 88 25 4F 88 05 4F 8A C7 E8 DE FF 88 25 4F 88 05 è%0è..0è|ò| è%0è.
000003F0 5B C3 51 52 32 E4 33 D2 B9 0A 00 F7 F1 80 CA 30 [ |QR2B3-|...≈±ÇA0
00000400 88 14 4E 33 D2 3D 0A 00 73 F1 3C 00 74 04 0C 30 è.N3т...s±<.t..0
00000410 88 04 5A 59 C3 B8 00 F0 8E C0 26 A0 FE FF 3C FF è.ZY|т...=A&ä. <
00000420 74 1C 3C FE 74 1E 3C FB 74 1A 3C FC 74 1C 3C FA t.<~t.<√t.<~t.<
00000430 71 1E 3C F8 74 20 3C FD 74 22 3C F9 74 24 BA 03 t.<~t <'t"<~t$||.
00000440 01 EB 22 90 BA 0E 01 EB 1C 90 BA 1C 01 EB 16 90 .ò"E||...ò.è||...ò.è
00000450 BA 27 01 EB 10 90 BA 3C 01 EB 0A 90 BA 51 01 EB ||'.ò.è||<.ò.è||Q.ò
00000460 04 90 BA 5E 01 B4 09 CD 21 C3 B4 30 CD 21 50 BE .è||^..|.=!|H0=|P|
00000470 75 01 83 C6 11 E8 7A FF 58 8A C4 83 C6 03 E8 71 u.â|..oz Xè=â|..oq
00000480 FF BA 75 01 B4 09 CD 21 BE 8D 01 83 C6 05 8A C7 ||u..|.=!|i.â|..e|
00000490 E8 5F FF BA 8D 01 B4 09 CD 21 BF 97 01 83 C7 14 0_||i..|.=!|i.â|..
000004A0 8B C1 E8 35 FF 83 EF 02 8A C3 E8 1C FF 89 05 BA iLq5 âñ.è|..ò. è..|
000004B0 97 01 B4 09 CD 21 C3 E8 5B FF E8 AD FF 32 C0 B4 ù..|.=!|ò| 2L|

```

### 3. Какова структура файла “хорошего” EXE? Чем он отличается от файла “плохого” EXE?

В “хорошем” EXE-файле сегменты кода, данных и стека должны быть отдельными. EXE-файл имеет заголовок, в нем находится сигнатура и данные, необходимые для загрузки EXE-файла, а также там находится таблица настройки адресов. EXE-файл может иметь любой размер.

В “плохом” EXE адресация начинается с 300h. В “хорошем” EXE 400h, т.к. теперь нет смещения в 100h (org 100h), а вместо этого отведено место под стек равное 200h.

```

00000000 4D 5A CD 01 03 00 01 00 20 00 00 00 FF 00 00 MZ=.....
00000010 00 02 62 E3 F9 00 2C 00 1E 00 00 00 01 00 FD 00 ..bmr.....
00000020 2C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000210 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000220 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000230 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000240 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000250 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000260 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000270 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000280 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000290 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000002A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000002B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000002C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000002D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000002E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000002F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000300 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000310 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000320 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000330 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000340 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000350 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000360 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000370 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000380 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000390 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000003A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000003B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000003C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000003D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000003E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000003F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000400 54 79 70 65 3A 20 50 43 0D 0A 24 54 79 70 65 3A Type: PC..$Type:
00000410 20 50 43 2F 58 54 0D 0A 24 54 79 70 65 3A 20 41 PC/XT..$Type: A
00000420 54 0D 0A 24 54 79 70 65 3A 20 50 53 32 20 6D 6F T..$Type: PS2 mo
00000430 64 65 6C 20 33 30 0D 0A 24 54 79 70 65 3A 20 50 del 30..$Type: P
00000440 53 32 20 6D 6F 64 65 6C 20 38 30 0D 0A 24 54 79 S2 model 80..$Ty
00000450 70 65 3A 20 50 53 6A 72 0D 0A 24 54 79 70 65 3A pe: PSjr..$Type:
00000460 20 50 43 20 43 6F 6E 76 65 72 74 69 62 6C 65 0D PC Convertible.
00000470 0A 24 4D 53 20 44 4F 53 20 56 65 72 73 69 6F 6E .SMS DOS Version
00000480 3A 20 20 20 2E 20 20 0D 0A 24 4F 45 4D 3A 20 20 : ..$OEM:
00000490 20 0D 0A 24 53 65 72 69 61 6C 20 6E 75 6D 62 65 ..$Serial numbe
000004A0 72 3A 20 20 20 20 20 20 20 20 20 20 20 20 20 20 r:
000004B0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 $.....
000004C0

```

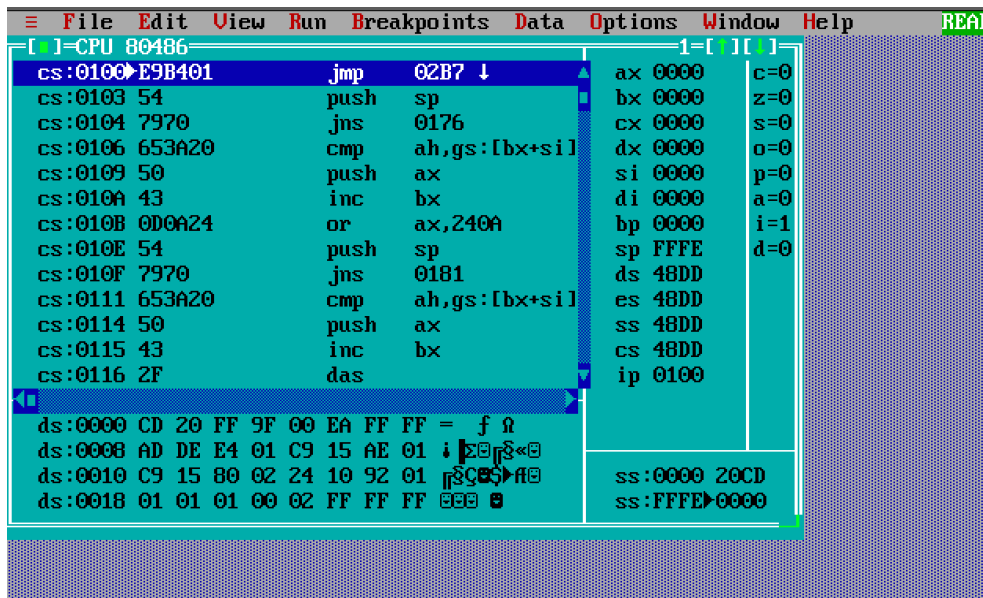
## Загрузка COM модуля в основную память

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

Сначала определяется место в ОП, в которое можно загрузить программу, COM-файл считывается и помещается в память. Код начинается со смещением в 100h.

Как видно в IP лежит 0100h.





2. Что располагается с адреса 0?

С адреса 0h располагается таблица настроек и PSP.

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

Сегментные регистры CS,ES,SS,CS равны 48DD(указывают на PSP), а SP равен FFFE и указывает на конец сегмента PSP.

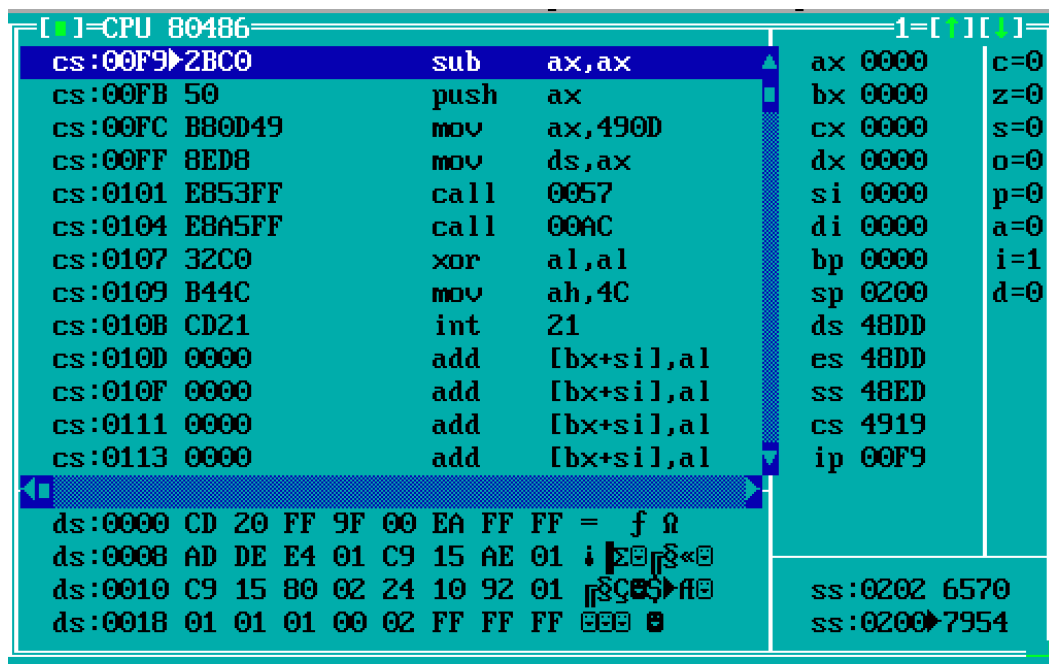
4. Как определяется стек? Какую область памяти он занимает? Какие адреса?

Стек получается автоматически. SS указывает на начало 0h, SP – на конец FFFEh, т.е. стек расположен с 0h по FFFEh.

### Загрузка “хорошего” EXE модуля в основную память

1. Как загружается “хороший” EXE? Какие значения имеют сегментные регистры?

EXE-файл загружается начиная с адреса 0100h, считывается информация заголовка PSP и выполняется перемещение адресов сегментов. DS, ES равны 48DD (указывают на начало PSP), SS равен 48ED (указывает на начало сегмента стека), CS равен 4919 (указывает на начало сегмента команд).



2. На что указывают регистры DS и ES?

DS и ES указывают на начало PSP.

3. Как определяется стек?

Стек определяется следующим образом:

```
AStack SEGMENT STACK
```

```
DW 256 DUP(?)
```

```
AStack ENDS
```

Где 256 – размер стека. Регистр SS указывает на начало сегмента стека, SP – на конец.

4. Как определяется точка входа?

Точка входа определяется при помощи директивы END. Эта директива обозначает конец программы, но также прописав END <entry point> можно указать точку входа в программу.

## ПРИЛОЖЕНИЕ Б

### КОД ПРОГРАММ

**Lab1.asm:**

```

TESTPC          SEGMENT
                  ASSUME  CS:TESTPC, DS:TESTPC, ES:NOTHING,
SS:NOTHING

                  ORG      100H
START:           JMP      BEGIN
;ДАННЫЕ

PC_STR           db          'Type: PC',0DH,0AH,'$'
PC_XT_STR db          'Type: PC/XT',0DH,0AH,'$'
AT_STR           db          'Type: AT',0DH,0AH,'$'
PS2_30_STR       db          'Type: PS2 model 30',0DH,0AH,'$'
PS2_80_STR       db          'Type: PS2 model 80',0DH,0AH,'$'
PSjr_STR         db          'Type: PSjr',0DH,0AH,'$'
PC_Conv_STR       db          'Type: PC Convertible',0DH,0AH,'$'
MS_DOS_VERSION   db          'MS DOS Version:  . ',0DH,0AH,'$'
OEM              db          'OEM: ',0DH,0AH,'$'
NUMBER           db          'Serial number:          $'

;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 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 ; в AL старшая цифра
                  pop     CX          ; в AH младшая

```

```

                                ret
BYTE_TO_HEX                    ENDP
;-----
WRD_TO_HEX                    PROC                        near
;перевод в 16 сс 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 10сс, 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

```

```

;-----
TYPE_IBM_PC PROC          near
                mov     ax,0F000h
                mov     es,ax
                mov     al,es:[0FFFEh]

                cmp     al,0FFh
                je      PC

                cmp     al,0FEh
                je      PC_XT
                cmp     al,0FBh
                je      PC_XT

                cmp     al,0FCh
                je      AT

                cmp     al,0FAh
                je      PS2_30

                cmp     al,0F8h
                je      PS2_80

                cmp     al,0FDh
                je      PSjr

                cmp     al,0F9h
                je      PC_Conv

PC:
                mov     dx,offset PC_STR
                jmp     write_type

PC_XT:
                mov     dx,offset PC_XT_STR
                jmp     write_type

AT:
                mov     dx,offset AT_STR
                jmp     write_type

PS2_30:
                mov     dx,offset PS2_30_STR
                jmp     write_type

PS2_80:
                mov     dx,offset PS2_80_STR
                jmp     write_type

```

PSjr:

```
mov dx,offset PSjr_STR
jmp write_type
```

PC\_Conv:

```
mov dx,offset PC_Conv_STR
```

write\_type:

```
mov AH,09h
int 21h
```

```
ret
```

TYPE\_IBM\_PC                      ENDP

;-----

OS\_VERSION                      PROC                      near

```
mov ah,30h
int 21h
```

```
push ax
mov si, offset MS_DOS_VERSION
add si,17 ;подвинемся на место первой цифры версии в
```

выводе

```
call BYTE_TO_DEC ; переводим число al в десятичное и
сохраняем в память куда указывает si
pop ax
```

```
mov al, ah
add si, 3; чуть подвинем указатель
call BYTE_TO_DEC
mov dx, offset MS_DOS_VERSION
```

```
mov AH,09h
int 21h
```

```
mov si, offset OEM
add si, 5
mov al, bh
call BYTE_TO_DEC
mov dx, offset OEM
```

```
mov AH,09h
int 21h
```

```
mov di, offset NUMBER
add di, 20
```

```

        mov ax, cx
        call WRD_TO_HEX ; уже занесли число в память.
        sub di, 2

        mov al, bl
        call BYTE_TO_HEX
        mov [di], ax
        mov dx, offset NUMBER

        mov AH, 09h
        int 21h

        ret
OS_VERSION ENDP

; КОД
BEGIN:

        call TYPE_IBM_PC
        call OS_VERSION

; ВЫХОД В DOS
        xor AL, AL
        mov AH, 4Ch
        int 21H
TESTPC ENDS
END START ;конец модуля, START - точка входа

```

### Lab1\_exe.asm:

```

AStack SEGMENT STACK

        DW 256 DUP(?)

AStack ENDS

DATA SEGMENT

PC_STR      db      'Type: PC', 0DH, 0AH, '$'

PC_XT_STR   db      'Type: PC/XT', 0DH, 0AH, '$'

```

```

AT_STR      db      'Type: AT',0DH,0AH,'$'
PS2_30_STR  db      'Type: PS2 model 30',0DH,0AH,'$'
PS2_80_STR  db      'Type: PS2 model 80',0DH,0AH,'$'
PSjr_STR    db      'Type: PSjr',0DH,0AH,'$'
PC_Conv_STR db      'Type: PC Convertible',0DH,0AH,'$'
MS_DOS_VERSION db    'MS DOS Version:  . ',0DH,0AH,'$'
OEM         db      'OEM:  ',0DH,0AH,'$'
NUMBER      db      'Serial number:          $'

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 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 ; в AL старшая цифра

pop     CX          ; в AH младшая

ret
```

BYTE\_TO\_HEX ENDP

;-----

WRD\_TO\_HEX PROC near

;перевод в 16 сс 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 10сс, 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_1:    pop     DX
                                pop     CX
                                ret
BYTE_TO_DEC      ENDP
;-----
TYPE_IBM_PC  PROC      near
                                mov     ax,0F000h
                                mov     es,ax
                                mov     al,es:[0FFFEh]

                                cmp     al,0FFh
                                je      PC

                                cmp     al,0FEh
                                je      PC_XT
                                cmp     al,0FBh
                                je      PC_XT

                                cmp     al,0FCh
                                je      AT

                                cmp     al,0FAh
                                je      PS2_30

```

```
    cmp  al,0F8h
    je    PS2_80
```

```
    cmp  al,0FDh
    je    PSjr
```

```
    cmp  al,0F9h
    je    PC_Conv
```

PC:

```
    mov  dx,offset PC_STR
    jmp  write_type
```

PC\_XT:

```
    mov  dx,offset PC_XT_STR
    jmp  write_type
```

AT:

```
    mov  dx,offset AT_STR
    jmp  write_type
```

PS2\_30:

```
    mov  dx,offset PS2_30_STR
    jmp  write_type
```

PS2\_80:

```
    mov  dx,offset PS2_80_STR
```

```
    jmp  write_type
```

PSjr:

```
    mov  dx,offset PSjr_STR
```

```
    jmp  write_type
```

PC\_Conv:

```
    mov  dx,offset PC_Conv_STR
```

write\_type:

```
    mov  AH,09h
```

```
    int  21h
```

```
    ret
```

```
TYPE_IBM_PC      ENDP
```

```
;-----
```

```
OS_VERSION      PROC      near
```

```
    mov  ah,30h
```

```
    int  21h
```

```
    push ax
```

```
    mov  si, offset MS_DOS_VERSION
```

```
    add  si,17 ;подвинемся на место первой цифры версии в
```

выводе

```
    call BYTE_TO_DEC ; переводим число al в десятичное и  
сохраняем в память куда указывает si
```

```
    pop ax
```

```
mov  al, ah  
add  si, 3; чуть подвинем указатель  
call BYTE_TO_DEC  
mov  dx, offset MS_DOS_VERSION
```

```
mov  AH,09h  
int  21h
```

```
mov  si, offset OEM  
add  si, 5  
mov  al, bh  
call BYTE_TO_DEC  
mov  dx, offset OEM
```

```
mov  AH,09h  
int  21h
```

```
mov  di, offset NUMBER  
add  di, 20  
mov  ax, cx  
call WRD_TO_HEX ; уже занесли число в память.  
sub  di, 2
```

```
mov  al, bl  
call  BYTE_TO_HEX  
mov  [di], ax  
mov  dx, offset NUMBER
```

```
mov  AH,09h  
int  21h
```

```
ret
```

```
OS_VERSION      ENDP
```

```
; КОД
```

```
Main      PROC      FAR
```

```
sub  AX,AX  
push AX  
mov  AX,DATA  
mov  DS,AX
```

```
call  TYPE_IBM_PC  
call  OS_VERSION
```

;ВЫХОД В DOS

```
    xor     AL,AL  
    mov     AH,4Ch  
    int     21H
```

Main ENDP

CODE ENDS

END Main