

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. 9383

Мосин К.К.

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.

Тип IBM PC хранится в байте по адресу 0F000:0FFFEh, в предпоследнем байте ROM BIOS. Соответствие кода и типа приведены в табл. 1.

Табл. 1 - Соответствие кода и типа PC

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

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

BH - серийный номер OEM (Original Equipment Manufacturer)

BL:CH – 24-битовый серийный номер пользователя

Функции и структуры данных.

Функции, используемые в программе, приведены в табл. 2.

Табл. 2 - Функции и структуры данных

Название	Описание
tetr_to_hex	Перевод из 4-ной с/с в 16-ную с/с
byte_to_hex	Перевод из 2-ной с/с в 16-ную с/с
word_to_hex	Перевод слова (2 байта) в 16-ную с/с
byte_to_dec	Перевод из 2-ной с/с в 10-ную с/с
print	Вывод сообщения на экран

Задание.

Шаг 1. Напишите текст исходного .COM модуля, определяющий тип PC и версию системы.

Шаг 2. Напишите текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1 и постройте и отладьте его.

Шаг 3. Сравните исходные тексты для .COM и .EXE модулей.

Шаг 4. Запустите FAR и откройте (F3/F4) файл загрузочного модуля .COM и файл “плохого” .EXE в 16-ном виде. Затем откройте (F3/F4) файл загрузочного модуля “хорошего” .EXE и сравните его с предыдущими файлами.

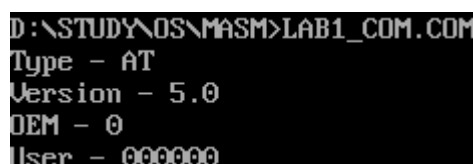
Шаг 5. Откройте отладчик TD.EXE и запустите .COM.

Шаг 6. Откройте отладчик TD.EXE и загрузите “хороший” .EXE.

Шаг 7. Оформление отчета.

Выполнение работы.

Пример .COM модуля, а также “хорошего” и “плохого” .EXE модулей проиллюстрированы на изображениях 1, 2 и 3.



```

D:\STUDY\OS\MASM>LAB1_COM.COM
Type - AT
Version - 5.0
OEM - 0
User - 000000

```

Рис 1 - пример .COM модуля

```

D:\STUDY\OS\MASM>LAB1_COM.EXE

                                0,0Type - PC
5 0
                                0,0Type - PC
0
0,0Type - PC
0000000
0,0Type - PC
0,

```

Рис 2 - пример “плохого” .EXE модуля

```

D:\STUDY\OS\MASM>LAB1_EXE.EXE
Type - AT
Version - 5.0
DEM - 0
User - 000000

```

Рис 3 - пример “хорошего” .EXE модуля

Отличие исходных текстов COM и EXE программ.

Сколько сегментов должна содержать COM-программа?

Код и данные находятся в одном сегменте, поэтому необходимо иметь один сегмент.

EXE-программа?

Для EXE-программы количество сегментов начинается с единицы, но сегменты кода, данных и стека всегда описываются отдельно друг от друга.

Какие директивы должны обязательно быть в тексте COM-программы

Так как первые 256 байт занимает префикс программного сегмента, необходима директива *org 100h*, обеспечивающая смещение. Программа выдаст ошибку при удалении директивы *assume*, так как она указывает на сегменты *cs*, *ds*, *es* и *ss*. В данном случае интересен *cs* и *ds*, в которых описаны сегмент кода и данных для запуска COM программы.

Все ли форматы команд можно использовать в COM-программе?

Команды вида `mov <регистр>` или `seg <сегмент>` не подлежат исполнению. В COM программе отсутствует таблица настроек, поэтому для решения этой проблемы необходима таблица настроек.

Отличие форматов файлов COM и EXE модулей.

Какова структура файла COM? С какого адреса располагается код?

В COM файле находится один сегмент, включающий в себя сегмент кода и данных. При этом стек генерируется автоматически. На проиллюстрированном ниже изображении (рис. 4) наблюдается, что код начинается в адреса 0.

00000000: E9 2C 01 54 79 70 65 20	2D 20 50 43 0D 0A 24 54	é,0Type - PC\
00000001: 79 70 65 20 2D 20 50 43	2F 58 54 0D 0A 24 54 79	ype - PC/XT\
00000002: 70 65 20 2D 20 41 54 0D	0A 24 54 79 70 65 20 2D	pe - AT\
00000003: 20 50 53 32 20 6D 6F 64	65 6C 20 33 30 0D 0A 24	PS2 model 30\
00000004: 54 79 70 65 20 2D 20 50	53 32 20 6D 6F 64 65 6C	Type - PS2 model
00000005: 20 35 30 20 6F 72 20 36	30 0D 0A 24 54 79 70 65	50 or 60\
00000006: 20 2D 20 50 53 32 20 6D	6F 64 65 6C 20 38 30 0D	- PS2 model 80\
00000007: 0A 24 54 79 70 65 20 2D	20 50 43 6A 72 0D 0A 24	\Type - PCjr\
00000008: 54 79 70 65 20 2D 20 50	43 20 43 6F 6E 76 65 72	Type - PC Conver
00000009: 74 69 62 6C 65 0D 0A 24	56 65 72 73 69 6F 6E 20	tible\
0000000A: 2D 20 78 2E 79 0D 0A 24	4F 45 4D 20 2D 20 3F 0D	- x.y\
0000000B: 0A 24 55 73 65 72 20 2D	20 3F 3F 3F 3F 3F 3F 0D	\User - ??????
0000000C: 0A 24 74 65 73 74 20 61	61 61 3D 3D 3D 3D 0D 0A	\test aaa===
0000000D: 24 24 0F 3C 09 76 02 04	07 04 30 C3 51 8A E0 E8	\$o<ov♦♦0AQ5aè
0000000E: EF FF 86 C4 B1 04 D2 E8	E8 E6 FF 59 C3 53 8A FC	iy†Ä±♦0èæyYÄS5ü
0000000F: E8 E9 FF 88 25 4F 88 25	4F 8A C7 E8 DE FF 88 25	èéy^%0^%05Çèpÿ^%
00000010: 4F 88 05 5B C3 51 52 32	E4 33 D2 B9 0A 00 F7 F1	O^+[\ÄQR2ä30¹ ÷ñ
00000011: 80 CA 30 88 14 4E 33 D2	3D 0A 00 73 F1 3C 00 74	€Ê0^¶N30= sñ< t
00000012: 04 0C 30 88 04 5A 59 C3	50 B4 09 CD 21 58 C3 B8	♦00^♦ZYÄP´oÍ!XÄ.
00000013: 00 F0 8E C0 26 A0 FE FF	3C FF 74 20 3C FE 74 25	ðŽÄ& pÿ<y† <pt%
00000014: 3C FB 74 21 3C FC 74 26	3C FA 74 2B 3C FC 74 30	<ût!<ût&<ût+<ût0
00000015: 3C F8 74 35 3C FD 74 3A	3C F9 74 3F BA 03 01 E8	<ø†5<y†:<ût?²♥0è
00000016: C6 FF EB 3D 90 BA 0F 01	E8 BD FF EB 34 90 BA 1E	Æyè=0°00è%yè40▲
00000017: 01 E8 B4 FF EB 2B 90 BA	2A 01 E8 AB FF EB 22 90	0è´yè+0°*0è«yè"0
00000018: BA 40 01 E8 A2 FF EB 19	90 BA 5C 01 E8 99 FF EB	°@0èçyè↓0°\0è™yè
00000019: 10 90 BA 72 01 E8 90 FF	EB 07 90 BA 80 01 E8 87	►0°r0è0yè•0°€0è†
0000001A: FF B4 30 CD 21 BE 98 01	83 C6 0A 50 E8 56 FF 58	y´0Í!%~°fÄPèVÿX
0000001B: 8A C4 83 C6 03 E8 4D FF	BA 98 01 E8 6A FF BE A8	ŠÄfÆ♥èMÿ°~0èjÿ%~

Рис 4 – 16-ное представление .COM файла

Какова структура файла "плохого" EXE? С какого адреса располагается код? Что располагается с адреса 0?

Сегмент кода и данных не разделены на разные сегменты, что приводит к некорректной работе EXE модуля. В плохом EXE файле располагаются MZ байты, заголовок и таблица настроек. По адресу 0h располагается MZ байты,

далее с адреса 2h заголовок и таблица настроек после него. Сам код располагается с адреса 300h.

```

D:\study\OS\MASM\LAB1_COM.EXE
00000000: 4D 5A F0 00 03 00 00 00 20 00 00 00 FF FF 00 00 MZ ̈♥  ŷŷ
000000010: 00 00 DA 93 00 01 00 00 1E 00 00 00 01 00 00 00 Ÿ" ̈ ▲ ̈
000000020: 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
000000040: 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
000000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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 2C 01 54 79 70 65 20 2D 20 50 43 0D 0A 24 54 é,@Type - PC,Type
000000310: 79 70 65 20 2D 20 50 43 2F 58 54 0D 0A 24 54 79 ype - PC/XT,Type
000000320: 70 65 20 2D 20 41 54 0D 0A 24 54 79 70 65 20 2D pe - AT,Type -
000000330: 20 50 53 32 20 6D 6F 64 65 6C 20 33 30 0D 0A 24 PS2 model 30,
000000340: 54 79 70 65 20 2D 20 50 53 32 20 6D 6F 64 65 6C Type - PS2 model
000000350: 20 35 30 20 6F 72 20 36 30 0D 0A 24 54 79 70 65 50 or 60,Type
000000360: 20 2D 20 50 53 32 20 6D 6F 64 65 6C 20 38 30 0D - PS2 model 80,
000000370: 0A 24 54 79 70 65 20 2D 20 50 43 6A 72 0D 0A 24 Type - PCjr,
000000380: 54 79 70 65 20 2D 20 50 43 20 43 6F 6E 76 65 72 Type - PC Conver
000000390: 74 69 62 6C 65 0D 0A 24 56 65 72 73 69 6F 6E 20 tible,Version
0000003A0: 2D 20 78 2E 79 0D 0A 24 4F 45 4D 20 2D 20 3F 0D - x.y,OEM - ??
0000003B0: 0A 24 55 73 65 72 20 2D 20 3F 3F 3F 3F 3F 3F 0D User - ??????

```

Рис 5 – 16-ное представление “плохого” .EXE файла

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

Теперь же присутствует ручное определение стека, а сегмент данных и сегмент кода определены отдельно друг от друга. В “хорошем” EXE модуле отсутствует смещение, определенное в COM модуле директивой *org 100h*.

D:\study\OS\MASM\LAB1_EXE.EXE																		
0000000000:	4D	5A	E4	00	03	00	01	00	20	00	00	00	FF	FF	00	00	MZä ♥ ☉	yy
0000000010:	00	01	5D	C6	5E	00	1C	00	1E	00	00	00	01	00	5F	00	☉]Æ^ L ▲	☉ _
0000000020:	1C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	L	
0000000030:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000040:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000050:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000060:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000070:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000080:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
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		
0000000100:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000110:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000120:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000130:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000140:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000150:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000160:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
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:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000250:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000260:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000270:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000280:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000290:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000002A0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000002B0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000002C0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000002D0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000002E0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000002F0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000300:	54	79	70	65	20	2D	20	50	43	0D	0A	24	54	79	70	65	Type - PC	☉\$Type
0000000310:	20	2D	20	50	43	2F	58	54	0D	0A	24	54	79	70	65	20	- PC/XT	☉\$Type
0000000320:	2D	20	41	54	0D	0A	24	54	79	70	65	20	2D	20	50	53	- AT	☉\$Type - PS
0000000330:	32	20	6D	6F	64	65	6C	20	33	30	0D	0A	24	54	79	70	2 model 30	☉\$Type
0000000340:	65	20	2D	20	50	53	32	20	6D	6F	64	65	6C	20	35	30	e - PS2 model 50	
0000000350:	20	6F	72	20	36	30	0D	0A	24	54	79	70	65	20	2D	20	or 60	☉\$Type -
0000000360:	50	53	32	20	6D	6F	64	65	6C	20	38	30	0D	0A	24	54	PS2 model 80	☉\$Type
0000000370:	79	70	65	20	2D	20	50	43	6A	72	0D	0A	24	54	79	70	ype - PCjr	☉\$Type
0000000380:	65	20	2D	20	50	43	20	43	6F	6E	76	65	72	74	69	62	e - PC Convertib	
0000000390:	6C	65	0D	0A	24	56	65	72	73	69	6F	6E	20	2D	20	78	le	☉\$Version - x
00000003A0:	2E	79	0D	0A	24	4F	45	4D	20	2D	20	3F	0D	0A	24	55	.y	☉\$OEM - ?
00000003B0:	73	65	72	20	2D	20	3F	3F	3F	3F	3F	3F	0D	0A	24	00	ser - ??????	☉\$Type

Рис 6 – 16-ное представление “хорошего” .EXE файла

Загрузка COM модуля в основную память.

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

Загрузка в память происходит считыванием COM файла с диска. Код располагается с адреса IP = 100.

The screenshot shows the DOSBox 0.74-3 interface. The title bar indicates 'DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: TD'. The menu bar includes File, Edit, View, Run, Breakpoints, Data, Options, Window, and Help. The main window displays assembly code for CPU 80486. The code is as follows:

Address	Code	Comment
cs:0100	E92C01	jmp 022F ↓
cs:0103	54	push sp
cs:0104	7970	jns 0176
cs:0106	65202D	and gs:[di],ch
cs:0109	205043	and [bx+si+43],dl
cs:010C	0D0A24	or ax,240A
cs:010F	54	push sp
cs:0110	7970	jns 0182
cs:0112	65202D	and gs:[di],ch
cs:0115	205043	and [bx+si+43],dl
cs:0118	2F	das
cs:0119	58	pop ax
cs:011A	54	push sp

Registers and flags are shown on the right:

Register	Value	Flag
ax	0000	c=0
bx	0000	z=0
cx	0000	s=0
dx	0000	o=0
si	0000	p=0
di	0000	a=0
bp	0000	i=1
sp	0000	d=0
ds	48DD	
es	48DD	
ss	48EC	
cs	48ED	
ip	0100	

Memory dump at the bottom shows:

Address	Hex	ASCII
ds:0000	CD 20 FF 9F 00 EA FF FF	= f 0
ds:0008	AD DE E4 01 C9 15 AE 01	1 2 3 4 5 6 7 8
ds:0010	C9 15 80 02 24 10 92 01	9 10 11 12 13 14 15 16
ds:0018	01 01 01 00 02 FF FF FF	17 18 19 20 21 22 23 24

At the bottom, the status bar shows: F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu.

Что располагается с адреса 0?

Префикс программного сегмента.

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

ds = es = 48DD, ss = 48EC, cs = 48ED. Данные регистры указывают на префикс программного сегмента.

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

Стек определяется автоматически. SS указывает на начало, SP на конец стека. Диапазон равен 0 – FFFh.

Загрузка “хорошего” EXE модуля в основную память

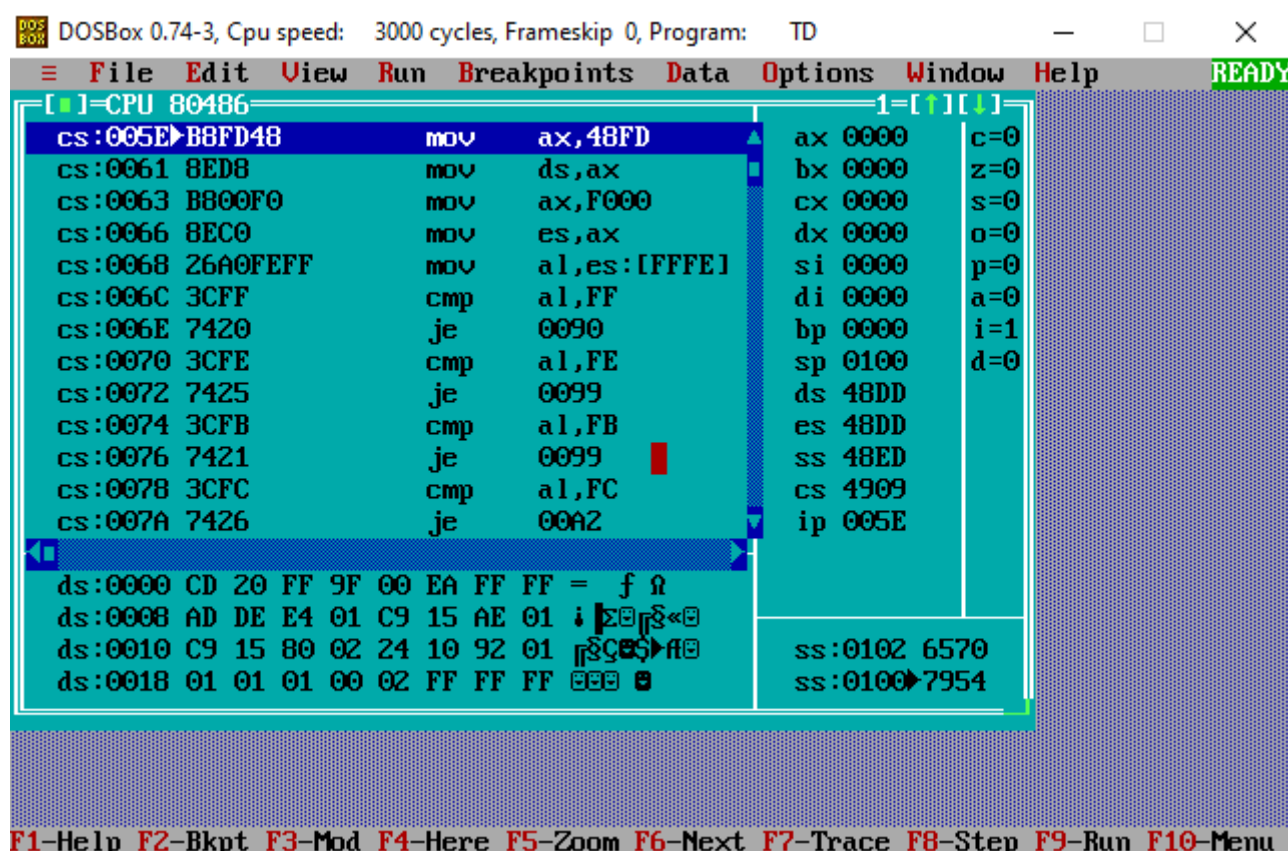
Как загружается "хороший" .EXE? Какие значения имеют сегментные регистры?

При загрузки происходит перемещение адресов сегментов:

ds=es=48DD,

ss=48ED,

cs=4909



На что указывают регистры DS и ES?

На префикс программного сегмента.

Как определяется стек?

Стек определяется директивой stack.

Как определяется точка входа?

Директивой END.

Выводы.

В ходе выполнения лабораторной работы были исследованы различия .COM и .EXE модулей.

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

pc db 'Type - PC',0dh,0ah,'\$'

pcxt db 'Type - PC/XT',0dh,0ah,'\$'

pcat db 'Type - AT',0dh,0ah,'\$'

ps2_30 db 'Type - PS2 model 30',0dh,0ah,'\$'

ps2_50_60 db 'Type - PS2 model 50 or 60',0dh,0ah,'\$'

ps2_80 db 'Type - PS2 model 80',0dh,0ah,'\$'

pcjr db 'Type - PCjr',0dh,0ah,'\$'

pc_convertible db 'Type - PC Convertible',0dh,0ah,'\$'

version db 'Version - x.y',0dh,0ah,'\$'

oem db 'OEM - ?',0dh,0ah,'\$'

user db 'User - ??????',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
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],ah
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
```

```
print proc near
push ax
mov ah,09h
int 21h
pop ax
ret
print endp
```

```

begin:
mov ax,0f000h
mov es,ax
mov al,es:[0fffeh]
cmp al,0ffh
je type_pc
cmp al,0feh
je type_pcxt
cmp al,0fbh
je type_pcxt
cmp al,0fch
je type_at
cmp al,0fah
je type_ps2_30
cmp al,0fch
je type_ps2_50_or_60
cmp al,0f8h
je type_ps2_80
cmp al,0fdh
je type_pcjr
cmp al,0f9h
je type_pc_convertible
type_pc:
mov dx,offset pc
call print
jmp exit
type_pcxt:
mov dx,offset pcxt
call print
jmp exit
type_at:

```



```
mov dx,offset pcat
call print
jmp exit
type_ps2_30:
mov dx,offset ps2_30
call print
jmp exit
type_ps2_50_or_60:
mov dx,offset ps2_50_60
call print
jmp exit
type_ps2_80:
mov dx,offset ps2_80
call print
jmp exit
type_pcjr:
mov dx,offset pcjr
call print
jmp exit
type_pc_convertible:
mov dx,offset pc_convertible
call print

exit:
mov ah,30h
int 21h
mov si,offset version
add si,10
push ax
call byte_to_dec
pop ax
```

```
mov al,ah
add si,3
call byte_to_dec
mov dx,offset version
call print
```

```
mov si,offset oem
add si,6
mov al,bh
call byte_to_dec
mov dx,offset oem
call print
```

```
mov di,offset user
add di,12
call wrd_to_hex
mov ax,cx
mov al,bl
call byte_to_hex
sub di,2
mov [di],ax
mov dx,offset user
call print
```

```
xor al,al
mov ah,4ch
int 21h
testpc ends
end start
```

Название файла: lab1_exe.asm

stacksg segment stack

dw 128 dup(?)

stacksg ends

datasg segment

pc db 'Type - PC',0dh,0ah,'\$'

pcxt db 'Type - PC/XT',0dh,0ah,'\$'

pcat db 'Type - AT',0dh,0ah,'\$'

ps2_30 db 'Type - PS2 model 30',0dh,0ah,'\$'

ps2_50_60 db 'Type - PS2 model 50 or 60',0dh,0ah,'\$'

ps2_80 db 'Type - PS2 model 80',0dh,0ah,'\$'

pcjr db 'Type - PCjr',0dh,0ah,'\$'

pc_convertible db 'Type - PC Convertible',0dh,0ah,'\$'

version db 'Version - x.y',0dh,0ah,'\$'

oem db 'OEM - ?',0dh,0ah,'\$'

user db 'User - ??????',0dh,0ah,'\$'

datasg ends

testpc segment

assume cs:testpc,ds:datasg,ss:stacksg

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],ah
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
```

```
print proc near
push ax
mov ah,09h
int 21h
pop ax
ret
print endp
```

```
main proc far
```

```

mov ax,datasg
mov ds,ax

mov ax,0f000h
mov es,ax
mov al,es:[0fffh]
cmp al,0ffh
je type_pc
cmp al,0feh
je type_pcxt
cmp al,0fbh
je type_pcxt
cmp al,0fch
je type_at
cmp al,0fah
je type_ps2_30
cmp al,0fch
je type_ps2_50_or_60
cmp al,0f8h
je type_ps2_80
cmp al,0fdh
je type_pcjr
cmp al,0f9h
je type_pc_convertible
type_pc:
mov dx,offset pc
call print
jmp exit
type_pcxt:
mov dx,offset pcxt
call print

```

```

jmp exit
type_at:
mov dx,offset pcat
call print
jmp exit
type_ps2_30:
mov dx,offset ps2_30
call print
jmp exit
type_ps2_50_or_60:
mov dx,offset ps2_50_60
call print
jmp exit
type_ps2_80:
mov dx,offset ps2_80
call print
jmp exit
type_pcjr:
mov dx,offset pcjr
call print
jmp exit
type_pc_convertible:
mov dx,offset pc_convertible
call print

exit:
mov ah,30h
int 21h
mov si,offset version
add si,10
push ax

```

```
call byte_to_dec
pop ax
mov al,ah
add si,3
call byte_to_dec
mov dx,offset version
call print
```

```
mov si,offset oem
add si,6
mov al,bh
call byte_to_dec
mov dx,offset oem
call print
```

```
mov di,offset user
add di,12
call wrd_to_hex
mov ax,cx
mov al,bl
call byte_to_hex
sub di,2
mov [di],ax
mov dx,offset user
call print
```

```
xor al,al
mov ah,4ch
int 21h
main endp
testpc ends
```


end main