

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

ОТЧЕТ
по практической работе № 1

по дисциплине «Операционные системы»

Тема: Исследование структур загрузочных модулей

Студент гр. 8383

Степанов В.Д.

Преподаватель

Губкин А.Ф.

Санкт-Петербург

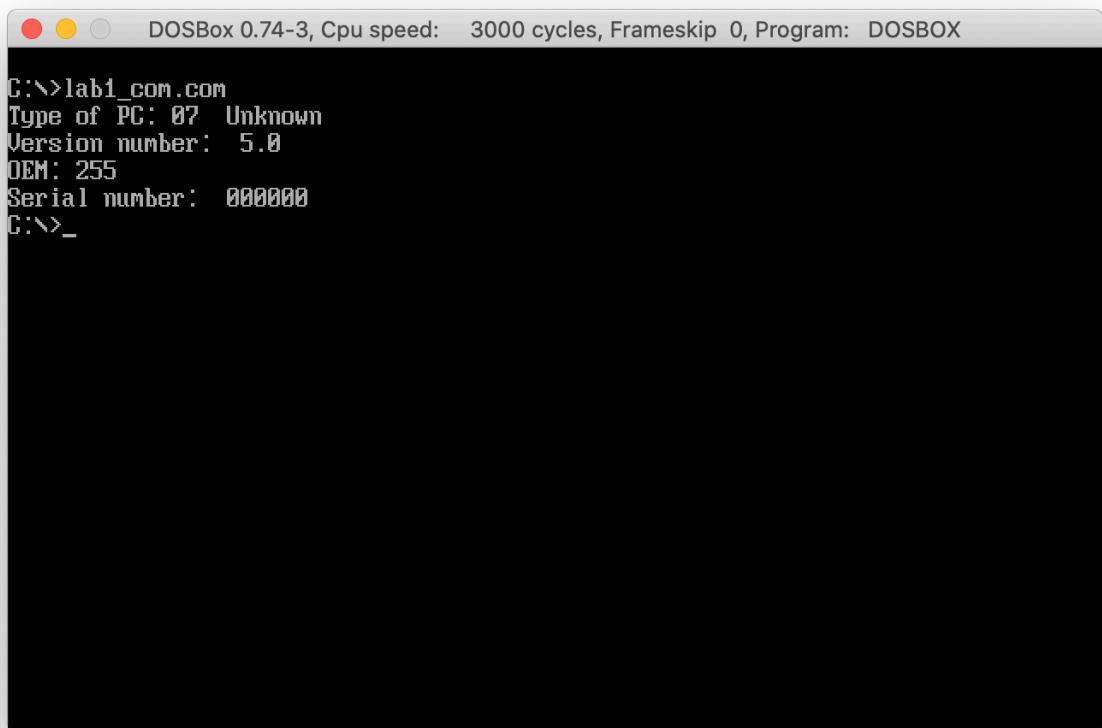
2020

Цель работы.

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

Выполнение работы.

Был написан текст исходного **.COM** модуля, который определяет тип РС и версию системы. Текст кода представлен в приложении А. Из написанного текста был получен «хороший» **.COM** модуль и «плохой» **.EXE** модуль. Работа модулей представлена на рисунке 1 и рисунке 2 соответственно.



DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\>lab1_com.com
Type of PC: 07 Unknown
Version number: 5.0
OEM: 255
Serial number: 000000
C:\>_

Рисунок 1 – Работа «хорошего» **.COM** модуля

Был написан текст для «хорошего» **.EXE** модуля, который выполняет те же функции, что и «хороший» **.COM** модуль. Работа модуля представлена на рисунке 3.

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\>lab1_com.exe

Type of PC: 07
Type of PC: 07
Type of PC: 5.0
Type of PC:
Type of PC: 255
Type of PC: 000000
Type of PC:
C:\>
```

Рисунок 2 – Работа «плохого» .EXE модуля

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\>lab1_exe.exe
Type of PC: 07 Unknown
Version number: 5.0
OEM: 255
Serial number: 000000
C:\>
```

Рисунок 3 – Работа «плохого» .EXE модуля

Сравнение текстов .COM и .EXE модулей

- 1) Сколько сегментов должна содержать СОМ-программа?
СОМ-программа содержит один сегмент не превышающий 64К.

2) EXE-программа?

EXE-программа может содержать множество сегментов.

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

ORG 100h, которая устанавливает в начале программы значение счетчика адресов равное 100h, т.к. операционная система при загрузке программы размещает в ее первые 100h байт префикс программного сегмента. Так же должна быть директива assume, которая кладет в регистр CS адрес сегмента программы. В случае её отсутствия программа, то вылетит ошибки на этапе ассемблирования.

ассемблирования.

Рисунок 4 – Содержание «плохого» .EXE модуля

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

Нет. Например нельзя использовать оператор seg, так как в СОМ-программах отсутствует заголовок.

В приложении FAR были открыты все созданные модули в шестнадцатеричном виде. Содержимое файлов приведено на рисунках 4, 5 и 6.

000000000000: E9 48 02 54 79 70 65 20	6F 66 20 50 43 3A 20 24	иН€Type of PC: \$
0000000010: 50 43 24 50 43 2F 58 54	24 41 54 24 50 53 32 20	PC\$PC/XT\$AT\$PS2
0000000020: 6D 6F 64 65 6C 20 33 30	24 50 53 32 20 6D 6F 64	model 30\$PS2 mod
0000000030: 65 6C 20 35 30 20 6F 72	20 36 30 24 50 53 32 20	e1 50 or 60\$PS2
0000000040: 6D 6F 64 65 6C 20 38 30	24 50 43 6A 72 24 50 43	model 80\$PCjr\$PC
0000000050: 20 63 6F 6E 76 65 72 74	61 62 6C 65 24 20 55 6E	convertable\$ Un
0000000060: 6B 6E 6F 77 6E 24 20 20	20 24 0D 0A 56 65 72 73	known\$ \$JQUsers
0000000070: 69 6F 6E 20 6E 75 6D 62	65 72 3A 20 24 20 20 2E	ion number: \$.
0000000080: 20 20 24 0D 0A 4F 45 4D	3A 20 24 20 20 20 20 20	\$JQ\$OEM: \$
0000000090: 20 24 0D 0A 53 65 72 69	61 6C 20 6E 75 6D 62 65	\$JQ\$Serial numbe
00000000A0: 72 3A 20 24 20 20 20 20	20 20 20 20 20 20 20 20	r: \$
00000000B0: 24 50 52 BA 03 01 E8 76	00 B8 00 F0 8E C0 26 A0	\$PRew@ни ё р?A&
00000000C0: FE FF B0 07 3C FF 74 34	3C FE 74 36 3C FB 74 32	юя°•<ят4<ют6<ят2
00000000D0: 3C FC 74 34 3C FA 74 36	3C FC 74 38 3C F8 74 3A	<ят4<ят6<ят8<ят:
00000000E0: 3C FD 74 3C 3C F9 74 3E	BF 66 01 E8 A2 00 89 05	<ят<<ят>иФ@иУ %
00000000F0: BA 66 01 E8 39 00 BA 5D	01 EB 2E 90 BA 10 01 EB	еf@и9 е]@л .?е►@л
0000000100: 28 90 BA 13 01 EB 22 90	BA 19 01 EB 1C 90 BA 1C	<?е!!@л ?е!@л ?е!
0000000110: 01 EB 16 90 BA 29 01 EB	10 90 BA 3C 01 EB 0A 90	@л_?е>@л!>е<@л@?
0000000120: BA 49 01 EB 04 90 BA 4E	01 E8 03 00 5A 58 C3 50	еI@л♦?еN@и♥ ZXGP
0000000130: 2B C0 B4 09 CD 21 58 C3	50 53 2B DB B3 10 B4 00	+A?оH!ХГРС+И?►?
0000000140: F6 F3 8B D0 8B D0 80 C2	30 2B C0 80 FA 3A 72 03	иу<Р<Р?В0+A?ь:г♥
0000000150: 80 C2 07 B4 02 CD 21 8A	D6 80 C2 30 2B C0 80 FA	?В?@Н!?Ц?В0+A?ь
0000000160: 3A 72 03 80 C2 07 B4 02	CD 21 5B 58 C3 53 8A FC	:г♥?В•?@Н! ZXGS?Ь
0000000170: E8 1D 00 88 25 4F 88 05	4F 8A C7 E8 12 00 88 25	и+ ?ХО?@О?Зи‡? ?Х
0000000180: 4F 88 05 5B C3 24 0F 3C	09 76 02 04 07 04 30 C3	О?+ЛГ\$* <kои@е♦+♦ог< td=""></kои@е♦+♦ог<>
0000000190: 51 8A E0 E8 EF FF 86 C4	B1 04 D2 E8 E6 FF 59	Q?апия+Д+♦ТиижяУ
00000001A0: C3 51 52 32 E4 33 D2 B9	0A 00 F7 F1 80 CA 30 88	GQR2д3TН@ чс?к0?
00000001B0: 14 4E 33 D2 3D 0A 00 73	F1 3C 00 74 04 0C 30 88	ЧН3T=0 sc< t♦@0?
00000001C0: 04 5A 59 C3 50 53 51 52	2B C0 B4 30 CD 21 BA 6A	♦ZYGPSQR+A?ОН!ej
00000001D0: 01 E8 5B FF 50 BE 7D 01	46 E8 C5 FF 58 8A C4 83	Ои[яР?]@ФиЕяХ?д?
00000001E0: C6 03 E8 BC FF BA 7D 01	E8 44 FF BA 83 01 E8 3E	Ж@и?яе>@иDяе?@и>
00000001F0: FF BE 8B 01 83 C6 02 8A	C7 E8 A5 FF BA 8B 01 E8	я?<@?Ж@?Зи?яе<@и
0000000200: 2D FF BA 92 01 E8 27 FF	BF A4 01 83 C7 06 8B C1	-яе'@и' яГ@?З@?3♦<Б
0000000210: E8 5A FF 8A C3 E8 78 FF	83 EF 02 89 05 BA A4 01	иЗя?Гихя?п@иФе@и
0000000220: E8 0C FF 5A 59 5B 58 C3	50 53 2B DB B3 0A B4 00	и@яZY ZXGPS+И?@?
0000000230: F6 F3 8B D0 80 C2 30 2B	C0 B4 02 CD 21 8A D6 80	иу<Р?В0+A?@Н!?Ц?
0000000240: C2 30 2B C0 B4 02 CD 21	5B 58 C3 E8 63 FE E8 73	В0+A?@Н! ZXГисюи@ яЗА?ЛН!
0000000250: FF 33 C0 B4 4C CD 21		

Рисунок 5 – Содержание «хорошего» .СОМ модуля

Отличия форматов файлов СОМ и EXE модулей

1) Какова структура файла СОМ? С какого адреса располагается код?

В файлах содержится только машинный код и данные программы. Код программы начинается с адреса 0h, но при загрузке происходит смещение на 100h.

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

У «плохого» .EXE модуля данные и код располагаются в одном сегменте, так как

он был получен текста для .COM модуля. Код располагается по адресу 300h, так как перед ним начиная с адреса 0h находится таблица настроек.

3) Какова структура файла «хорошего» EXE модуля? Чем он отличается от файла «плохого» EXE?

Так как «плохой» .EXE был получен из текста для .COM модуля, поэтому у него данные и код располагаются в одном сегменте, в отличие от «хорошего» .EXE. Так же в «плохом» .EXE отсутсвует выделение стека.

000000000000: 4D 5A 36 00 03 00 01 00	20 00 00 00 FF FF 00 00	MZ6 ♥ ⊖ яя
000000000010: 02 00 00 00 61 01 0C 00	3E 00 00 00 01 00 FB 50	⊕ a⊕♀ > ⊕ мР
000000000020: 6A 72 00 00 00 00 00 00	00 00 00 00 00 00 00 00	jr
000000000030: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 65 01	e⊖
000000000040: 0C 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	♀
000000000050: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000000060: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000000070: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000000080: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000000090: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000000A0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000000B0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000000C0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000000D0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000000E0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000000F0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001000: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001100: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001200: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001300: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001400: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001500: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001600: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001700: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001800: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001900: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001A00: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001B00: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001C00: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001D00: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001E00: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000001F00: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000002000: 00 01 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000002100: 54 29 70 65 20 6F 66 20	50 43 3A 20 24 50 43 24	Type of PC: \$PC\$
000000002200: 50 43 2F 58 54 24 41 54	24 50 53 32 20 6D 6F 64	PC/XT\$AT\$PS2 mod
000000002300: 65 6C 20 33 30 24 50 53	32 20 6D 6F 64 65 6C 20	e1 30\$PS2 model
000000002400: 35 30 20 6F 72 20 36 30	24 50 53 32 20 6D 6F 64	50 or 60\$PS2 mod
000000002500: 65 6C 20 38 30 24 50 43	6A 72 24 50 43 20 63 6F	e1 80\$PCj\$PC co
000000002600: 6E 26 65 72 24 61 62 6C	65 24 20 55 6E 6B 6E 6F	nvertable\$ Unkn
000000002700: 77 6E 24 20 20 20 24 0D	0A 56 65 72 73 69 6F 6E	wn\$ \$J\$Qversion
000000002800: 20 6E 75 6D 62 65 72 3A	20 24 20 20 20 20 20 24	number: \$. \$
000000002900: 0D 0A 4F 45 4D 3A 20 24	20 20 20 20 20 20 20 24	J\$OEM: \$. \$P
000000002A00: 0A 53 65 72 69 61 6C 20	6E 75 6D 62 65 72 3A 20	Serial number:
000000002B00: 24 20 20 20 20 20 20 20	20 20 20 20 20 20 24 00	\$ \$
000000002C00: 50 52 BA 00 00 E8 76 00	B8 00 F0 8E C0 26 A0 FE	PRe ии ё р?A& ю
000000002D00: FF B0 07 3C FF 74 34 3C	FE 74 36 3C FB 74 32 3C	я°<ят4<т6<т2<
000000002E00: FC 74 34 3C FA 74 36 3C	FC 74 38 3C FB 74 3A 3C	ьт4<т6<т8<т:<<
000000002F00: FD 74 3C 3C F9 74 3E BF	63 00 E8 6D 00 89 05 BA	зт(<тт)< ии ×д
000000003000: 63 00 E8 39 00 BA 5A 00	EB 24 90 BA 00 00 EB 28	с и9 єZ л.?еJ л<
000000003100: 90 BA 10 00 EB 22 90 BA	16 00 EB 1C 90 BA 19 00	?е> л?"е_ л-?е↓
000000003200: EB 16 90 BA 26 00 EB 10	90 BA 39 00 EB 0A 90 BA	л-?е& л-?е9 л?е
000000003300: 46 00 EB 04 90 BA 4B 00	E8 03 00 5A 58 C3 50 2B	F л-?еK ии ZХГР+
000000003400: C0 B4 09 CD 21 58 C3 53	8A FC E8 1D 00 88 25 4F	A?ОН!ХГ\$?и+?Х0
000000003500: 88 05 4F 8A C7 E8 12 00	88 25 4F 88 05 5B C3 24	?ХО?Зи+ ?ХО?ФЛГ\$
000000003600: 0F 3C 09 76 02 04 07 04	30 C3 51 8A E0 0E EF FF	*Ков\$♦*ФЛQ?ампя
000000003700: 86 C4 B1 04 D2 E8 E8 E6	FF 59 C3 51 52 32 E4 33	+Д+ФТинжУГQR2л3
000000003800: D2 B9 0A 00 F7 F1 80 CA	30 88 14 4E 33 D2 3D 0A	THQ ч?ХО?ФN3T=0
000000003900: 00 73 F1 3C 00 74 00 0C	30 88 04 5A 59 C3 50 53	sc< т*90?♦ZУГР\$
000000003A00: 2B C0 B4 30 CD 21 BA 67	00 E8 92 FF 50 BE 2A 00	+А?ОН!leg и'яР?з
000000003B00: 46 E8 C7 FF 58 8A C4 83	C6 03 E8 BE FF BA 2A 00	РиЗяХ?Д?Ж?и?яез
000000003C00: E8 7B FF BA 80 00 E8 75	FF BE 88 00 83 C6 02 8A	и<яе? ииа?? ?Ж?
000000003D00: C7 E8 A7 FF BA 88 00 E8	64 FF BA 8F 00 E8 5E FF	ЗиЯяе? ииае? и^я
000000003E00: BF A1 00 83 C7 06 8B C1	E8 5C FF 8A C3 E8 2A FF	Y9 ?3* Би\я?Гизя
000000003F00: 83 EF 02 89 05 BA A1 00	E8 43 FF 5B 58 C3 50 53	?и?и?и?и?и?и?и?и?
000000004000: 2B DB B3 0A B4 00 F6 F3	8B D0 80 C2 30 2B C0 B4	+И?О? цу<Р?В0+A?
000000004100: 02 CD 21 8A D6 80 C2 30	2B C0 B4 02 CD 21 5B 58	ИН?И?И?И?И?И?И?
000000004200: C3 50 2B C0 B8 01 00 8E	D8 58 E8 93 FE E8 6E FF	ГР+АéО ?ИИи"юипя
000000004300: 2B C0 B4 4C CD 21		+А?ЛН!

Рисунок 6 – Содержание «хорошего» .EXE модуля

При помощи отладчика TD.EXE был загружен .COM модуль. Результат загрузки показан на рисунке 7.

Загрузка .COM модуля в основную память

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

Код, данные, стек и PSP располагаются в одном сегменте. Код располагается с адреса 0100h.

- 2) Что располагается с адреса 0?

С адреса 0h располагается PSP и заканчивается перед адресом 100h.

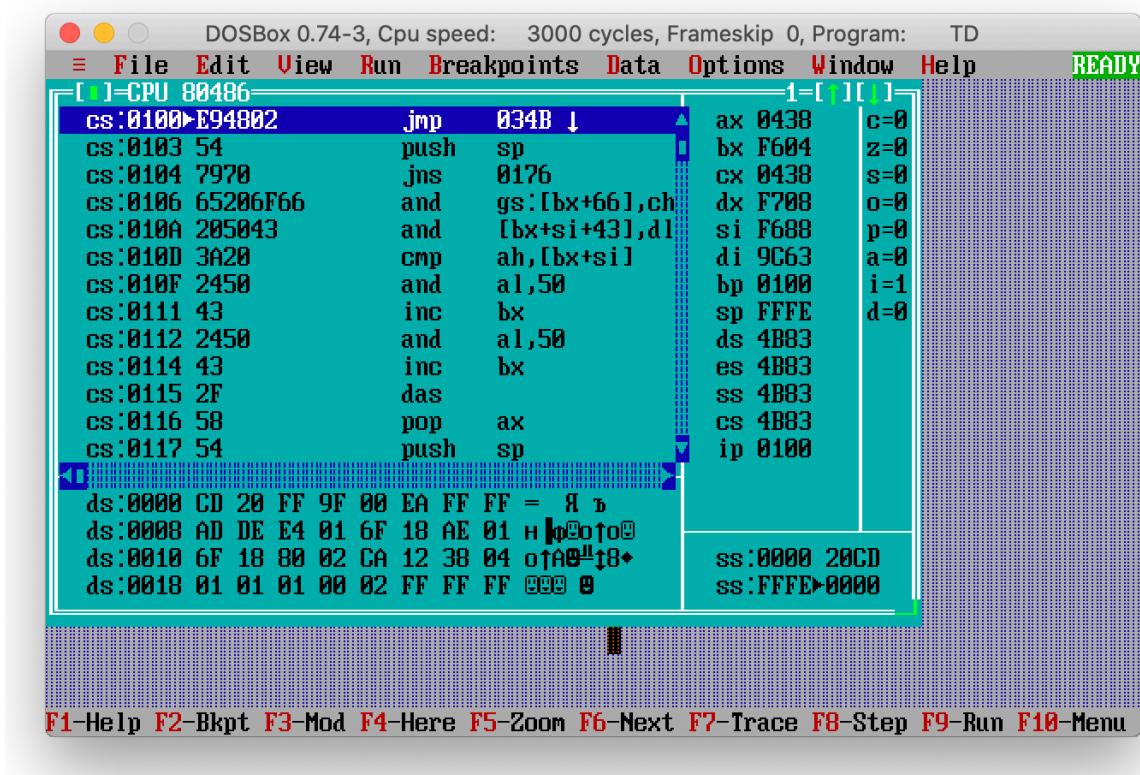


Рисунок 7 – Открытие .COM модуля в TD.EXE

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

Все сегментные регистры указывают на начало единственного регистра (начало PSP).

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

Стек занимает место, которое на занято PSP и машинным кодом. Регистр SP указывает на адрес FFFEh.

При помощи отладчика TD.EXE был загружен «хороший» .EXE модуль. Результат загрузки показан на рисунке 8.

Загрузка «хорошего» .EXE модуля в основную память

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

Определяется свободный участок памяти, в который можно будет загрузить программу. Выделяется сегмент для PSP . CS указывает на начало кода, SP указывает на начало стека.

2) На что указывают регистры DS и ES?

DS и ES указывают на начало PSP.

3) Как определяется стек?

Стек определяется вручную через директиву ASSUME: SS:LAB1STACK. Если объявление отсутствует, то стек создается по адресу FFFEh.

4) Как определяется точка входа?

При помощи директивы END.

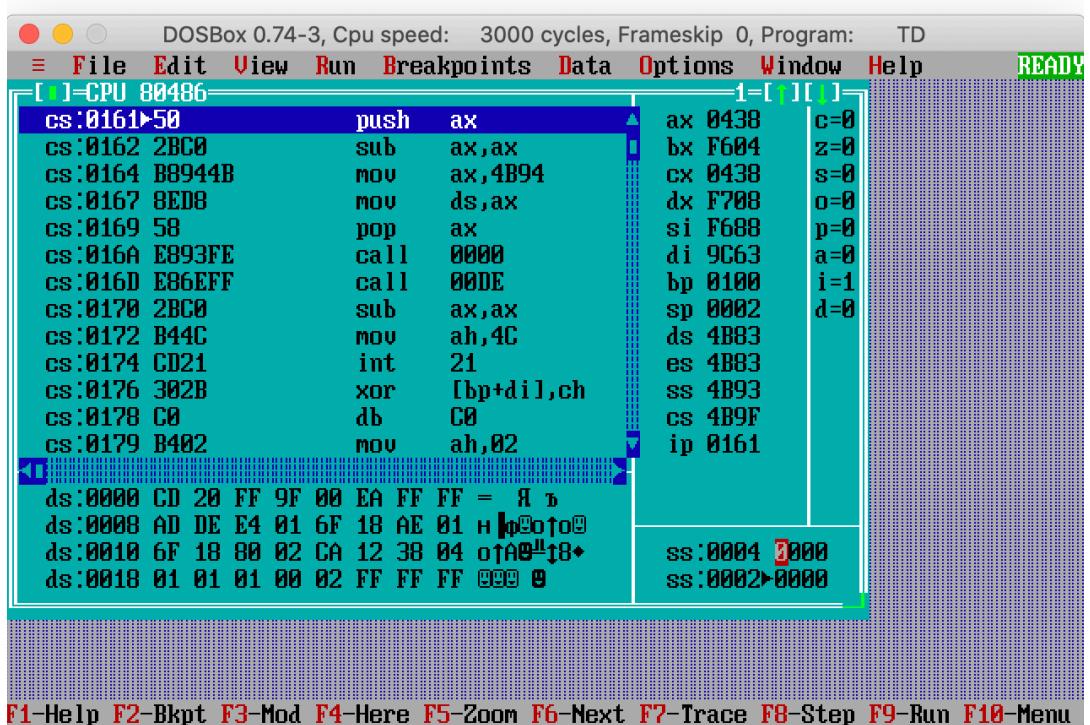


Рисунок 8 – Открытие «хорошего» .EXE модуля в TD.EXE

Выводы.

В ходе лабораторной работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов загрузки в основную память.

ПРИЛОЖЕНИЕ А

КОД ДЛЯ .COM МОДУЛЯ

```
LAB1 SEGMENT
    ASSUME CS:LAB1, DS:LAB1, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN

;-----
TYPE_OF_PC db "Type of PC: $"
PC db "PC$"
PC_XT db "PC/XT$"
AT db "AT$"
PS2_30 db "PS2 model 30$"
PS2_50 db "PS2 model 50 or 60$"
PS2_80 db "PS2 model 80$"
PCjr db "PCjr$"
PC_Con db "PC convertable$"
UNKNOWN_TYPE db " Unknown$"
OUT_UNKNOWN db "      $"
VER_NUM db 13, 10, "Version number: $"

OUT_VER_NUM db "  .  $"

OEM_NUM db 13, 10, "OEM: $"
OUT_OEM db "          $"
SER_NUM db 13, 10, "Serial number: $"
OUT_SER_NUM DB "                  $"
;-----

TYPE_PC PROC near

    push ax
    push dx

    mov dx, offset TYPE_OF_PC
    call PRINT

    mov ax, 0F000h
    mov es, ax
    mov al, es:[0FFEh]

    mov al, 7h
```

```
    cmp al, 0FFh
        je W_PC
    cmp al, 0FEh
        je W_PC_XT
    cmp al, 0FBh
        je W_PC_XT
    cmp al, 0FCh
        je W_AT
    cmp al, 0FAh
        je W_PS2_30
    cmp al, 0FCh
        je W_PS2_50
    cmp al, 0F8h
        je W_PS2_80
    cmp al, 0FDh
        je W_PCjr
    cmp al, 0F9h
        je W_PC_Con
```

```
    mov di, offset OUT_UNKNOWN
    call BYTE_TO_HEX
    mov [di], ax
    mov dx, offset OUT_UNKNOWN
    CALL PRINT
```

```
    mov DX, offset UNKNOWN_TYPE
    jmp ESC_PROC
```

W_PC:

```
    mov DX, offset PC
    jmp ESC_PROC
```

W_PC_XT:

```
    mov DX, offset PC_XT
    jmp ESC_PROC
```

W_AT:

```
    mov DX, offset AT
    jmp ESC_PROC
```

W_PS2_30:

```

        mov DX, offset PS2_30
        jmp ESC_PROC

W_PS2_50:
        mov DX, offset PS2_50
        jmp ESC_PROC

W_PS2_80:
        mov DX, offset PS2_80
        jmp ESC_PROC

W_PCjr:
        mov DX, offset PCjr
        jmp ESC_PROC

W_PC_Con:
        mov DX, offset PC_Con

ESC_PROC:
        call PRINT

pop dx
pop ax

ret
TYPE_PC ENDP

;-----

PRINT PROC near

push ax
sub ax, ax
mov ah, 9h
int 21h
pop ax

ret
PRINT ENDP

;-----


WRD_TO_HEX    PROC    near

```

```

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

WRD_TO_HEX ENDP

;-----

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 ;â AL ñòàðøàÿ öèôðà
        pop       CX           ;â AH ìëàäøàÿ
        ret

BYTE_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

```

-----;

```

SYSTEM_VERSION PROC

    push ax
    push bx
    push cx
    push dx

    sub ax, ax
    mov ah, 30h
    int 21h

    mov dx, offset VER_NUM
    call PRINT

    push ax
    mov si, offset OUT_VER_NUM
    inc si
    call BYTE_TO_DEC

```

```

pop ax
mov al, ah
add si, 3
call BYTE_TO_DEC

mov dx, offset OUT_VER_NUM
call print

mov DX, offset OEM_NUM
call PRINT

mov si, offset OUT_OEM
add si, 2
mov al, bh
call BYTE_TO_DEC
mov dx, OFFSET OUT_OEM
call PRINT

mov dx, offset SER_NUM
call PRINT

mov di, offset OUT_SER_NUM
add di, 6
mov ax, cx

call WRD_TO_HEX
mov al, bl
call BYTE_TO_HEX
sub di, 2
mov [di], ax
mov dx, offset OUT_SER_NUM
CALL PRINT

pop DX
pop CX
pop BX
pop AX
ret
SYSTEM_VERSION ENDP
;
```

```
PRINT_NUMBER_DEC PROC
```

```
    push AX
```

```
    push BX
```

```
    sub BX, BX
```

```
    mov Bl, 10
```

```
    mov AH, 0
```

```
    div Bl
```

```
    mov DX, AX
```

```
    add DL, '0'
```

```
    sub AX, AX
```

```
    mov AH, 02h
```

```
    int 21h
```

```
    mov DL, DH
```

```
    add DL, '0'
```

```
    sub AX, AX
```

```
    mov AH, 02h
```

```
    int 21h
```

```
    pop BX
```

```
    pop AX
```

```
    ret
```

```
PRINT_NUMBER_DEC ENDP
```

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

```
BEGIN:
```

```
    call TYPE_PC
```

```
    call SYSTEM_VERSION
```

```
    xor AX, AX
```

```
    mov AH, 4Ch
```

```
    int 21h
```

```
LAB1 ENDS
```

```
END START
```

ПРИЛОЖЕНИЕ Б

КОД ДЛЯ «ХОРОШЕГО».EXE МОДУЛЯ

```
LAB1STACK SEGMENT STACK
    dw 100h
LAB1STACK ENDS

DATA SEGMENT

    TYPE_OF_PC db "Type of PC: $"
    PC db "PC$"
    PC_XT db "PC/XT$"
    AT db "AT$"
    PS2_30 db "PS2 model 30$"
    PS2_50 db "PS2 model 50 or 60$"
    PS2_80 db "PS2 model 80$"
    PCjr db "PCjr$"
    PC_Con db "PC convertable$"
    UNKNOWN_TYPE db " Unknown$"
    OUT_UNKNOWN db "      $"
    VER_NUM db 13, 10, "Version number: $"

    OUT_VER_NUM db " .   $"

    OEM_NUM db 13, 10, "OEM: $"
    OUT_OEM db "        $"
    SER_NUM db 13, 10, "Serial number: $"
    OUT_SER_NUM DB "                  $""

DATA ENDS

LAB1CODE SEGMENT
ASSUME CS: LAB1CODE, DS:DATA, ES:NOTHING, SS:LAB1STACK

;-----



TYPE_PC PROC near

    push ax
    push dx

    mov dx, offset TYPE_OF_PC
```

```

call PRINT

mov ax, 0F000h
mov es, ax
mov al, es:[0FFEh]

mov al, 7h

cmp al, 0FFh
    je W_PC
cmp al, 0FEh
    je W_PC_XT
cmp al, 0FBh
    je W_PC_XT
cmp al, 0FCCh
    je W_AT
cmp al, 0FAh
    je W_PS2_30
cmp al, 0FCCh
    je W_PS2_50
cmp al, 0F8h
    je W_PS2_80
cmp al, 0FDh
    je W_PCjr
cmp al, 0F9h
    je W_PC_Con

mov di, offset OUT_UNKNOWN
call BYTE_TO_HEX
mov [di], ax
mov dx, offset OUT_UNKNOWN
CALL PRINT

mov DX, offset UNKNOWN_TYPE
jmp ESC_PROC

W_PC:
    mov DX, offset PC
    jmp ESC_PROC

W_PC_XT:
    mov DX, offset PC_XT

```

```

        jmp ESC_PROC

W_AT:
    mov DX, offset AT
    jmp ESC_PROC

W_PS2_30:
    mov DX, offset PS2_30
    jmp ESC_PROC

W_PS2_50:
    mov DX, offset PS2_50
    jmp ESC_PROC

W_PS2_80:
    mov DX, offset PS2_80
    jmp ESC_PROC

W_PCjr:
    mov DX, offset PCjr
    jmp ESC_PROC

W_PC_Con:
    mov DX, offset PC_Con

ESC_PROC:
    call PRINT

    pop dx
    pop ax

    ret
TYPE_PC ENDP

;-----


PRINT PROC near

    push ax
    sub ax, ax
    mov ah, 9h
    int 21h
    pop ax

```

```

    ret
PRINT ENDP

;-----

WRD_TO_HEX PROC near
    push     BX
    mov      BH, AH
    call     BYTE_TO_HEX
    mov      [DI], AH
    dec      DI
    mov      [DI], AL
    dec      DI
    mov      AL, BH
    call     BYTE_TO_HEX
    mov      [DI], AH
    dec      DI
    mov      [DI], AL
    pop     BX
    ret
WRD_TO_HEX ENDP

;-----


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 ;â AL ñòàðøàÿ öèôðà
        pop       CX           ;â AH ìëàäøàÿ
        ret
BYTE_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
```

;

```
SYSTEM_VERSION PROC
```

```
push ax
push bx

sub ax, ax
mov ah, 30h
int 21h

mov dx, offset VER_NUM
```

```

call PRINT

push ax
mov si, offset OUT_VER_NUM
inc si
call BYTE_TO_DEC
pop ax
mov al, ah
add si, 3
call BYTE_TO_DEC

mov dx, offset OUT_VER_NUM
call print

mov DX, offset OEM_NUM
call PRINT

mov si, offset OUT_OEM
add si, 2
mov al, bh
call BYTE_TO_DEC
mov dx, OFFSET OUT_OEM
call PRINT

mov dx, offset SER_NUM
call PRINT

mov di, offset OUT_SER_NUM
add di, 6
mov ax, cx

call WRD_TO_HEX
mov al, bl
call BYTE_TO_HEX
sub di, 2
mov [di], ax
mov dx, offset OUT_SER_NUM
CALL PRINT

pop bx
pop ax

```

```
    ret  
SYSTEM_VERSION ENDP
```

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

```
PRINT_NUMBER_DEC PROC
```

```
    push AX  
    push BX
```

```
    sub BX, BX  
    mov Bl, 10  
    mov AH, 0  
    div Bl  
    mov DX, AX
```

```
    add DL, '0'  
    sub AX, AX  
    mov AH, 02h  
    int 21h
```

```
    mov DL, DH
```

```
    add DL, '0'  
    sub AX, AX  
    mov AH, 02h  
    int 21h
```

```
    pop BX  
    pop AX  
    ret
```

```
PRINT_NUMBER_DEC ENDP
```

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

```
MAIN PROC FAR
```

```
    push AX  
    sub AX, AX  
    mov ax, data  
    mov ds, ax  
    POP AX
```

```
    call TYPE_PC  
    call SYSTEM_VERSION
```

```
sub AX, AX  
mov AH, 4Ch  
int 21h
```

```
MAIN ENDP  
LAB1CODE ENDS  
END MAIN
```