

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

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

Студентка гр. 8381

Звегинцева Е.Н.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

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

Необходимые сведения для составления программы.

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

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:

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.

Постановка задачи.

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

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

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

Выполнение работы.

Выполнение работы производилось на базе операционной системы Windows 10. Сборка и отладка модулей производилась с помощью компиляторов MASM и TASM и отладчика TD в эмуляторе DOSBox.

Типы исполняемых файлов в DOS исследовались на примере программы, которая выводит версию операционной системы. Программа содержит таблицу кодов, обозначающих различные типы PC. Она считывает код из памяти BIOS, ищет его по таблице и выводит строку с названием этого типа (функции LOOKUP_MODEL и PRINT_MODEL). Затем загружается информация о версии DOS, и отображается в функции

PRINT_VERSION_INFO. Вывод чисел выполняется при помощи функций печати чисел в строку BYTE_TO_DEC, BYTE_TO_HEX и WRD_TO_HEX из методички. Эта программа была написана таким образом, чтобы из неё можно было получить загрузочный модуль типа COM (LAB1COM.ASM). После этого она была модифицирована в программу EXE (LAB1EXE.ASM).

Пример сборки .COM модуля показан на рис. 1.

```
C:\>tasm lab1com.asm
Turbo Assembler Version 4.0 Copyright (c) 1988, 1993 Borland International

Assembling file:   lab1com.asm
Error messages:    None
Warning messages:  None
Passes:           1
Remaining memory:  466k

C:\>tlink /t lab1com.obj
Turbo Link Version 4.01 Copyright (c) 1991 Borland International
```

Рисунок 1 – Полученный .COM модуль

Во время линковки «плохого» .EXE модуля было выведено предупреждение об отсутствии сегмента стека, представлено на рис. 2.

Запуск «плохого» .EXE модуля.

```
C:\>link lab1com.obj

Microsoft (R) Overlay Linker Version 3.64
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.

Run File [LAB1COM.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment
```

Рисунок 2 – Предупреждение во время линковки «плохого» .EXE



Рисунок 3 – Вывод плохого .EXE модуля

```
C:\>lab1com.com
AT or PS/2 model 50 or 60
DOS version: 5.0
OEM number: FF
User serial number: 000000
```

Рисунок 4 — Вывод .COM модуля

```
C:\>lab1exe.exe
AT or PS/2 model 50 or 60
DOS version: 5.0
OEM number: FF
User serial number: 000000
```

Рисунок 5 — Вывод «хорошего» .EXE модуля

Отличия исходных текстов COM и EXE программ.

1. Сколько сегментов должна содержать COM-программа?

Один: сегмент кода и данных. Сегмент стека создаётся при запуске автоматически.

2. EXE программа?

Произвольное число сегментов (в частности, стек нужно создавать вручную, т.к. он не создаётся автоматически).

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

Директива ORG 100h (смещение 100h), так как при загрузке COM-файла в память DOS занимает первые 256 байт (100h) блоком данных PSP и

располагает код программы только после этого блока. Также обязательна директива ASSUME, которая устанавливает значения регистров CS, DS на адрес сегмента программы.

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

Нельзя использовать команды, связанные с адресом сегмента, так как он неизвестен до загрузки этого сегмента в память, т.к. в .COM файле отсутствует таблица настройки с информацией о типе адресов и их местоположении.

Отличия форматов файлов COM и EXE модулей.

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

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

00000000	E9 C1 01 50 43 24 50 43	2F 58 54 24 41 54 20 6F	o PC\$PC/XT\$AT o
00000010	72 20 50 53 2F 32 20 6D	6F 64 65 6C 20 35 30 20	r PS/2 model 50
00000020	6F 72 20 36 30 24 50 53	2F 32 20 6D 6F 64 65 6C	or 60\$PS/2 model
00000030	20 33 30 24 50 53 2F 32	20 6D 6F 64 65 6C 20 38	30\$PS/2 model 8
00000040	30 24 50 43 6A 72 24 50	43 20 43 6F 6E 76 65 72	0\$PCjr\$PC Conver
00000050	74 69 62 6C 65 24 55 6E	6B 6E 6F 77 6E 20 6D 6F	tible\$Unknown mo
00000060	64 65 6C 24 03 01 06 01	06 01 0C 01 26 01 34 01	del\$. & . 4 .
00000070	42 01 47 01 56 01 FF FE	FB FC FA F8 FD F9 00 0D	B . G . V
00000080	0A 24 44 4F 53 20 76 65	72 73 69 6F 6E 3A 20 24	. \$DOS version: \$
00000090	4F 45 4D 20 6E 75 6D 62	65 72 3A 20 24 55 73 65	OEM number: \$Use
000000A0	72 20 73 65 72 69 61 6C	20 6E 75 6D 62 65 72 3A	r serial number:
000000B0	20 24 53 33 DB 8A 97 76	01 84 D2 74 07 3A D0 74	\$S3\$èuv. ätt. : \$t
000000C0	03 43 EB F1 D1 E3 88 97	64 01 5B C3 52 BA 7F 01	. C5±πiùd. [R △.
000000D0	B4 09 CD 21 5A C3 52 B4	09 CD 21 E8 EE FF 5A C3	. . = ! Z R . = ! ± z Z
000000E0	24 0F 3C 09 76 02 04 07	04 30 C3 51 8A E0 E8 EF	\$. < . v 0 Qèαφn
000000F0	FF 86 C4 B1 04 D2 E8 E8	E6 FF 59 C3 53 8A FC E8	ä - Y - Sèñφ
00000100	E9 FF 88 25 4F 88 05 4F	8A C7 E8 DE FF 88 25 4F	e è%0è. 0è φ è%0
00000110	88 05 5B C3 51 52 32 E4	33 D2 B9 0A 00 F7 F1 80	è. [QR2Σ3 . . ~ ± Ç
00000120	CA 30 88 14 4E 33 D2 3D	0A 00 73 F1 3C 00 74 04	Δ0è. N3T = . . s ± < . t .
00000130	0C 30 88 04 5A 59 C3 52	50 55 56 57 50 BA 82 01	. 0è. ZY RPUVWP é.
00000140	B4 09 CD 21 58 83 EC 07	8B EC 50 8D 76 02 C6 44	. . = ! Xä∞. i∞Piv. D
00000150	01 24 E8 BF FF 8D 54 01	B4 09 CD 21 B4 02 B2 2E	. \$φ iT. . = ! . .
00000160	CD 21 58 8A C4 8D 76 02	E8 A9 FF 8D 54 01 B4 09	= ! Xè - iv. φ - iT. .
00000170	CD 21 E8 57 FF BA 90 01	B4 09 CD 21 8A C7 E8 6A	= ! φW É. . = ! è φj
00000180	FF 50 8A D0 B4 02 CD 21	58 8A D4 B4 02 CD 21 E8	Pè . = ! Xè . = ! φ
00000190	3A FF BA 9D 01 B4 09 CD	21 8D 76 05 C6 44 01 24	: ¥. . = ! iv. D. \$
000001A0	8B FE 8B C1 E8 55 FF 8A	C3 E8 3F FF 88 44 FB 88	i . i LφU è φ? èD√è
000001B0	64 FC 8B D5 B4 09 CD 21	E8 11 FF 83 C4 07 5F 5E	d^n i . = ! φ. ä - . ^
000001C0	5D 58 5A C3 1E B8 00 F0	8E D8 BB FE FF 8A 07 1F] XZ . . = A . è . .
000001D0	E8 DF FE E8 00 FF B8 00	30 CD 21 E8 59 FF B8 00	φ ■ . φ. φ . 0 = ! φY φ .
000001E0	4C CD 21		L = !

Рисунок 6 – Вид COM файла в шестнадцатеричном виде

COM файл состоит из одного сегмента и содержит данные и машинные команды. размер файла не превышает 64 КБ.

Файл начинается с E9 C1 01 – jmp к адресу 0x103 + 0x1c1 (0x103 – значение IP при выполнении этой команды), с которого начинается функция

MAIN. После кода программы в файле также ничего нет. По файлу карты памяти, который выдаёт линковщик, видно, что длина файла 1E316 равна длине единственного сегмента за вычетом длины PSP (10016 байт)

Start	Stop	Length	Name	Class
-------	------	--------	------	-------

000000H	002E2H	002E3H	TEXT	
---------	--------	--------	------	--

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

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

00000000	4D 5A E3 00 03 00 00 00	20 00 00 00 FF FF 00 00	MZ.....
00000010	00 00 50 E0 00 01 00 00	1E 00 00 00 01 00 00 00	..Pα.....
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	E9 C1 01 50 43 24 50 43	2F 58 54 24 41 54 20 6F	oL.PC\$PC/XT\$AT o
00000110	72 20 50 53 2F 32 20 6D	6F 64 65 6C 20 35 30 20	r PS/2 model 50
00000120	6F 72 20 36 30 24 50 53	2F 32 20 6D 6F 64 65 6C	or 60\$PS/2 model
00000130	20 33 30 24 50 53 2F 32	20 6D 6F 64 65 6C 20 38	30\$PS/2 model 8
00000140	30 24 50 43 6A 72 24 50	43 20 43 6F 6E 76 65 72	0\$PCjr\$PC Conver
00000150	74 69 62 6C 65 24 55 6E	6B 6E 6F 77 6E 20 6D 6F	tible\$Unknown mo
00000160	64 65 6C 24 03 01 06 01	06 01 0C 01 26 01 34 01	del\$.&.4.
00000170	42 01 47 01 56 01 FF FE	FB FC FA F8 FD F9 00 0D	B.G.V. √n.°2..
00000180	0A 24 44 4F 53 20 76 65	72 73 69 6F 6E 3A 20 24	.\$DOS version: \$
00000190	4F 45 4D 20 6E 75 6D 62	65 72 3A 20 24 55 73 65	OEM number: \$Use
000001A0	72 20 73 65 72 69 61 6C	20 6E 75 6D 62 65 72 3A	r serial number:
000001B0	20 24 53 33 DB 8A 97 76	01 84 D2 74 07 3A D0 74	\$S3ëüv.ä.t.:lt
000001C0	03 43 EB F1 D1 E3 8B 97	64 01 5B C3 52 BA 7F 01	.C5±πiüd.[R o.
000001D0	B4 09 CD 21 5A C3 52 B4	09 CD 21 E8 EE FF 5A C3	+.!= Z R+.!= e Z
000001E0	24 0F 3C 09 76 02 04 07	04 30 C3 51 8A E0 E8 EF	\$.<.v....0 Qèαφn
000001F0	FF 86 C4 B1 04 D2 E8 E8	E6 FF 59 C3 53 8A FC E8	ä-_.T±μ Y Sèñ±
00000200	E9 FF 88 25 4F 88 05 4F	8A C7 E8 DE FF 88 25 4F	e è%Oè.Oè ± è%0
00000210	88 05 5B C3 51 52 32 E4	33 D2 B9 0A 00 F7 F1 80	è.[QR2z3T ...±ç
00000220	CA 30 88 14 4E 33 D2 3D	0A 00 73 F1 3C 00 74 04	±Oè.N3T...s±<.t.
00000230	0C 30 88 04 5A 59 C3 52	50 55 56 57 50 BA 82 01	.0è.ZY RPUVWP é.
00000240	B4 09 CD 21 58 83 EC 07	8B EC 50 8D 76 02 C6 44	+.!= Xâ∞.i∞Piv. D
00000250	01 24 E8 BF FF 8D 54 01	B4 09 CD 21 B4 02 B2 2E	.\$± i T .+= .±.
00000260	CD 21 58 8A C4 8D 76 02	E8 A9 FF 8D 54 01 B4 09	=!Xè-iv.φ- i T .±
00000270	CD 21 E8 57 FF BA 90 01	B4 09 CD 21 8A C7 E8 6A	=!±W É .+= ±j
00000280	FF 50 8A D0 B4 02 CD 21	58 8A D4 B4 02 CD 21 E8	Pè .+= Xè .= ±
00000290	3A FF BA 9D 01 B4 09 CD	21 8D 76 05 C6 44 01 24	: Y .+= iv. D.\$
000002A0	8B FE 8B C1 E8 55 FF 8A	C3 E8 3F FF 88 44 FB 88	i·i±U è ±? èDvè
000002B0	64 FC 8B D5 B4 09 CD 21	E8 11 FF 83 C4 07 5F 5E	dñi . .= ±. â-._^
000002C0	5D 58 5A C3 1E B8 00 F0	8E D8 BB FE FF 8A 07 1F]XZ .±.=A ±. è..
000002D0	E8 DF FE E8 00 FF B8 00	30 CD 21 E8 59 FF B8 00	±±.±. ±.0= ±Y ±.
000002E0	4C CD 21		L=!

Рисунок 7 – Вид «плохого» EXE файла

В «плохом» EXE файле данные и код содержатся в одном сегменте. Код располагается с адреса 300h. С адреса 0h располагается Relocation Table

(таблица разметки). Также 100h резервируются командой ORG 100h (что вызовет отличие в структуре «плохого» и «хорошего» EXE).

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

Вид «хорошего» EXE в шестнадцатеричном виде представлен на рис.8

без_названия- x	LAB1EXE.EXE x		
00000000	4D 5A E6 01 03 00 01 00	20 00 00 00 FF FF 00 00	MZμ.....
00000010	00 02 CA 66 12 01 2B 00	1E 00 00 00 01 00 17 01	..lf+.
00000020	2B 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
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	50 43 24 50 43 2F 58 54	24 41 54 20 6F 72 20 50	PC\$PC/XT\$AT or P
00000410	53 2F 32 20 6D 6F 64 65	6C 20 35 30 20 6F 72 20	S/2 model 50 or
00000420	36 30 24 50 53 2F 32 20	6D 6F 64 65 6C 20 33 30	60\$PS/2 model 30
00000430	24 50 53 2F 32 20 6D 6F	64 65 6C 20 38 30 24 50	\$PS/2 model 80\$P
00000440	43 6A 72 24 50 43 20 43	6F 6E 76 65 72 74 69 62	Cjr\$PC Convertib
00000450	6C 65 24 55 6E 6B 6E 6F	77 6E 20 6D 6F 64 65 6C	le\$Unknown model
00000460	24 00 00 03 00 03 00 09	00 23 00 31 00 3F 00 44	\$.#.1?.D
00000470	00 53 00 FF FE FB FC FA	F8 FD F9 00 0D 0A 24 44	..S. .√n.°²...\$D
00000480	4F 53 20 76 65 72 73 69	6F 6E 3A 20 24 4F 45 4D	OS version: \$OEM
00000490	20 6E 75 6D 62 65 72 3A	20 24 55 73 65 72 20 73	number: \$User s
000004A0	65 72 69 61 6C 20 6E 75	6D 62 65 72 3A 20 24 00	erial number: \$.
000004B0	53 33 DB 8A 97 73 00 84	D2 74 07 3A D0 74 03 43	S3ëùs.äTt.:lt.C
000004C0	EB F1 D1 E3 8B 97 61 00	5B C3 52 BA 7C 00 B4 09	5±πiüa.[R . .
000004D0	CD 21 5A C3 52 B4 09 CD	21 E8 EE FF 5A C3 24 0F	== Z R .!= æ Z \$.
000004E0	3C 09 76 02 04 07 04 30	C3 51 8A E0 E8 EF FF 86	<.v....0 Qëαφn ä
000004F0	C4 B1 04 D2 E8 E8 E6 FF	59 C3 53 8A FC E8 E9 FF	- .T±μ Y Sè"±0
00000500	88 25 4F 88 05 4F 8A C7	E8 DE FF 88 25 4F 88 05	ê%0ë.Oè ± ê%0ë.
00000510	5B C3 51 52 32 E4 33 D2	B9 0A 00 F7 F1 80 CA 30	[QR2Σ3T ...±zC 0
00000520	88 14 4E 33 D2 3D 0A 00	73 F1 3C 00 74 04 0C 30	ê.N3T=...s±<.t..0
00000530	88 04 5A 59 C3 52 50 55	56 57 50 BA 7F 00 B4 09	ê.ZY RPUVWP . . .
00000540	CD 21 58 83 EC 07 8B EC	50 8D 76 02 C6 44 01 24	==!Xâ∞.i∞Piv. D.\$
00000550	E8 BF FF 8D 54 01 B4 09	CD 21 B4 02 B2 2E CD 21	φγ iT. . = . . .==!
00000560	58 8A C4 8D 76 02 E8 A9	FF 8D 54 01 B4 09 CD 21	Xê-iv.φ- iT. . =!
00000570	E8 57 FF BA 8D 00 B4 09	CD 21 8A C7 E8 6A FF 50	φW i. . =!è ±j P
00000580	8A D0 B4 02 CD 21 58 8A	D4 B4 02 CD 21 E8 3A FF	è . =!Xè . =!±:
00000590	BA 9A 00 B4 09 CD 21 8D	76 05 C6 44 01 24 8B FE	U. . =!iv. D.\$i.
000005A0	8B C1 E8 55 FF 8A C3 E8	3F FF 88 44 FB 88 64 FC	ĩL±U è ±? èD√edⁿ
000005B0	8B D5 B4 09 CD 21 E8 11	FF 83 C4 07 5F 5E 5D 58	ĩ . =!±. â-._^]X
000005C0	5A C3 1E 33 C0 50 B8 20	00 8E D8 1E B8 00 F0 8E	Z .3Lpγ .Ä+.γ.≡Ä
000005D0	D8 BB FE FF 8A 07 1F E8	D6 FE E8 F7 FE B8 00 30	†γ. è...φγ.±±.γ.0
000005E0	CD 21 E8 50 FF CB		==!±P γ

Рисунок 8 – Вид «хорошего» EXE файла

В «хорошем» файле EXE содержится информация для загрузчика, сегмент стека, сегмент данных и сегмент кода (3 сегмента вместо одного в «плохом» EXE). Если из исходного текста EXE-программы убрать сегмент стека, то код будет располагаться с адреса 200h. Отличие от «плохого» EXE в том, что в «хорошем» не резервируется дополнительно 100h, которые в COM

файле требовались для PSP, поэтому адреса начала кода отличаются на 100h + S, где S – размер стека.

Здесь сегмент стека начинается по адресу 0x200 от начала файла, данные – 0x400 и код – 0x4B0; все адреса будут на 20016 меньше, если их брать от начала образа программы. Это видно в файле карты памяти:

```
Start Stop Length Name Class
00000H 001FFH 00200H STACK
00200H 002AEH 000AFH DATA
002B0H 003E5H 00136H TEXT
```

Загрузка COM модуля в основную память.

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

Запуск файла .COM в отладчике TD.EXE представлен на рис 9.

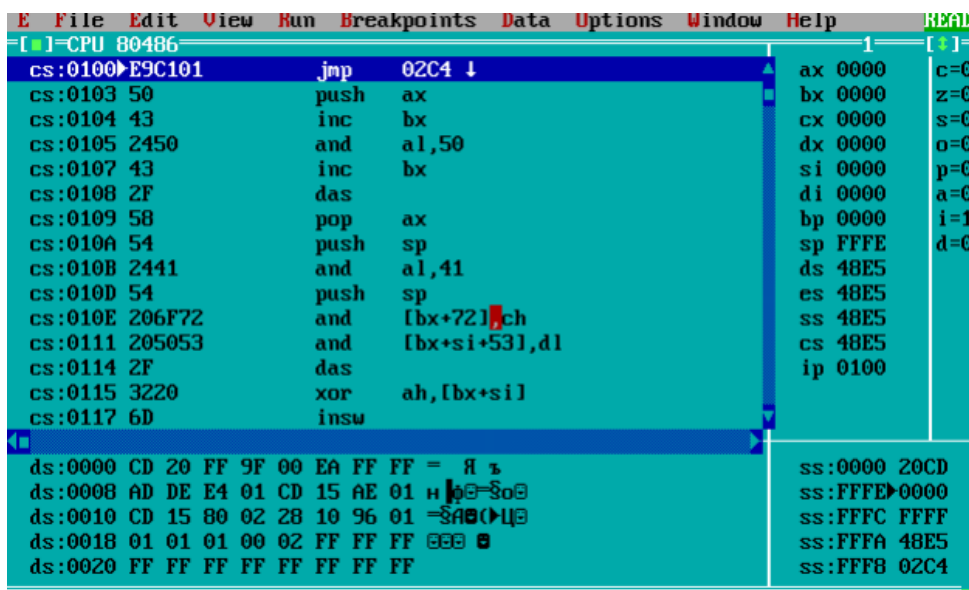


Рисунок 9 – Отладка файла .COM

Всё содержимое файла COM загружается в один и тот же сегмент по сдвигу 0x100; исполнение начинается с этого же адреса

2. Что располагается с 0 адреса?

PSP – Program Segment Prefix. Это структура, содержащая, в частности, команду int 20h (завершение программы), объём доступной памяти в

«параграфах», адреса подпрограмм, обрабатывающих некоторые события, 2 параметрические области и буфер передачи данных (DTA), который после запуска программы содержит остаток командной строки.

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

Сегментные регистры имеют значения 48DDh и указывают на PSP.

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

В COM модуле нельзя объявить стек, он создается автоматически. Указатель стека устанавливается на конец сегмента – FFFEH. Стек занимает оставшуюся память, а его адреса изменяются от больших к меньшим, то есть от FFFEH к 0000h.

Загрузка «хорошего» EXE модуля в основную память.

Запуск «хорошего» EXE модуля в отладчике TD.EXE представлен на рис.10.

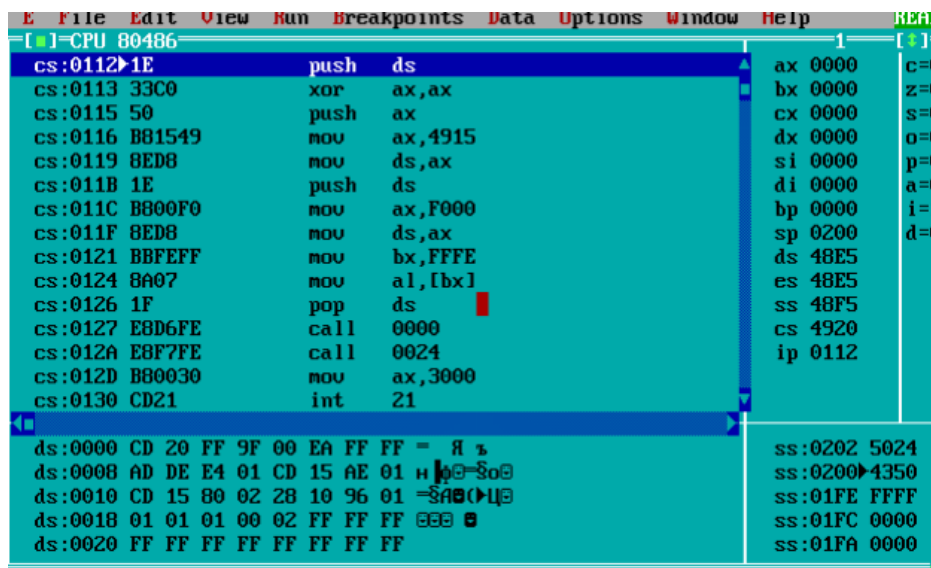


Рисунок 10 – Загрузка «хорошего» EXE модуля в память

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

В начале, модуль загружается в начальный сегмент. Потом, обрабатывается таблица настройки, т.е. изменяются адреса в программе, которые зависят от адреса начального сегмента. 8 Сегментный регистр CS

будет содержать адрес сегмента кода, регистр SS – адрес сегмента стека, а DS и ES – адрес сегмента, в котором по адресу 0 расположен PSP. Значение регистра DS, как правило, перезаписывается программой в начале выполнения адресом её сегмента данных. Перед этим его значение и слово, равное нулю добавляются в стек, чтобы из главной функции программы можно было выйти при помощи инструкции `ret far`. Это возможно, поскольку по адресу 0 в PSP лежит инструкция `int 20h`, которая завершает программу. В примере на скриншоте, образ программы загружен сразу за PSP. Начальный сегмент имеет адрес 0x48F5.

2. На что указывают регистры DS и ES?

Они указывают на сегмент, в котором расположен PSP.

3. Как определяется стек?

Стек может быть объявлен при помощи директивы `ASSUME`, которая устанавливает сегментный регистр SS на начало сегмента стека, а также задает значение SP, указанное в заголовке. Также стек может быть объявлен с помощью директивы `STACK`. Если стек не объявлять, то он будет создан автоматически.

4. Как определяется точка входа?

Смещение точки входа в программу загружается в указатель команд IP и определяется операндом (функцией или меткой, с которой необходимо начать программу) директивы `END`.

Вывод.

В ходе выполнения лабораторной работы были исследованы два формата загрузочных модулей в операционной системе DOS – COM и EXE (MZ). Исследованы различия в исходных текстах модулей, а также в их устройстве, формате и способе загрузки в память.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. LAB1COM.ASM

TEXT SEGMENT

ASSUME CS:TEXT, DS:TEXT

ORG 100h

START:

jmp MAIN

MODEL_FF	DB	'PC\$'
MODEL_FE_FB	DB	'PC/XT\$'
MODEL_FC	DB	'AT or PS/2 model 50 or 60\$'
MODEL_FA	DB	'PS/2 model 30\$'
MODEL_F8	DB	'PS/2 model 80\$'
MODEL_FD	DB	'PCjr\$'
MODEL_F9	DB	'PC Convertible\$'
MODEL_UNKNOWN	DB	'Unknown model\$'

MODEL_TAB	DW	OFFSET MODEL_FF
	DW	OFFSET MODEL_FE_FB
	DW	OFFSET MODEL_FE_FB
	DW	OFFSET MODEL_FC
	DW	OFFSET MODEL_FA
	DW	OFFSET MODEL_F8
	DW	OFFSET MODEL_FD
	DW	OFFSET MODEL_F9
	DW	OFFSET MODEL_UNKNOWN

MODEL_CHARS	DB	0FFh, 0FEh, 0FBh, 0FCh
	DB	0FAh, 0F8h, 0FDh, 0F9h, 0

NEWLINE_STRING DB 13, 10, '\$'

DOS_VERSION_STRING DB 'DOS version: \$'

```

OEM_NUMBER_STRING  DB    'OEM number: $'
USER_SERIAL_STRING DB    'User serial number: $'


LOOKUP_MODEL PROC

    push    BX
    xor BX,  BX

    ;; we don't save DX as we return into it
    anyway

    _lookup_model_loop:
        mov DL,  MODEL_CHARS[BX]

        ;; zero => unknown model
        test    DL,  DL
        jz  _lookup_model_break

        ;; model code matches => exit
        cmp DL,  AL
        jz  _lookup_model_break

        inc BX
        jmp  _lookup_model_loop

    _lookup_model_break:
        ;; we're addressing 2-byte words
        shl BX,  1
        mov DX,  MODEL_TAB[BX]

        pop BX
        ret
LOOKUP_MODEL ENDP


NEWLINE PROC

    push    DX

```

```

        mov DX,  OFFSET NEWLINE_STRING
        mov AH,  09h
        int 21h
        pop DX
        ret
NEWLINE ENDP

PRINT_MODEL PROC
        push     DX

        ;; assume the model name is in DX
        mov AH,  09h
        int 21h

        call     NEWLINE

        pop DX
        ret
PRINT_MODEL 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
                                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],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
                                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_VERSION_INFO PROC

```

```

        push    DX
        push    AX
        push    BP
        push    SI
        push    DI

        push    AX

```

```

        mov     DX,  OFFSET DOS_VERSION_STRING
        mov     AH,  09h
        int     21h

```

```

        pop     AX

```

```

;; reserver max buffer we're going to need
sub     SP,    7
mov     BP,    SP

```

```

;; print first line
push    AX

```



```

    lea SI,    [BP+2]
    mov BYTE PTR [SI+1],    '$'
    call      BYTE_TO_DEC

    lea DX,    [SI+1]
    mov AH,    09h
    int 21h

    mov AH,    02h
    mov DL,    '.'
    int 21h

    pop AX

    mov AL,    AH
    lea SI,    [BP+2]
    call      BYTE_TO_DEC

    lea DX,    [SI+1]
    mov AH,    09h
    int 21h

    call      NEWLINE

;; print second line
    mov DX,    OFFSET OEM_NUMBER_STRING
    mov AH,    09h
    int 21h

    mov AL,    BH
    call      BYTE_TO_HEX

    push      AX
    mov DL,    AL
    mov AH,    02h

```

```

int 21h

pop AX
mov DL,  AH
mov AH,  02h
int 21h

call      NEWLINE

;; print third line
mov DX,   OFFSET USER_SERIAL_STRING
mov AH,   09h
int 21h

lea SI,   [BP+5]
mov BYTE PTR [SI+1], '$'
mov DI,   SI
mov AX,   CX
call      WRD_TO_HEX

mov AL,   BL
call      BYTE_TO_HEX
mov [SI-5], AL
mov [SI-4], AH

mov DX,   BP
mov AH,   09h
int 21h

call      NEWLINE

add SP,   7

pop DI
pop SI

```

```

        pop BP
        pop AX
        pop DX
        ret
PRINT_VERSION_INFO ENDP

MAIN PROC

        ;; AX <- 0F000:0FFFEh
        push    DS
        mov AX, 0F000h
        mov DS, AX
        mov BX, 0FFFEh
        mov AL, BYTE PTR [BX]
        pop DS

        ;; DX <- model name string offset (rel. to
DS)

        call    LOOKUP_MODEL

        call    PRINT_MODEL

        ;; AL == 00h => return OEM number
        ;; AL <- DOS version major
        ;; AH <- DOS version minor
        ;; BL:CX <- `user serial number'
        ;; BH <- MS-DOS OEM number
        mov AX, 3000h
        int 21h

        call    PRINT_VERSION_INFO

        mov AX, 4C00h
        int 21h

MAIN ENDP

```

TEXT ENDS

END START

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. LAB1EXE.ASM

```
STACK SEGMENT STACK
```

```
        DW 100h DUP(?)
```

```
STACK ENDS
```

```
        ASSUME CS:TEXT, DS:DATA, SS:STACK
```

```
DATA SEGMENT
```

```
MODEL_FF      DB  'PC$'
```

```
MODEL_FE_FB    DB  'PC/XT$'
```

```
MODEL_FC      DB  'AT or PS/2 model 50 or 60$'
```

```
MODEL_FA      DB  'PS/2 model 30$'
```

```
MODEL_F8      DB  'PS/2 model 80$'
```

```
MODEL_FD      DB  'PCjr$'
```

```
MODEL_F9      DB  'PC Convertible$'
```

```
MODEL_UNKNOWN DB  'Unknown model$'
```

```
MODEL_TAB     DW  OFFSET MODEL_FF
```

```
             DW  OFFSET MODEL_FE_FB
```

```
             DW  OFFSET MODEL_FE_FB
```

```
             DW  OFFSET MODEL_FC
```

```
             DW  OFFSET MODEL_FA
```

```
             DW  OFFSET MODEL_F8
```

```
             DW  OFFSET MODEL_FD
```

```
             DW  OFFSET MODEL_F9
```

```
             DW  OFFSET MODEL_UNKNOWN
```

```
MODEL_CHARS   DB  0FFh, 0FEh, 0FBh, 0FCh
```

```
             DB  0FAh, 0F8h, 0FDh, 0F9h, 0
```

```
NEWLINE_STRING DB  13, 10, '$'
```

```

DOS_VERSION_STRING DB 'DOS version: $'
OEM_NUMBER_STRING DB 'OEM number: $'
USER_SERIAL_STRING DB 'User serial number: $'

DATA ENDS

TEXT SEGMENT

LOOKUP_MODEL PROC
    push    BX
    xor BX,  BX

    ;; we don't save DX as we return into it
    anyway

    _lookup_model_loop:
        mov DL,  MODEL_CHARS[BX]

        ;; zero => unknown model
        test    DL,  DL
        jz  _lookup_model_break

        ;; model code matches => exit
        cmp DL,  AL
        jz  _lookup_model_break

        inc BX
        jmp _lookup_model_loop

    _lookup_model_break:
        ;; we're addressing 2-byte words
        shl BX,  1
        mov DX,  MODEL_TAB[BX]

        pop BX
        ret

```

```
LOOKUP_MODEL ENDP
```

```
NEWLINE PROC
```

```
    push    DX
    mov DX,  OFFSET NEWLINE_STRING
    mov AH,  09h
    int 21h
    pop DX
    ret
```

```
NEWLINE ENDP
```

```
PRINT_MODEL PROC
```

```
    push    DX

    ;; assume the model name is in DX
    mov AH,  09h
    int 21h

    call    NEWLINE

    pop DX
    ret
```

```
PRINT_MODEL 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
        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],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
        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_VERSION_INFO PROC

```

```

                                push     DX
                                push     AX
                                push     BP
                                push     SI
                                push     DI

```

```

                                push     AX

```

```

                                mov DX,  OFFSET DOS_VERSION_STRING
                                mov AH,  09h
                                int 21h

```

```

                                pop AX

```

```

;; reserver max buffer we're going to need
sub SP,  7
mov BP,  SP

```

```

;; print first line
push     AX

lea SI,   [BP+2]
mov BYTE PTR [SI+1], '$'
call     BYTE_TO_DEC

lea DX,   [SI+1]
mov AH,   09h
int 21h

mov AH,   02h
mov DL,   '.'
int 21h

pop AX

mov AL,   AH
lea SI,   [BP+2]
call     BYTE_TO_DEC

lea DX,   [SI+1]
mov AH,   09h
int 21h

call     NEWLINE

;; print second line
mov DX,   OFFSET OEM_NUMBER_STRING
mov AH,   09h
int 21h

mov AL,   BH
call     BYTE_TO_HEX

```

```

push     AX
mov DL,  AL
mov AH,  02h
int 21h

pop AX
mov DL,  AH
mov AH,  02h
int 21h

call     NEWLINE

;; print third line
mov DX,  OFFSET USER_SERIAL_STRING
mov AH,  09h
int 21h

lea SI,  [BP+5]
mov BYTE PTR [SI+1],  '$'
mov DI,  SI
mov AX,  CX
call     WRD_TO_HEX

mov AL,  BL
call     BYTE_TO_HEX
mov [SI-5],  AL
mov [SI-4],  AH

mov DX,  BP
mov AH,  09h
int 21h

call     NEWLINE

```

```

        add SP, 7

        pop DI
        pop SI
        pop BP
        pop AX
        pop DX
        ret

PRINT_VERSION_INFO ENDP

MAIN PROC FAR

        push DS
        xor AX, AX
        push AX

        mov AX, DATA
        mov DS, AX

        ;; AX <- 0F000:0FFFEh
        push DS
        mov AX, 0F000h
        mov DS, AX
        mov BX, 0FFFEh
        mov AL, BYTE PTR [BX]
        pop DS

        ;; DX <- model name string offset (rel. to
DS)

        call LOOKUP_MODEL

        call PRINT_MODEL

        ;; AL == 00h => return OEM number
        ;; AL <- DOS version major
        ;; AH <- DOS version minor

```

```

;; BL: CX <- `user serial number'
;; BH <- MS-DOS OEM number
mov AX, 3000h
int 21h

call PRINT_VERSION_INFO

ret

MAIN ENDP

TEXT ENDS

END MAIN

```