

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

ЛАБОРАТОРНАЯ РАБОТА № 1
по дисциплине «Операционные системы»
Тема: Исследование структур загрузочных модулей.

Студент гр. 9381

Матвеев А. Н.

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.

Ход работы.

1) На основе шаблона был написан текст исходного .COM модуля, который определяет тип PC и версию системы. Были получены “хороший” .COM модуль и “плохой” .EXE модуль. Для личного удобства на этом шаге использовался компилятор TASM.

Файл lab1_com.asm -> lab1_com.obj -> lab1_com.com (хороший com-модуль)

Файл lab1_com.asm -> lab1_com.obj -> lab1_com.exe (построение плохого exe модуля)

2) Был написан код программы, получен и отлажен “хороший” .EXE модуль. (выполняет то же, что и .COM модуль). На этом шаге для удобства использовался MASM.

Файл lab1_exe.asm -> lab1_exe.obj -> lab1_exe.exe

Названия процедур	Назначение
TETR_TO_HEX	Перевод десятичной цифры в код символа.
BYTE_TO_HEX	Перевод байта в 16-ной с/с в символьный код
WRD_TO_HEX	Перевод слова в 16-ной с/с в символьный код
BYTE_TO_DEC	Перевод байта в 16-ной с/с в символьный код в 10-ной с/с
WRITE	Вывод строки.
GET_PC_TYPE	Вывод типа PC
GET_VERSION	Вывод версии, OEM и серийного номера пользователя

3) Структуры полученных загрузочных модулей были сравнены и проанализированы при помощи отладчика TD.exe и менеджера Far, сделаны необходимые выводы, даны ответы на поставленные вопросы.

Ответы на вопросы

Отличия исходных текстов COM и EXE программ

1. Сколько сегментов должна содержать COM-программа?

Один сегмент, в нём находятся данные и код.

2. EXE программа?

Программы в формате EXE могут иметь любое количество сегментов. В EXE-программе сегменты для кода, данных и стека отдельные.

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

ORG 100h устанавливает значение программного счетчика в 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. Все программы, которые компилируются в файлы типа COM, должны начинаться с этой директивы. Последствия отсутствия org 100h:

```
F:\>tasm lab1.asm
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file:   lab1.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  464k

F:\>tlink /t lab1
Turbo Link Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International
Fatal: Cannot generate COM file : invalid initial entry point address

F:\>_
```

Невозможно создать com-файл: неверный адрес начальной точки входа.

Также нужна директива **ASSUME**, когда её не было, при компиляции исходного файла были выявлены ошибки:

```
***Error** lab1.asm(137) Near jump or call to different CS
***Error** lab1.asm(140) Near jump or call to different CS
***Error** lab1.asm(143) Near jump or call to different CS
***Error** lab1.asm(146) Near jump or call to different CS
***Error** lab1.asm(149) Near jump or call to different CS
***Error** lab1.asm(152) Near jump or call to different CS
***Error** lab1.asm(157) Near jump or call to different CS
***Error** lab1.asm(164) Near jump or call to different CS
***Error** lab1.asm(185) Near jump or call to different CS
***Error** lab1.asm(188) Near jump or call to different CS
***Error** lab1.asm(195) Near jump or call to different CS
***Error** lab1.asm(199) Near jump or call to different CS
***Error** lab1.asm(206) Near jump or call to different CS
***Error** lab1.asm(210) Near jump or call to different CS
***Error** lab1.asm(213) Near jump or call to different CS
***Error** lab1.asm(216) Near jump or call to different CS
***Error** lab1.asm(226) Near jump or call to different CS
***Error** lab1.asm(227) Near jump or call to different CS
Error messages: 39
Warning messages: None
Passes: 1
Remaining memory: 464k
```

С помощью директивы **ASSUME** ассемблеру сообщается информация о соответствии между сегментными регистрами, и программными сегментами.

Для того чтобы использовать сегменты памяти как сегменты кода, данных или стека, необходимо предварительно сказать транслятору об этом, для чего используют специальную директиву **ASSUME**. Эта директива сообщает транслятору о том, какой сегмент к какому сегментному регистру привязан. Это позволяет компилятору корректно связывать имена, определенные в сегментах. Привязка сегментов к сегментным регистрам осуществляется с помощью операндов этой директивы.

Ключевое слово **NOTHING** можно использовать вместо аргумента <Сегмент>, в этом случае будет разрываться связь между сегментом с именем <Сегмент> и соответствующим сегментным регистром.

Также нужно использование директивы **END** для завершения программы.

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

Нет.

COM-программа подразумевает наличие **лишь одного сегмента**, а значит, можно использовать только near-переходы, так как в far-переходах подразумевается использование нескольких сегментов.

В COM-программах в DOS не содержится таблицы настройки, которая содержит описание адресов, зависящих от размещения загрузочного модуля в оперативной памяти, поэтому нельзя использовать команды, связанные с адресом сегмента (адрес сегмента до загрузки неизвестен). Отсюда вытекает, что нельзя использовать, например, оператор SEG NAME, дающий доступ к началу сегмента NAME.

Отличия форматов файлов COM и EXE модулей

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

.COM-файл состоит из команд, процедур и данных, используемых в программе. Код (и данные) начинается с **нулевого** адреса (смотреть скриншот ниже). Код, данные и стек располагаются в одном сегменте.

D:\OS\Far\LAB1_COM.COM																	
0000000000:	E9	EF	01	74	79	70	65	20	6F	66	20	50	43	20	2D	20	йн@type of PC -
0000000010:	24	50	43	0D	0A	24	50	43	2F	58	54	0D	0A	24	41	54	\$PC)PC/XT)AT
0000000020:	0D	0A	24	50	53	32	20	6D	6F	64	65	6C	20	33	30	0D)PS2 model 30)
0000000030:	0A	24	50	53	32	20	6D	6F	64	65	6C	20	35	30	20	6F)PS2 model 50 o
0000000040:	72	20	36	30	0D	0A	24	50	53	32	20	6D	6F	64	65	6C	r 60)PS2 model
0000000050:	20	38	30	0D	0A	24	50	43	6A	72	0D	0A	24	50	43	20	80)PCjr)PC
0000000060:	43	6F	6E	76	65	72	74	69	62	6C	65	0D	0A	24	20	20	Convertible)\$
0000000070:	20	20	20	65	72	72	6F	72	2E	20	55	6E	6B	6E	6F	77	error. Unknow
0000000080:	6E	0D	0A	24	53	79	73	74	65	6D	20	76	65	72	73	69	n)\$System versi
0000000090:	6F	6E	3A	20	20	2E	20	20	0D	0A	24	4F	45	4D	3A	20	on: .)OEM:
00000000A0:	20	20	0D	0A	24	53	65	72	69	61	6C	20	75	73	65	72)Serial user
00000000B0:	20	6E	75	6D	62	65	72	3A	20	20	20	20	20	20	20	0D	number:)
00000000C0:	0A	24	24	0F	3C	09	76	02	04	07	04	30	C3	51	8A	E0)\$<ov@•••GQ)а
00000000D0:	E8	EF	FF	86	C4	B1	04	D2	E8	E8	E6	FF	59	C3	53	8A	ипя†Д±•ТиижяYGS)ь
00000000E0:	FC	E8	E9	FF	88	25	4F	88	05	4F	8A	C7	E8	DE	FF	88	ыйя€%O€•O)зиюя€
00000000F0:	25	4F	88	05	5B	C3	51	52	32	E4	33	D2	B9	0A	00	F7	%O€+[ГQR2дЗТ№)ч
0000000100:	F1	80	CA	30	88	14	4E	33	D2	3D	0A	00	73	F1	3C	00	сЪK0€)N3T=)sc<
0000000110:	74	04	0C	30	88	04	5A	59	C3	50	B4	09	CD	21	58	C3	t•Q0€•ZYГPгoH!XГ
0000000120:	50	52	06	B8	00	F0	8E	C0	26	A0	FE	FF	BA	03	01	E8	PR•ё рЪA& юяє♥@и
0000000130:	E7	FF	3C	FF	74	23	3C	FE	74	25	3C	FB	74	21	3C	FC	зя<ят#<ют%<ыт!<ь
0000000140:	74	23	3C	FA	74	25	3C	FC	74	27	3C	F8	74	29	3C	FD	t#<ьт%<ьт'<шт)<э
0000000150:	74	2B	3C	F9	74	2D	EB	31	90	BA	11	01	EB	3A	90	BA	t+<щт-л1)є◀@л:)є
0000000160:	16	01	EB	34	90	BA	1E	01	EB	2E	90	BA	23	01	EB	28	◀@л4)є▲@л.)є#@л(
0000000170:	90	BA	32	01	EB	22	90	BA	47	01	EB	1C	90	BA	56	01)є2@л")єG@лL)єV@
0000000180:	EB	16	90	BA	5D	01	EB	10	90	BA	6E	01	50	E8	3D	FF	л-)є]@л▶)єп@Ри=я
0000000190:	8B	F2	88	04	46	88	24	58	E8	7E	FF	07	5A	58	C3	50	<т€•F€\$Хи~я•ZXГP
00000001A0:	52	B4	30	CD	21	50	56	BE	84	01	83	C6	10	E8	46	FF	Rг0H!PVs,,@гЖ▶иFя
00000001B0:	83	C6	03	8A	C4	E8	3E	FF	5E	58	8A	C7	BE	9B	01	83	гЖ♥)ди>я^Х)зs>@г
00000001C0:	C6	07	E8	31	FF	8A	C3	E8	03	FF	BF	A5	01	83	C7	14	Ж•и1я)Ги♥яїГ@гЗ)Г
00000001D0:	89	05	8B	C1	BF	A5	01	83	C7	19	E8	01	FF	BA	84	01	‰+<БїГ@гЗ↓и@яє,,@
00000001E0:	E8	36	FF	BA	9B	01	E8	30	FF	BA	A5	01	E8	2A	FF	5A	ибяє>@и@яєГ@и*яZ
00000001F0:	58	C3	E8	2B	FF	E8	A7	FF	32	C0	B4	4C	CD	21			XГи+яи\$я2AгLH!

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

Что располагается с 0 адреса?

Рассмотрим скриншоты:

D:\OS\Far\LAB1_COM.EXE																		
0000000000:	4D	5A	FE	00	03	00	00	00	20	00	00	00	FF	FF	00	00	MZю ♥	яя
0000000010:	00	00	00	00	00	01	00	00	3E	00	00	00	01	00	FB	71	⊙ >	⊙ ыq
0000000020:	6A	72	00	00	00	00	00	00	00	00	00	00	00	00	00	00	jr	
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		
1Help2Text3Quit4Dump56Edit7S																		

00000000290:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000002A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000002B0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000002C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000002D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000002E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000002F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000300:	E9 EF 01 74 79 70 65 20	6F 66 20 50 43 20 2D 20	йп@type of PC -
00000000310:	24 50 43 0D 0A 24 50 43	2F 58 54 0D 0A 24 41 54	\$PC!\$PC/XT!\$AT
00000000320:	0D 0A 24 50 53 32 20 6D	6F 64 65 6C 20 33 30 0D	!\$PS2 model 30!
00000000330:	0A 24 50 53 32 20 6D 6F	64 65 6C 20 35 30 20 6F	\$PS2 model 50 o
00000000340:	72 20 36 30 0D 0A 24 50	53 32 20 6D 6F 64 65 6C	r 60!\$PS2 model
00000000350:	20 38 30 0D 0A 24 50 43	6A 72 0D 0A 24 50 43 20	80!\$PCjr!\$PC
00000000360:	43 6F 6E 76 65 72 74 69	62 6C 65 0D 0A 24 20 20	Convertible!\$
00000000370:	20 20 20 65 72 72 6F 72	2E 20 55 6E 6B 6E 6F 77	error. Unknow
00000000380:	6E 0D 0A 24 53 79 73 74	65 6D 20 76 65 72 73 69	n!\$System versi
00000000390:	6F 6E 3A 20 20 2E 20 20	0D 0A 24 4F 45 4D 3A 20	on: . !\$OEM:
000000003A0:	20 20 0D 0A 24 53 65 72	69 61 6C 20 75 73 65 72	!\$Serial user
000000003B0:	20 6E 75 6D 62 65 72 3A	20 20 20 20 20 20 20 0D	number: !
000000003C0:	0A 24 24 0F 3C 09 76 02	04 07 04 30 C3 51 8A E0	\$ \$<ov0♦♦0ГQЉа
000000003D0:	E8 EF FF 86 C4 B1 04 D2	E8 E8 E6 FF 59 C3 53 8A	ипя†Д±♦ТиижяҮГSЉ
000000003E0:	FC E8 E9 FF 88 25 4F 88	05 4F 8A C7 E8 DE FF 88	ыйя€%O€+OЉ3иЮя€
000000003F0:	25 4F 88 05 5B C3 51 52	32 E4 33 D2 B9 0A 00 F7	%O€+[ГQR2д3ТЉ ч
00000000400:	F1 80 CA 30 88 14 4E 33	D2 3D 0A 00 73 F1 3C 00	сЉK0€ЉN3Т=Љ sс<
00000000410:	74 04 0C 30 88 04 5A 59	C3 50 B4 09 CD 21 58 C3	t♦Q0€♦ZYГР҃оH!XГ
00000000420:	50 52 06 B8 00 F0 8E C0	26 A0 FE FF BA 03 01 E8	PR♦è рЉA& юя€♥@и
00000000430:	E7 FF 3C FF 74 23 3C FE	74 25 3C FB 74 21 3C FC	зя<ят#<ют%<йт!<ь
00000000440:	74 23 3C FA 74 25 3C FC	74 27 3C F8 74 29 3C FD	т#<ът%<ът'<шт)<э
00000000450:	74 2B 3C F9 74 2D EB 31	90 BA 11 01 EB 3A 90 BA	т+<шт-л1ћє◀@л:ћє
00000000460:	16 01 EB 34 90 BA 1E 01	EB 2E 90 BA 23 01 EB 28	¬@л4ћє▲@л.ћє#@л(
00000000470:	90 BA 32 01 EB 22 90 BA	47 01 EB 1C 90 BA 56 01	ћє2@л"ћєG@лLћєV@
00000000480:	EB 16 90 BA 5D 01 EB 10	90 BA 6E 01 50 E8 3D FF	л¬ћє]@л¬ћєп@ри=я
00000000490:	8B F2 88 04 46 88 24 58	E8 7E FF 07 5A 58 C3 50	<т€♦F€\$Хи¬¬я•ZXГР
000000004A0:	52 B4 30 CD 21 50 56 BE	84 01 83 C6 10 E8 46 FF	RГ0H!PVs,,@ГЖ►иFя
000000004B0:	83 C6 03 8A C4 E8 3E FF	5E 58 8A C7 BE 9B 01 83	ГЖ♥Љди>я^ХЉ3s>@Г
000000004C0:	C6 07 E8 31 FF 8A C3 E8	03 FF BF A5 01 83 C7 14	Ж•и1яЉГи♥яіГ@Г3Љ
000000004D0:	89 05 8B C1 BF A5 01 83	C7 19 E8 01 FF BA 84 01	§♦<БіГ@Г3↓и@я€,@
000000004E0:	E8 36 FF BA 9B 01 E8 30	FF BA A5 01 E8 2A FF 5A	иБя€>@и@я€Г@и*яZ
000000004F0:	58 C3 E8 2B FF E8 A7 FF	32 C0 B4 4C CD 21	XГи+яи\$я2AГLH!

Отсюда видно, что:

- данные и код расположены в одном сегменте.
- Код (и данные) начинается с адреса 300h.
- С адреса 0h располагается заголовок и таблица настройки адресов.

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

D:\OS\Far\LAB1_EXE.EXE															
00000000:	4D	5A	17	00	03	00	01	00	20	00	11	00	FF	FF	22 00
00000001:	00	01	FF	0C	10	00	00	00	1E	00	00	00	01	00	48 01
00000002:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000003:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000004:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000005:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000006:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000007:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000008:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000009:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000A:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000B:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000C:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000D:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000E:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000F:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000010:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000011:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000012:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000013:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000014:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000015:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000016:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000017:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000018:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000019:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000001A:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000001B:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000001C:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000001D:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000001E:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000001F:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000020:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

```

00000000210: E9 34 01 24 0F 3C 09 76   02 04 07 04 30 C3 51 8A   й40$о<ov0♦♦♦0ГQЪ
00000000220: E0 E8 EF FF 86 C4 B1 04   D2 E8 E8 E6 FF 59 C3 53   аипя†Д±♦ТиижяYГS
00000000230: 8A FC E8 E9 FF 88 25 4F   88 05 4F 8A C7 E8 DE FF   льийя€%O€+Oль3июя
00000000240: 88 25 4F 88 05 5B C3 51   52 32 E4 33 D2 B9 0A 00   €%O€+[ГQR2д3Т№
00000000250: F7 F1 80 CA 30 88 14 4E   33 D2 3D 0A 00 73 F1 3C   чсЪK0€JN3T= sc<
00000000260: 00 74 04 0C 30 88 04 5A   59 C3 50 B4 09 CD 21 58   т♦Q0€♦ZYГPгoH!X
00000000270: C3 50 52 06 B8 00 F0 8E   C0 26 A0 FE FF BA 08 00   ГPR♦ё рЪA& юяе
00000000280: E8 E7 FF 3C FF 74 23 3C   FE 74 25 3C FB 74 21 3C   изя<ят#<ют%<ыт!<
00000000290: FC 74 23 3C FA 74 25 3C   FC 74 27 3C F8 74 29 3C   ьт#<ьт%<ьт'<шт)<
000000002A0: FD 74 2B 3C F9 74 2D EB   31 90 BA 16 00 EB 3A 90   эт+<шт-л1ћє= л:ћ
000000002B0: BA 1B 00 EB 34 90 BA 23   00 EB 2E 90 BA 28 00 EB   є← л4ћє# л.ћє( л
000000002C0: 28 90 BA 37 00 EB 22 90   BA 4C 00 EB 1C 90 BA 5B   (ћє7 л"ћєL ллћє[
000000002D0: 00 EB 16 90 BA 62 00 EB   10 90 BA 73 00 50 E8 3D   л-ћєb л-ћєs Ри=
000000002E0: FF 8B F2 88 04 46 88 24   58 E8 7E FF 07 5A 58 C3   я<т€♦F€$Хи~я•ZXГ
000000002F0: 50 52 B4 30 CD 21 50 56   8D 36 89 00 83 C6 10 E8   PRГ0H!PVК6% фЖ>и
00000000300: 45 FF 83 C6 03 8A C4 E8   3D FF 5E 58 8A C7 8D 36   ЕяГЖ♥льДи=я^ХльЗК6
00000000310: A0 00 83 C6 07 E8 2F FF   8A C3 E8 01 FF 8D 3E AA   фЖ•и/яльГиояК>€
00000000320: 00 83 C7 14 89 05 8B C1   8D 3E AA 00 83 C7 19 E8   фЗГ%+<БК>€ фЗ↓и
00000000330: FD FE BA 89 00 E8 32 FF   BA A0 00 E8 2C FF BA AA   эю€% и2яє и,яє€
00000000340: 00 E8 26 FF 5A 58 C3 B8   15 00 8E D8 E8 22 FF E8   и&яZXГё$ Ѓши"яи
00000000350: 9E FF 32 C0 B4 4C CD 21   74 79 70 65 20 6F 66 20   ћя2AГLH!type of
00000000360: 50 43 20 2D 20 24 50 43   0D 0A 24 50 43 2F 58 54   PC - $PC!$PC/XT
00000000370: 0D 0A 24 41 54 0D 0A 24   50 53 32 20 6D 6F 64 65   !$AT!$PS2 mode
00000000380: 6C 20 33 30 0D 0A 24 50   53 32 20 6D 6F 64 65 6C   l 30!$PS2 model
00000000390: 20 35 30 20 6F 72 20 36   30 0D 0A 24 50 53 32 20   50 or 60!$PS2
000000003A0: 6D 6F 64 65 6C 20 38 30   0D 0A 24 50 43 6A 72 0D   model 80!$PCjr!
000000003B0: 0A 24 50 43 20 43 6F 6E   76 65 72 74 69 62 6C 65   $PC Convertible
000000003C0: 0D 0A 24 20 20 20 20 20   65 72 72 6F 72 2E 20 55   !$ error. U
000000003D0: 6E 6B 6E 6F 77 6E 0D 0A   24 53 79 73 74 65 6D 20   nknown!$System
000000003E0: 76 65 72 73 69 6F 6E 3A   20 20 2E 20 20 0D 0A 24   version: . !$
000000003F0: 4F 45 4D 3A 20 20 20 0D   0A 24 53 65 72 69 61 6C   OEM: !$Serial
00000000400: 20 75 73 65 72 20 6E 75   6D 62 65 72 3A 20 20 20   user number:
00000000410: 20 20 20 20 0D 0A 24      !$

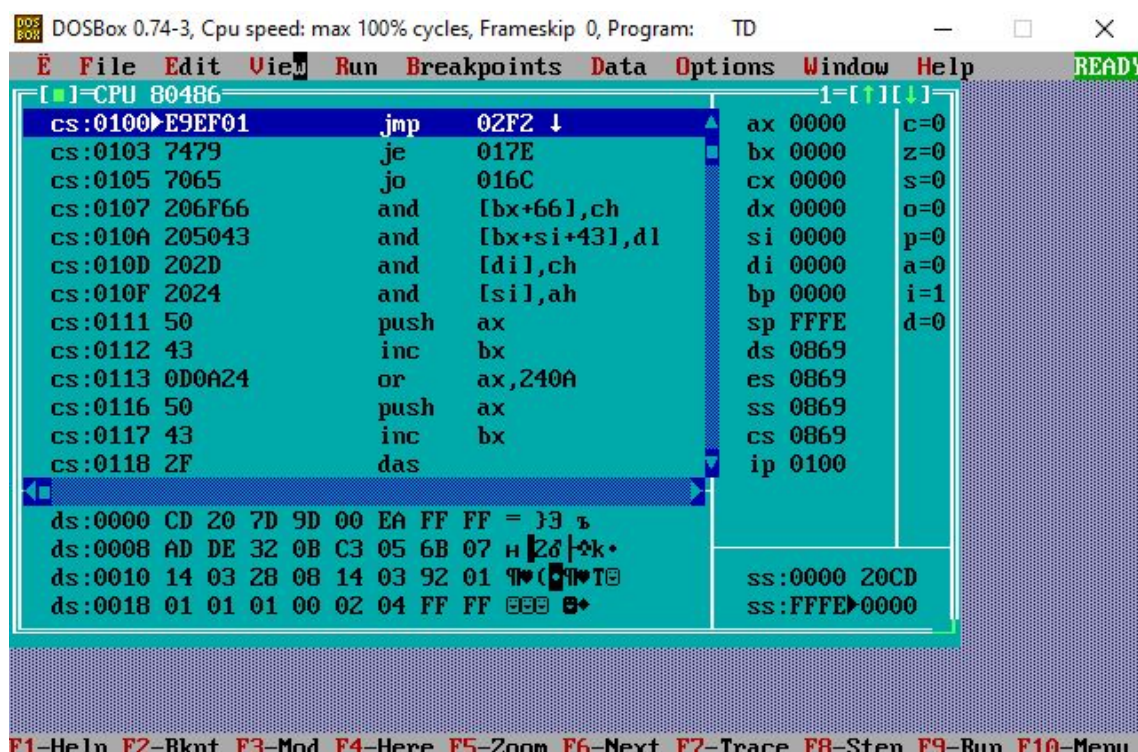
```

Структура хорошего EXE: стек, дата, код – отдельные сегменты, в отличие от плохого, в котором один они объединены в один сегмент.

Хороший EXE-файл, в отличие от плохого, не содержит директивы ORG 100h, выделяющей память под PSP.

Загрузка COM модуля в основную память

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



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TD
E File Edit View Run Breakpoints Data Options Window Help
[CPU 80486]
cs:0100 E9EF01 jmp 02F2
cs:0103 7479 je 017E
cs:0105 7065 jo 016C
cs:0107 206F66 and [bx+66],ch
cs:010A 205043 and [bx+si+43],dl
cs:010D 202D and [di],ch
cs:010F 2024 and [si],ah
cs:0111 50 push ax
cs:0112 43 inc bx
cs:0113 0D0A24 or ax,240A
cs:0116 50 push ax
cs:0117 43 inc bx
cs:0118 2F das
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 FFFE d=0
ds 0869
es 0869
ss 0869
cs 0869
ip 0100
ds:0000 CD 20 7D 9D 00 EA FF FF = }3 ь
ds:0008 AD DE 32 0B C3 05 6B 07 H 2d k+
ds:0010 14 03 28 08 14 03 92 01 70 C 70 T
ds:0018 01 01 01 00 02 04 FF FF 88 8+
ss:0000 20CD
ss:FFFE 0000
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu
```

При загрузке COM-файла в память DOS занимает первые 256 байт (100h) блоком данных PSP и располагает код программы только после этого блока. Поэтому код располагается с адреса 100h. После загрузки COM-программы в память сегментные регистры указывают на начало PSP.

2. Что располагается с 0 адреса?

С адреса 0 располагается префикс программного сегмента (PSP).

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

Сегментные регистры имеют значения, соответствующие началу сегмента, в который модуль был помещен управляющей программой. Все сегментные регистры имеют значения 0869, т.к. указывают на один и тот же сегмент памяти - PSP.

```
ds 0869
es 0869
ss 0869
cs 0869
```

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

Стек создается автоматически.

Регистр ss (сегментный регистр стека) указывает на начало PSP (0h).

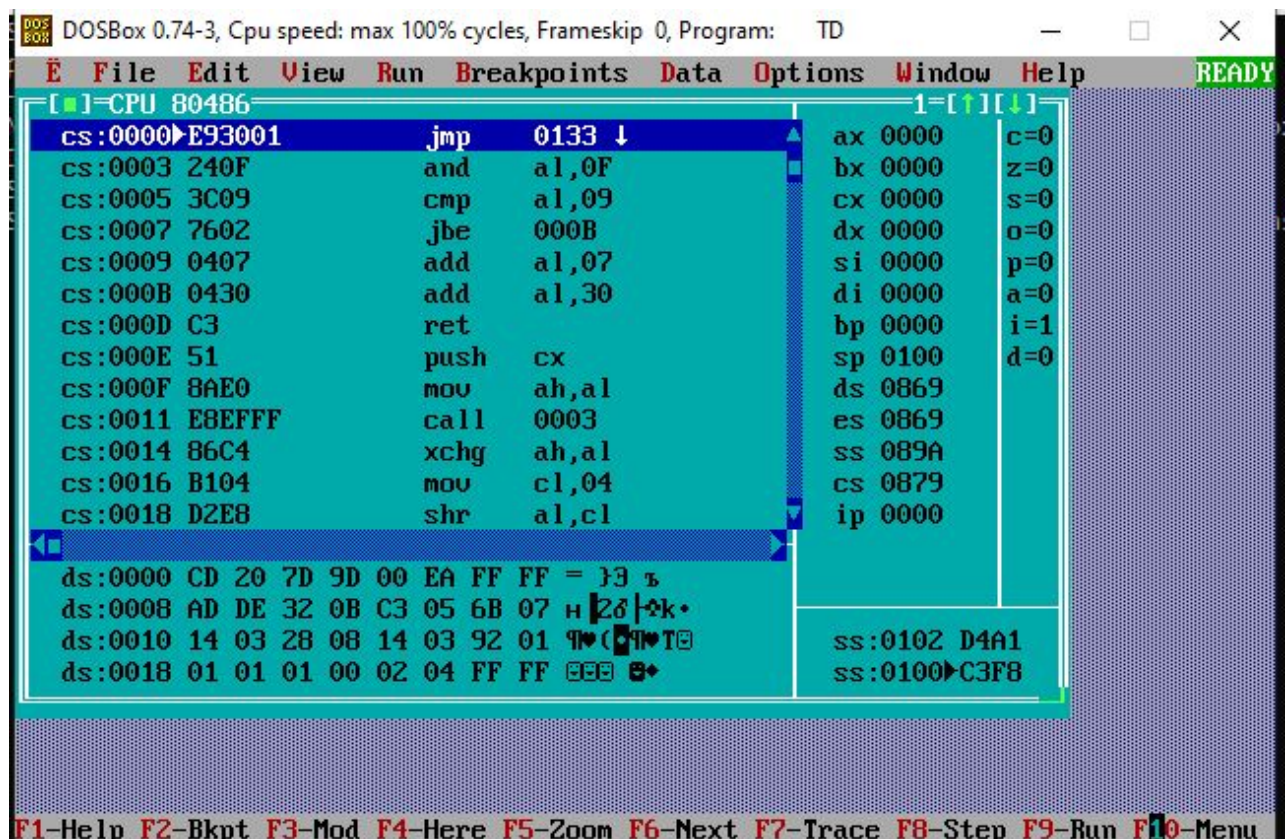
Регистр sp указывает на вершину стека – FFFh. Он занимает оставшуюся память, адреса изменяются от больших к меньшим, то есть от FFFh к 000h.

План загрузки модуля .COM в основную память.

- 1) В основной памяти выделяется свободный сегмент.
- 2) Первые 100h байт уходят под PSP, а в оставшаяся область выделенного сегмента под саму программу.

Загрузка «хорошего» EXE модуля в память

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



«хороший» EXE загружается следующим образом:

- В области памяти после резидентной части выполняющей загрузку программы строится PSP. Префикс Программного сегмента строится Загрузчиком и содержит системную информацию, необходимую программе;
- Стандартная часть таблицы настроек считывается в рабочую область Загрузчика.

- Определяется длина тела загрузочного модуля
- В зависимости от признака, указывающего загружать задачу в конец памяти или в начало, определяется сегментный адрес для загрузки. Этот сегмент называется начальным сегментом.
- Загрузочный модуль считывается в начальный сегмент.
- Таблица настройки после стандартной части порциями считывается в рабочую память.
- Для каждого элемента таблицы настройки к полю сегмента прибавляется сегментный адрес начального сегмента. В результате элемент таблицы указывает на нужное слово в памяти; к этому слову прибавляется сегментный адрес начального сегмента.
- Когда таблица настройки адресов обработана, регистрам SS и SP придаются значения, указанные в заголовке
- к SS прибавляется сегментный адрес начального сегмента.
- В ES и DS засылается сегментный адрес начала PSP.
- Управление передается загруженной задаче по адресу, указанному в заголовке

DS и ES указывают на начало PSP (0869),

CS – на начало сегмента кода (0879),

SS – на начало сегмента стека (089A).

ds	0869
es	0869
ss	089A
cs	0879

2. На что указывают регистры DS и ES?

Изначально регистры DS и ES указывают на начало сегмента PSP.

3. Как определяется стек?

Стек определяется при объявлении сегмента стека, в котором указывается, сколько памяти необходимо выделить. В регистры SS и SP записываются значения, указанные в заголовке, а к SS прибавляется сегментный адрес начального сегмента. Регистр SS указывает на начало сегмента стека, а регистр SP – на конец этого сегмента.

4. Как определяется точка входа?

При помощи директивы END, операндом которой является адрес, с которого начинается выполнение программы.

Заключение.

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

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

lab1_com.asm:

TESTPC SEGMENT

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

ORG 100H

START: JMP BEGIN

;ДАННЫЕ

pct_type_text DB 'type of PC - ', '\$'

pct_1 DB 'PC', 0DH, 0AH, '\$'

pct_2 DB 'PC/XT', 0DH, 0AH, '\$'

pct_3 DB 'AT', 0DH, 0AH, '\$'

pct_4 DB 'PS2 model 30', 0DH, 0AH, '\$'

pct_5 DB 'PS2 model 50 or 60', 0DH, 0AH, '\$'

pct_6 DB 'PS2 model 80', 0DH, 0AH, '\$'

pct_7 DB 'PCjr', 0DH, 0AH, '\$'

pct_8 DB 'PC Convertible', 0DH, 0AH, '\$'

pct_unknown DB ' error. Unknown', 0DH, 0AH, '\$'

System_version DB 'System version: . ', 0DH, 0AH, '\$'

OEM DB 'OEM: ', 0DH, 0AH, '\$'

user_number DB 'Serial user number: ', 0DH, 0AH, '\$'

;процедуры

TETR_TO_HEX PROC near ;представляет 4 младших бита al в виде цифры
16-ой с.сч. и представляет её в символьном виде

and AL,0Fh

cmp AL,09

jbe NEXT

add AL,07

NEXT:

add AL,30h ; результат в al

ret

TETR_TO_HEX ENDP

;-----

BYTE_TO_HEX PROC near

; байт в al переводится в 2 символа шест. числа в AX

```

    push CX
    mov AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX ;в AL старшая цифра
    pop CX ;в AH младшая
    ret
BYTE_TO_HEX ENDP
;-----
WRD_TO_HEX PROC near
; перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
    push BX
    mov BH,AH
    call BYTE_TO_HEX
    mov [DI],AH
    dec DI
    mov [DI],AL
    dec DI
    mov AL,BH
    call BYTE_TO_HEX
    mov [DI],AH
    dec DI
    mov [DI],AL
    pop BX
    ret
WRD_TO_HEX ENDP
;-----
BYTE_TO_DEC PROC near
; перевод в 10 с/с. SI - адрес поля младшей цифры
    push CX
    push DX
    xor AH,AH
    xor DX,DX
    mov CX,10

```



```

loop_bd: div CX
        or DL,30h
        mov [SI],DL
        dec SI
        xor DX,DX
        cmp AX,10
        jae loop_bd
        cmp AL,00h
        je end_1
        or AL,30h
        mov [SI],AL

end_1:
        pop DX
        pop CX
        ret

BYTE_TO_DEC ENDP

WRITE PROC NEAR
        push ax
        mov  AH, 9
        int  21h ; Вызов функции DOS по прерыванию
        pop ax
        ret

WRITE ENDP

GET_PC_TYPE PROC NEAR ; получение типа PC
        push ax
        push dx
        push es
        mov ax, 0F000h
        mov es, ax
        mov al, es:[0FFFEh]
        mov dx, offset pc_type_text
        call WRITE

        cmp al, 0FFh ;распознавание типа PC по специальной таблице
        je pct_1_case
        cmp al, 0FEh

```

```

        je pct_2_case
        cmp al, 0FBh
        je pct_2_case
        cmp al, 0FCh
        je pct_3_case
        cmp al, 0FAh
        je pct_4_case
        cmp al, 0FCh
        je pct_5_case
        cmp al, 0F8h
        je pct_6_case
        cmp al, 0FDh
        je pct_7_case
        cmp al, 0F9h
        je pct_8_case
        jmp pct_unknown_case

pct_1_case:
        mov dx, offset pct_1 ; загрузка в зависимости от значения al
        смещения нужной строки
        jmp final_step
pct_2_case:
        mov dx, offset pct_2
        jmp final_step
pct_3_case:
        mov dx, offset pct_3
        jmp final_step
pct_4_case:
        mov dx, offset pct_4
        jmp final_step
pct_5_case:
        mov dx, offset pct_5
        jmp final_step
pct_6_case:
        mov dx, offset pct_6
        jmp final_step
pct_7_case:
        mov dx, offset pct_7

```

```

        jmp final_step
pct_8_case:
        mov dx, offset pct_8
        jmp final_step

pct_unknown_case: ; в случае несоответствия ни одному элементу таблицы
вывод сообщения о неизвестном типе
        mov dx, offset pct_unknown
        push ax
        call BYTE_TO_HEX
        mov si, dx
        mov [si], al
        inc si
        mov [si], ah ; в начало сообщения записывается код "ошибки" в
16-ричном виде
        pop ax
final_step:
        call write
end_of_proc:
        pop es
        pop dx
        pop ax
        ret
GET_PC_TYPE ENDP

GET_VERSION PROC NEAR
        push ax
        push dx
        MOV AH, 30h
        INT 21h
        ;write version number

;Сначала надо обработать al, а потом ah и записать в конец
System_version
        push ax
        push si
        lea si, System_version
        add si, 16

```

```

        call BYTE_TO_DEC
        add si, 3
        mov al, ah
        call BYTE_TO_DEC
        pop ax
        pop si
;OEM
        mov al, bh
        lea si, OEM
        add si, 7
        call BYTE_TO_DEC

;get_user_number
        mov al, bl
        call BYTE_TO_HEX ;
        lea di, user_number
        add di, 20
        mov [di], ax
        mov ax, cx
        lea di, user_number
        add di, 25
        call WRD_TO_HEX

version_:
        mov dx, offset System_version
        call WRITE
get_OEM:
        mov dx, offset OEM
        call write
get_user_number:
        mov dx, offset user_number
        call write
end_of_proc_2:
        pop dx
        pop ax
        ret
GET_VERSRION ENDP
;-----

```



```

;КОД
BEGIN: ;ВЫЗОВ ОСНОВНЫХ ПРОЦЕДУР
call GET_PC_TYPE
call GET_VERSION
;ВЫХОД В ДОС
    xor AL,AL
    mov AH,4Ch
    int 21H

TESTPC ENDS
    END START
end
; КОНЕЦ МОДУЛЯ, START - ТОЧКА ВХОДА

```

lab1_exe.asm:

```

DOSSEG
.model small
.stack 100h

;ДАННЫЕ
.data
pc_type_text DB 'type of PC - ', '$'
pct_1 DB 'PC', 0DH, 0AH, '$'
pct_2 DB 'PC/XT', 0DH, 0AH, '$'
pct_3 DB 'AT', 0DH, 0AH, '$'
pct_4 DB 'PS2 model 30', 0DH, 0AH, '$'
pct_5 DB 'PS2 model 50 or 60', 0DH, 0AH, '$'
pct_6 DB 'PS2 model 80', 0DH, 0AH, '$'
pct_7 DB 'PCjr', 0DH, 0AH, '$'
pct_8 DB 'PC Convertible', 0DH, 0AH, '$'
pct_unknown DB '      error. Unknown', 0DH, 0AH, '$'
System_version DB 'System version:  . ', 0DH, 0AH, '$'
OEM DB 'OEM: ', 0DH, 0AH, '$'
user_number DB 'Serial user number:      ', 0DH, 0AH, '$'

```

```

.code
START:
    jmp BEGIN
;процедуры
TETR_TO_HEX PROC near ;представляет 4 младших бита al в виде цифры
16-ой с.сч. и представляет её в символьном виде
    and AL,0Fh
    cmp AL,09
    jbe NEXT
    add AL,07
NEXT:
    add AL,30h ; результат в al
    ret
TETR_TO_HEX ENDP
;-----

BYTE_TO_HEX PROC near
; байт в al переводится в 2 символа шест. числа в AX
    push CX
    mov AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX ;в AL старшая цифра
    pop CX ;в AH младшая
    ret
BYTE_TO_HEX ENDP
;-----

WRD_TO_HEX PROC near
; перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
    push BX

```

```

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

;-----
BYTE_TO_DEC PROC near
; перевод в 10 с/с. SI - адрес поля младшей цифры
    push CX
    push DX
    xor AH,AH
    xor DX,DX
    mov CX,10
loop_bd: div CX
            or DL,30h
            mov [SI],DL
            dec SI
            xor DX,DX
            cmp AX,10
            jae loop_bd
            cmp AL,00h
            je end_1
            or AL,30h
            mov [SI],AL
end_1:

```

```

        pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP

```

```

WRITE PROC NEAR
    push ax
    mov  AH, 9
    int  21h ; Вызов функции DOS по прерыванию
    pop ax
    ret
WRITE ENDP

```

```

GET_PC_TYPE PROC NEAR
    push ax
    push dx
    push es
    mov ax, 0F000h
    mov es, ax
    mov al, es:[0FFFEh]
    mov dx, offset pc_type_text
    call WRITE

    cmp al, 0FFh
    je pct_1_case
    cmp al, 0FEh
    je pct_2_case
    cmp al, 0FBh
    je pct_2_case
    cmp al, 0FCh
    je pct_3_case
    cmp al, 0FAh
    je pct_4_case
    cmp al, 0FCh
    je pct_5_case

```

```

        cmp al, 0F8h
        je pct_6_case
        cmp al, 0FDh
        je pct_7_case
        cmp al, 0F9h
        je pct_8_case
        jmp pct_unknown_case

pct_1_case:
        mov dx, offset pct_1
        jmp final_step
pct_2_case:
        mov dx, offset pct_2
        jmp final_step
pct_3_case:
        mov dx, offset pct_3
        jmp final_step
pct_4_case:
        mov dx, offset pct_4
        jmp final_step
pct_5_case:
        mov dx, offset pct_5
        jmp final_step
pct_6_case:
        mov dx, offset pct_6
        jmp final_step
pct_7_case:
        mov dx, offset pct_7
        jmp final_step
pct_8_case:
        mov dx, offset pct_8
        jmp final_step

```

pct_unknown_case:; в случае несоответствия ни одному элементу
таблицы вывод сообщения о неизвестном типе

```

        mov dx, offset pct_unknown
        push ax
        call BYTE_TO_HEX
        mov si, dx
        mov [si], al
        inc si
        mov [si], ah
        pop ax
final_step:
        call write
end_of_proc:
        pop es
        pop dx
        pop ax
        ret
GET_PC_TYPE ENDP

```

```

GET_VERSION PROC NEAR

```

```

        push ax
        push dx
        MOV AH,30h
        INT 21h
        ;write version number

```

```

;Сначала надо обработать al, а потом ah и записать в конец
System_version

```

```

        push ax
        push si
        lea si, System_version
        add si, 16
        call BYTE_TO_DEC
        add si, 3
        mov al, ah
        call BYTE_TO_DEC
        pop si

```



```

        pop ax
;OEM
        mov al, bh
        lea si, OEM
        add si, 7
        call BYTE_TO_DEC

;get_user_number
        mov al, bl
        call BYTE_TO_HEX
        lea di, user_number
        add di, 20
        mov [di], ax
        mov ax, cx
        lea di, user_number
        add di, 25
        call WRD_TO_HEX

version_:
        mov dx, offset System_version
        call WRITE
get_OEM:
        mov dx, offset OEM
        call write
get_user_number:
        mov dx, offset user_number
        call write
end_of_proc_2:
        pop dx
        pop ax
        ret
GET_VERSION ENDP
;-----

```

```
BEGIN:
    mov ax, @data
    mov ds, ax
    call GET_PC_TYPE
    call GET_VERSION
    ;выход в ДОС
    xor AL,AL
    mov AH,4Ch
    int 21H
END START
; КОНЕЦ МОДУЛЯ
```