

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

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

Студент гр. 9381

Гурин С.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.

Порядок выполнения работы.

Шаг 1. Напишите текст исходного **.COM** модуля, который определяет тип РС и версию системы. Это довольно простая задача и для тех, кто уже имеет опыт программирования на ассемблере, это будет небольшой разминкой. Для тех, кто раньше не сталкивался с программированием на ассемблере, это неплохая задача для первого опыта.

За основу возьмите шаблон, приведенный в разделе «Основные сведения». Необходимые сведения о том, как извлечь требуемую информацию, представлены в следующем разделе.

Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип РС и выводить строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код переводиться в символьную строку, содержащую запись шестнадцатеричного числа и выводиться на экран в виде соответствующего сообщения.

Затем определяется версия системы. Ассемблерная программа должна по значениям регистров AL и AH формировать текстовую строку в формате xx.yy, где xx – номер основной версии, а yy - номер модификации в десятичной системе счисления, формировать строки с серийным номером OEM и серийным номером пользователя. Полученные строки выводятся на экран.

Отладьте полученный исходный модуль.

Результатом выполнения этого шага будет «хороший» **.COM** модуль, а также необходимо построить «плохой» **.EXE**, полученный из исходного текста для **.COM** модуля.

Шаг 2. Напишите текст исходного **.EXE** модуля, который выполняет те же функции, что и модуль в Шаге 1 и постройте и отладьте его. Таким образом, будет получен «хороший» **.EXE**.

Шаг 3. Сравните исходные тексты для **.COM** и **.EXE** модулей. Ответьте на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

Шаг 4. Запустите FAR и откройте (F3/F4) файл загрузочного модуля **.COM** и файл «плохого» **.EXE** в шестнадцатеричном виде. Затем откройте (F3/F4) файл загрузочного модуля «хорошего» **.EXE** и сравните его с предыдущими файлами. Ответьте на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

Шаг 5. Откройте отладчик **TD.EXE** и загрузите **.COM**. Ответьте на контрольные вопросы «Загрузка COM модуля в основную память». Представьте в отчете план загрузки модуля **.COM** в основную память.

Шаг 6. Откройте отладчик **TD.EXE** и загрузите «хороший» **.EXE**. Ответьте на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

Шаг 7. Оформление отчета в соответствии с требованиями. В отчете необходимо привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей – в отладчике.

Выполнение работы.

Шаг 1. Пользуясь данным шаблоном, был реализован исходный код **.COM** модуля, который определяет тип PC и версию системы (см. *прил. А*). В результате данной реализации получились так называемые “хороший” **.COM** модуль, а так же “плохой” **.EXE** модуль. Далее были продемонстрированы выводы данных модулей.

```
C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>LAB1_COM.EXE

Type of PC:

Type of PC:          5 0

Type of PC:          0

Type of PC:
Type of PC:
000000
Type of PC:
Type of PC:
```

Рис. 1

```
C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>LAB1_COM.COM
Type of PC: AT
Version of MS DOS: 5.0
Serial number of OEM: 0
Serial number of user: 000000
```

Рис. 2

Шаг 2. На основе ранее реализованного **.COM** модуля был реализован исходный код “хорошего” **.EXE** модуля. Далее был продемонстрирован вывод данного модуля.

```
C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>LAB1_EXE.EXE
Type of PC: AT
Version of MS DOS: 5.0
Serial number of OEM: 0
Serial number of user: 000000
```

Рис. 3

Шаг 3. Ответы на контрольные вопросы:

1) Сколько сегментов должна содержать COM-программа?

COM-программа должна содержать только один сегмент. Стек генерируется автоматически. Все данные, а так же сегменты кода находятся в одном сегменте

2) EXE-программа

EXE-программа содержит более одного сегмента. Код, данные, а так же стек хранятся в разных сегментах.

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

В исходном коде COM-программы обязательно должны быть директива `ORG 100h`. Она позволяет сместить всю адресацию на 256 байт. Это необходимо из-за того, что первые 256 байт занимает блок PSP, а все сегментные регистры при загрузке указывают именно на него. Еще должна использоваться директива `ASSUME`. Она позволяет привязать сегмент данных и сегмент кода на один общий сегмент.

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

Адрес сегмента до загрузки неизвестен, следовательно нельзя использовать такие команды, как: `mov reg, seg`. COM-программа не содержит описание адресов. Так как адреса зависят от размещения загрузочного модуля в оперативной памяти, следовательно поэтому нельзя использовать команды, дающие доступ к началу сегментов.

Шаг 4. Ответы на контрольные вопросы:

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

.COM файл состоит из одного сегмента и может быть размером не больше 64КБ, содержит команды и данные. Код начинается с `0h`

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	E9	BC	01	54	79	70	65	20	6F	66	20	50	43	3A	20	24	йj.Type of PC: \$
00000010	50	43	0D	0A	24	50	43	2F	58	54	0D	0A	24	41	54	0D	PC..\$PC/XT..\$AT.
00000020	0A	24	50	53	32	20	6D	6F	64	65	6C	20	33	30	0D	0A	.\$PS2 model 30..
00000030	24	50	53	32	20	6D	6F	64	65	6C	20	38	30	0D	0A	24	\$PS2 model 80..\$
00000040	50	43	6A	72	0D	0A	24	50	43	20	43	6F	6E	76	65	72	PCjr..\$PC Conver
00000050	74	69	62	6C	65	0D	0A	24	56	65	72	73	69	6F	6E	20	tible..\$Version
00000060	6F	66	20	4D	53	20	44	4F	53	3A	20	20	2E	20	20	0D	of MS DOS: . .
00000070	0A	24	53	65	72	69	61	6C	20	6E	75	6D	62	65	72	20	.\$Serial number
00000080	6F	66	20	4F	45	4D	3A	20	20	20	24	53	65	72	69	61	of OEM: \$Seria
00000090	6C	20	6E	75	6D	62	65	72	20	6F	66	20	75	73	65	72	l number of user
000000A0	3A	20	20	20	20	20	20	20	20	20	24	0D	0A	24	24	0F	: \$..\$\$.
000000B0	3C	09	76	02	04	07	04	30	C3	51	8A	E0	E8	EF	FF	86	<.v....0ГQЪаипят
000000C0	C4	B1	04	D2	E8	E8	E6	FF	59	C3	53	8A	FC	E8	E9	FF	Д±.ТиижяУГ\$Ъийя
000000D0	88	25	4F	88	05	4F	8A	C7	E8	DE	FF	88	25	4F	88	05	€%0€..0ЪзиЮя€%0€.
000000E0	5B	C3	51	52	32	E4	33	D2	B9	0A	00	F7	F1	80	CA	30	[ГQR2д3ТН..чсЪК0
000000F0	88	14	4E	33	D2	3D	0A	00	73	F1	3C	00	74	04	0C	30	€.N3T=...sc<.t..0
00000100	88	04	5A	59	C3	BA	03	01	B4	09	CD	21	B8	00	F0	8E	€.ZYГe..г.Н!ë.pђ
00000110	C0	26	A0	FE	FF	3C	FF	74	1C	3C	FE	74	1E	3C	FB	74	А& юя<ят.<ют.<џ
00000120	1A	3C	FC	74	1C	3C	FA	74	1E	3C	F8	74	20	3C	FD	74	.<џt.<џt.<џt <џt
00000130	22	3C	F9	74	24	BA	10	01	EB	25	90	BA	15	01	EB	1F	"<џt\$e..л%ђe..л.
00000140	90	BA	1D	01	EB	19	90	BA	22	01	EB	13	90	BA	31	01	ђe..л.ђe".л.ђel.
00000150	EB	0D	90	BA	40	01	EB	07	90	BA	47	01	EB	01	90	B4	л.ђe@.л.ђeG.л.ђг
00000160	09	CD	21	C3	B4	30	CD	21	50	BE	58	01	83	C6	13	E8	.Н!Гг0Н!РsX.ђЖ.и
00000170	70	FF	58	8A	C4	83	C6	03	E8	67	FF	BA	58	01	B4	09	ряXђДђЖ.игяeX.г.
00000180	CD	21	BE	72	01	83	C6	16	8A	C7	E8	55	FF	BA	72	01	Н!sr.ђЖ.ЪзиУяer.
00000190	B4	09	CD	21	BA	AB	01	B4	09	CD	21	BF	8B	01	83	C7	г.Н!e«.г.Н!i<.ђЗ
000001A0	1C	8B	C1	E8	24	FF	8A	C3	E8	0E	FF	83	EF	02	89	05	.<Ви\$яЪГи.яђп.%.
000001B0	BA	8B	01	B4	09	CD	21	BA	AB	01	B4	09	CD	21	C3	E8	e<.г.Н!e«.г.Н!Ги
000001C0	43	FF	E8	9F	FF	32	C0	B4	4C	CD	21						Сяиця2ArLH!

Рис. 4

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

В “плохом” .EXE файле код и данные хранятся в одном сегменте. С адреса 300h располагается сам сегмент. С адреса 0h располагается таблица настроек адресов, маркер MZ и заголовок.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	4D	5A	CB	00	03	00	00	00	20	00	00	00	FF	FF	00	00	MZЛ..... ..яя..
00000010	00	00	C3	56	00	01	00	00	1E	00	00	00	01	00	00	00	..ГV.....
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

Рис. 5

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000280	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000290	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000002A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000002B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000002C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000002D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000002E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000002F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000300	E9	BC	01	54	79	70	65	20	6F	66	20	50	43	3A	20	24	йж.Type of PC: \$
00000310	50	43	0D	0A	24	50	43	2F	58	54	0D	0A	24	41	54	0D	PC..\$PC/XT..\$AT.
00000320	0A	24	50	53	32	20	6D	6F	64	65	6C	20	33	30	0D	0A	.\$PS2 model 30..
00000330	24	50	53	32	20	6D	6F	64	65	6C	20	38	30	0D	0A	24	\$PS2 model 80..\$
00000340	50	43	6A	72	0D	0A	24	50	43	20	43	6F	6E	76	65	72	PCjr..\$PC Conver
00000350	74	69	62	6C	65	0D	0A	24	56	65	72	73	69	6F	6E	20	tible..\$Version
00000360	6F	66	20	4D	53	20	44	4F	53	3A	20	20	2E	20	20	0D	of MS DOS: . . .
00000370	0A	24	53	65	72	69	61	6C	20	6E	75	6D	62	65	72	20	.\$Serial number
00000380	6F	66	20	4F	45	4D	3A	20	20	24	53	65	72	69	61		of OEM: \$Serial
00000390	6C	20	6E	75	6D	62	65	72	20	6F	66	20	75	73	65	72	l number of user
000003A0	3A	20	20	20	20	20	20	20	20	24	0D	0A	24	24	0F		: ..\$\$.
000003B0	3C	09	76	02	04	07	04	30	C3	51	8A	E0	E8	EF	FF	86	<.v....0GQбайпн
000003C0	C4	B1	04	D2	E8	E8	E6	FF	59	C3	53	8A	FC	E8	E9	FF	Дт.ТиияяYTSльийя
000003D0	88	25	4F	88	05	4F	8A	C7	E8	DE	FF	88	25	4F	88	05	€%O€.OлЪиOя€%O€.
000003E0	5B	C3	51	52	32	E4	33	D2	B9	0A	00	F7	F1	80	CA	30	[QQR2д3TН..чсБК0
000003F0	88	14	4E	33	D2	3D	0A	00	73	F1	3C	00	74	04	0C	30	€.N3T=..sc<.t..0
00000400	88	04	5A	59	C3	BA	03	01	B4	09	CD	21	B8	00	F0	8E	€.ZYTe..r.Н!э.рН
00000410	C0	26	A0	FE	FF	3C	FF	74	1C	3C	FE	74	1E	3C	FB	74	A& ня<ят.<ют.<ют
00000420	1A	3C	FC	74	1C	3C	FA	74	1E	3C	F8	74	20	3C	FD	74	.<ьт.<ьт.<шт <эт
00000430	22	3C	F9	74	24	BA	10	01	EB	25	90	BA	15	01	EB	1F	"<шт\$е..л%је..л.
00000440	90	BA	1D	01	EB	19	90	BA	22	01	EB	13	90	BA	31	01	је..л.је".л.жел.
00000450	EB	0D	90	BA	40	01	EB	07	90	BA	47	01	EB	01	90	B4	л.је@.л.јеG.л.жг
00000460	09	CD	21	C3	B4	30	CD	21	50	BE	58	01	83	C6	13	E8	.Н!Гг0Н!PsX.фЖ.и
00000470	70	FF	58	8A	C4	83	C6	03	E8	67	FF	BA	58	01	B4	09	ряХЛдфЖ.игяеХ.г.
00000480	CD	21	BE	72	01	83	C6	16	8A	C7	E8	55	FF	BA	72	01	Н!sr.фЖ.лЪиUяer.
00000490	B4	09	CD	21	BA	AB	01	B4	09	CD	21	BF	8B	01	83	C7	г.Н!е«.г.Н!й<.фЗ
000004A0	1C	8B	C1	E8	24	FF	8A	C3	E8	0E	FF	83	EF	02	89	05	.<Би\$ялГи.яфп.%.
000004B0	BA	8B	01	B4	09	CD	21	BA	AB	01	B4	09	CD	21	C3	E8	е<.г.Н!е«.г.Н!Ги
000004C0	43	FF	E8	9F	FF	32	C0	B4	4C	CD	21						Сяиця2ArLH!

Рис. 6

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

В “хорошем” .EXE файле стек и код разделены по сегментам, следовательно файл может быть больше, чем 64КБ. Код не использует директивы ORG 100h, следовательно память не идет под PSP модуль, а код начинается с адреса 300h.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	4D	5A	D6	00	03	00	01	00	20	00	00	00	FF	FF	00	00	MZЦ..... ..яя..
00000010	00	01	55	23	11	01	1B	00	1E	00	00	00	01	00	15	01	..U#.....
00000020	1B	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

Рис. 7

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000280	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000290	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000002A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000002B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000002C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000002D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000002E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000002F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000300	54	79	70	65	20	6F	66	20	50	43	3A	20	24	50	43	0D	Type of PC: \$PC.
00000310	0A	24	50	43	2F	58	54	0D	0A	24	41	54	0D	0A	24	50	.\$PC/XI..\$AT..\$P
00000320	53	32	20	6D	6F	64	65	6C	20	33	30	0D	0A	24	50	53	S2 model 30..\$PS
00000330	32	20	6D	6F	64	65	6C	20	38	30	0D	0A	24	50	43	6A	2 model 80..\$PCj
00000340	72	0D	0A	24	50	43	20	43	6F	6E	76	65	72	74	69	62	r..\$PC Convertib
00000350	6C	65	0D	0A	24	56	65	72	73	69	6F	6E	20	6F	66	20	le..\$Version of
00000360	4D	53	20	44	4F	53	3A	20	20	2E	20	20	0D	0A	24	53	MS DOS: . . . \$S
00000370	65	72	69	61	6C	20	6E	75	6D	62	65	72	20	6F	66	20	erial number of
00000380	4F	45	4D	3A	20	20	20	24	53	65	72	69	61	6C	20	6E	OEM: \$Serial n
00000390	75	6D	62	65	72	20	6F	66	20	75	73	65	72	3A	20	20	umber of user:
000003A0	20	20	20	20	20	20	20	24	0D	0A	24	00	00	00	00	00	\$.\$. \$.....
000003B0	24	0F	3C	09	76	02	04	07	04	30	C3	51	8A	E0	E8	EF	\$.<.v....0TQbair
000003C0	FF	86	C4	B1	04	D2	E8	E8	E6	FF	59	C3	53	8A	FC	E8	я+Д±.ТижжЯУТ\$Ъьи
000003D0	E9	FF	88	25	4F	88	05	4F	8A	C7	E8	DE	FF	88	25	4F	йя%0€.0б\$иЮя€%0
000003E0	88	05	5B	C3	51	52	32	E4	33	D2	B9	0A	00	F7	F1	80	€. [QQR2л3TМ...чсб
000003F0	CA	30	88	14	4E	33	D2	3D	0A	00	73	F1	3C	00	74	04	K0€.N3T=..sc<.t.
00000400	0C	30	88	04	5A	59	C3	BA	00	00	B4	09	CD	21	B8	00	.0€.ZYTe..r.H!ë.
00000410	F0	8E	C0	26	A0	FE	FF	3C	FF	74	1C	3C	FE	74	1E	3C	pHAg юя<ят.<ят.<
00000420	FB	74	1A	3C	FC	74	1C	3C	FA	74	1E	3C	F8	74	20	3C	ят.<ьт.<ьт.<шт <
00000430	FD	74	22	3C	F9	74	24	BA	0D	00	EB	25	90	BA	12	00	шт"<шт\$е..л\$е..
00000440	EB	1F	90	BA	1A	00	EB	19	90	BA	1F	00	EB	13	90	BA	л.е..л.е..л.е
00000450	2E	00	EB	0D	90	BA	3D	00	EB	07	90	BA	44	00	EB	01	..л.е=..л.еD.л.
00000460	90	B4	09	CD	21	C3	B4	30	CD	21	50	BE	55	00	83	C6	ђr.H!Tr0H!PsU.фЖ
00000470	13	E8	70	FF	58	8A	C4	83	C6	03	E8	67	FF	BA	55	00	иpяXлДфЖ.игяеU.
00000480	B4	09	CD	21	BE	6F	00	83	C6	16	8A	C7	E8	55	FF	BA	r.H!so.фЖ.л\$иUяе
00000490	6F	00	B4	09	CD	21	BA	A8	00	B4	09	CD	21	BF	88	00	о.r.H!eE.r.H!i€.
000004A0	83	C7	1C	8B	C1	E8	24	FF	8A	C3	E8	0E	FF	83	EF	02	ф\$.<Би\$ялГи.яфп.
000004B0	89	05	BA	88	00	B4	09	CD	21	BA	A8	00	B4	09	CD	21	%.е€.r.H!eE.r.H!
000004C0	C3	2B	C0	50	B8	10	00	8E	D8	E8	3B	FF	E8	97	FF	32	Г+APë..лши; яи-я2
000004D0	C0	B4	4C	CD	21	C3											ArLH!Г

Рис.8

Шаг 5. Ответы на контрольные вопросы:

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

Сначала определяется сегментный адрес участка ОП, способного вместить загрузку программы, затем создается блок памяти для PSP и программы, COM-файл считывается помещается в память с 100h. После сегментные регистры устанавливаются на начало PSP. SP устанавливается на конец PSP, 0000h помещается в стек, в IP записывается 100h. Код располагается с адреса 100h

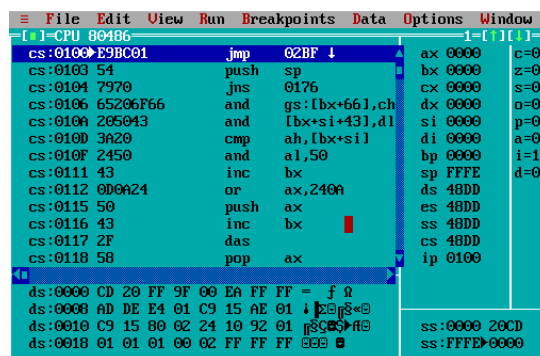


Рис. 9

2) Что располагается с адреса 0?

С адреса 0 располагаются PSP сегмент и таблица настроек

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

Сегментные регистры имеют значения 48DD, они указывают на PSP.

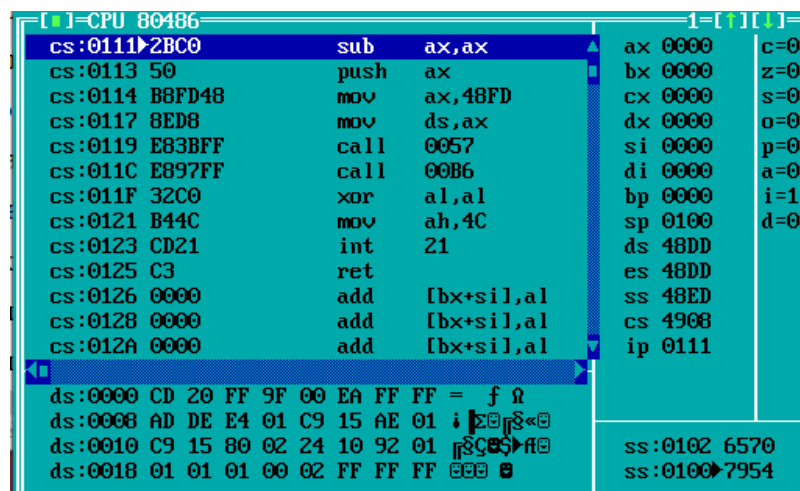
4) Как определяется стек? Какую область памяти он занимает? Какие адреса?

Стек генерируется автоматически. SP указывает на конец стека, а SS—на начало. Адреса расположены в диапазоне 0h–FFFEh (потому что это последний адрес, который кратен двум).

Шаг 6. Ответы на контрольные вопросы:

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

Загружается “хороший” EXE считыванием информации заголовка EXE, далее выполняется перемещение адресов сегментов, ES и DS устанавливаются на начало PSP, SS – на начало сегмента стека, а CS – на начало сегмента команд.



The screenshot shows a debugger window with the following content:

Address	Disassembly	Comment	Register/Value
cs:0111	sub ax,ax		ax 0000
cs:0113	push ax		bx 0000
cs:0114	mov ax,48FD		cx 0000
cs:0117	mov ds,ax		dx 0000
cs:0119	call 0057		si 0000
cs:011C	call 00B6		di 0000
cs:011F	xor al,al		bp 0000
cs:0121	mov ah,4C		sp 0100
cs:0123	int 21		ds 48DD
cs:0125	ret		es 48DD
cs:0126	add [bx+si],al		ss 48ED
cs:0128	add [bx+si],al		cs 4908
cs:012A	add [bx+si],al		ip 0111

Below the assembly list, memory addresses are shown:

Address	Value
ds:0000	CD 20 FF 9F 00 EA FF FF = f 0
ds:0008	AD DE E4 01 C9 15 AE 01
ds:0010	C9 15 80 02 24 10 92 01
ds:0018	01 01 01 00 02 FF FF FF

On the right side, the status bar shows:

Register	Value
ax	0000
bx	0000
cx	0000
dx	0000
si	0000
di	0000
bp	0000
sp	0100
ds	48DD
es	48DD
ss	48ED
cs	4908
ip	0111

Рис. 10

2) На что указывают регистры DS и ES?

DS и ES указывают на начало PSP

3) Как определяется стек?

Стек определяется с помощью директивы `.stack` с указанием размера стека. `SS` указывает на начало сегмента стека, а `SP` – на конец.

4) Как определяется точка входа?

Точка входа определяется директивой `END`.

Вывод:

В ходе выполнения лабораторной работы была написана программа для определения типа и версии РС, изучены 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 ПРОГРАММЫ

Название файла: lab1_com.asm

```
TESTPC SEGMENT

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

                ORG      100H

START:          JMP      BEGIN


type_of         db          'Type of PC: $'
pc              db          'PC',0DH,0ah,'$'
pc_xt           db          'PC/XT',0DH,0ah,'$'
at_type         db          'AT',0DH,0ah,'$'
ps2_m30         db          'PS2 model 30',0DH,0ah,'$'
ps2_m80         db          'PS2 model 80',0DH,0ah,'$'
pcjr            db          'PCjr',0DH,0ah,'$'
pc_conv         db          'PC Convertible',0DH,0ah,'$'
version_pc      db          'Version of MS DOS: . ',0DH,0ah,'$'
sn_oem          db          'Serial number of OEM:   $'
sn_of_user      db          'Serial number of user:           $'
at_end          db          0DH,0ah,'$'


;ПРОЦЕДУРЫ
;-----

TETR_TO_HEX     PROC  near

                and  al,0Fh
                cmp  al,09
                jbe  NEXT
                add  al,07

NEXT:           add  al,30h
                ret

TETR_TO_HEX     ENDP

;-----

BYTE_TO_HEX     PROC  near

; байт в al переводится в два символа шестн. числа в AX

                push cx
                mov  ah,al
                call TETR_TO_HEX
                xchg al,ah
```

```

        mov     cl,4
        shr     al,cl
        call    TETR_TO_HEX ;в al старшая цифра
        pop     cx ;в ah младшая
        ret

BYTE_TO_HEX      ENDP
;-----
WRD_TO_HEX      PROC near
;перевод в 16 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 10с/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_l
                                or      al,30h
                                mov     [SI],al

                                end_l:
                                pop     dx
                                pop     cx
                                ret

BYTE_TO_DEC                     ENDP

print_type                      PROC near
                                mov     dx, offset type_of

                                mov     ah, 09h
                                int     21h

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

                                cmp     al, 0FFh
                                je      print_pc
                                cmp     al, 0FEh
                                je      print_pc_xt
                                cmp     al, 0FBh
                                je      print_pc_xt
                                cmp     al, 0FCh
                                je      print_at
                                cmp     al, 0FAh
                                je      print_ps2_m30
                                cmp     al, 0F8h
                                je      print_ps2_m80
                                cmp     al, 0FDh
                                je      print_pcjr
                                cmp     al, 0F9h

```

```

                                je      print_pc_conv

print_pc:
                                mov     dx, offset pc
                                jmp     type_end

print_pc_xt:
                                mov     dx, offset pc_xt
                                jmp     type_end

print_at:
                                mov     dx, offset at_type
                                jmp     type_end

print_ps2_m30:
                                mov     dx, offset ps2_m30
                                jmp     type_end

print_ps2_m80:
                                mov     dx, offset ps2_m80
                                jmp     type_end

print_pcjr:
                                mov     dx, offset pcjr
                                jmp     type_end

print_pc_conv:
                                mov     dx, offset pc_conv
                                jmp     type_end

type_end:
                                mov     ah, 09h
                                int     21h
                                ret

print_type                      ENDP

print_ver                      PROC  near

```

```

mov  ah, 30h
int  21h

push ax
mov  si, offset version_pc
add  si, 19
call BYTE_TO_DEC
pop  ax
mov  al, ah
add  si, 3
call BYTE_TO_DEC
mov  dx, offset version_pc

mov  ah,09h
int  21h

mov  si, offset sn_oem
add  si, 22
mov  al, bh
call BYTE_TO_DEC
mov  dx, offset sn_oem

mov  ah,09h
int  21h

mov  dx, offset at_end

mov  ah,09h
int  21h

mov  di, offset sn_of_user
add  di, 28
mov  ax, cx
call WRD_TO_HEX
mov  al, bl
call BYTE_TO_HEX
sub  di, 2
mov  [di], ax

```

```

                                mov  dx, offset sn_of_user

                                mov  ah,09h
                                int   21h

                                mov  dx, offset at_end

                                mov  ah,09h
                                int   21h

                                ret

print_ver      ENDP

BEGIN:
; . . . . .

                                call print_type
                                call print_ver

; . . . . .
; Выход в DOS

                                xor   al,al
                                mov   ah,4Ch
                                int    21h

TESTPC        ENDS

END  START ;конец модуля, START - точка входа

```

Название файла: lab1_exe.asm

```

SSTACK        SEGMENT  STACK

                                DW 128 DUP(?)

SSTACK ENDS

```

```

DATA          SEGMENT

TYPE_OF       DB            'TYPE OF PC: $'
PC             DB            'PC',0DH,0AH,'$'
PC_XT         DB            'PC/XT',0DH,0AH,'$'
AT_TYPE       DB            'AT',0DH,0AH,'$'
PS2_M30       DB            'PS2 MODEL 30',0DH,0AH,'$'
PS2_M80       DB            'PS2 MODEL 80',0DH,0AH,'$'
PCJR          DB            'PCJR',0DH,0AH,'$'

```

```

PC_CONV      DB      'PC CONVERTIBLE',0DH,0AH,'$'
VERSION_PC   DB      'VERSION OF MS DOS:  .  ',0DH,0AH,'$'
SN_OEM       DB      'SERIAL NUMBER OF OEM:  $'
SN_OF_USER   DB      'SERIAL NUMBER OF USER:          $'
AT_END       DB      0DH,0AH,'$'

```

```
DATA      ENDS
```

```
CODE      SEGMENT
```

```
      ASSUME      CS:CODE, DS:DATA, SS:SSTACK
```

```
;ПРОЦЕДУРЫ
```

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

```

TETR_TO_HEX  PROC  NEAR
                AND  AL,0FH
                CMP  AL,09
                JBE  NEXT
                ADD  AL,07

```

```

NEXT:          ADD  AL,30H
                RET

```

```
TETR_TO_HEX  ENDP
```

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

```
BYTE_TO_HEX  PROC  NEAR
```

```
; БАЙТ В AL ПЕРЕВОДИТСЯ В ДВА СИМВОЛА ШЕСТН. ЧИСЛА В AX
```

```

                PUSH CX
                MOV  AH,AL
                CALL TETR_TO_HEX
                XCHG AL,AH
                MOV  CL,4
                SHR  AL,CL
                CALL TETR_TO_HEX ;В AL СТАРШАЯ ЦИФРА
                POP  CX ;В AH МЛАДШАЯ
                RET

```

```
BYTE_TO_HEX  ENDP
```

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

```
WRD_TO_HEX    PROC  NEAR
```

```
;ПЕРЕВОД В 16 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 10С/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_L
        OR     AL,30H
        MOV    [SI],AL
END_L:
        POP    DX
        POP    CX
        RET

```

```

BYTE_TO_DEC      ENDP

PRINT_TYPE      PROC NEAR
MOV     DX, OFFSET TYPE_OF

MOV     AH, 09H
INT     21H

MOV     AX, 0F000H
MOV     ES, AX
MOV     AL, ES:[0FFFEH]

CMP     AL, 0FFH
JE      PRINT_PC
CMP     AL, 0FEH
JE      PRINT_PC_XT
CMP     AL, 0FBH
JE      PRINT_PC_XT
CMP     AL, 0FCH
JE      PRINT_AT
CMP     AL, 0FAH
JE      PRINT_PS2_M30
CMP     AL, 0F8H
JE      PRINT_PS2_M80
CMP     AL, 0FDH
JE      PRINT_PCJR
CMP     AL, 0F9H
JE      PRINT_PC_CONV

PRINT_PC:
MOV     DX, OFFSET PC
JMP     TYPE_END

PRINT_PC_XT:
MOV     DX, OFFSET PC_XT
JMP     TYPE_END

PRINT_AT:

```

```

        MOV     DX, OFFSET AT_TYPE
        JMP     TYPE_END

PRINT_PS2_M30:
        MOV     DX, OFFSET PS2_M30
        JMP     TYPE_END

PRINT_PS2_M80:
        MOV     DX, OFFSET PS2_M80
        JMP     TYPE_END

PRINT_PCJR:
        MOV     DX, OFFSET PCJR
        JMP     TYPE_END

PRINT_PC_CONV:
        MOV     DX, OFFSET PC_CONV
        JMP     TYPE_END

TYPE_END:
        MOV     AH, 09H
        INT     21H
        RET

PRINT_TYPE      ENDP

PRINT_VER      PROC NEAR
        MOV     AH, 30H
        INT     21H

        PUSH    AX
        MOV     SI, OFFSET VERSION_PC
        ADD     SI, 19
        CALL    BYTE_TO_DEC
        POP     AX
        MOV     AL, AH
        ADD     SI, 3
        CALL    BYTE_TO_DEC

```

MOV DX, OFFSET VERSION_PC

MOV AH, 09H

INT 21H

MOV SI, OFFSET SN_OEM

ADD SI, 22

MOV AL, BH

CALL BYTE_TO_DEC

MOV DX, OFFSET SN_OEM

MOV AH, 09H

INT 21H

MOV DX, OFFSET AT_END

MOV AH, 09H

INT 21H

MOV DI, OFFSET SN_OF_USER

ADD DI, 28

MOV AX, CX

CALL WRD_TO_HEX

MOV AL, BL

CALL BYTE_TO_HEX

SUB DI, 2

MOV [DI], AX

MOV DX, OFFSET SN_OF_USER

MOV AH, 09H

INT 21H

MOV DX, OFFSET AT_END

MOV AH, 09H

INT 21H

RET

```

PRINT_VER      ENDP

MAIN           PROC FAR
                SUB    AX,AX
                PUSH   AX
                MOV     AX,DATA
                MOV     DS,AX
; . . . . .
                CALL   PRINT_TYPE
                CALL   PRINT_VER
; . . . . .
; ВЫХОД В DOS
                XOR     AL,AL
                MOV     AH,4CH
                INT     21H
                RET
MAIN           ENDP
CODE           ENDS
END    MAIN

```