

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. 0382

Гудов Н.Р.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

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

Задание.

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.
2. Написать текст исходного **.EXE** модуля, который выполняет те же функции, что и модуль в шаге 1 и отладить его. Таким образом, будет получен «хороший» **.EXE**.

3. Сравнить исходные тексты для **.COM** и **.EXE** модулей. Ответить на вопросы «Отличия исходных текстов **COM** и **EXE** программ».
4. Запустить **FAR** и открыть файл загрузочного модуля **.COM** и файл «плохого» **.EXE** в шестнадцатеричном виде. Затем открыть файл загрузочного модуля «хорошего» **.EXE** и сравнить его с предыдущими файлами. Ответить на контрольные вопросы «Отличия форматов файлов **COM** и **EXE** модулей».
5. Открыть отладчик **TD.EXE** и загрузить **CO**. Ответить на контрольные вопросы «Загрузка **COM** модуля в основную память». Представить в отчете план загрузки модуля **.COM** в основную память.
6. Открыть отладчик **TD.EXE** и загрузить «хороший» **.EXE**. Ответить на контрольные вопросы «Загрузка «хорошего» **EXE** в основную память».
7. Оформить отчет в соответствии с требованиями. Привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей — в отладчике.

Выполнение работы.

При работе был использован шаблон из методического пособия с реализацией процедур перевода двоичных кодов в шестнадцатеричные и десятичные числа. При выполнении были разработаны и реализованы следующие процедуры: *write*, для вывода строки; *Model_type* – вывода типа ПК, путем считывания предпоследнего байта ROM BIOS по адресу 0F000:0FFFEh.; *Dos_Version* – для определения версии MS DOS через функцию 30H прерывания 21H; а также *Oem* и *User_Number* для вывода серийных номеров OEM и пользователя.

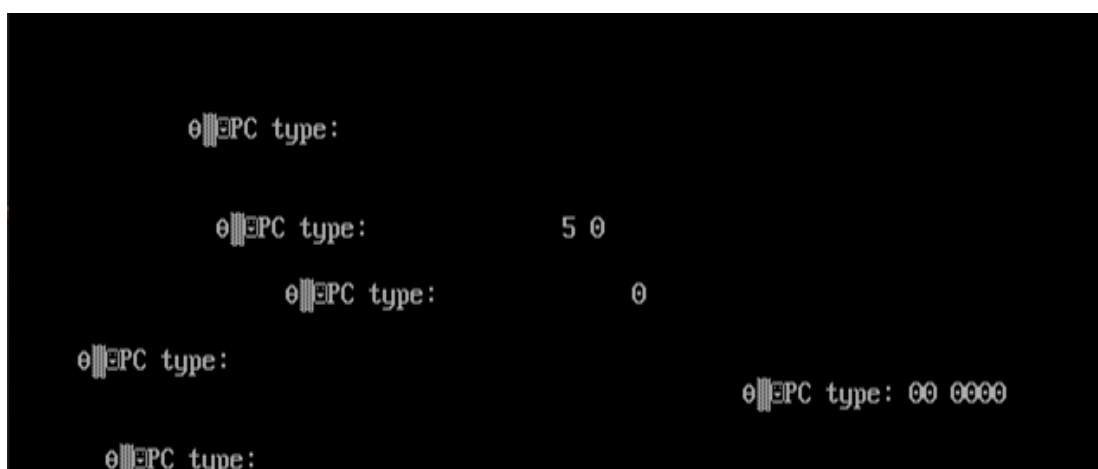
Путем использования шаблона, из методических указаний получаем “хороший” .com модуль, содержащий один сегмент и правильно работающую программу. Рисунок 1. При попытке компиляции такой программы как .exe модуль получаем “плохой” .exe модуль. Рисунок 2.

В результате шага имеем “хороший” .COM модуль и “плохой” .EXE модуль. Выводы, полученные при их запуске, представлены на рисунке 1 и рисунке 2 соответственно.



```
B:\>LAB1_COM.COM
PC type: AT
DOS version: 5.0
OEM number: 0
User number: 00:0000
```

Рисунок 1 –запуск модуля.com



```
PC type:
PC type: 5 0
PC type: 0
PC type:
PC type: 00 0000
PC type:
```

Рисунок 2 –запуск модуля .exe

Организация “хорошего” exe. Модуля требует разбиения программы на сегменты кода, данных и стека. Такой подход не обязывает устанавливать CS:IP на конец PSP, так как блок расположен вне сегмента кода. Также требуется написать главную функцию, которая будет являться точкой входа в программу. Все остальные функции в обеих программах выполняются одинаково.

Правильно написанный .exe модуль выдает результат аналогичный правильному .com модулю.

Контрольные вопросы.

Отличия исходных текстов COM и EXE программ:

1. Сколько сегментов должна содержать COM-программа?
COM программа содержит один сегмент.
2. EXE-программа?
EXE программа содержит следующие сегменты: сегмент стека, сегмент данных, сегмент кода.
3. Какие директивы должны быть обязательно в тексте COM-программы?
ORG 100h – для смещения адресации, так как первые 256 байт будут выделены под PSP. ASSUME – для сопоставления сегментным регистром единственного сегмента.
SEGMENT – используется для описания сегмента единого.
4. Все ли форматы команд можно использовать в COM-программе?
Нет. Так как в .com файле отсутствует relocation table, команды вида `mov *register*, seg *segment_name*` не поддерживаются.

Отличия форматов файлов .com и .exe модулей:

1. Какова структура файла .COM? С какого адреса располагается код?
Код располагается с нулевого адреса, но фактическое его размещение происходит после смещения на 100h, из-за расположения там PSP. Программа имеет следующую структуру: PSP, код, данные, стек в одном сегменте.

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	e9	b2	01	50	43	20	74	79	70	65	3a	20	24	50	43	20	йI.PC type: \$PC
00000010	0a	24	50	43	2f	58	54	20	0a	24	41	54	20	0a	24	50	.\$PC/XT .\$AT .\$P
00000020	53	32	20	6d	6f	64	65	6c	20	33	30	20	0a	24	50	53	S2 model 30 .\$PS
00000030	32	20	6d	6f	64	65	6c	20	38	30	20	0a	24	50	43	6a	2 model 80 .\$PCj
00000040	72	20	0a	24	50	43	20	43	6f	6e	76	65	72	74	69	62	r .\$PC Convertib
00000050	6c	65	20	0a	24	55	6e	6b	6e	6f	77	6e	20	62	79	74	le .\$Unknown byt
00000060	65	3a	20	20	20	0a	24	44	4f	53	20	76	65	72	73	69	e: .\$DOS versi
00000070	6f	6e	3a	20	20	2e	20	20	20	0a	24	4f	45	4d	20	6e	on: . .\$OEM n
00000080	75	6d	62	65	72	3a	20	20	20	20	0a	24	55	73	65	72	umber: .\$User
00000090	20	6e	75	6d	62	65	72	3a	20	24	20	20	3a	20	20	20	number: \$:
000000a0	20	0a	24	24	0f	3c	09	76	02	04	07	04	30	c3	51	8a	.\$\$.<.v....0GQЪ
000000b0	e0	e8	ef	ff	86	c4	b1	04	d2	e8	e8	e6	ff	59	c3	53	аипя†Д±.ТиижяYGS
000000c0	8a	fc	e8	e9	ff	88	25	4f	88	05	4f	8a	c7	e8	de	ff	Ъийя€%O€.OЪзиЮя
000000d0	88	25	4f	88	05	5b	c3	51	52	32	e4	33	d2	b9	0a	00	€%O€. [GQR2дЗТН..
000000e0	f7	f1	80	ca	30	88	14	4e	33	d2	3d	0a	00	73	f1	3c	чсЪK0€.N3T=..sc<
000000f0	00	74	04	0c	30	88	04	5a	59	c3	b4	09	cd	21	c3	b8	.t..0€.ZYГг.Н!Гё
00000100	00	f0	8e	c0	26	a0	fe	ff	ba	03	01	e8	ec	ff	3c	ff	.рѢA&.юяє..имя<я

Бинарный вид com модуля.

2. Какова структура файла «плохого» EXE? С какого адреса располагается код?
Что располагается с адреса 0?
«Плохой» .exe файл состоит из единственного сегмента. Код располагается с адреса 300h.

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	4d	5a	c7	00	03	00	00	00	20	00	00	00	ff	ff	00	00	MZS..... ..яя..
00000010	00	00	66	1f	00	01	00	00	1e	00	00	00	01	00	00	00	..f.....
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	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000200	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000210	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000220	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000230	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000240	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000250	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000260	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000270	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
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	b2	01	50	43	20	74	79	70	65	3a	20	24	50	43	20	ИI.PC type: \$PC
00000310	0a	24	50	43	2f	58	54	20	0a	24	41	54	20	0a	24	50	.\$PC/XT.\$AT.\$P
00000320	53	32	20	6d	6f	64	65	6c	20	33	30	20	0a	24	50	53	S2 model 30.\$PS
00000330	32	20	6d	6f	64	65	6c	20	38	30	20	0a	24	50	43	6a	2 model 80.\$PCj
00000340	72	20	0a	24	50	43	20	43	6f	6e	76	65	72	74	69	62	r.\$PC Convertib
00000350	6c	65	20	0a	24	55	6e	6b	6e	6f	77	6e	20	62	79	74	1e.\$Unknown byt
00000360	65	3a	20	20	20	0a	24	44	4f	53	20	76	65	72	73	69	e:.\$DOS versi
00000370	6f	6e	3a	20	20	2e	20	20	20	0a	24	4f	45	4d	20	6e	on:.\$OEM n
00000380	75	6d	62	65	72	3a	20	20	20	20	0a	24	55	73	65	72	umber:.\$User
00000390	20	6e	75	6d	62	65	72	3a	20	24	20	20	3a	20	20	20	number: \$:
000003a0	20	0a	24	24	0f	3c	09	76	02	04	07	04	30	c3	51	8a	.\$\$.<.v....0GQБ
000003b0	e0	e8	ef	ff	86	c4	b1	04	d2	e8	e8	e6	ff	59	c3	53	аппя†Д†.ТиияяYTS
000003c0	8a	fc	e8	e9	ff	88	25	4f	88	05	4f	8a	c7	e8	de	ff	Ъийяе%OE.OБзиЮя
000003d0	88	25	4f	88	05	5b	c3	51	52	32	e4	33	d2	b9	0a	00	е%OE.[QQR2дЗТМ..
000003e0	f7	f1	80	ca	30	88	14	4e	33	d2	3d	0a	00	73	f1	3c	чсБК0E.N3T=..sc<
000003f0	00	74	04	0c	30	88	04	5a	59	c3	b4	09	cd	21	c3	b8	.t..OE.ZYTr.N!Гe
00000400	00	f0	8e	c0	26	a0	fe	ff	ba	03	01	e8	ec	ff	3c	ff	.pнA&.юяе..имя<я
00000410	75	06	ba	0d	01	eb	4f	90	3c	fe	74	04	3c	fb	75	06	u.e..ло.<ют.<му.
00000420	ba	12	01	eb	41	90	3c	fc	75	06	ba	1a	01	eb	37	90	e..лA.<yu.e..л7.
00000430	3c	fa	75	06	ba	1f	01	eb	2d	90	3c	f8	75	06	ba	2e	<yu.e..л-<шу.e.
00000440	01	eb	23	90	3c	fd	75	06	ba	3d	01	eb	19	90	3c	f9	.л#. <yu.e=.л..<щ
00000450	75	06	ba	44	01	eb	0f	90	ba	55	01	e8	50	ff	bf	55	u.eD.л..eU.иPяiU
00000460	01	83	c7	0e	89	05	e8	91	ff	c3	b4	30	cd	21	be	74	.fз.к.и'яГrOH!st
00000470	01	e8	63	ff	83	c6	03	8a	c4	e8	5b	ff	ba	67	01	e8	.исяfЖ.ЪДи[яег.и

Плохой .exe модуль

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

В отличие от “плохого” EXE в “хорошем” EXE присутствуют три сегмента: сегмент кода, сегмент данных и сегмент стека (в “плохом” EXE один сегмент, совмещающий код и данные). Смещение кода теперь равно 400h, так как была выделена память под стек (200h), но была удалена директива `org 100h`.

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	4d	5a	c9	01	03	00	01	00	20	00	00	00	ff	ff	00	00	MЗЙ..... ..яя..
00000010	00	02	58	54	12	01	2a	00	1e	00	00	00	01	00	13	01	..ХТ..*.....
00000020	2a	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	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Хороший .exe модуль

Загрузка .com модуля в основную память:

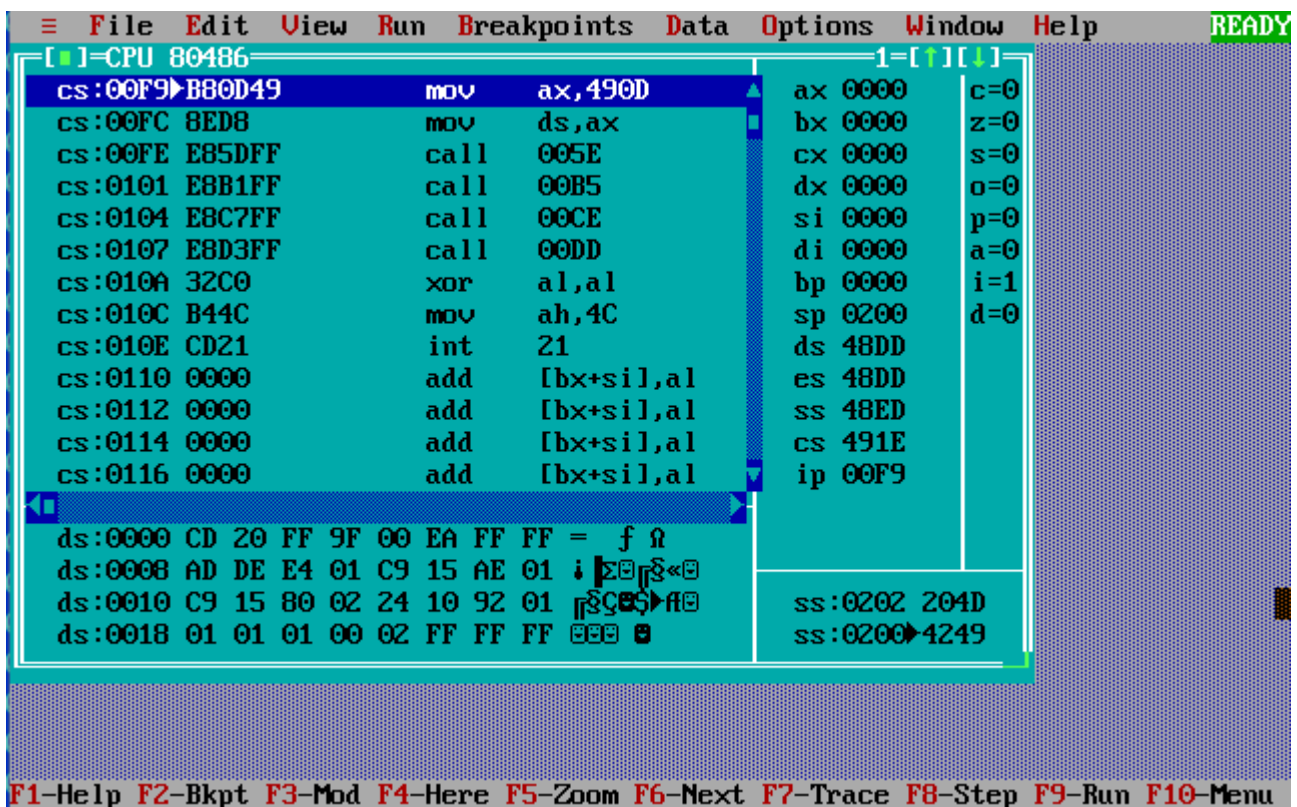
1. Какой формат загрузки модуля COM? С какого адреса располагается код?
Выделяется свободный сегмент памяти, и его адрес заносится в сегментные регистры. В первые 256 байт этого сегмента записывается PSP, после чего записывается содержимое файла .COM.
2. Что располагается с адреса 0?
PSP (англ. Program Segment Prefix) размером в 256 байт, резервируемый операционной системой.

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

Все сегментные регистры имеют значение 489D, и указывают на сегмент памяти, выделенный под программу.

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

Под стек отведен весь сегмент, в который загружена программа. SS=48DD указывает на начало сегмента SP=FFFE указывает на последний адрес сегмента кратный двум. Адреса: 48DD:0000 – 48DD:FFFE.



Загрузка .com модуля

Загрузка «хорошего» .exe модуля в основную память:

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

Сначала в память загружается PSP, после которого загружается .exe модуль в соответствии с информацией в заголовке. Значение регистров см. на рисунке 9.

2. На что указывают DS и ES?

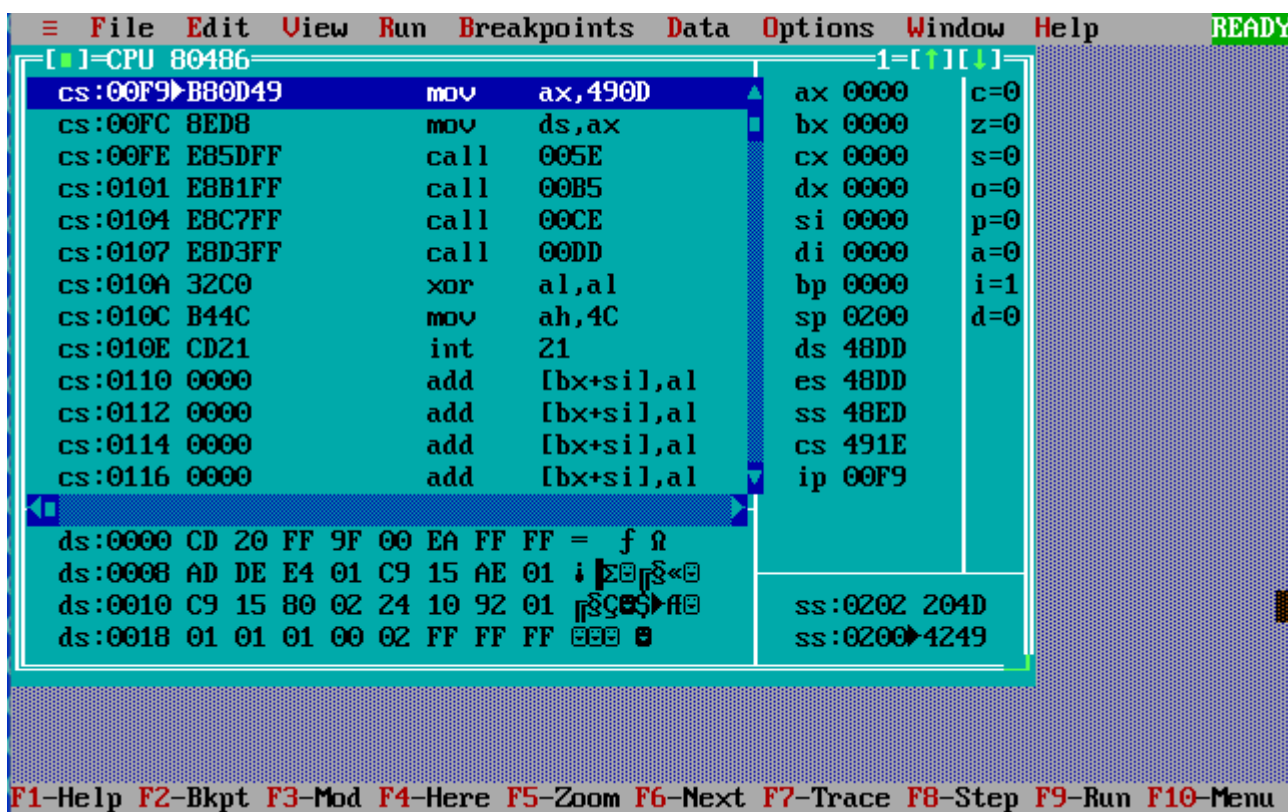
На начало PSP.

3. Как определяется стэк?

Стэк определяется при помощи описание стэкового сегмента в коде и директивы ASSUME. SS указывает на начало сегмента стэка, а SP – на конец стэка (на рисунке 8 видно, что SP = 0100h так как размер стэка – 256 байт).

4. Как определяется точка входа?

Точка входа определяется при помощи директивы END.



Загрузка .exe модуля

Вывод.

Были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память. Написана программа, выводящая информацию о компьютере.

ПРИЛОЖЕНИЕ А.

Исходный код модулей

lab1_com.asm:

```
PC Segment
    Assume CS:PC, DS:PC, ES:NOTHING, SS:NOTHING
    ORG 100H
```

```
START: JMP BEGIN
```

;ДАННЫЕ

```
Type_Message db 'PC type: ', '$'
Model_Message db 'PC ', 0ah, '$'
XT_Model_Message db 'PC/XT ', 0ah, '$'
AT_Model_Message db 'AT ', 0ah, '$'
PS2_Model_30_Message db 'PS2 model 30 ', 0ah, '$'
PS2_Model_80_Message db 'PS2 model 80 ', 0ah, '$'
Jr_Model_Message db 'PCjr ', 0ah, '$'
Convertible_Model_Message db 'PC Convertible ', 0ah, '$'
Unknown_Message db 'Unknown byte: ', 0ah, '$'
Dos_Ver_Message db 'DOS version: . ', 0ah, '$'
Oem_Message db 'OEM number: ', 0ah, '$'
User_Num_Message db 'User number: ', '$'
User_Num db ' : ', 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 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 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

```

write PROC NEAR
    mov ah, 09h
    int 21h
    ret
write ENDP

```

```

Model_type PROC NEAR
    mov ax, 0F000h
    mov es, ax
    mov al, es:[0FFFEh]

```

```

    mov dx, offset Type_Message
    call write

```

```

Model:
    cmp al, 0FFh
    jne Xt_Model
    mov dx, offset Model_Message
    jmp _out

```

```

Xt_Model:
    cmp al, 0FEh
    je pc_xt_process
    cmp al, 0FBh
    jne At_Model

pc_xt_process:
    mov dx, offset XT_Model_Message
    jmp _out

At_Model:
    cmp al, 0FCh
    jne Ps2_Model_30
    mov dx, offset AT_Model_Message
    jmp _out

Ps2_Model_30:
    cmp al, 0FAh
    jne Ps2_Model_80
    mov dx, offset PS2_Model_30_Message
    jmp _out

Ps2_Model_80:
    cmp al, 0F8h
    jne Jr_Model
    mov dx, offset PS2_Model_80_Message
    jmp _out

Jr_Model:
    cmp al, 0FDh
    jne Convertible_Model
    mov dx, offset Jr_Model_Message
    jmp _out

Convertible_Model:
    cmp al, 0F9h
    jne Unknown_type

    mov dx, offset Convertible_Model_Message
    jmp _out

Unknown_type:
    mov dx, offset Unknown_Message
    call byte_to_hex
    mov di, offset Unknown_Message
    add di, 14
    mov [di], ax

_out:
    call write
    ret

Model_type ENDP

Dos_Version PROC NEAR
    mov ah, 30h
    int 21h

```



```

    mov si, offset Dos_Ver_Message + 13
    call byte_to_dec
    add si, 3
    mov al, ah
    call byte_to_dec
    mov dx, offset Dos_Ver_Message
    call write
    ret

Dos_Version ENDP

Oem PROC NEAR
    mov si, offset Oem_Message + 13
    mov al, bh
    call byte_to_dec
    mov dx, offset Oem_Message
    call write
    ret
Oem ENDP

User_Number PROC NEAR
    mov dx, offset User_Num_Message
    call write

    mov ah, 30h
    int 21h
    mov di, offset User_Num

    mov al, bl
    call byte_to_hex
    mov [di], ax
    add di, 6

    mov ax, cx
    call wrd_to_hex
    mov dx, offset User_Num
    call write
    ret

User_Number ENDP

BEGIN:
    call Model_type
    call Dos_Version
    call Oem
    call User_Number

    ; Выход в DOS
    xor al, al
    mov ah, 4Ch
    int 21H

PC ENDS
        END        START

```

exe.asm:

AStack SEGMENT STACK

DW 512 DUP(?)

AStack ENDS

DATA SEGMENT

; ДАННЫЕ

Type_Message db 'PC type: ', '\$'

Model_Message db 'PC ', 0ah, '\$'

XT_Model_Message db 'PC/XT ', 0ah, '\$'

AT_Model_Message db 'AT ', 0ah, '\$'

PS2_Model_30_Message db 'PS2 model 30 ', 0ah, '\$'

PS2_Model_80_Message db 'PS2 model 80 ', 0ah, '\$'

Jr_Model_Message db 'PCjr ', 0ah, '\$'

Convertible_Model_Message db 'PC Convertible ', 0ah, '\$'

Unknown_Message db 'Unknown byte: ', 0ah, '\$'

Dos_Ver_Message db 'DOS version: . ', 0ah, '\$'

Oem_Message db 'OEM number: ', 0ah, '\$'

User_Num_Message db 'User number: ', '\$'

User_Num db ' : ', 0ah, '\$'

DATA ENDS

PC Segment

Assume CS:PC, DS:DATA, SS:AStack

; 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 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 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

write PROC NEAR
    mov ah, 09h
    int 21h
    ret
write ENDP

Model_type PROC NEAR
    mov ax, 0F000h
    mov es, ax
    mov al, es:[0FFFEh]

    mov dx, offset Type_Message
    call write

Model:
    cmp al, 0FFh
    jne Xt_Model
    mov dx, offset Model_Message
    jmp _out

Xt_Model:
    cmp al, 0FEh

```

```

je pc_xt_process
cmp al, 0FBh
jne At_Model

pc_xt_process:
    mov dx, offset XT_Model_Message
    jmp _out

At_Model:
    cmp al, 0FCh
    jne Ps2_Model_30
    mov dx, offset AT_Model_Message
    jmp _out

Ps2_Model_30:
    cmp al, 0FAh
    jne Ps2_Model_80
    mov dx, offset PS2_Model_30_Message
    jmp _out

Ps2_Model_80:
    cmp al, 0F8h
    jne Jr_Model
    mov dx, offset PS2_Model_80_Message
    jmp _out

Jr_Model:
    cmp al, 0FDh
    jne Convertible_Model
    mov dx, offset Jr_Model_Message
    jmp _out

Convertible_Model:
    cmp al, 0F9h
    jne Unknown_type

    mov dx, offset Convertible_Model_Message
    jmp _out

Unknown_type:

```

```

        mov dx, offset Unknown_Message
        call byte_to_hex
        mov di, offset Unknown_Message
        add di, 14
        mov [di], ax

_out:
        call write
        ret

Model_type ENDP

Dos_Version PROC NEAR
        mov ah, 30h
        int 21h

        mov si, offset Dos_Ver_Message + 13
        call byte_to_dec
        add si, 3
        mov al, ah
        call byte_to_dec
        mov dx, offset Dos_Ver_Message
        call write
        ret

Dos_Version ENDP

Oem PROC NEAR
        mov si, offset Oem_Message + 13
        mov al, bh
        call byte_to_dec
        mov dx, offset Oem_Message
        call write
        ret

Oem ENDP

User_Number PROC NEAR
        mov dx, offset User_Num_Message
        call write

```



```

mov ah, 30h
int 21h
mov di, offset User_Num

mov al, bl
call byte_to_hex
mov [di], ax
add di, 6

mov ax, cx
call wrd_to_hex
mov dx, offset User_Num
call write
ret

User_Number ENDP

MAIN PROC FAR
    mov ax, data
    mov ds, ax

    call Model_type
    call Dos_Version
    call Oem
    call User_Number

    ; Выход в DOS
    xor al, al
    mov ah, 4Ch
    int 21H

MAIN ENDP

PC ENDS
                END      MAIN

```