

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

Тюленев Т.В.

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.

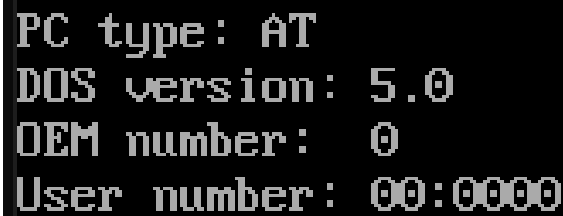
1. Напишите текст исходного **.COM** модуля, который определяет тип РС и версию системы. Это довольно простая задача и для тех, кто уже имеет опыт программирования на ассемблере, это будет небольшой разминкой. Для тех, кто раньше не сталкивался с программированием на ассемблере, это неплохая задача для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

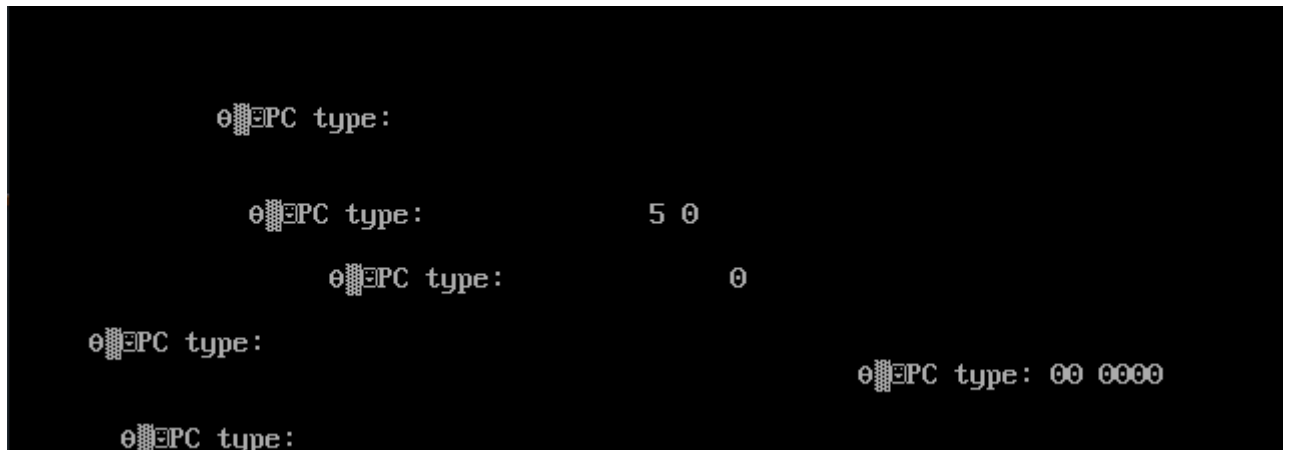
За основу был взят шаблон **.COM** модуля из методического пособия, в котором реализованы процедуры преобразования двоичных кодов в символы шестнадцатеричных и десятичных чисел. Для определения типа РС и версии системы были написаны процедуры: **IBM_TYPE**, **DOS_VER**, **OEMN**, **USERN**. Тип IBM PC был получен согласно байту по адресу 0F000:0FFFFh. Для определения версии системы же использовалась функция 30H прерывания 21H. Ее выходными параметрами являются: **AL** – номер основной версии, **AH** – номер модификации, **BH** – серийный номер **OEM**, **BL:CX** – 24-битовый серийный номер пользователя.

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



```
PC type: AT
DOS version: 5.0
OEM number: 0
User number: 00:0000
```

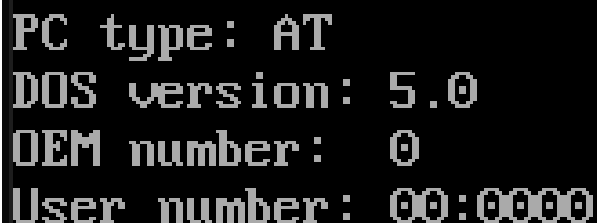
Рисунок 1 – результат запуска модуля com.com



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

Рисунок 2 – результат запуска модуля com.exe

Для написания “хорошего” .EXE модуля разобьем программу на сегменты кода, данных и стека, также добавим главную процедуру. В .COM модуле имелась директива org 100h, что нужна, поскольку при загрузке COM модуля в память DOS первые 256 байт блоком данных занимает PSP, код программы располагается лишь после этого блока. В .EXE модуле же мы в этом не нуждаемся, поскольку блок PSP расположен вне сегмента кода. На рисунке 3 представлены результаты запуска данного модуля.



```
PC type: AT
DOS version: 5.0
OEM number: 0
User number: 00:0000
```

Рисунок 3 – результат запуска модуля exe.exe

Контрольные вопросы.

Отличия исходных текстов COM и EXE программ:

1. Сколько сегментов должна содержать COM-программа?
Один сегмент размером 64Кб.
2. EXE-программа?
EXE программа содержит три сегмента. Сегмент стека, сегмент данных и сегмент кода.
3. Какие директивы должны быть обязательно в тексте COM-программы?
ORG 100h – для смещения адресации на 256 байт. Первые 256 байт будут выделены для PSP.
ASSUME – для сопоставления сегментным регистрам единственного сегмента (или обозначение отсутствия какого либо сегмента, как в случае со стеком).
SEGMENT – используется для описания сегмента данных.
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?
 .COM файл состоит из единственного сегмента, началом которого инициализируются все сегментные регистры. Код располагается с адреса 0h, однако при загрузке программы в память в начало этого сегмента будет добавлен PSP размером 100h байт, поэтому выставляется смещение 100h.

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....0ГQЪ
000000b0	e0	e8	ef	ff	86	c4	b1	04	d2	e8	e8	e6	ff	59	c3	53	аипя†Д±.ТиияYTS
000000c0	8a	fc	e8	e9	ff	88	25	4f	88	05	4f	8a	c7	e8	de	ff	Ъийя€%œ.œЪзиЮя
000000d0	88	25	4f	88	05	5b	c3	51	52	32	e4	33	d2	b9	0a	00	€%œ. [ГQR2дЗТѠ..
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	.рѠА&.юя€.имя<я

Рисунок 4 – .com модуль в бинарном виде

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 (512 байт с адреса 0 – заголовок и relocation table, 256 байт – смещение).

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	MZ3..... .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	йI.PC type: \$PC
00000310	0a	24	50	43	2f	58	54	20	0a	24	41	54	20	0a	24	50	.\$PC/XT .SAT .\$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	le . \$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....0ГQБ
000003b0	e0	e8	ef	ff	86	c4	b1	04	d2	e8	e8	e6	ff	59	c3	53	аипя†Д†.ТиияяYTS
000003c0	8a	fc	e8	e9	ff	88	25	4f	88	05	4f	8a	c7	e8	de	ff	Ъиййяе%ОЕ.ОЪзиЮя
000003d0	88	25	4f	88	05	5b	c3	51	52	32	e4	33	d2	b9	0a	00	е%ОЕ. [ГQР2д3ТМ..
000003e0	f7	f1	80	ca	30	88	14	4e	33	d2	3d	0a	00	73	f1	3c	чсЪК0Е.N3T=...sc<
000003f0	00	74	04	0c	30	88	04	5a	59	c3	b4	09	cd	21	c3	b8	.t..0Е.ZYTr.N!Гё
00000400	00	f0	8e	c0	26	a0	fe	ff	ba	03	01	e8	ec	ff	3c	ff	.рЪА&.юяе..имя<я
00000410	75	06	ba	0d	01	eb	4f	90	3c	fe	74	04	3c	fb	75	06	u.e...ло.<ют.<ьу.
00000420	ba	12	01	eb	41	90	3c	fc	75	06	ba	1a	01	eb	37	90	e...ла.<ьу.e...л7.
00000430	3c	fa	75	06	ba	1f	01	eb	2d	90	3c	f8	75	06	ba	2e	<ьу.e...л-<шу.e.
00000440	01	eb	23	90	3c	fd	75	06	ba	3d	01	eb	19	90	3c	f9	.л#. <ьу.e=.л..<щ
00000450	75	06	ba	44	01	eb	0f	90	ba	55	01	e8	50	ff	bf	55	u.eD.л..eU.иРяiU
00000460	01	83	c7	0e	89	05	e8	91	ff	c3	b4	30	cd	21	be	74	.fз.%.и'яГр0H!st
00000470	01	e8	63	ff	83	c6	03	8a	c4	e8	5b	ff	ba	67	01	e8	.исягЖ.Ъди[яег.и

Рисунок 5 – Плохой .exe модуль в бинарном виде

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

В отличии 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	МЗЙ..... ..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

Рисунок 6 – 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**. В связи с этим код начинается с адреса CS:100h.
2. Что располагается с адреса 0?
PSP (англ. Program Segment Prefix) размером в 256 байт, резервируемый операционной системой.
3. Какие значение 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 (см. рисунок 9).

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 стек?

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

4. Как определяется точка входа?

Точка входа определяется при помощи директивы END.

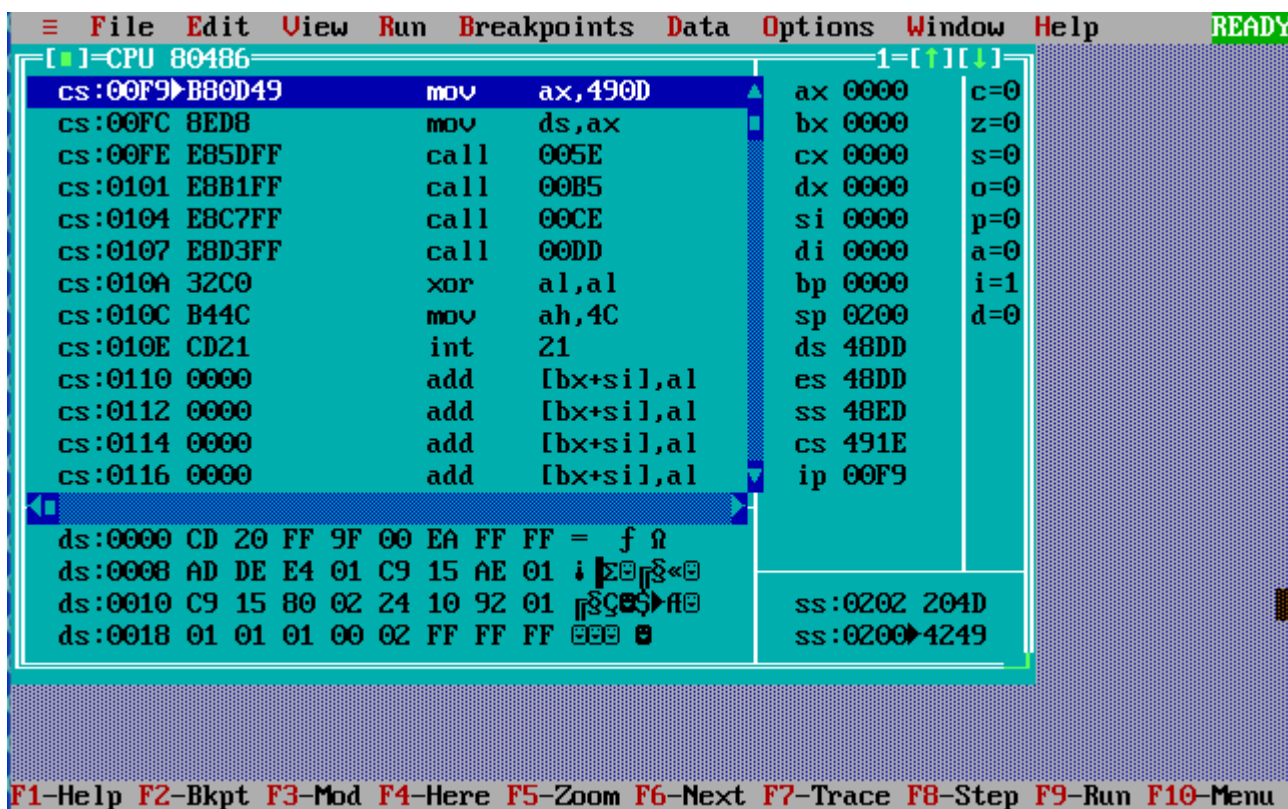


Рисунок 7 – Начальное состояние загруженного .exe модуля

Вывод.

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

ПРИЛОЖЕНИЕ А.

Исходный код модулей

com.asm:

```
TESTPC Segment
    Assume CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN
```

; Данные

```
pc_type_message db 'PC type: ', '$'
pc_model_message db 'PC ', 0ah, '$'
pc_xt_model_message db 'PC/XT ', 0ah, '$'
pc_at_model_message db 'AT ', 0ah, '$'
pc_ps2_model_30_message db 'PS2 model 30 ', 0ah, '$'
pc_ps2_model_80_message db 'PS2 model 80 ', 0ah, '$'
pc_jr_model_message db 'PCjr ', 0ah, '$'
pc_convertible_model_message db 'PC Convertible ', 0ah, '$'
unknown_message db 'Unknown byte: ', 0ah, '$'
dos_version_message db 'DOS version: . ', 0ah, '$'
oem_message db 'OEM number: ', 0ah, '$'
user_number_message db 'User number: ', '$'
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

```

```

print PROC NEAR
    ; процедура вывода
    ; dx - смещение сообщения
    mov ah, 09h
    int 21h
    ret
print ENDP

```

type_model_type PROC NEAR ; процедура выводит на экран тип модели
компьютера.

```

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

    mov dx, offset pc_type_message
    call print

```

```

pc_model: ; сравнение с каждым типом
    cmp al, 0FFh
    jne pc_xt_model

```



```

    mov dx, offset pc_model_message
    jmp _out

pc_xt_model:
    cmp al, 0FEh
    je pc_xt_process

    cmp al, 0FBh
    jne pc_at_model

pc_xt_process:
    mov dx, offset pc_xt_model_message
    jmp _out

pc_at_model:
    cmp al, 0FCh
    jne pc_ps2_model_30

    mov dx, offset pc_at_model_message
    jmp _out

pc_ps2_model_30:
    cmp al, 0FAh
    jne pc_ps2_model_80

    mov dx, offset pc_ps2_model_30_message
    jmp _out

pc_ps2_model_80:
    cmp al, 0F8h
    jne pc_jr_model

    mov dx, offset pc_ps2_model_80_message
    jmp _out

pc_jr_model:
    cmp al, 0FDh
    jne pc_convertible_model

    mov dx, offset pc_jr_model_message
    jmp _out

pc_convertible_model:
    cmp al, 0F9h
    jne unknown_type

    mov dx, offset pc_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 print

```

```

        ret

type_model_type ENDP

type_dos_version PROC NEAR ; печатает версию MS DOS
    mov ah, 30h
    int 21h

    mov si, offset dos_version_message + 13
    call byte_to_dec
    add si, 3
    mov al, ah
    call byte_to_dec
    mov dx, offset dos_version_message
    call print
    ret

type_dos_version ENDP

type_oem PROC NEAR ; печатает серийный номер OEM
    mov si, offset OEM_message + 13
    mov al, bh
    call byte_to_dec
    mov dx, offset OEM_message
    call print
    ret
type_oem ENDP

type_user_number PROC NEAR ; печатает 24-битовый серийный номер
пользователя
    mov dx, offset user_number_message
    call print

    mov ah, 30h
    int 21h

    mov di, offset user_num

    ; bl:cx - серийный номер пользователя
    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 print
    ret

type_user_number ENDP

BEGIN:
    call type_model_type
    call type_dos_version
    call type_oem
    call type_user_number

    ; Выход в DOS

```

```

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

TESTPC  ENDS
        END      START

```

exe.asm:

```

AStack SEGMENT STACK
    DW 256 DUP(?)
AStack ENDS

```

```

DATA SEGMENT
    ; Данные
    pc_type_message db 'PC type: ', '$'
    pc_model_message db 'PC ', 0ah, '$'
    pc_xt_model_message db 'PC/XT ', 0ah, '$'
    pc_at_model_message db 'AT ', 0ah, '$'
    pc_ps2_model_30_message db 'PS2 model 30 ', 0ah, '$'
    pc_ps2_model_80_message db 'PS2 model 80 ', 0ah, '$'
    pc_jr_model_message db 'PCjr ', 0ah, '$'
    pc_convertible_model_message db 'PC Convertible ', 0ah, '$'
    unknown_message db 'Unknown byte: ', 0ah, '$'
    dos_version_message db 'DOS version: . ', 0ah, '$'
    oem_message db 'OEM number: ', 0ah, '$'
    user_number_message db 'User number: ', '$'
    user_num db ' : ', 0ah, '$'
DATA ENDS

```

```

TESTPC Segment
    Assume CS:TESTPC, 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

```

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 PROC NEAR ; процедура вывода
; dx - смещение сообщения
    mov ah, 09h
    int 21h
    ret
print ENDP

```

```

type_model_type PROC NEAR ; процедура выводит на экран тип модели
; персонального компьютера.
    mov ax, 0F000h
    mov es, ax
    mov al, es:[0FFFEh]

    mov dx, offset pc_type_message
    call print

```

```

pc_model: ; сравнение с каждым типом
    cmp al, 0FFh
    jne pc_xt_model

    mov dx, offset pc_model_message
    jmp _out

pc_xt_model:
    cmp al, 0FEh
    je pc_xt_process

    cmp al, 0FBh
    jne pc_at_model

    pc_xt_process:
        mov dx, offset pc_xt_model_message
        jmp _out

pc_at_model:
    cmp al, 0FCh
    jne pc_ps2_model_30

    mov dx, offset pc_at_model_message
    jmp _out

pc_ps2_model_30:
    cmp al, 0FAh
    jne pc_ps2_model_80

    mov dx, offset pc_ps2_model_30_message
    jmp _out

pc_ps2_model_80:
    cmp al, 0F8h
    jne pc_jr_model

    mov dx, offset pc_ps2_model_80_message
    jmp _out

pc_jr_model:
    cmp al, 0FDh
    jne pc_convertible_model

    mov dx, offset pc_jr_model_message
    jmp _out

pc_convertible_model:
    cmp al, 0F9h
    jne unknown_type

    mov dx, offset pc_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 print
    ret

type_model_type ENDP

type_dos_version PROC NEAR ; печатает версию MS DOS
    mov ah, 30h
    int 21h

    mov si, offset dos_version_message + 13
    call byte_to_dec
    add si, 3
    mov al, ah
    call byte_to_dec
    mov dx, offset dos_version_message
    call print
    ret

type_dos_version ENDP

type_oem PROC NEAR ; печатает серийный номер OEM
    mov si, offset OEM_message + 13
    mov al, bh
    call byte_to_dec
    mov dx, offset OEM_message
    call print
    ret
type_oem ENDP

type_user_number PROC NEAR ; печатает 24-битовый серийный номер
пользователя
    mov dx, offset user_number_message
    call print

    mov ah, 30h
    int 21h

    mov di, offset user_num

    ; bl:cx - серийный номер пользователя
    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 print
    ret

type_user_number ENDP

main PROC FAR
    mov ax, data
    mov ds, ax

```

```
call type_model_type
call type_dos_version
call type_oem
call type_user_number

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

TESTPC ENDS
      END      main
```