

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

Шангичев В. А.

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.

Задание.

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.

В начале выполнения данной лабораторной работы был написан файл исходного кода `com_file.asm`. Цель написания кода в данном файле – получение общей информации о модели компьютера. Программа компилируется и компоуется с помощью утилит `masm` и `link`, а затем подается на вход утилите `exe2bin` для получения файла типа `.com`. При запуске файла `com_file.com` был получен следующий вывод:



```
C:\DOS>com_file.com
PC type: AT
DOS version: 5.0
OEM number: 0
User number: 00:0000

C:\DOS>_
```

Если же запустить программу с типом .exe, то в консоли появятся следующие символы:

```
C:\DOS>com_file.exe

    PC type:

    PC type:      5 0
    PC type:      0

    PC type:
                                PC type: 00 0000

    PC type:
C:\DOS>
```

Для получения корректного вывода была написана программа exe_file.exe, в исходном коде которой была добавлена разметка сегментов. Вывод данной программы:

```
C:\DOS>exe_file.exe
PC type: AT
DOS version: 5.0
OEM number: 0
User number: 00:0000

C:\DOS>_
```

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 обязательно быть 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?

Не все форматы поддерживаются. Нельзя использовать команды вида mov <регистр>, seg <имя сегмента>, так как в .com-программе отсутствует таблица настроек (содержит описание адресов, которые зависят от размещения загрузочного модуля в ОП).

Отличия форматов файлов .COM и .EXE модулей.

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

Файл данного вида содержит один сегмент, в котором расположен как код, так и данные. Код начинается с адреса 0.

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±.ТиияяYGS
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€. [ГQR2д3ТН..
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€.ZYГr.Н!Гё
00000100	00	f0	8e	c0	26	a0	fe	ff	ba	03	01	e8	ec	ff	3c	ff	.рЪА&.юя€.имя<я

2. Какова структура “плохого” EXE? С какого адреса располагается код? Что располагается с адреса 0?

Данный файл состоит из заголовка, таблицы настройки адресов и единственного сегмента. Первые два байта – буквы М и Z – обозначают начало заголовка. Данные байты подобраны так, что машинные команды, соответствующие им, не могут быть расположены в начале корректной .COM программы. Таким образом, буквы М и Z (первые буквы имени и фамилии лидирующего разработчика MS-DOS) в начале программы однозначно идентифицируют .EXE файл, и не могут быть спутаны с началом машинных команд .COM файла.

Код располагается с адреса 300h, так как перед этим 200h занимает заголовок и таблица настроек, и еще 100h – смещение, прописанное директивой org.

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 . $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 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....0TQЪ
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 Ъийя€%œ.ОЪзиЮя
000003d0 88 25 4f 88 05 5b c3 51 52 32 e4 33 d2 b9 0a 00 €%œ. [ГQR2дЗТ№..
000003e0 f7 f1 80 ca 30 88 14 4e 33 d2 3d 0a 00 73 f1 3c чсЪK0€.N3T=..sc<
000003f0 00 74 04 0c 30 88 04 5a 59 c3 b4 09 cd 21 c3 b8 .t...0€.ZYTr.Н!Гё
00000400 00 f0 8e c0 26 a0 fe ff ba 03 01 e8 ec ff 3c ff .рђА&.юяе..имя<я
00000410 75 06 ba 0d 01 eb 4f 90 3c fe 74 04 3c fb 75 06 u.e...ло.<ют.<ьи.
00000420 ba 12 01 eb 41 90 3c fc 75 06 ba 1a 01 eb 37 90 e...ла.<ьи.e...л7.
00000430 3c fa 75 06 ba 1f 01 eb 2d 90 3c f8 75 06 ba 2e <ьи.e...л-<шу.e.
00000440 01 eb 23 90 3c fd 75 06 ba 3d 01 eb 19 90 3c f9 .л#. <ьи.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 .ѓз.%и'яГr0H!st
00000470 01 e8 63 ff 83 c6 03 8a c4 e8 5b ff ba 67 01 e8 .исяѓЖ.Љди[яег.и

```

3. Какова структура файла “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	..XT..*.....
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

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?

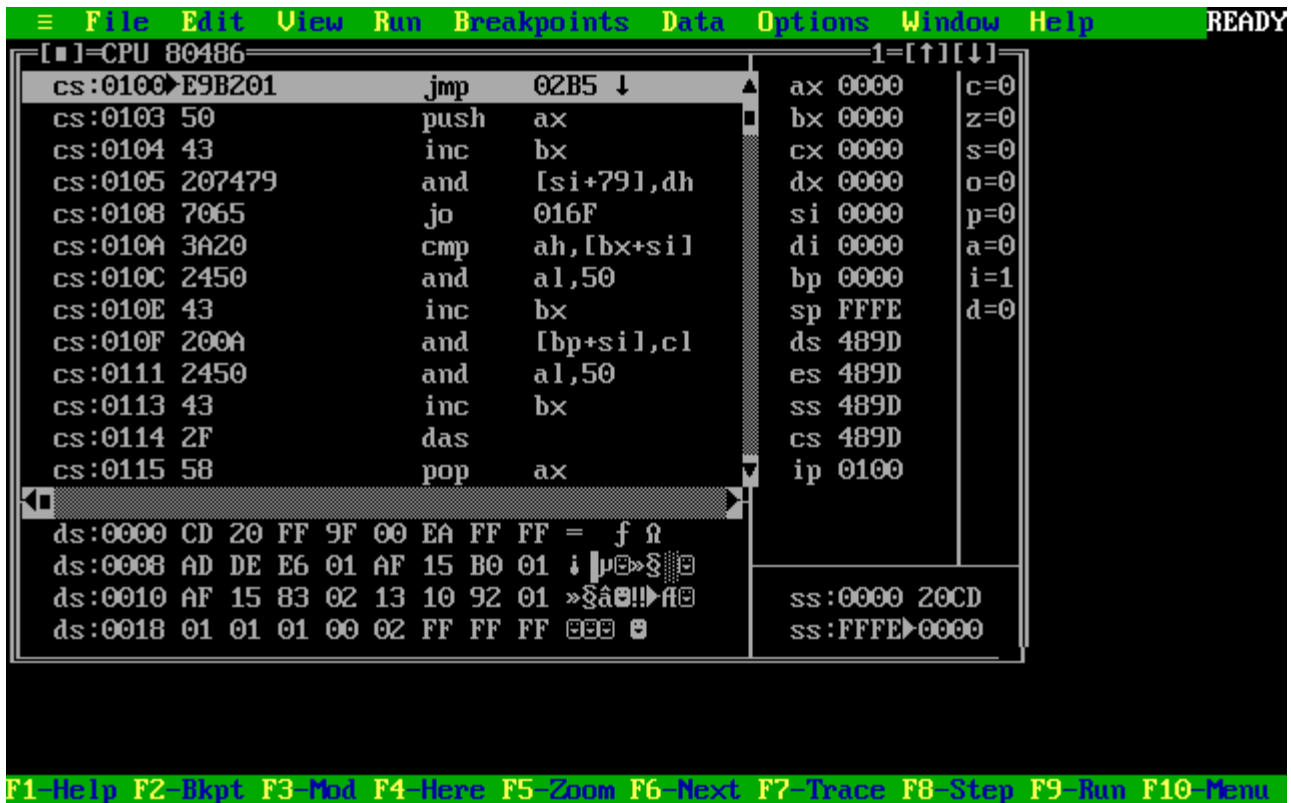
PSP (англ. Program Segment Prefix) размером в 256 байт, резервируемый операционной системой.

3. Какие 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?

Так как в этой программе всего один сегмент, то и стек располагается в этом же сегменте (ss=cs=ds=489D). SP указывает на последний адрес, кратный двум (FFFE). То есть стек занимает адреса с 0h по FFFeh.



```
File Edit View Run Breakpoints Data Options Window Help
[CPU 80486] 1=[↑][↓]
cs:0100>E9B201 jmp 02B5 ↓
cs:0103 50 push ax
cs:0104 43 inc bx
cs:0105 207479 and [si+79],dh
cs:0108 7065 jo 016F
cs:010A 3A20 cmp ah,[bx+si]
cs:010C 2450 and al,50
cs:010E 43 inc bx
cs:010F 200A and [bp+si],cl
cs:0111 2450 and al,50
cs:0113 43 inc bx
cs:0114 2F das
cs:0115 58 pop ax
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 489D
es 489D
ss 489D
cs 489D
ip 0100
ds:0000 CD 20 FF 9F 00 EA FF FF = f Ω
ds:0008 AD DE E6 01 AF 15 B0 01 ↓ μΩ§Ω
ds:0010 AF 15 83 02 13 10 92 01 »§âΩ!!Ω#Ω
ds:0018 01 01 01 00 02 FF FF FF ΩΩΩ Ω
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
```

Загрузка “хорошего” .exe модуля в основную память:

1. Как загружается “хороший” .exe? Какие значения имеют сегментные регистры?

Сначала берется свободный сегмент памяти, и в него загружается PSP. Затем используется заголовок файла .EXE для его загрузки в память. Значения регистров можно видеть на скриншоте ниже.

2. На что указывают регистры DS и ES?

На сегмент, в котором расположен PSP.

3. Как определяется стек?

Стек определяется парой регистров ss и sp. SS указывает на сегмент стека, а sp на его вершину. В программе под стек было выделено 256 байт, и, как можно видеть, значение sp в начале выполнения программы равно 200h.

4. Как определяется точка входа?

Адрес метки, указанной после директивы END.

The screenshot shows the DOS DEBUG program interface. The menu bar at the top includes File, Edit, View, Run, Breakpoints, Data, Options, Window, and Help. The status bar at the bottom shows function key shortcuts: F1-Help, F2-Bkpt, F3-Mod, F4-Here, F5-Zoom, F6-Next, F7-Trace, F8-Step, F9-Run, and F10-Menu.

The main window is divided into several sections:

- Assembly List:** Displays instructions with their addresses and hex values. The current instruction is at address 0112: `cs:0112 B8CD48 mov ax,48CD`.
- Registers:** A table showing the current values of 16-bit registers:

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	0200	d	=0
ds	489D		
es	489D		
ss	48AD		
cs	48D7		
ip	0112		
- Memory Dump:** Shows a hex dump of memory starting at address 0000. The first line is `ds:0000 CD 20 FF 9F 00 EA FF FF = f Ω`.
- Stack Pointer:** At the bottom right, it shows `ss:0202 7420` and `ss:0200 4350`.

Выводы.

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 ПРОГРАММЫ

Файл com_file.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
        ; персонального компьютера.
        mov ax, 0F000h
        mov es, ax
        mov al, es:[0FFFEh]

        mov dx, offset pc_type_message
        call print

```

```

; сравнение с каждым типом
pc_model:
    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

```

Файл exe_file.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

;-----

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
    ; персонального компьютера.
    mov ax, 0F000h
    mov es, ax
    mov al, es:[0FFFEh]

    mov dx, offset pc_type_message
    call print

    ; сравнение с каждым типом
pc_model:
    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
```