

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

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

Студент гр. 8381

Преподаватель

Облизов А.Д.

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

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

Основные теоретические положения.

Тип IBM PC хранится в байте по адресу 0F000:0FFFFeh, в предпоследнем байте ROM BIOS. Соответствие кода и типа представлены в табл.1.

Таблица 1 – Соответствие кода и типа PC

Тип IBM PC	Код
PC	FF
PC/XT	FE, FB
AT	FC
PS2, модель 30	FA
PS2, модель 50 или 60	FC
PS2, модель 80	F8
PCjr	FD
PC Convertible	F9

Для определения версии MS DOS следует воспользоваться функцией 30H прерывания 21H. Входным параметром является номер функции в AH. Выходными параметрами являются:

- AL - номер основной версии. Если 0, то < 2.0
- AH - номер модификации
- BH - серийный номер OEM (Original Equipment Manufacturer)
- BL:CH - 24-битовый серийный номер пользователя.

Выполнение работы.

Написание работы производилось на базе операционной системы Windows 10 в редакторе Visual Code. Сборка, отладка производились в эмуляторе DOS DOSBox 0.74-3, а также с помощью Far Manager.

Был написан текст исходного .COM модуля, который определяет тип PC и информацию о системе. Полученный модуль был отлажен, и в результате были получены «плохой» .EXE модуль и, с помощью программы EXE2BIN, «хороший» .COM модуль. Во время линковки было выведено предупреждение об отсутствии сегмента стека, представленное на рис. 1.

```
D:\>link com.obj

Microsoft (R) Overlay Linker  Version 3.64
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Run File [COM.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment
```

Рисунок 1 – Предупреждение во время линковки

Результат выполнения «плохого» .EXE модуля представлен на рис. 2.

```
D:\>com.exe

  0=PC
0=PC
    5 0
    0
    00 0000
    0=PC
                                0=PC

D:\>_
```

Рисунок 2 – Вывод «плохого» .EXE модуля

Результат выполнения «хорошего» .COM модуля представлен на рис. 3.

```
D:\>com.com
IBM PC type is: AT
MSDOS version is: 5.0
OEM number is: 0
Serial number is: 00 0000
```

Рисунок 3 – Вывод «хорошего» .COM модуля

Был написан текст исходного .EXE модуля, который выполняет те же функции, что и .COM модуль. В результате постройки и отладки был получен «хороший» .EXE модуль. Результат его выполнения представлен на рис. 4.

```
D:\>exe.exe
IBM PC type is: AT
MSDOS version is: 5.0
OEM number is: 0
Serial number is: 00 0000
```

Рисунок 4 – Вывод «хорошего» .EXE модуля

Отличия исходных текстов COM и EXE программ

1. Сколько сегментов должна содержать COM-программа?

COM-программа должна содержать только один сегмент (модель памяти tiny)

2. EXE-программа?

EXE-программа может содержать несколько сегментов. При использовании модели памяти small, в программе должен содержаться один сегмент данных и один сегмент кода. При других моделях памяти (напр. large) есть возможность использования нескольких сегментов данных и (или) нескольких сегментов кода. Помимо этого, в программе может быть описан сегмент стека – стек в любом случае использует операционная система при обработке прерываний, и создаст его автоматически, если он не был описан.

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

При запуске COM-программы первые 100h байт необходимо зарезервировать для префикса программного сегмента (PSP). Для этого используется директива ORG, которая устанавливает относительный адрес для начала выполнения программы: **ORG 100h.**

При компиляции COM модуля без директивы ASSUME возникают ошибки – эта директива обязательна, чтобы сегментные регистры CS, DS указывали на адрес сегмента программы. Ошибки представлены на рис. 5.

```
C:\MASM>masm com.asm
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All rights reserved

Object filename [com.OBJ]:
Source listing [NUL.LST]:
Cross-reference [NUL.CRF]:
com.asm(4): error A2062: Missing or unreachable CS
com.asm(25): error A2062: Missing or unreachable CS
com.asm(33): error A2062: Missing or unreachable CS
com.asm(38): error A2062: Missing or unreachable CS
com.asm(42): error A2062: Missing or unreachable CS
com.asm(54): error A2062: Missing or unreachable CS
com.asm(72): error A2062: Missing or unreachable CS
com.asm(79): error A2062: Missing or unreachable CS
com.asm(90): error A2062: Missing or unreachable CS
com.asm(96): error A2062: Missing or unreachable CS
com.asm(126): error A2062: Missing or unreachable CS
com.asm(130): error A2062: Missing or unreachable CS
com.asm(134): error A2062: Missing or unreachable CS
com.asm(138): error A2062: Missing or unreachable CS
com.asm(142): error A2062: Missing or unreachable CS
com.asm(146): error A2062: Missing or unreachable CS
com.asm(150): error A2062: Missing or unreachable CS
com.asm(154): error A2062: Missing or unreachable CS
com.asm(158): error A2062: Missing or unreachable CS
com.asm(161): error A2062: Missing or unreachable CS
com.asm(165): error A2062: Missing or unreachable CS
com.asm(173): error A2062: Missing or unreachable CS
com.asm(177): error A2062: Missing or unreachable CS
com.asm(183): error A2062: Missing or unreachable CS
com.asm(187): error A2062: Missing or unreachable CS
com.asm(198): error A2062: Missing or unreachable CS
com.asm(202): error A2062: Missing or unreachable CS

49138 + 436761 Bytes symbol space free

0 Warning Errors
27 Severe Errors

C:\MASM>
```

Рисунок 5 – Ошибки при компиляции при отсутствии ASSUME

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

В .COM файле отсутствует таблица настройки с информацией о типе адресов и их местоположении в коде. Поэтому нельзя использовать команды, связанные с адресом сегмента, так как адрес сегмента неизвестен вплоть до загрузки этого сегмента в память. Загрузчику необходима информация о местоположении в файле загрузочного модуля полей адресов.

Отличия форматов файлов COM и EXE модулей.

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

Вид файла COM в шестнадцатеричном формате представлен на рис. 6.

D:\OS\COM.EXE																	
000000000:	4D	5A	D6	00	03	00	00	00	00	00	00	00	FF	FF	00	00	MZÖ ▼ ŷŷ
000000010:	00	00	00	23	69	00	01	00	00	1E	00	00	00	01	00	00	#i 0 ▲ 0
000000020:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000030:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000040:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000050:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000060:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000070:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000080:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000090:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000A0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000B0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000C0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000D0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000E0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000F0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000100:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000110:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000120:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000130:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000140:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000150:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000160:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000170:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000180:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000190:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000001A0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000001B0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000001C0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000001D0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000001E0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000001F0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000200:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000210:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000220:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000230:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000240:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000250:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000260:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000270:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000280:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000290:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000002A0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000002B0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000002C0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000002D0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000002E0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000002F0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000300:	E9	16	01	50	43	0D	0A	24	50	43	2F	58	54	0D	0A	24	é=0PC;w\$PC/XT;w\$
000000310:	41	54	0D	0A	24	50	53	32	20	6D	6F	64	65	6C	20	33	AT;w\$PS2 model 3
000000320:	30	0D	0A	24	50	53	32	20	6D	6F	64	65	6C	20	35	30	0;w\$PS2 model 50
000000330:	2F	36	30	0D	0A	24	50	53	32	20	6D	6F	64	65	6C	20	/60;w\$PS2 model
000000340:	38	30	0D	0A	24	50	53	6A	72	0D	0A	24	50	43	20	63	80;w\$Psr;w\$PC c
000000350:	6F	6E	76	65	72	74	69	62	6C	65	0D	0A	24	49	42	4D	onvertible;w\$IBM
000000360:	20	50	43	20	74	79	70	65	20	69	73	3A	20	24	4D	53	PC type is: \$MS
000000370:	44	4F	53	20	76	65	72	73	69	6F	6E	20	69	73	3A	20	DOS version is:
000000380:	20	2E	20	0D	0A	24	4F	45	4D	20	6E	75	6D	62	65	72	. ;w\$OEM number
000000390:	20	69	73	3A	20	20	20	20	20	0D	0A	24	53	65	72	69	is: ;w\$Seri
0000003A0:	61	6C	20	6E	75	6D	62	65	72	20	69	73	3A	20	20	20	al number is:
0000003B0:	20	20	20	0D	0A	24	50	B4	09	CD	21	58	C3	24	0F	3C	;w\$P`oi!XASo<
0000003C0:	09	76	02	04	07	04	30	C3	51	8A	C4	E8	EF	FF	86	C4	ov0*+0AQ\$AeiYtA
0000003D0:	B1	04	D2	E8	E8	E6	FF	59	C3	53	8A	FC	E8	E9	FF	88	±+0èèyYAS\$üèéy`
0000003E0:	25	4F	88	05	4F	8A	C7	32	E4	E4	E8	DC	FF	88	25	4F	%0*+0\$C2aèÜy`%0`
0000003F0:	05	5B	C3	51	52	50	32	E4	33	D2	B9	0A	00	F7	F1	80	+[AQRP2a30*# ÷nè
000000400:	CA	30	88	14	4E	33	D2	3D	0A	00	73	F1	3D	00	00	76	E0*JN30= sñ= v
000000410:	04	0C	30	88	04	58	5A	59	C3	52	50	BA	5D	01	E8	95	♦00`♦XZYARP*]0è♦
000000420:	FF	B8	00	F0	8E	C0	26	A0	FE	FF	3C	FF	74	20	3C	FE	ŷ. ðZÅè þy<y't <þ
000000430:	74	22	3C	FB	74	1E	3C	FC	74	20	3C	FA	74	22	3C	FC	t"<ùtA<ùt <ùt"<ü
000000440:	74	24	3C	F8	74	26	3C	FD	74	28	3C	F9	74	2A	BA	03	t\$<øt&<y't(<ùt*øv
000000450:	01	EB	2B	90	BA	08	01	EB	25	90	BA	10	01	EB	1F	90	0è+0ø0ø0ø0ø0ø0ø0
000000460:	BA	15	01	EB	19	90	BA	24	01	EB	13	90	BA	36	01	EB	ø\$øè0øøøøøøøøøø
000000470:	0D	90	BA	45	01	EB	07	90	BA	4C	01	EB	01	90	E8	35	J0øE0èøøøøøøøøøø
000000480:	FF	B4	30	CD	21	8D	36	6E	01	83	C6	12	E8	64	FF	83	y`0!06n0fAteidyf
000000490:	C6	03	8A	C4	E8	5C	FF	BA	6E	01	E8	19	FF	8A	C7	8D	Aw\$Aè\y`n0èiy\$C0
0000004A0:	36	86	01	83	C6	0F	E8	4A	FF	BA	86	01	E8	07	FF	8A	6t0fAèèJy`t0è♦y\$
0000004B0:	C3	8D	36	9C	01	83	C6	12	E8	0D	FF	89	04	83	C6	06	Å06ø0fAteiyè♦fA♦
0000004C0:	8B	FE	8B	C1	E8	12	FF	BA	9C	01	E8	E9	FE	58	5A	32	<þAèty`øøèèþXZ2
0000004D0:	C0	B4	4C	CD	21	C3			A`LI!A								

Рисунок 8 – Вид «плохого» EXE файла

В «плохом» EXE-файле код располагается с адреса 300h. С нулевого адреса располагается управляющая информация для загрузчика. В управляющую информацию входит таблица настроек, но также и другая информация.

Также 100h резервируются командой ORG 100h (что вызовет отличие в структуре «плохого» и «хорошего» EXE)

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

Вид файла «хорошего» EXE в шестнадцатеричном виде представлен на рис. 9. Ввиду большого числа пустых строк на рисунке представлена только часть файла.

D:\OS\EXE.EXE	
00000002F0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000300: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000310: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000320: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000330: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000340: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000350: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000360: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000370: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000380: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000390: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000003A0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000003B0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000003C0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000003D0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000003E0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000003F0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000400: 50 43 0D 0A 24 50 43 2F	58 54 0D 0A 24 41 54 0D PC!\$PC/XT!\$AT!
0000000410: 0A 24 50 53 32 20 6D 6F	64 65 6C 20 33 30 0D 0A \$PS2 model 30!
0000000420: 24 50 53 32 20 6D 6F 64	65 6C 20 35 30 2F 36 30 \$PS2 model 50/60
0000000430: 0D 0A 24 50 53 32 20 6D	6F 64 65 6C 20 38 30 0D !\$PS2 model 80!
0000000440: 0A 24 50 53 6A 72 0D 0A	24 50 43 20 63 6F 6E 76 \$PSjr!\$PC conv
0000000450: 65 72 74 69 62 6C 65 0D	0A 24 49 42 4D 20 50 43 ertible!\$IBM PC
0000000460: 20 74 79 70 65 20 69 73	3A 20 24 4D 53 44 4F 53 type is: \$MSDOS
0000000470: 20 76 65 72 73 69 6F 6E	20 69 73 3A 20 20 2E 20 version is: .
0000000480: 0D 0A 24 4F 45 4D 20 6E	75 6D 62 65 72 20 69 73 !\$OEM number is
0000000490: 3A 20 20 20 20 20 0D 0A	24 53 65 72 69 61 6C 20 : !\$Serial
00000004A0: 6E 75 6D 62 65 72 20 69	73 3A 20 20 20 20 20 20 number is:
00000004B0: 0D 0A 24 00 00 00 00 00	!\$
00000004C0: 50 B4 09 CD 21 58 C3 24	0F 3C 09 76 02 04 07 04 P'oi!X\$so<ov0♦♦
00000004D0: 30 C3 51 8A C4 E8 EF FF	86 C4 B1 04 D2 E8 E8 E6 0AQ\$Aeiŷt!A±♦0èèè
00000004E0: FF 59 C3 53 8A FC E8 E9	FF 88 25 4F 88 05 4F 8A ŷYAS\$üèèŷ~%0~*OS
00000004F0: C7 32 E4 E8 DC FF 88 25	4F 88 05 5B C3 51 52 50 Ç2äèÜŷ~%0~*[AQRP
0000000500: 32 E4 33 D2 B9 0A 00 F7	F1 80 CA 30 88 14 4E 33 2ä30!è ÷ñ€È0~JN3
0000000510: D2 3D 0A 00 73 F1 3D 00	00 76 04 0C 30 88 04 58 0=è sñ= v♦00~*X
0000000520: 5A 59 C3 52 50 B8 20 00	8E D8 BA 5A 00 E8 90 FF ZYARP, Ž0Z è8ŷ
0000000530: B8 00 F0 8E C0 26 A0 FE	FF 3C FF 74 20 3C FE 74 . ðŽA& bŷ<ŷt <bt
0000000540: 22 3C FB 74 1E 3C FC 74	20 3C FA 74 22 3C FC 74 "<ût!<ût <ût"<ût
0000000550: 24 3C F8 74 26 3C FD 74	28 3C F9 74 2A BA 00 00 \$<øt&<ŷt(<ût*ø
0000000560: EB 2B 90 BA 05 00 EB 25	90 BA 0D 00 EB 1F 90 BA è+0ø+ è%0ø! è▼0ø
0000000570: 12 00 EB 19 90 BA 21 00	EB 13 90 BA 33 00 EB 0D ‡ èŷ0ø! è!!0ø3 è!
0000000580: 90 BA 42 00 EB 07 90 BA	49 00 EB 01 90 E8 30 FF 0øB è•0øI è00è0ŷ
0000000590: B4 30 CD 21 8D 36 6B 00	83 C6 12 E8 5F FF 83 C6 '0Í!06k fA†è_ŷfA
00000005A0: 03 8A C4 E8 57 FF BA 6B	00 E8 14 FF 8A C7 8D 36 ♥\$AèWŷøk èJŷ\$Ç06
00000005B0: 83 00 83 C6 0F E8 45 FF	BA 83 00 E8 02 FF 8A C3 f fAèèEŷøf è0ŷ\$A
00000005C0: 8D 36 99 00 83 C6 12 E8	08 FF 89 04 83 C6 06 8B 06™ fA†èJŷ%♦fA<
00000005D0: FE 8B C1 E8 0D FF BA 99	00 E8 E4 FE 58 5A 32 C0 b<AèJŷø™ èäpXZ2A
00000005E0: B4 4C CD 21 CB	ˆLÍ!È

Рисунок 9 – Вид «хорошего» EXE файла

В «хорошем» EXE с нулевого адреса также располагается таблица настроек (управляющая информация для загрузчика). Также перед кодом располагается сегмент стека. Так, при размере стека 200h код располагается с адреса 400h. Если из исходного текста .EXE-программы убрать сегмент стека, то код будет располагаться с адреса 200h. Отличие от «плохого» EXE в том, что в «хорошем» не резервируется дополнительно 100h, которые в COM файле требовались для PSP, поэтому адреса начала кода отличаются на $100h + S$, где S – размер стека.

Загрузка COM модуля в основную память.

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

Запуск файла .COM в отладчике TD.EXE представлен на рис. 10.

The screenshot shows the TD.EXE debugger interface. The main window displays assembly code with addresses, hex values, and mnemonics. The right window shows the state of various registers.

Address	Hex	Mnemonic	Register	Value
cs:0100	E91601	jmp	0219	↓
cs:0103	50	push	ax	
cs:0104	43	inc	bx	
cs:0105	0D0A24	or	ax, 240A	
cs:0108	50	push	ax	
cs:0109	43	inc	bx	
cs:010A	2F	das		
cs:010B	58	pop	ax	
cs:010C	54	push	sp	
cs:010D	0D0A24	or	ax, 240A	
cs:0110	41	inc	cx	
cs:0111	54	push	sp	
cs:0112	0D0A24	or	ax, 240A	

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

Segment	Address	Hex	Value
ds	0000	CD 20 FF 9F 00 EA FF FF	= f 0
ds	0008	AD DE E4 01 C9 15 AE 01	↓ 20 18 00
ds	0010	C9 15 80 02 24 10 92 01	↓ 00 00 00
ds	0018	01 01 01 00 02 FF FF FF	000 0

Segment	Address	Value
ss	0000	20CD
ss	FFFE	0000

Рисунок 10 – Отладка файла .COM

После загрузки COM-программы в память сегментные регистры указывают на начало PSP. Код располагается с адреса 100h, регистр IP смещения следующей команды также имеет значение 100h.

2. Что располагается с адреса 0?

С нулевого адреса располагается сегмент PSP

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

Сегментные регистры имеют значения 48DDh и указывают на PSP.

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

В COM модуле нельзя объявить стек, он создается автоматически. На рис. 8 видно, что SP указывает на FFFEH. Стек занимает оставшуюся память, а его адреса изменяются от больших к меньшим, то есть от FFFEH к 0000h.

Загрузка «хорошего» EXE модуля в основную память.

Запуск «хорошего» EXE модуля в отладчике TD.EXE представлен на рис.

11.

The screenshot shows a debugger window with two main panes. The left pane displays assembly code with addresses and instructions. The right pane shows the current values of the 16-bit registers. Below the assembly list, there is a memory dump showing hex values and their ASCII representations.

Address	Instruction
cs:0063	52 push dx
cs:0064	50 push ax
cs:0065	B80D49 mov ax, 490D
cs:0068	8ED8 mov ds, ax
cs:006A	BA5A00 mov dx, 005A
cs:006D	E890FF call 0000
cs:0070	B800F0 mov ax, F000
cs:0073	8EC0 mov es, ax
cs:0075	26A0FEFF mov al, es:[FFFE]
cs:0079	3CFF cmp al, FF
cs:007B	7420 je 009D
cs:007D	3CFE cmp al, FE
cs:007F	7422 je 00A3

Register	Value
ax	0000
bx	0000
cx	0000
dx	0000
si	0000
di	0000
bp	0000
sp	0200
ds	48DD
es	48DD
ss	48ED
cs	4919
ip	0063

Address	Hex	ASCII
ds:0000	CD 20 FF 9F 00 EA FF FF	= f Ω
ds:0008	AD DE E4 01 C9 15 AE 01	i 20 78 <0
ds:0010	C9 15 80 02 24 10 92 01	SSCS>ff0
ds:0018	01 01 01 00 02 FF FF FF	000 0

Register	Value
ss	0202 0A0D
ss	0200 4350

Рисунок 11 – Отладка «хорошего» EXE модуля

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

Для PSP и программы выделяется блок памяти. После запуска программы DS и ES указывают на начало PSP (48DDh), CS – на начало сегмента команд (4919h), а SS – на начало сегмента стека (48EDh). IP имеет ненулевое значение,

так как в программе есть дополнительные процедуры, расположенные до основной.

2. На что указывают регистры DS и ES?

Изначально регистры DS и ES указывают на начало сегмента PSP. Именно поэтому в начале программы для корректной работы с данными необходимо загрузить в DS адрес сегмента данных.

3. Как определяется стек?

Стек может быть объявлен при помощи директивы ASSUME, которая устанавливает сегментный регистр SS на начало сегмента стека, а также задает значение SP, указанное в заголовке. Также стек может быть объявлен с помощью директивы STACK. Если стек не объявлять, то он будет создан автоматически таким же образом, как в COM-модуле. Вид программы EXE модуля без объявленного стека после команды push в отладчике представлен на рис. 12.

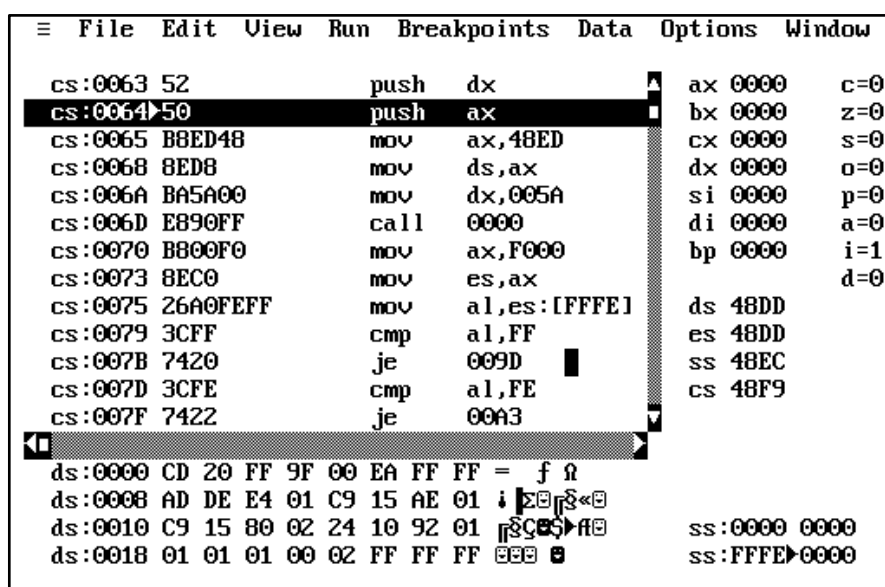


Рисунок 12 – Отладка EXE модуля без объявленного стека

4. Как определяется точка входа?

Смещение точки входа в программу загружается в указатель команд IP и определяется операндом директивы END, который называется точкой входа. Операндом является функция или метка, с которой необходимо начать программу.

Выводы.

В ходе выполнения лабораторной работы были изучены COM и EXE модули и их различия. Так же были получены два «хороших» модуля EXE.EXE и COM.COM, а также один «плохой» модуль COM.EXE.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. COM.ASM

TESTPC SEGMENT

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

ORG 100H ; резервирование места для PSP

START: JMP BEGIN

;DATA SEGMENT

PC_TYPE db 'PC', 0DH, 0AH, '\$'

PC_XT_TYPE db 'PC/XT', 0dh, 0ah, '\$'

AT_TYPE db 'AT', 0dh, 0ah, '\$'

PS2_30_TYPE db 'PS2 model 30', 0dh, 0ah, '\$'

PS2_5060_TYPE db 'PS2 model 50/60', 0dh, 0ah, '\$'

PS2_80_TYPE db 'PS2 model 80', 0dh, 0ah, '\$'

PCjr_TYPE db 'PSjr', 0dh, 0ah, '\$'

PC_CONVERTIBLE db 'PC convertible', 0dh, 0ah, '\$'

IBM_PC_NAME db 'IBM PC type is: ', '\$'

OS_NAME db 'MSDOS version is: . ', 0dh, 0ah, '\$'

OEM_NAME db 'OEM number is: ', 0dh, 0ah, '\$'

SERIAL_NAME db 'Serial number is: ', 0dh, 0ah, '\$'

;DATA ENDS

;CODE SEGMENT

PRINT_STRING PROC near

push AX

mov AH, 09h

int 21h

pop AX

ret

PRINT_STRING ENDP

;-----

--

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

```

        push    cx
        mov     al, ah
        call    TETR_TO_HEX
        xchg    al, ah
        mov     cl, 4
        shr     al, cl
        call    TETR_TO_HEX
        pop     cx
        ret

```

```

BYTE_TO_HEX      ENDP

```

```

;-----
--

```

```

WRD_TO_HEX      PROC    near
        push    bx
        mov     bh, ah
        call    BYTE_TO_HEX
        mov     [di], ah
        dec     di
        mov     [di], al
        dec     di
        mov     al, bh
        xor     ah, ah
        call    BYTE_TO_HEX
        mov     [di], ah
        dec     di
        mov     [di], al
        pop     bx
        ret

```

```

WRD_TO_HEX      ENDP

```

```

;-----
--

```

```

BYTE_TO_DEC      PROC    near
        push    cx
        push    dx
        push    ax
        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     ax, 00h
        jbe     end_1

```



```

        or            al, 30h
        mov          [si], al
end_1:   pop          ax
        pop          dx
        pop          cx
        ret

BYTE_TO_DEC      ENDP

BEGIN:
;PC INFO OUT
        push DX
        push AX

        mov DX, offset IBM_PC_NAME
        call PRINT_STRING

        mov AX, 0F000H
        mov ES, AX
        mov AL, ES:[0FFFEH]
        cmp AL, 0FFh
        je PC_WRITE
        cmp AL, 0FEh
        je PC_XT_WRITE
        cmp AL, 0FBh
        je PC_XT_WRITE
        cmp AL, 0FCh
        je AT_WRITE
        cmp AL, 0FAh
        je PS2_30_WRITE
        cmp AL, 0FCh
        je PS2_5060_WRITE
        cmp AL, 0F8h
        je PS2_80_WRITE
        cmp AL, 0FDh
        je PCjr_WRITE
        cmp AL, 0F9H
        je PC_CONVERTIBLE_WRITE

PC_WRITE:
        mov DX, offset PC_TYPE
        jmp TYPE_WRITE

PC_XT_WRITE:
        mov DX, offset PC_XT_TYPE
        jmp TYPE_WRITE

AT_WRITE:

```

```

        mov DX, offset AT_TYPE
        jmp TYPE_WRITE

PS2_30_WRITE:
        mov DX, offset PS2_30_TYPE
        jmp TYPE_WRITE

PS2_5060_WRITE:
        mov DX, offset PS2_5060_TYPE
        jmp TYPE_WRITE

PS2_80_WRITE:
        mov DX, offset PS2_80_TYPE
        jmp TYPE_WRITE

PCjr_WRITE:
        mov DX, offset PCjr_TYPE
        jmp TYPE_WRITE

PC_CONVERTIBLE_WRITE:
        mov DX, offset PC_CONVERTIBLE
        jmp TYPE_WRITE

TYPE_WRITE:
        call PRINT_STRING

OS_INFO_GET:
        mov AH, 30h
        int 21h

OS_VERSION_SET:
        lea     SI, OS_NAME
        add     SI, 18
        call    BYTE_TO_DEC
        add     SI, 3
        mov     AL, AH
        call    BYTE_TO_DEC

OS_VERSION_WRITE:
        mov DX, offset OS_NAME
        call PRINT_STRING

OEM_SET:
        mov     AL, BH
        lea     SI, OEM_NAME
        add     SI, 15
        call    BYTE_TO_DEC

```

```

OEM_WRITE:
    mov     DX, offset OEM_NAME
    call    PRINT_STRING

SERIAL_SET:
    mov     AL, BL
    lea     SI, SERIAL_NAME
    add     SI, 18
    call    BYTE_TO_HEX
    mov     [SI], AX
    add     SI, 6
    mov     DI, SI
    mov     AX, CX
    call    WRD_TO_HEX

SERIAL_WRITE:
    mov     DX, offset SERIAL_NAME
    call    PRINT_STRING

ENDING:
    pop     AX
    pop     DX

    xor     AL, AL
    mov     AH, 4ch
    int     21h
    ret

TESTPC    ENDS
END       START

```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. EXE.ASM

```
AStack      SEGMENT  STACK
                DW 100h DUP(?)
```

```
AStack      ENDS
```

```
DATA SEGMENT
```

```
    PC_TYPE          db    'PC', 0DH, 0AH, '$'
    PC_XT_TYPE        db    'PC/XT', 0dh, 0ah, '$'
    AT_TYPE           db    'AT', 0dh, 0ah, '$'
    PS2_30_TYPE        db    'PS2 model 30', 0dh, 0ah, '$'
    PS2_5060_TYPE      db    'PS2 model 50/60', 0dh, 0ah, '$'
    PS2_80_TYPE        db    'PS2 model 80', 0dh, 0ah, '$'
    PCjr_TYPE          db    'PSjr', 0dh, 0ah, '$'
    PC_CONVERTIBLE     db    'PC convertible', 0dh, 0ah, '$'

    IBM_PC_NAME        db    'IBM PC type is: ', '$'
    OS_NAME            db    'MSDOS version is: . ', 0dh, 0ah, '$'
    OEM_NAME           db    'OEM number is:      ', 0dh, 0ah, '$'
    SERIAL_NAME        db    'Serial number is:      ', 0dh, 0ah, '$'
```

```
DATA ENDS
```

```
CODE SEGMENT
```

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

```
PRINT_STRING PROC near
```

```
    push  AX
    mov   AH, 09h
    int   21h
    pop   AX
    ret
```

```
PRINT_STRING ENDP
```

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

```
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
                push  cx
```

```

        mov     al, ah
        call    TETR_TO_HEX
        xchg    al, ah
        mov     cl, 4
        shr     al, cl
        call    TETR_TO_HEX
        pop     cx
        ret
BYTE_TO_HEX      ENDP

```

```

;-----
--

```

```

WRD_TO_HEX      PROC    near
        push    bx
        mov     bh, ah
        call    BYTE_TO_HEX
        mov     [di], ah
        dec     di
        mov     [di], al
        dec     di
        mov     al, bh
        xor     ah, ah
        call    BYTE_TO_HEX
        mov     [di], ah
        dec     di
        mov     [di], al
        pop     bx
        ret
WRD_TO_HEX      ENDP

```

```

;-----
--

```

```

BYTE_TO_DEC     PROC    near
        push    cx
        push    dx
        push    ax
        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     ax, 00h
        jbe     end_1
        or      al, 30h

```

```

        mov     [si], al
end_1:   pop     ax
        pop     dx
        pop     cx
        ret

BYTE_TO_DEC      ENDP

Main PROC far
;PC INFO OUT
        push DX
        push AX

        mov ax, DATA
        mov  ds, ax

        mov DX, offset IBM_PC_NAME
        call PRINT_STRING

        mov AX, 0F000H
        mov ES, AX
        mov AL, ES:[0FFFEH]
        cmp AL, 0FFh
        je PC_WRITE
        cmp AL, 0FEh
        je PC_XT_WRITE
        cmp AL, 0FBh
        je PC_XT_WRITE
        cmp AL, 0FCh
        je AT_WRITE
        cmp AL, 0FAh
        je PS2_30_WRITE
        cmp AL, 0FCh
        je PS2_5060_WRITE
        cmp AL, 0F8h
        je PS2_80_WRITE
        cmp AL, 0FDh
        je PCjr_WRITE
        cmp AL, 0F9H
        je PC_CONVERTIBLE_WRITE

PC_WRITE:
        mov DX, offset PC_TYPE
        jmp TYPE_WRITE

PC_XT_WRITE:
        mov DX, offset PC_XT_TYPE
        jmp TYPE_WRITE

```



```

AT_WRITE:
    mov DX, offset AT_TYPE
    jmp TYPE_WRITE

PS2_30_WRITE:
    mov DX, offset PS2_30_TYPE
    jmp TYPE_WRITE

PS2_5060_WRITE:
    mov DX, offset PS2_5060_TYPE
    jmp TYPE_WRITE

PS2_80_WRITE:
    mov DX, offset PS2_80_TYPE
    jmp TYPE_WRITE

PCjr_WRITE:
    mov DX, offset PCjr_TYPE
    jmp TYPE_WRITE

PC_CONVERTIBLE_WRITE:
    mov DX, offset PC_CONVERTIBLE
    jmp TYPE_WRITE

TYPE_WRITE:
    call PRINT_STRING

OS_INFO_GET:
    mov AH, 30h
    int 21h

OS_VERSION_SET:
    lea     SI, OS_NAME
    add     SI, 18
    call    BYTE_TO_DEC
    add     SI, 3
    mov     AL, AH
    call    BYTE_TO_DEC

OS_VERSION_WRITE:
    mov DX, offset OS_NAME
    call PRINT_STRING

OEM_SET:
    mov     AL, BH
    lea     SI, OEM_NAME

```

```

        add    SI, 15
        call   BYTE_TO_DEC

OEM_WRITE:
        mov    DX, offset OEM_NAME
        call   PRINT_STRING

SERIAL_SET:
        mov     AL, BL
        lea     SI, SERIAL_NAME
        add     SI, 18
        call    BYTE_TO_HEX
        mov     [SI], AX
        add     SI, 6
        mov     DI, SI
        mov     AX, CX
        call    WRD_TO_HEX

SERIAL_WRITE:
        mov    DX, offset SERIAL_NAME
        call   PRINT_STRING

ENDING:
        pop     AX
        pop     DX

        xor     AL, AL
        mov     AH, 4ch
        int     21h
        ret

Main    ENDP
CODE    ENDS

        END     Main

```