

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

Андрющенко К.С.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

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

Задание.

1. Написать текст исходного .COM модуля, который определяет тип РС и версию системы. Результатом выполнения этого шага будет «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM.
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 СО. 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.

Порядок выполнения работы.

1. Из методического пособия возьмем шаблон .COM модуля, в котором реализованы процедуры преобразования двоичных кодов в символы 16-х и 10-х чисел.

1.1.Процедуры IBM_TYPE, DOS_VER, OEMN, USERN написаны для определения типа PC и версии системы.

1.2.Для определения версии системы использовалась функция 30H прерывания 21H, где:

- AL – номер основной версии,
- AH – номер модификации,
- BH – серийный номер OEM,
- BL:CH – 24-битовый серийный номер пользователя.

1.3.Запуск «хорошего» .COM модуля и «плохого» .EXE модуля, полученных в результате предыдущих шагов, представлен на скриншотах.

```
C:\>LAB1_C.EXE

          5 0
PC TYPE: PC
255
0000
PC TYPE: PC
PC TYPE: PC
PC TYPE: PC
C:\>S_
```

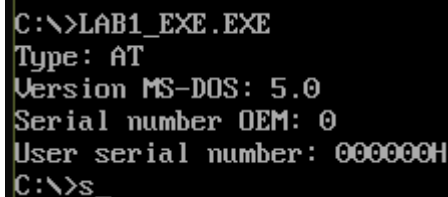
```
C:\>EXE2BIN.COM LAB1_C.EXE GOODCOM.COM
C:\>GOODCOM.COM
PC TYPE: AT
MS DOS VERSION: 5.0
OEM SERIAL N255ER:
USER SERIAL 0000ER:
C:\>s
```

Рис.1. Результат запуска .COM

2. Для создания «хорошего» .EXE модуля разобьем программу на сегменты кода, данных и стека, добавим главную процедуру.

2.1. Разобьем программу на сегменты кода, данных, стека и главную процедуру, чтобы создать «хороший» .EXE модуль. В .EXE модуле отсутствует директива `org 100h`¹

2.2. Представим результат запуска данного модуля:



```
C:\>LAB1_EXE.EXE
Type: AT
Version MS-DOS: 5.0
Serial number OEM: 0
User serial number: 000000H
C:\>s_
```

Рис.1. Результат запуска .EXE

Отличия исходных текстов COM и EXE программ:

1. Сколько сегментов должна содержать COM-программа?

Один сегмент: код, данные.

2. EXE-программа?

Два и более. EXE-программа должна содержать сегмент кода (один обязательный сегмент) и может иметь сегменты данных, стека (если сегмент стека не объявлен, то используется стек DOS) описывающихся отдельно друг от друга.

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

ORG (так как первые 256 байт занимает блок данных PSP, нужно, чтобы адресация имела смещение в 256 байт от нулевого адреса) и ASSUME (позволяет сегменту данных и сегменту кода указывать на один сегмент).

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

Нет. Нельзя использовать команды с указанием сегментов, так как отсутствует таблица настройки, адреса сегментных регистров определяются при запуске, а не линковке.

Отличия форматов файлов .com и .exe модулей.

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

¹ При загрузке COM модуля в память DOS первые 256 байт блоком данных занимает PSP, код программы располагается после этого блока.

Рассмотрим шестнадцатеричные виды модулей.

.COM модуль состоит из одного сегмента, содержащего код (располагается с нулевого адреса. PSP занимает 100h байт, значит, устанавливается смещение 100h) и данные.

начало сегмента →	00000000: E9 09 02 49 42 4D 20 74 79 70 65 3A 20 20 0D 0A	й00IBM type: 2
	000000010: 24 49 42 4D 20 74 79 70 65 3A 20 50 43 0D 0A 24	\$IBM type: PC2
	000000020: 49 42 4D 20 74 79 70 65 3A 20 50 43 2F 58 54 0D	IBM type: PC/XT2
	000000030: 0A 24 49 42 4D 20 74 79 70 65 3A 20 41 54 0D 0A	\$IBM type: AT2
	000000040: 24 49 42 4D 20 74 79 70 65 3A 20 50 53 32 20 6D	\$IBM type: PS2 m
	000000050: 6F 64 65 6C 20 33 30 0D 0A 24 49 42 4D 20 74 79	odel 302\$IBM ty
	000000060: 70 65 3A 20 50 53 32 20 6D 6F 64 65 6C 20 35 30	pe: PS2 model 50
	000000070: 20 6F 72 20 36 30 0D 0A 24 49 42 4D 20 74 79 70	or 602\$IBM typ
	000000080: 65 3A 20 50 53 32 20 6D 6F 64 65 6C 20 38 30 0D	e: PS2 model 802
	000000090: 0A 24 49 42 4D 20 74 79 70 65 3A 20 50 43 6A 72	\$IBM type: PCjr
	0000000A0: 0D 0A 24 49 42 4D 20 74 79 70 65 3A 20 50 43 20	\$IBM type: PC
	0000000B0: 43 6F 6E 76 65 72 74 69 62 6C 65 0D 0A 24 4D 53	Convertible2\$MS
	0000000C0: 20 44 4F 53 20 76 65 72 73 69 6F 6E 3A 20 20 2E	DOS version: .
	0000000D0: 20 20 0D 0A 24 4F 45 4D 20 6E 75 6D 62 65 72 3A	\$OEM number:
	0000000E0: 20 20 20 0D 0A 24 55 73 65 72 5F 73 20 6E 75 6D	\$User_s num
	0000000F0: 62 65 72 3A 20 20 20 20 20 20 20 68 0D 0A 24 24	ber: h2\$\$\$
	000000100: 0F 3C 09 76 02 04 07 04 30 C3 51 8A E0 E8 EF FF	о<ov0♦♦♦0ГQьаипя
	000000110: 86 C4 B1 04 D2 E8 E8 E6 FF 59 C3 53 8A FC E8 E9	тД+♦ТиикяYFSьий
	000000120: FF 88 25 4F 88 05 4F 8A C7 E8 DE FF 88 25 4F 88	я€%0€♦0ь3и0я€%0€
	000000130: 05 5B C3 51 52 32 E4 33 D2 B9 0A 00 F7 F1 80 CA	♦[ГQR2д3ТН€ чсЪK
	000000140: 30 88 14 4E 33 D2 3D 0A 00 73 F1 3C 00 74 04 0C	0€ЅN3T= sс< т♦Q
	000000150: 30 88 04 5A 59 C3 50 B4 09 CD 21 58 C3 53 52 06	0€♦ZYГProHIXГSR♦
	000000160: BB 00 F0 8E C3 26 A0 FE FF 3C FF 74 2A 3C FE 74	» рћГ& юя<ят*<ют
	000000170: 2C 3C FB 74 28 3C FC 74 2A 3C FA 74 2C 3C F8 74	,<ыт(<ьт*<ьт,<шт
	000000180: 2E 3C FD 74 30 3C F9 74 32 BF 0D 01 E8 7B FF 89	.<эт0<шт2ї,0и{я%
	000000190: 05 BA 03 01 EB 28 90 BA 11 01 EB 22 90 BA 20 01	♦е▼0л(ће<0л"ће 0
	0000001A0: EB 1C 90 BA 32 01 EB 16 90 BA 41 01 EB 10 90 BA	л_ће20л=ћеA0л=ће
	0000001B0: 79 01 EB 0A 90 BA 92 01 EB 04 90 BA A3 01 E8 95	у0лће'0л♦ћеJ0и•
	0000001C0: FF 07 5A 58 C3 50 53 51 33 C0 B4 30 CD 21 BE CE	я•ZXГPSQ3Ar0H!s0
	0000001D0: 01 E8 5F FF 8A C4 83 C6 03 E8 57 FF BA BE 01 E8	0и_яьДгЖвиwyes0и
	0000001E0: 74 FF BE E2 01 8A C7 E8 49 FF BA D5 01 E8 66 FF	тяsv0ь3иIяeX0иfя
	0000001F0: BF FA 01 8B C1 E8 23 FF 8A C3 E8 0D FF BF F5 01	їь0<Би#яьГиJяїx0
	000000200: 89 05 BA E6 01 E8 4E FF 59 5B 58 C3 B8 00 F0 8E	ї♦еж0иNяY[ХГё рћ
	000000210: C0 26 A0 FF FF E8 45 FF E8 AA FF 32 C0 B4 4C CD	A& яяиЕяиЕя2ArLH
конец сегмента →	000000220: 21	!

Рис.3. .COM в HEX

2. Какова структура файла «плохого» EXE? С какого адреса располагается код?

Что располагается с адреса 0?

1.1. Состоит из одного сегмента.

1.2. По адресу 0 располагается заголовок модуля.

1.3. Код располагается по адресу 300h.

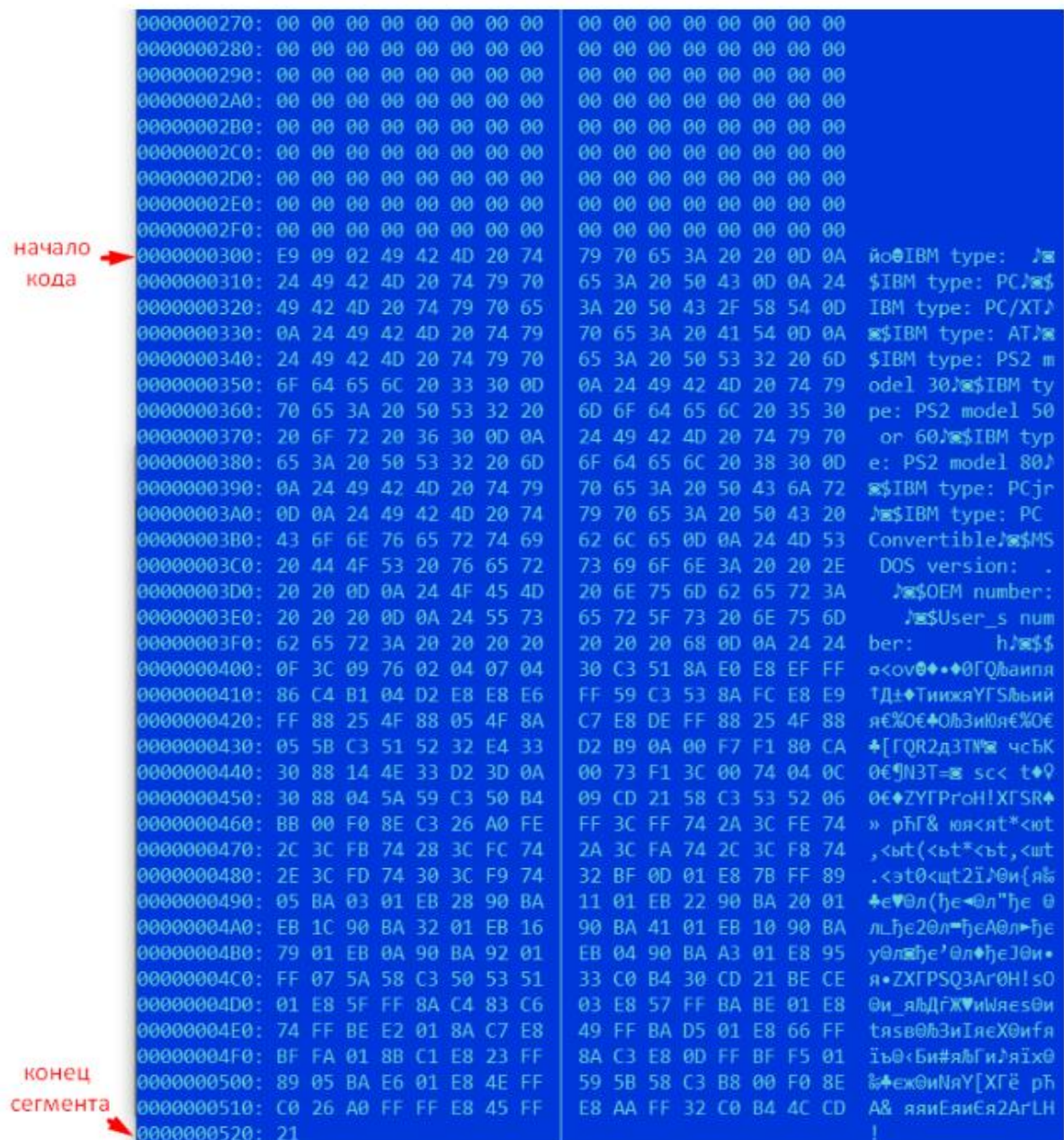


Рис.4. «плохой» .EXE в HEX

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

Структура .EXE файла:

3.1. Файл состоит из двух частей - управляющей информации для загрузчика и загрузочного модуля.

1.2. Отличие «хорошего» .EXE модуля от плохого:

1.2.1. В «хорошем» .EXE файле для данных, кода и стека отводится по сегменту, в плохом все находится в одном сегменте.

	0000000170:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
	0000000180:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
	0000000190:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
	00000001A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
	00000001B0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
	00000001C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
	00000001D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
	00000001E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
начало	00000001F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
сегмента	0000000200:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
стека	0000000210:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
	0000000220:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
	0000000230:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
начало	0000000240:	49 42 4D 20 74 79 70 65	3A 20 20 0D 0A 24 54 79	IBM type: Jm\$Ty
сегмента	0000000250:	70 65 3A 20 50 43 0D 0A	24 54 79 70 65 3A 20 50	pe: PCm\$Type: P
данных	0000000260:	43 2F 58 54 0D 0A 24 54	79 70 65 3A 20 41 54 0D	C/XTJm\$Type: ATJ
	0000000270:	0A 24 54 79 70 65 3A 20	50 53 32 20 6D 6F 64 65	m\$Type: PS2 mode
	0000000280:	6C 20 33 30 0D 0A 24 54	79 70 65 3A 20 50 53 32	l 30Jm\$Type: PS2
	0000000290:	20 6D 6F 64 65 6C 20 35	30 20 6F 72 20 36 30 0D	model 50 or 60J
	00000002A0:	0A 24 54 79 70 65 3A 20	50 53 32 20 6D 6F 64 65	m\$Type: PS2 mode
	00000002B0:	6C 20 38 30 0D 0A 24 54	79 70 65 3A 20 50 43 6A	l 80Jm\$Type: PCj
	00000002C0:	72 0D 0A 24 54 79 70 65	3A 20 50 43 20 43 6F 6E	rJm\$Type: PC Con
	00000002D0:	76 65 72 74 69 62 6C 65	0D 0A 24 4D 53 20 44 4F	vertibleJm\$MS DO
	00000002E0:	53 20 76 65 72 73 69 6F	6E 3A 20 20 2E 20 20 0D	S version: . J
	00000002F0:	0A 24 4F 45 4D 20 6E 75	6D 62 65 72 3A 20 20 20	m\$OEM number:
	0000000300:	0D 0A 24 55 73 65 72 5F	73 20 6E 75 6D 62 65 72	Jm\$User_s number
	0000000310:	3A 20 20 20 20 20 20 20	68 0D 0A 24 00 00 00 00	: hJm\$
начало	0000000320:	24 0F 3C 09 76 02 04 07	04 30 C3 51 8A E0 E8 EF	\$o<ov0+*0GQ/аип
сегмента	0000000330:	FF 86 C4 B1 04 D2 E8 E8	E6 FF 59 C3 53 8A FC E8	л!д±0ТиикпYFSльи
кода	0000000340:	E9 FF 88 25 4F 88 05 4F	8A C7 E8 DE FF 88 25 4F	йяё%0€+0лЗийлн€%0
	0000000350:	88 05 5B C3 51 52 32 E4	33 D2 B9 0A 00 F7 F1 80	€+IGQR2д3Tи% чсб
	0000000360:	CA 30 88 14 4E 33 D2 3D	0A 00 73 F1 3C 00 74 04	K0€JN3T= sc< t+
	0000000370:	0C 30 88 04 5A 59 C3 50	B4 09 CD 21 58 C3 50 52	Q0€+ZYTProHIXGPR
	0000000380:	06 B8 00 F0 8E C0 26 A0	FE FF 3C FF 74 2A 3C FE	€ё рhA& юя<ят*<ю
	0000000390:	74 2C 3C FB 74 28 3C FC	74 2A 3C FA 74 2C 3C F8	t,<wt(<ьt*<ьt,<ш
	00000003A0:	74 2E 3C FD 74 30 3C F9	74 32 BF 0A 00 E8 7B FF	t.<эт0<шт2iи и{я
	00000003B0:	89 05 BA 00 00 EB 28 90	BA 0E 00 EB 22 90 BA 19	й€ё л(йёД л"йе4
	00000003C0:	00 EB 1C 90 8A 27 00 EB	16 90 BA 32 00 EB 10 90	л.йе' л"йе2 л"й
	00000003D0:	BA 62 00 EB 0A 90 BA 77	00 EB 04 90 BA 84 00 E8	сб лмйев л"йе,, и
	00000003E0:	95 FF 07 5A 58 C3 50 53	51 33 C0 B4 30 CD 21 BE	•я•ZXGPSQ3Ar0HIs
	00000003F0:	AB 00 E8 5F FF 8A C4 83	C6 03 E8 57 FF BA 9B 00	и и.ялйДfXиWяе>
	0000000400:	E8 74 FF 8E BF 00 8A C7	E8 49 FF BA B2 00 E8 66	итяяi лЗиIяеI иф
	0000000410:	FF BF D7 00 8B C1 E8 23	FF 8A C3 E8 0D FF BF D2	лiЧ <Би#ялйГл'яiT
	0000000420:	00 89 05 BA C3 00 E8 4E	FF 59 5B 58 C3 88 04 00	й€ёГ иNяY[XГё€
	0000000430:	8E D8 E8 49 FF E8 AE FF	32 C0 B4 4C CD 21	ТшйIяи"я2ArLH!

Рис.5. «хороший» .EXE в HEX

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

1.3. Выделение свободного сегмента в основной памяти.

1.4. Генерация PSP в первых 256 байтах файла.

1.5. Запись программы.

Код располагается с адреса CS:0100 = 48DD:0100.

2. Что располагается с адреса 0?

С адреса 0 располагается PSP.

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

Указывают на начало PSP. Значение 48DD.

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

Указатель на стек SP в начале равен FFFE (последний чётный доступный адрес). Стэк заполняется «с конца», следовательно, имеет адреса от FFFE до 0. Стэк определяется автоматически и находится в общем сегменте (для всей программы сегмент только один).

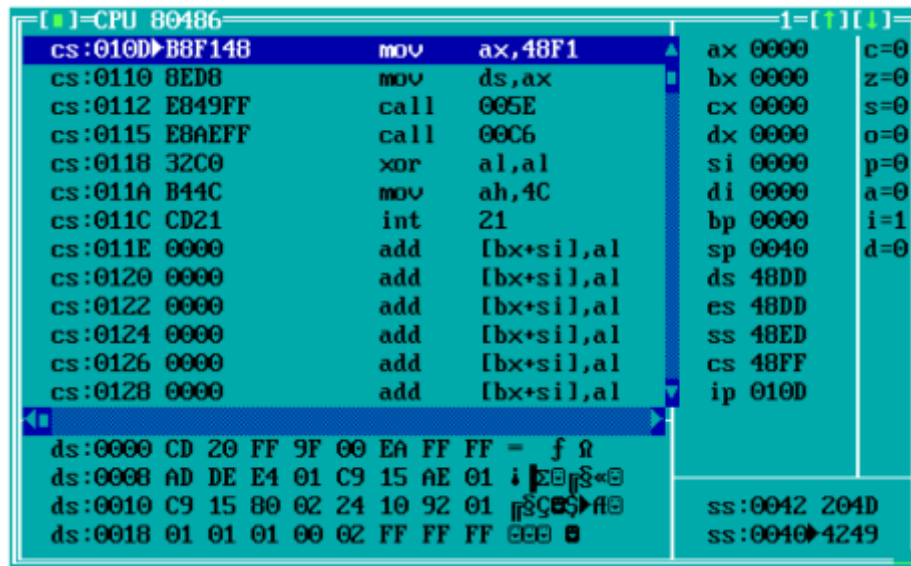


Рис.6. «хороший» .COM в отладчике

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

1.1. Определение сегментного адреса свободного участка памяти для возможности размещения программы.

1.2. Создание и заполнение блока памяти для переменных среды.

1.3. Создание блока памяти для PSP (0000h) и программы (сегмент+0010h:0000h).

1.4. Заполнение полей PSP.

1.5. В рабочую область загрузчика считывается форматированная часть заголовка .EXE файла.

1.6. Инициализация регистров.

ES=DS=PSP=48DD. SS = 48ED. CS=497D.

1.7. Выполнение программы.

2. На что указывают DS и ES?

На начало PSP.

3. Как определяется стек?

Программа: при помощи регистров SS и SP.

Код: при помощи директивы ASSUME и описания стекового сегмента в коде.

4. Как определяется точка входа?

При помощи директивы END.

Вывод.

В результате работы были изучены различия в структурах исходных текстов модулей типов .COM и .EXE, файлов загрузочных моделей, способов загрузки их в основную память и инициализации начального состояния. Написана программа, выводящая строковую информацию о версии операционной системы DOS.

ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД МОДУЛЕЙ

Файл LAB1_C.ASM

```
; Шаблон текста программы на ассемблере для модуля типа .COM
TESTPC SEGMENT
```

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

```
    ORG 100H
```

```
START: JMP BEGIN
```

```
; ДАННЫЕ
```

```
PC db 'PC TYPE: PC', 0DH, 0AH, '$'
```

```
PC_XT db 'PC TYPE: PC/XT', 0DH, 0AH, '$'
```

```
AT db 'PC TYPE: AT', 0DH, 0AH, '$'
```

```
MODEL_30 db 'PC TYPE: PS2 model 30', 0DH, 0AH, '$'
```

```
MODEL_50_OR_60 db 'PC TYPE: PS2 model 50 or 60', 0DH, 0AH, '$'
```

```
MODEL_80 db 'PC TYPE: PS2 model 80', 0DH, 0AH, '$'
```

```
PCjr db 'PC TYPE: PCjr', 0DH, 0AH, '$'
```

```
PC_CONVERTIBLE db 'PC TYPE: PC Convertible', 0DH, 0AH, '$'
```

```
MS_DOS_VERSION db 'MS DOS VERSION:  . ', 0DH, 0AH, '$'
```

```
OEM db 'OEM SERIAL NUMBER:      ', 0DH, 0AH, '$'
```

```
USER db 'USER SERIAL NUMBER:      ', 0DH, 0AH, '$'
```

```
;ПРОЦЕДУРЫ
```

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

```
TETR_TO_HEX PROC near
```

```
    and AL,0Fh
```

```
    cmp AL,09
```

```
    jbe NEXT
```

```
    add AL,07
```

```
NEXT: add AL,30h
```

```
    ret
```

```
TETR_TO_HEX ENDP
```

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

```
BYTE_TO_HEX PROC near
```

```
; байт в AL переводится в два символа шестн. числа в AX
```

```
    push CX
```

```
    mov AH,AL
```

```
    call TETR_TO_HEX
```

```
    xchg AL,AH
```

```
    mov CL,4
```

```
    shr AL,CL
```

```

    call TETR_TO_HEX ;в AL старшая цифра
    pop CX ;в AH младшая
    ret
BYTE_TO_HEX ENDP
;-----
WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
    push BX
    mov BH,AH
    call BYTE_TO_HEX
    mov [DI],AH
    dec DI
    mov [DI],AL
    dec DI
    mov AL,BH
    call BYTE_TO_HEX
    mov [DI],AH
    dec DI
    mov [DI],AL
    pop BX
    ret
WRD_TO_HEX ENDP
;-----
BYTE_TO_DEC PROC near
; перевод в 10с/с, SI - адрес поля младшей цифры
    push CX
    push DX
    xor AH,AH
    xor DX,DX
    mov CX,10
loop_bd: div CX
    or DL,30h
    mov [SI],DL
    dec SI
    xor DX,DX
    cmp AX,10
    jae loop_bd
    cmp AL,00h
    je end_1
    or AL,30h
    mov [SI],AL

```

```

end_1: pop DX
      pop CX
      ret
BYTE_TO_DEC ENDP
;-----
; КОД
; вовод сообщения

PRINT_MESSAGE:
      mov AH, 09h
      int 21h

      pop DX
      pop ES
      pop AX

      ret
; вывод информации
PRINT_PC_TYPE PROC NEAR
      push AX
      push ES
      push DX

      mov AX, 0F000h
      mov ES, AX
      mov AL, ES:[0FFFEh]

      cmp AL, 0FFh
      je PC_TYPE_MES

      cmp AL, 0FEh
      je PC_XT_MES

      cmp AL, 0FBh
      je PC_XT_MES

      cmp AL, 0FCh
      je AT_MES

      cmp AL, 0FAh
      je MODEL_30_MES

```



```
    cmp AL, 0FCh
    je MODEL_50_60_MES

    cmp AL, 0F8h
    je MODEL_80_MES

    cmp AL, 0FDh
    je PCGR_TYPE

    cmp AL, 0F9h
    je PC_CONVERTIBLE_MES
```

```
PC_TYPE_MES:
    mov DX, offset PC
    jmp PRINT_MESSAGE
```

```
PC_XT_MES:
    mov DX, offset PC_XT
    jmp PRINT_MESSAGE
```

```
AT_MES:
    mov DX, offset AT
    jmp PRINT_MESSAGE
```

```
MODEL_30_MES:
    mov DX, offset MODEL_30
    jmp PRINT_MESSAGE
```

```
MODEL_50_60_MES:
    mov DX, offset MODEL_50_OR_60
    jmp PRINT_MESSAGE
```

```
MODEL_80_MES:
    mov DX, offset MODEL_80
    jmp PRINT_MESSAGE
```

```
PCGR_TYPE:
    mov DX, offset PCjr
    jmp PRINT_MESSAGE
```

```
PC_CONVERTIBLE_MES:
    mov DX, offset PC_CONVERTIBLE
```

```
jmp PRINT_MESSAGE
```

```
PRINT_PC_TYPE ENDP
```

```
PRINT_MES_SYS PROC near
```

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

```
PRINT_MES_SYS ENDP
```

```
PRINT_SYSTEM_VERSION PROC near
```

```
    push AX  
    push BX  
    push CX  
    push DI  
    push SI
```

```
    sub AX, AX  
    mov AH, 30h  
    int 21h
```

```
    mov SI, offset MS_DOS_VERSION  
    add SI, 16  
    call BYTE_TO_DEC  
    mov AL, AH ; AH - DOS VERSION  
    add SI, 3  
    call BYTE_TO_DEC  
    mov DX, offset MS_DOS_VERSION  
    call PRINT_MES_SYS
```

```
    mov SI, offset OEM  
    add SI, 14  
    mov AL, BH ; BH - OEM NUMBER  
    call BYTE_TO_DEC  
    mov DX, offset OEM  
    call PRINT_MES_SYS
```

```
    mov DI, offset USER
```

```

    add DI, 15
    mov AX, CX
    call WRD_TO_HEX
    mov AL, BL
    call BYTE_TO_HEX
    mov DX, offset USER
    call PRINT_MES_SYS

    pop SI
    pop DI
    pop CX
    pop BX
    pop AX

    ret
PRINT_SYSTEM_VERSION ENDP

BEGIN:

    call PRINT_PC_TYPE
    call PRINT_SYSTEM_VERSION

; Выход в DOS

    xor AL,AL
    mov AH,4Ch
    int 21H
TESTPC ENDS
    END START ;конец модуля, START - точка входа

```

Файл LAB1_E.ASM

```
AStack    SEGMENT    STACK
            DW 128 DUP(?)
AStack    ENDS
```

```
DATA    SEGMENT
```

```
    PC db    'Type: PC',0DH,0AH,'$'
    PC_XT db 'Type: PC/XT',0DH,0AH,'$'
    AT db    'Type: AT',0DH,0AH,'$'
    MODEL_30 db 'Type: PS2 модель 30',0DH,0AH,'$'
    MODEL_50_OR_60 db 'Type: PS2 модель 50 или
60',0DH,0AH,'$'
    MODEL_80 db 'Type: PS2 модель 80',0DH,0AH,'$'
    PCjr db 'Type: PCjr',0DH,0AH,'$'
    PC_CONVERTIBLE db 'Type: PC Convertible',0DH,0AH,'$'

    VERSIONS db 'Version MS-DOS: . ',0DH,0AH,'$'
    SERIAL_NUMBER db 'Serial number OEM: ',0DH,0AH,'$'
    USER_NUMBER db 'User serial number:      H $'
DATA ENDS
```

```
CODE SEGMENT
```

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

```
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 AH,AL
    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
    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
    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_1
    or AL,30h
    mov [SI],AL

```

```

end_1:
    pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP

WRITESTRING PROC near
    mov AH,09h
    int 21h
    ret
WRITESTRING ENDP

PC_TYPE PROC near
    mov ax, 0f000h
    mov es, ax
    mov al, es:[0ffffh]

    cmp al, 0ffh
    je pc_mes
    cmp al, 0feh
    je pc_xt_mes
    cmp al, 0fbh
    je pc_xt_mes
    cmp al, 0fch
    je pc_at_mes
    cmp al, 0fah
    je pc_ps2_m30
    cmp al, 0f8h
    je pc_ps2_m80
    cmp al, 0fdh
    je pc_jr
    cmp al, 0f9h
    je pc_conv
pc_mes:
    mov dx, offset PC
    jmp writetype
pc_xt_mes:
    mov dx, offset PC_XT
    jmp writetype
pc_at_mes:
    mov dx, offset AT
    jmp writetype

```

```

pc_ps2_m30:
    mov dx, offset MODEL_30
    jmp writetype
pc_ps2_m50_60:
    mov dx, offset MODEL_50_OR_60
    jmp writetype
pc_ps2_m80:
    mov dx, offset MODEL_80
    jmp writetype
pc_jr:
    mov dx, offset PCjr
    jmp writetype
pc_conv:
    mov dx, offset PC_CONVERTIBLE
    jmp writetype
writetype:
    call WRITESTRING
    ret
PC_TYPE ENDP

```

```

OS_VER PROC near
    mov ah, 30h
    int 21h
    push ax

    mov si, offset VERSIONS
    add si, 16
    call BYTE_TO_DEC
pop ax
mov al, ah
add si, 3
    call BYTE_TO_DEC
    mov dx, offset VERSIONS
    call WRITESTRING

    mov si, offset SERIAL_NUMBER
    add si, 19
    mov al, bh
    call BYTE_TO_DEC
    mov dx, offset SERIAL_NUMBER
    call WRITESTRING

```

```

    mov di, offset USER_NUMBER
    add di, 25
    mov ax, cx
    call WRD_TO_HEX
    mov al, bl
    call BYTE_TO_HEX
    sub di, 2
    mov [di], ax
    mov dx, offset USER_NUMBER
    call WRITESTRING
    ret
OS_VER ENDP

Main PROC FAR
    sub    AX,AX
    push   AX
    mov    AX,DATA
    mov    DS,AX
    call   PC_TYPE
    call   OS_VER
    xor    AL,AL
    mov    AH,4Ch
    int    21H
Main ENDP
CODE ENDS
    END Main

```