

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.

Студентка гр. 0382

Михайлова О.Д.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

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.

1. Напишите 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 модуля.

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.

3. Сравните исходные тексты для .COM и .EXE модулей. Ответьте на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

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».

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.

6. Откройте отладчик TD.EXE и загрузите «хороший» .EXE. Ответьте на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

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.

Для выполнения задания был использован шаблон из методических указаний с процедурами перевода двоичных кодов в символы шестнадцатеричных чисел и десятичное число: TETR_TO_HEX, BYTE_TO_HEX, WRD_TO_HEX, BYTE_TO_DEC. Также были добавлены следующие процедуры:


- PRINT_STRING – вывод строки на экран
- PC_TYPE - определение типа PC и вывод строки с названием модели
- S_VERSION - определение версии системы

Было написано два файла с исходным кодом:

1. com_file.asm – файл, в результате отладки которого был получен «хороший» .com модуль и «плохой» .exe.

2. exe_file.asm – был получен из первого файла путем разбиения кода на сегменты. В результате отладки данного файла был получен «хороший» .exe.

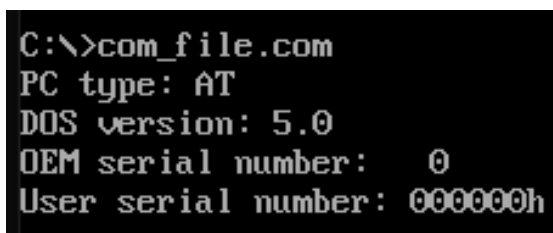
Результаты работы «хороших» и «плохих» файлов .com и .exe представлены на рисунках 1-3.



```
C:\>com_file.exe

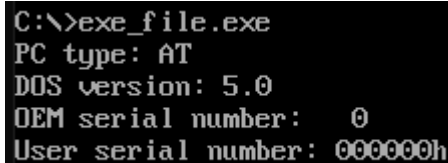
5 0
PC type: PC
0
C
000000
PC type: PC
PC type: PC
PC type: P
```

Рисунок 1 - Результат работы "плохого" .exe



```
C:\>com_file.com
PC type: AT
DOS version: 5.0
OEM serial number: 0
User serial number: 000000h
```

Рисунок 2 - Результат работы "хорошего" .com



```
C:\>exe_file.exe
PC type: AT
DOS version: 5.0
OEM serial number: 0
User serial number: 000000h
```

Рисунок 3 - Результат работы "хорошего" .exe

Далее было проведено сравнение исходных файлов для .exe и .com модулей.

После этого .com и .exe файлы были открыты в FAR и отладчике TD.

Ответы на контрольные вопросы.

Отличия исходных текстов COM и EXE программ

1. Сколько сегментов должна содержать COM-программа?

Один.

2. EXE-программа?

Может иметь любое количество сегментов. EXE-программа предполагает отдельные сегменты для кода, данных и стека.

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

Директива ORG 100h, которая задает смещение для программы на 256 байт для PSP. Директива ASSUME, которая указывает ассемблеру, с каким сегментом или группой сегментов связан тот или иной сегментный регистр.

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

Нет, не все. Нельзя использовать команды, связанные с адресом сегмента, так как он неизвестен до загрузки, потому что в com-программах не содержится таблица настроек.

Отличия форматов файлов COM и EXE модулей

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

COM-файл состоит из единственного сегмента, в котором располагаются код, данные и стек. Код начинается с адреса 0h. (рис. 4)

```

D:\DOSBox-0.74-3\ос_лаб\COM_FILE.COM
00000000: E9 F8 01 50 43 20 74 79 70 65 3A 20 50 43 0D 0A  éøPC type: PC
00000001: 24 50 43 20 74 79 70 65 3A 20 50 43 2F 58 54 0D  $PC type: PC/XT
00000002: 0A 24 50 43 20 74 79 70 65 3A 20 41 54 0D 0A 24  $PC type: AT
00000003: 50 43 20 74 79 70 65 3A 20 50 53 32 20 6D 6F 64  PC type: PS2 mod
00000004: 65 6C 20 33 30 0D 0A 24 50 43 20 74 79 70 65 3A  el 30
00000005: 20 50 53 32 20 6D 6F 64 65 6C 20 35 30 20 6F 72  PS2 model 50 or
00000006: 20 36 30 0D 0A 24 50 43 20 74 79 70 65 3A 20 50 60  $PC type: P
00000007: 53 32 20 6D 6F 64 65 6C 20 38 30 0D 0A 24 50 43 S2 model 80
00000008: 20 74 79 70 65 3A 20 50 D0 A1 6A 72 0D 0A 24 50  type: PDjr
00000009: 43 20 74 79 70 65 3A 20 50 43 20 43 6F 6E 76 65  C type: PC Conve
0000000A: 72 74 69 62 6C 65 0D 0A 24 44 4F 53 20 76 65 72  rtible
0000000B: 73 69 6F 6E 3A 20 20 2E 20 20 0D 0A 24 4F 45 4D  sion: .
0000000C: 20 73 65 72 69 61 6C 20 6E 75 6D 62 65 72 3A 20  serial number:
0000000D: 20 20 0D 0A 24 55 73 65 72 20 73 65 72 69 61 6C  $User serial
0000000E: 20 6E 75 6D 62 65 72 3A 20 20 20 20 20 20 20 68  number: h
0000000F: 0D 0A 24 24 0F 3C 09 76 02 04 07 04 30 C3 51 8A  $o<ov
00000010: E0 E8 EF FF 86 C4 B1 04 D2 E8 E8 E6 FF 59 C3 53  àèÿ†Ä±ðèæÿYÄS
00000011: 8A FC E8 E9 FF 88 25 4F 88 05 4F 8A C7 E8 DE FF  Šüëÿ~%0^*0ŠÇèÿ
00000012: 88 25 4F 88 05 5B C3 51 52 32 E4 33 D2 B9 0A 00  ^%0^*[ÄQR2ä30¹
00000013: F7 F1 80 CA 30 88 14 4E 33 D2 3D 0A 00 73 F1 3C  ÷ñ€Ê0^ŊN30= sñ<
00000014: 00 74 04 0C 30 88 04 5A 59 C3 B4 09 CD 21 C3 B8  t00^ZYÄ°óÍ!Ä.
00000015: 00 F0 8E C0 26 A0 FE FF 3C FF 74 20 3C FE 74 22  ðŽÄ& þÿ<ÿt <pt"
00000016: 3C FB 74 1E 3C FC 74 20 3C FA 74 22 3C FC 74 24  <ût▲<ût <üt"<üt$
00000017: 3C F8 74 26 3C FD 74 28 3C F9 74 2A BA 03 01 EB  <øt&<ÿt(<üt*0♥0ë
00000018: 2B 90 BA 11 01 EB 25 90 BA 22 01 EB 1F 90 BA 30  +0°◀0ë%0°"0ë▼0°0
00000019: 01 EB 19 90 BA 48 01 EB 13 90 BA 66 01 EB 0D 90  0ë↓0°H0ë!!0°f0ë)0
0000001A: BA 7E 01 EB 07 90 BA 8F 01 EB 01 90 E8 9B FF C3  °~0ë•0°00ë00è>ÿÄ
0000001B: B4 30 CD 21 BE A9 01 83 C6 0D E8 6A FF 8A C4 83  °Í!%00fÄjÿŠÄf
0000001C: C6 03 E8 62 FF BA A9 01 E8 7F FF BE BD 01 83 C6  Ä♥ëÿ°00ë0ÿ%00fÄ
0000001D: 15 8A C7 E8 51 FF BA BD 01 E8 6E FF BF D5 01 83  ŠŠÇèQÿ°%0èny¿Ö0f
0000001E: C7 19 8B C1 E8 28 FF 8A C3 E8 12 FF BF D5 01 83  Ç↓<Äè(ÿŠÄè†ÿ¿Ö0f
0000001F: C7 14 89 05 BA D5 01 E8 50 FF C3 E8 51 FF E8 AF  ÇŊ%◀°Ö0èPÿÄèQÿè~
00000020: FF 32 C0 B4 4C CD 21  ÿ2Ä`LÍ!
1Help 2Dump 3Quit 4Text 5 6Edit 7Search 8ANSI 9

```

Рисунок 4 - "хороший" COM модуль

2. Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с адреса 0?

«Плохой» EXE-файл состоит из единственного сегмента. Код начинается с адреса 300h. С адреса 0h располагается заголовок и таблица настройки адресов. (рис. 5-6)

D:\DOSBox-0.74-3\ос_лаб\COM_FILE.EXE																		
0000000000:	4D	5A	07	01	03	00	00	00	20	00	00	00	FF	FF	00	00	MZ•♥	ÿÿ
0000000010:	00	00	D3	31	00	01	00	00	1E	00	00	00	01	00	00	00	Ó1 ☹ ▲	☹
0000000020:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000030:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000040:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000050:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000060:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000070:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000080:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000090:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000000A0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000000B0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000000C0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		

Рисунок 5 - "плохой" EXE модуль

D:\DOSBox-0.74-3\ос_лаб\COM_FILE.EXE																	
00000001D0:	00	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	00
00000001F0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000200:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000210:	00	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	00
0000000230:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000240:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000250:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000260:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000270:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000280:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000290:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000002A0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000002B0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000002C0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000002D0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000002E0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000002F0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000300:	E9	F8	01	50	43	20	74	79	70	65	3A	20	50	43	0D	0A	éøPC type: PC
0000000310:	24	50	43	20	74	79	70	65	3A	20	50	43	2F	58	54	0D	\$PC type: PC/XT
0000000320:	0A	24	50	43	20	74	79	70	65	3A	20	41	54	0D	0A	24	\$PC type: AT
0000000330:	50	43	20	74	79	70	65	3A	20	50	53	32	20	6D	6F	64	PC type: PS2 mod
0000000340:	65	6C	20	33	30	0D	0A	24	50	43	20	74	79	70	65	3A	e1 30\$PC type:
0000000350:	20	50	53	32	20	6D	6F	64	65	6C	20	35	30	20	6F	72	PS2 model 50 or
0000000360:	20	36	30	0D	0A	24	50	43	20	74	79	70	65	3A	20	50	60\$PC type: P
0000000370:	53	32	20	6D	6F	64	65	6C	20	38	30	0D	0A	24	50	43	S2 model 80\$PC
0000000380:	20	74	79	70	65	3A	20	50	D0	A1	6A	72	0D	0A	24	50	type: PDjrn\$P
1Help	2Dump	3Quit	4Text	5	6Edit	7Search	8ANSI	9									

Рисунок 6 - "плохой" EXE модуль

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

В «хорошем» EXE-файле код, данные и стек находятся в разных сегментах, в отличие от «плохого». Кроме того, «хороший» EXE- файл не содержит директивы ORG 100h, поэтому код начинается без смещения. (рис. 7)

```
D:\DOSBox-0.74-3\ос_лаб\EXE_FILE.EXE
00000004E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000004F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000500: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000510: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000520: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000530: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000540: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000550: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000560: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000570: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000580: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000590: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000005A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000005B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000005C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000005D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000005E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000005F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000600: 50 43 20 74 79 70 65 3A 20 50 43 0D 0A 24 50 43 PC type: PC
0000000610: 20 74 79 70 65 3A 20 50 43 2F 58 54 0D 0A 24 50 type: PC/XT
0000000620: 43 20 74 79 70 65 3A 20 41 54 0D 0A 24 50 43 20 C type: AT
0000000630: 74 79 70 65 3A 20 50 53 32 20 6D 6F 64 65 6C 20 type: PS2 model
0000000640: 33 30 0D 0A 24 50 43 20 74 79 70 65 3A 20 50 53 30 PC type: PS
0000000650: 32 20 6D 6F 64 65 6C 20 35 30 20 6F 72 20 36 30 2 model 50 or 60
0000000660: 0D 0A 24 50 43 20 74 79 70 65 3A 20 50 53 32 20 PC type: PS2
0000000670: 6D 6F 64 65 6C 20 38 30 0D 0A 24 50 43 20 74 79 model 80
0000000680: 70 65 3A 20 50 D0 A1 6A 72 0D 0A 24 50 43 20 74 pe: P0ijr
0000000690: 79 70 65 3A 20 50 43 20 43 6F 6E 76 65 72 74 69 ype: PC Converte

1Help 2Dump 3Quit 4Text 5 6Edit 7Search 8ANSI 9
```

Рисунок 7 - "хороший" EXE модуль

Загрузка COM модуля в основную память

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

В основной памяти выделяется свободный сегмент, далее 256 байт занимает PSP, далее располагается код. Код начинается с адреса 100h. (рис. 8)

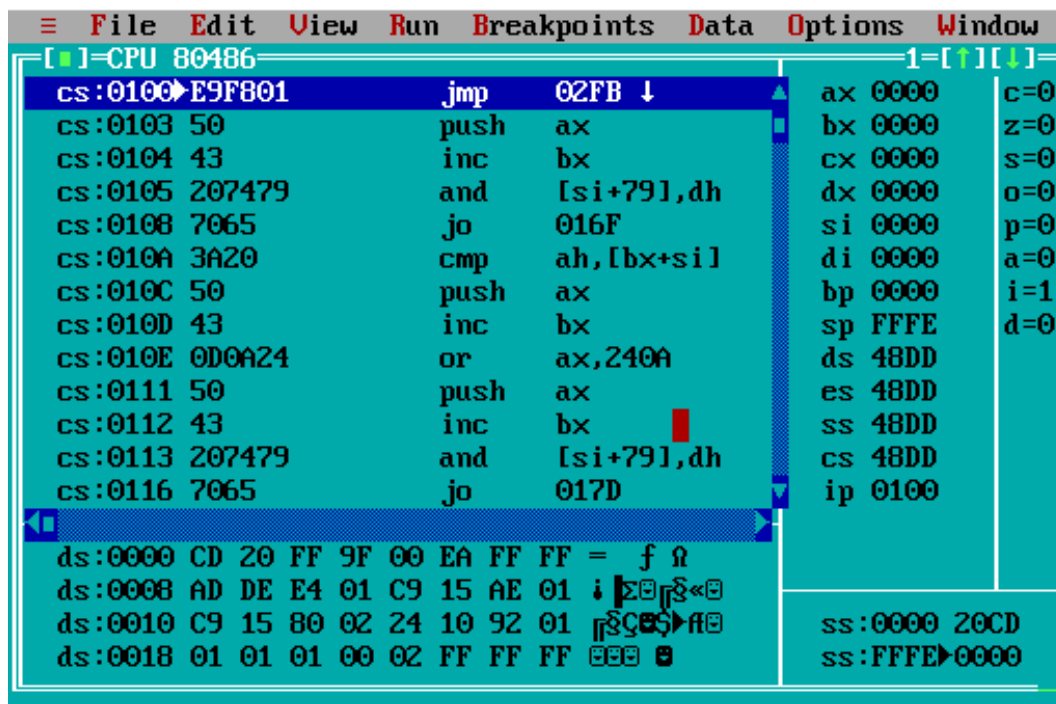


Рисунок 8 - "хороший" COM модуль в отладчике

2. Что располагается с адреса 0?

PSP.

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

Все сегментные регистры указывают на начало PSP и имеют значение 48DD. (рис. 8)

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

Стек создается автоматически. Под него отводится весь сегмент. SS указывает на начало стека (0h), а SP на конец (FFFEh). То есть стек расположен между адресами SS:0000h – SS:FFFEh.

Загрузка «хорошего» EXE модуля в основную память

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

Сначала в память загружается PSP, потом загружается EXE модуль в соответствии с информацией в заголовке. Значения регистров: ES=DS=48DD, SS = 48ED, CS = 493C. (рис. 9)

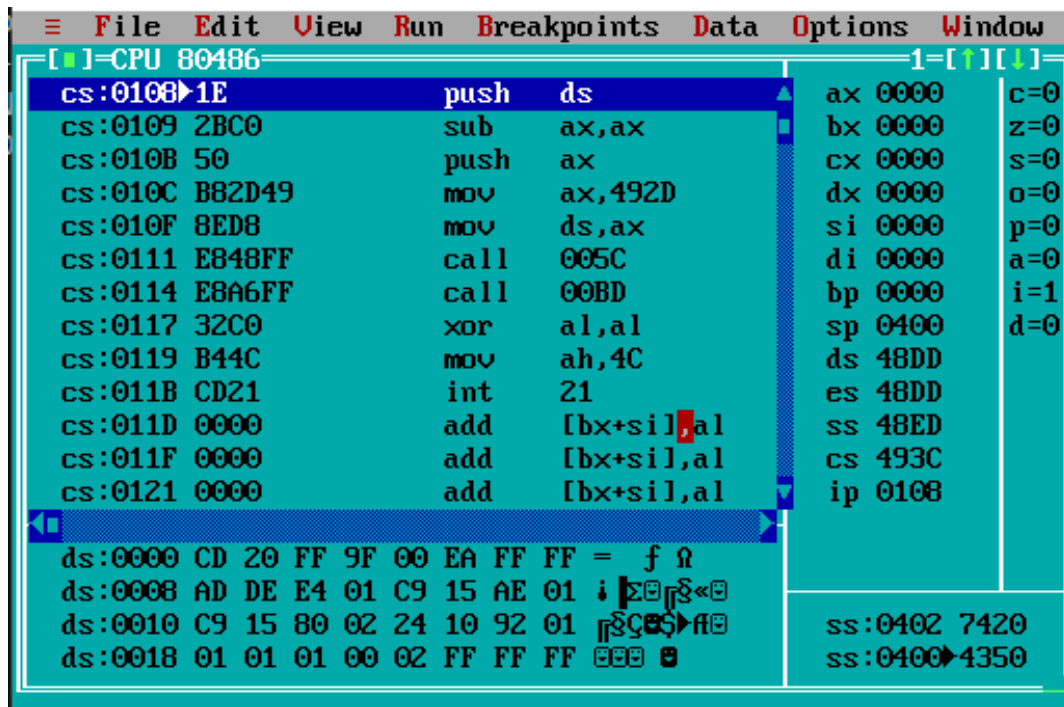


Рисунок 9 - "хороший" EXE модуль в отладчике

2. На что указывают регистры DS и ES?

На начало сегмента PSP.

3. Как определяется стек?

Стек определяется пользователем с помощью директивы SEGMENT STACK. Регистр SS указывает на начало сегмента стека, SP – на конец.

4. Как определяется точка входа?

С помощью директивы END.

Выводы.

В ходе работы были изучены различия исходных файлов для .COM и .EXE модулей, структур файлов загрузочных модулей и способов их загрузки в основную память.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: com_file.asm

```
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,'$'
PS2_30 db 'PC type: PS2 model 30',0DH,0AH,'$'
PS2_50_or_60 db 'PC type: PS2 model 50 or 60',0DH,0AH,'$'
PS2_80 db 'PC type: PS2 model 80',0DH,0AH,'$'
PCJR db 'PC type: PCjr',0DH,0AH,'$'
PC_CONVERTIBLE db 'PC type: PC Convertible',0DH,0AH,'$'

DOS_VERS db 'DOS version: . ',0DH,0AH,'$'
OEM_NUMBER db 'OEM serial number: ',0DH,0AH,'$'
USER_NUMBER db 'User serial number:      h', 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 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_STRING PROC near
    mov AH, 09h
    int 21h
    ret
PRINT_STRING ENDP

PC_TYPE PROC near
    mov AX, 0F000h
    mov ES, AX
    mov AL, ES:[0FFFEh]

    cmp AL, 0FFh
    je t_pc

    cmp AL, 0FEh
    je t_pc_xt

    cmp AL, 0FBh
    je t_pc_xt

```

```

        cmp AL, 0FCh
        je t_at

        cmp AL, 0FAh
        je t_ps2_30

        cmp AL, 0FCh
        je t_ps2_50_or_60

        cmp AL, 0F8h
        je t_ps2_80

        cmp AL, 0FDh
        je t_pcjr

        cmp AL, 0F9h
        je t_pc_convertible

t_pc:
    mov DX, offset PC
    jmp final_1

t_pc_xt:
    mov DX, offset PC_XT
    jmp final_1

t_at:
    mov DX, offset AT
    jmp final_1

t_ps2_30:
    mov DX, offset PS2_30
    jmp final_1

t_ps2_50_or_60:
    mov DX, offset PS2_50_or_60
    jmp final_1

t_ps2_80:
    mov DX, offset PS2_80
    jmp final_1

t_pcjr:
    mov DX, offset PCJR
    jmp final_1

t_pc_convertible:
    mov DX, offset PC_CONVERTIBLE
    jmp final_1

final_1:
    call PRINT_STRING
    ret
PC_TYPE ENDP

```

```

S_VERSION PROC near
    mov AH,30h
    int 21h
    ;push AX

    mov SI, offset DOS_VERS
    add SI, 13
    call BYTE_TO_DEC
    ;pop AX
    mov AL,AH
    add SI,3
    call BYTE_TO_DEC
    mov DX, offset DOS_VERS
    call PRINT_STRING

    mov SI, offset OEM_NUMBER
    add SI, 21
    mov AL, BH
    call BYTE_TO_DEC
    mov DX, offset OEM_NUMBER
    call PRINT_STRING

    mov DI, offset USER_NUMBER
    add DI, 25
    mov AX, CX
    call WRD_TO_HEX
    mov AL, BL
    call BYTE_TO_HEX
    mov DI, offset USER_NUMBER
    add DI, 20
    mov [DI], AX
    mov DX, offset USER_NUMBER
    call PRINT_STRING

    ret

S_VERSION ENDP

```

```

; Код
BEGIN:
    call PC_TYPE
    call S_VERSION
    xor AL,AL
    mov AH,4Ch
    int 21H
TESTPC ENDS
END START

```

Название файла: exe_file.asm

```

AStack SEGMENT STACK
    DW 512 DUP(?)
AStack ENDS

```

```

DATA SEGMENT
    PC db  'PC type: PC',0DH,0AH,'$'

```

```

PC_XT db 'PC type: PC/XT',0DH,0AH,'$'
AT db 'PC type: AT',0DH,0AH,'$'
PS2_30 db 'PC type: PS2 model 30',0DH,0AH,'$'
PS2_50_or_60 db 'PC type: PS2 model 50 or 60',0DH,0AH,'$'
PS2_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,'$'

DOS_VERS db 'DOS version: . ',0DH,0AH,'$'
OEM_NUMBER db 'OEM serial number: ',0DH,0AH,'$'
USER_NUMBER db 'User serial number: h', 0DH, 0AH,'$'
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

```

```

;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_STRING PROC near
    mov AH, 09h
    int 21h
    ret
PRINT_STRING ENDP

PC_TYPE PROC near
    mov AX, 0F000h
    mov ES, AX
    mov AL, ES:[0FFFEh]

    cmp AL, 0FFh
    je t_pc

    cmp AL, 0FEh
    je t_pc_xt

    cmp AL, 0FBh
    je t_pc_xt

    cmp AL, 0FCh
    je t_at

    cmp AL, 0FAh
    je t_ps2_30

    cmp AL, 0FCh
    je t_ps2_50_or_60

```

```

        cmp AL, 0F8h
        je t_ps2_80

        cmp AL, 0FDh
        je t_pcjr

        cmp AL, 0F9h
        je t_pc_convertible

t_pc:
    mov DX, offset PC
    jmp final_1

t_pc_xt:
    mov DX, offset PC_XT
    jmp final_1

t_at:
    mov DX, offset AT
    jmp final_1

t_ps2_30:
    mov DX, offset PS2_30
    jmp final_1

t_ps2_50_or_60:
    mov DX, offset PS2_50_or_60
    jmp final_1

t_ps2_80:
    mov DX, offset PS2_80
    jmp final_1

t_pcjr:
    mov DX, offset PCJR
    jmp final_1

t_pc_convertible:
    mov DX, offset PC_CONVERTIBLE
    jmp final_1

final_1:
    call PRINT_STRING
    ret
PC_TYPE ENDP

S_VERSION PROC near
    mov AH, 30h
    int 21h
    ;push AX

    mov SI, offset DOS_VERS
    add SI, 13
    call BYTE_TO_DEC
    ;pop AX
    mov AL, AH

```



```

    add SI,3
    call BYTE_TO_DEC
    mov DX, offset DOS_VERS
    call PRINT_STRING

    mov SI, offset OEM_NUMBER
    add SI, 21
    mov AL, BH
    call BYTE_TO_DEC
    mov DX, offset OEM_NUMBER
    call PRINT_STRING

    mov DI, offset USER_NUMBER
    add DI, 25
    mov AX, CX
    call WRD_TO_HEX
    mov AL, BL
    call BYTE_TO_HEX
    mov DI, offset USER_NUMBER
    add DI, 20
    mov [DI], AX
    mov DX, offset USER_NUMBER
    call PRINT_STRING

    ret

S_VERSION ENDP

Main PROC FAR
    push DS
    sub AX, AX
    push AX
    mov AX, DATA
    mov DS, AX

    call PC_TYPE
    call S_VERSION
    xor AL,AL
    mov AH,4Ch
    int 21H

Main ENDP
CODE ENDS
END Main

```