

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

Морева Е.С.

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.

Шаг 1. Написать текст исходно .COM модуля, который определяет тип РС и версию системы. Построить «плохой» .EXE модуль, полученный из исходного текста для .COM модуля.

Шаг 2. Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1, построить и отладить его. Таким образом будет получен «хороший» .EXE.

Шаг 3. Сравнить исходные тексты для .COM и .EXE модулей. Ответить на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

Шаг 4. Запустить FAR и открыть файлы 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».

Шаг 5. Открыть отладчик TD.EXE и загрузить .COM. Ответить на контрольные вопросы «Загрузка COM модуля в основную память». Представить в отчете план загрузки модуля .COM в основную память.

Шаг 6. Открыть отладчик TD.EXE и загрузить «хороший» .EXE. Ответить на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

Выполнение работы.

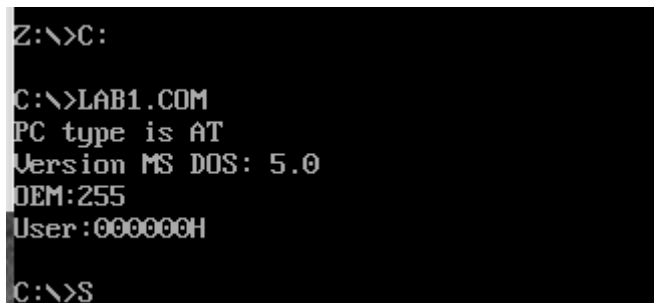
В ходе работы были написаны две процедуры: CHECK_PC_TYPE и CHECK_OS_VERS. Они реализованы с использованием шаблона из методических материалов.

CHECK_PC_TYPE: получаем информацию о типе PC из последнего байта ROMBIOS и выводим ее с помощью процедуры PRINT.

CHECK_OS_VERS: с помощью функции 30h прерывания 21h получаем информацию о версии MS DOS, затем с помощью процедур BYTE_TO_DEC, WRD_TO_HEX и BYTE_TO_HEX записываем полученную информацию в строки, далее выводим их.

В ходе работы было написано два файла lab1.asm (“Хороший” .com модуль и “плохой” .exe модуль) и prog_exe.asm (“Хороший” .exe модуль).

Результаты их запусков:



```
Z:\>C:
C:\>LAB1.COM
PC type is AT
Version MS DOS: 5.0
OEM:255
User:000000H
C:\>S
```

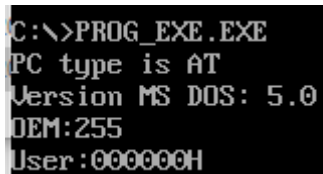
Рисунок 1 – “Хороший” .com модуль



```
C:\>LAB1.EXE
PC type is PC
Version MS DOS: 5.0
OEM: 255
User: 000000H
C:\>S_
```

Рисунок 2 – “Плохой” .exe модуль

Для того, чтобы получить “хороший” .exe модуль следует выделить сегменты стека, кода и данных, убрать ненужные директивы и установить сегмент данных.



```
C:\>PROG_EXE.EXE  
PC type is AT  
Version MS DOS: 5.0  
OEM:255  
User:000000H
```

Рисунок 3 – “Хороший” .exe модуль

Исходный программный код смотрите в приложении А.

Контрольные вопросы.

1. Отличие исходных текстов COM и EXE программ:

- Сколько сегментов должна содержать COM-программа?

Ответ: Один сегмент. Код и данные находятся в одном сегменте, а стек генерируется автоматически.

- EXE?

Ответ: один и больше

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

Ответ: директива `org 100h`, которая смещает адресацию на размер PSP, а именно на 256 байтов, а также `ASSUME`, т.к. без этой директивы компилятор не будет знать какой сегмент относится к какому сегментному регистру.

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

Ответ: нет, не все, в число запрещенных входят команды вида `mov <регистр>, seg <имя сегмента>` так как в COM-файле отсутствует таблица настроек.

2. Отличия форматов файлов COM и EXE модулей:

- Какова структура файла COM? С какого адреса располагается код?

Ответ: COM файлы состоят из одного сегмента, адресация которого начинается с `0h`, но при загрузке программы в память, код сместиться на `100h`, и будет начинаться сразу после PSP.

```
0000000010: 0D 0A 24 50 43 20 74 79 70 65 20 69 73 20 50 43  J$PC type is PC
0000000020: 2F 58 54 0D 0A 24 50 43 20 74 79 70 65 20 69 73  /XTJ$PC type is
0000000030: 20 41 54 0D 0A 24 50 43 20 74 79 70 65 20 69 73  ATJ$PC type is
0000000040: 20 50 53 32 20 D0 BC D0 BE D0 B4 D0 B5 D0 BB D1  PS2 D%D'D'DµD»N
0000000050: 8C 20 33 30 0D 0A 24 50 43 20 74 79 70 65 20 69  PC type i
0000000060: 73 20 50 53 32 20 D0 BC D0 BE D0 B4 D0 B5 D0 BB  s PS2 D%D'D'DµD»
0000000070: D1 8C 20 35 30 20 D0 B8 D0 BB D0 B8 20 36 30 0D  N 50 D.D»D. 60J
0000000080: 0A 24 50 43 20 74 79 70 65 20 69 73 20 50 53 32  $PC type is PS2
0000000090: 20 D0 BC D0 BE D0 B4 D0 B5 D0 BB D1 8C 20 38 30  D%D'D'DµD»N 80
00000000A0: 0D 0A 24 50 43 20 74 79 70 65 20 69 73 20 50 D0  J$PC type is PD
00000000B0: A1 6A 72 0D 0A 24 50 43 20 74 79 70 65 20 69 73  jjrJ$PC type is
00000000C0: 20 50 43 20 43 6F 6E 76 65 72 74 69 62 6C 65 0D  PC ConvertibleJ
00000000D0: 0A 24 50 43 20 74 79 70 65 20 69 73 20 0D 0A 24  $PC type is J$
00000000E0: 56 65 72 73 69 6F 6E 20 4D 53 20 44 4F 53 3A 20  Version MS DOS:
00000000F0: 20 2E 20 20 0D 0A 24 4F 45 4D 3A 20 20 0D 0A 24  . J$OEM: J$
0000000100: 55 73 65 72 3A 20 20 20 20 20 20 48 0D 0A 24 24  User: HJ$
0000000110: 0F 3C 09 76 02 04 07 04 30 C3 51 8A E0 E8 EF FF  o<ov♦♦♦0AQ5àèiy
0000000120: 86 C4 B1 04 D2 E8 E8 E6 FF 59 C3 53 8A FC E8 E9  †Ã±♦ÔèèæÿYÃS5üèé
0000000130: FF 88 25 4F 88 05 4F 8A C7 E8 DE FF 88 25 4F 88  y"%O"♦05Çèpy"%O"
0000000140: 05 5B C3 51 52 32 E4 33 D2 B9 0A 00 F7 F1 80 CA  †[ÃQR2ã30†± ÷ñÊ
0000000150: 30 88 14 4E 33 D2 3D 0A 00 73 F1 3C 00 74 04 0C  0~¶N30= sñ< t♦Q
0000000160: 30 88 04 5A 59 C3 50 B4 09 CD 21 58 C3 B8 00 F0  0♦ZVÃP´oÍ!XÃ. ð
0000000170: 8E C0 26 8A 26 FE FF 80 FC FF 74 3B 80 FC FE 74  Ž&5&pyëÿt;€ÿt
0000000180: 3C 80 FC FB 74 37 80 FC FC 74 38 80 FC FA 74 39  <ëÿt7€ÿt8€ÿt9
0000000190: 80 FC FC 74 3A 80 FC F8 74 38 80 FC FD 74 3C 80  €ÿt:€ÿt;€ÿt<€
00000001A0: FC F9 74 3D BF D2 01 83 C7 0B 8A C4 E8 6B FF 89  ÿt=;00fCđ5Ãèky%
00000001B0: 05 BA D2 01 EB 31 90 BA 03 01 EB 2B 90 BA 13 01  †00è100♥0è+00!!0
00000001C0: EB 25 90 BA 26 01 EB 1F 90 BA 36 01 EB 19 90 BA  è%00&00è▼0060è↓00
00000001D0: 57 01 EB 13 90 BA 82 01 EB 0D 90 BA A3 01 EB 07  W0è!!00,0èJ00f0è•
00000001E0: 90 BA B6 01 EB 01 90 E8 7C FF C3 B4 30 CD 21 50  00¶0è00è|ÿÃ´0Í!P
00000001F0: BE E0 01 83 C6 10 E8 4A FF 58 83 C6 03 8A C4 E8  %à0fA»-èJÿXfA♥5Ãè
0000000200: 41 FF BA E0 01 E8 5E FF BE F7 01 83 C6 06 8A C7  Ayèà0è^ÿ%+0fA•5Ç
0000000210: E8 30 FF BA F7 01 E8 4D FF BF 00 02 83 C7 0A 8B  è0ÿè+0èMÿ; 0fCè<
0000000220: C1 E8 07 FF 8A C3 E8 F1 FE 83 EF 02 89 05 BA 00  Áè•ÿ5Ãèñpfi0%†è
0000000230: 02 E8 32 FF C3 E8 35 FF E8 B0 FF 32 C0 B4 4C CD  0è2ÿÃè5ÿè°ÿ2Ã´LÍ
0000000240: 21 !

1Help 2Text 3Quit 4Dump 5 6Edit 7Search 8ANSI 9 10Quit 11Plugins 12Screen
```

Рисунок 4 – “Хороший” СОМ модуль в 16-ричном виде

- Какова структура файла “плохого” EXE? С какого адреса располагается код? Что располагается с адреса 0?

Ответ: В «плохом» EXE данные и код располагаются в одном сегменте, что для EXE файла некорректно, так как код и данные должны быть разделены на отдельные сегменты. Код располагается с адреса 300h, а с адреса 0h идёт таблица настроек.


```
C:\education\programming\tools\PROG EXE.EXE |h|1252|1611|Col 0|100%|0:50
0000000370: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 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 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 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 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 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 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 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 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 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000400: 50 43 20 74 79 70 65 20 69 73 20 50 43 0D 0A 24 PC type is PC!$
0000000410: 50 43 20 74 79 70 65 20 69 73 20 50 43 2F 58 54 PC type is PC/XT
0000000420: 0D 0A 24 50 43 20 74 79 70 65 20 69 73 20 41 54 !$PC type is AT
0000000430: 0D 0A 24 50 43 20 74 79 70 65 20 69 73 20 50 53 !$PC type is PS
0000000440: 32 20 D0 BC D0 BE D0 B4 D0 B5 D0 BB D1 8C 20 33 2 0%0%0'Dµ0»ÑE 3
0000000450: 30 0D 0A 24 50 43 20 74 79 70 65 20 69 73 20 50 0!$PC type is P
0000000460: 53 32 20 D0 BC D0 BE D0 B4 D0 B5 D0 BB D1 8C 20 S2 0%0%0'Dµ0»ÑE
0000000470: 35 30 20 D0 B8 D0 BB D0 B8 20 36 30 0D 0A 24 50 50 D,D»D, 60!$P
0000000480: 43 20 74 79 70 65 20 69 73 20 50 53 32 20 D0 BC C type is PS2 0%
0000000490: D0 BE D0 B4 D0 B5 D0 BB D1 8C 20 38 30 0D 0A 24 0%D'Dµ0»ÑE 80!$
00000004A0: 50 43 20 74 79 70 65 20 69 73 20 50 D0 A1 6A 72 PC type is PD!jr
00000004B0: 0D 0A 24 50 43 20 74 79 70 65 20 69 73 20 50 43 !$PC type is PC
00000004C0: 20 43 6F 6E 76 65 72 74 69 62 6C 65 0D 0A 24 50 Convertible!$P
00000004D0: 43 20 74 79 70 65 20 69 73 20 0D 0A 24 56 65 72 C type is !$Ver
00000004E0: 73 69 6F 6E 20 4D 53 20 44 4F 53 3A 20 20 2E 20 sion MS DOS: .
00000004F0: 20 0D 0A 24 4F 45 4D 3A 20 20 0D 0A 24 55 73 65 !$OEM: !$Use
0000000500: 72 3A 20 20 20 20 20 20 48 0D 0A 24 00 00 00 00 r: H!$
0000000510: 24 0F 3C 09 76 02 04 07 04 30 C3 51 8A E0 E8 EF $o<ov0♦♦0AQ5àèi
0000000520: FF 86 C4 B1 04 D2 E8 E8 E6 FF 59 C3 53 8A FC E8 y†Ã±♦0èèyYÄSüè
0000000530: E9 FF 88 25 4F 88 05 4F 8A C7 E8 DE FF 88 25 4F éy^%0^+O5Cèpy^%0
0000000540: 88 05 5B C3 51 52 32 E4 33 D2 B9 0A 00 F7 F1 80 ^+[ÄQR2à30^1  ÷ñ€
0000000550: CA 30 88 14 4E 33 D2 3D 0A 00 73 F1 3C 00 74 04 Ê0^JN30= sñ< t♦
0000000560: 0C 30 88 04 5A 59 C3 50 B4 09 CD 21 58 C3 B8 00 90♦ZYÄP^oÍ!XÄ,
0000000570: F0 8E C0 26 8A 26 FE FF 80 FC FF 74 3B 80 FC FE ðŽÄ&S&pyëüyt;ëüp
0000000580: 74 3C 80 FC FB 74 37 80 FC FC 74 38 80 FC FA 74 t<ëüüt7ëüüt8ëüüt
0000000590: 39 80 FC FC 74 3A 80 FC F8 74 3B 80 FC FD 74 3C 9ëüüt:ëüöt;ëüýt<
00000005A0: 80 FC F9 74 3D BF CF 00 83 C7 0B 8A C4 E8 6B FF ëüüt=¿I fçðŠÄèky
00000005B0: 89 05 BA CF 00 EB 31 90 BA 00 00 EB 2B 90 BA 10 %♦eI ë1  ë+  ►
00000005C0: 00 EB 25 90 BA 23 00 EB 1F 90 BA 33 00 EB 19 90 ë%  # ë▼ 3 ë↓
00000005D0: BA 54 00 EB 13 90 BA 7F 00 EB 0D 90 BA A0 00 EB 0T ë!!  ò ëJ  ë
00000005E0: 07 90 BA B3 00 EB 01 90 E8 7C FF C3 B4 30 CD 21 • ³ ë  ëlyÄ^oÍ!
00000005F0: 50 BE DD 00 83 C6 10 E8 4A FF 58 83 C6 03 8A C4 P%Y fÆ»ëJYxfÆ♥ŠÄ
0000000600: E8 41 FF BA DD 00 E8 5E FF BE F4 00 83 C6 06 8A èAyºY è^y%0 fÆ♦Š
0000000610: C7 E8 30 FF BA F4 00 E8 4D FF BF FD 00 83 C7 0A Çè0yº0 èMy¿y fÇ
0000000620: 8B C1 E8 07 FF 8A C3 E8 F1 FE 83 EF 02 89 05 BA <Äè•yŠÄèñpfio  ♦º
0000000630: FD 00 E8 32 FF C3 1E 33 C0 50 B8 20 00 8E D8 E8 ý è2yÄ▲3ÄP, Ž0è
0000000640: 2C FF E8 A7 FF 32 C0 B4 4C CD 21 ,yèšy2Ä^LÍ!
1Help 2Text 3Quit 4Dump 5 6Edit 7Search 8ANSI 9 10Quit 11Plugin 12Screen
```

Рисунок 6 – “Хороший” EXE модуль в 16-ричном виде

2. Загрузка COM модуля в основную память:

- Какой формат загрузки модуля COM? С какого адреса располагается код?

Ответ: Определяется сегментный адрес участка ОП, у которого достаточно места для загрузки программы, образ COM-файла считывается с диска и помещается в память, начиная с PSP:0100h. После загрузки двоичного образа COM-программы сегментные регистры CS, DS, ES и SS указывают на PSP(в данном случае сегментные регистры указывают на 48DD), SP указывает на конец сегмента PSP(обычно FFFE), слово 00H помещено в стек, IP содержит 100H в результате команды JMP PSP:100H.

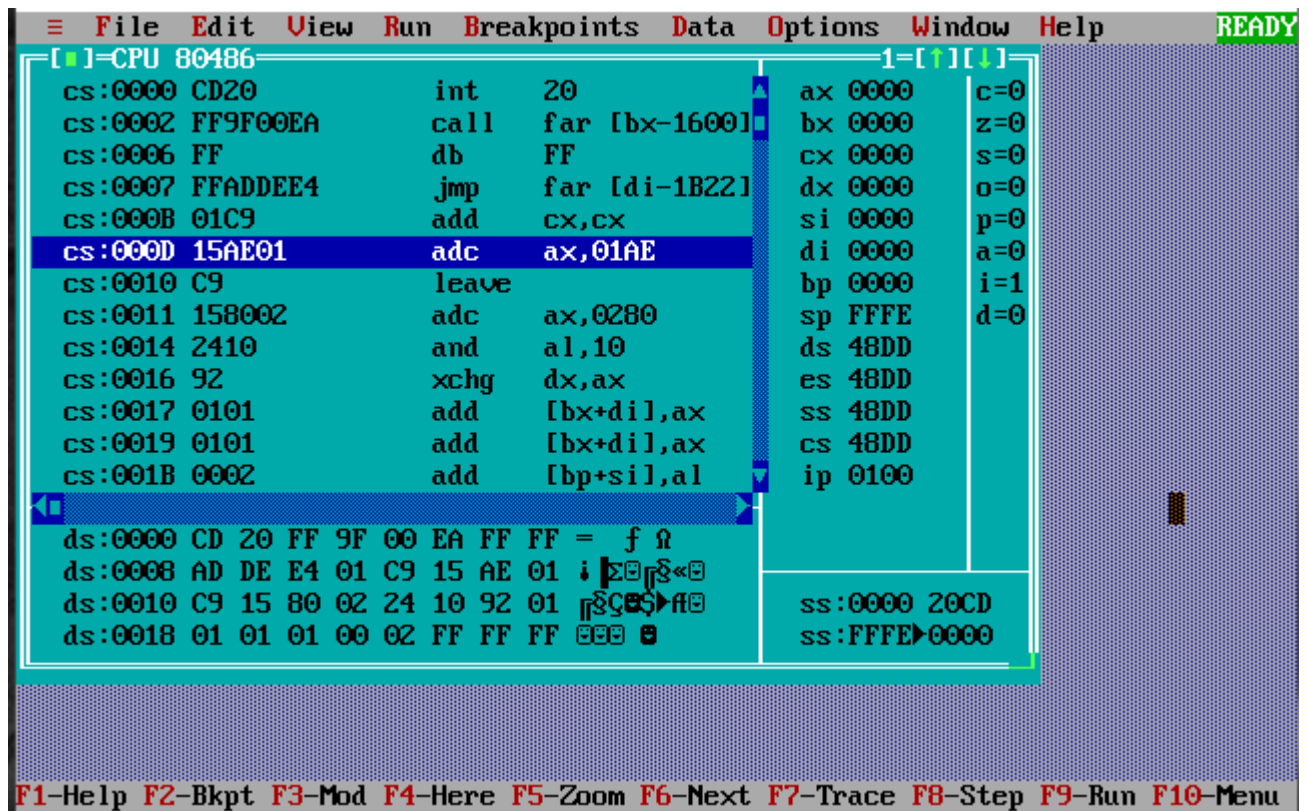


Рисунок 7 – “Хорошая” COM программа в отладчике

- Что располагается с адреса 0?

Ответ: PSP (Program Segment Prefix).

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

Ответ: Все сегментные регистры имеют значение 48DD и указывают на начало PSP.

- Как определяется стек? Какую область памяти он занимает?

Какие адреса?

Ответ: Стек генерируется автоматически при создании COM-программы. SS – на начало (0h), регистр SP указывает на конец стека (FFFEh). Таким образом, стек занимает адреса SS:0000h-SS:FFFEh.

3. Загрузка “хорошего” EXE модуля в основную память

- Как загружается “хороший” EXE? Какие значения имеют сегментные регистры?

Ответ: Аналогично COM модулю EXE загружается со смещением относительно PSP – 100h. Значения регистров DS=ES=48DD, CS=491E, SS=48ED.

- На что указывают регистры DS и ES?

Ответ: На начало сегменты PSP.

- Как определяется стек?

Ответ: он определяется вручную с помощью директивы SEGMENT STACK, в которой указывается размер стека.

- Как определяется точка входа?

Ответ: с помощью директивы END.

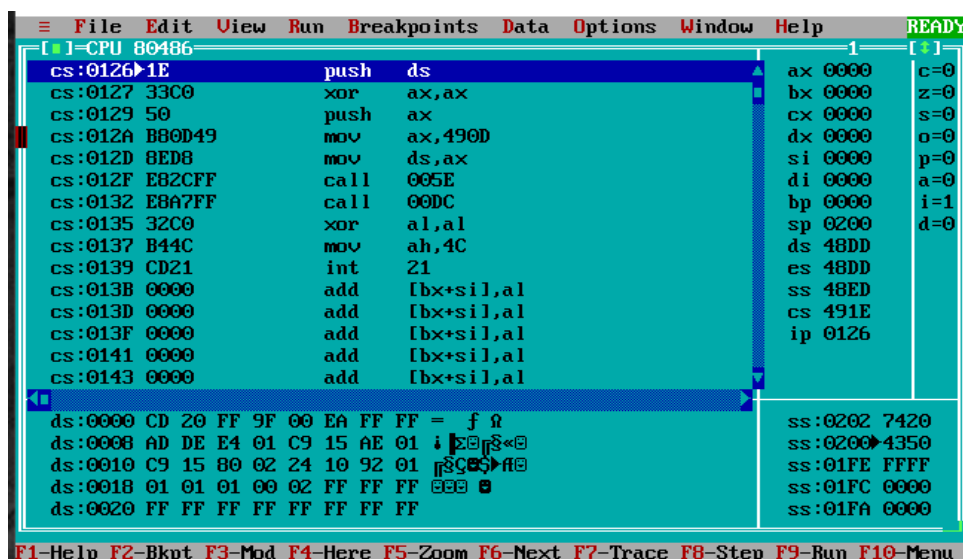


Рисунок 8 – “Хороший” EXE модуль в отладчике

Выводы.

В ходе лабораторной работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab1.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:TESTPC, SS:TESTPC
    ORG 100H
START: JMP BEGIN
; Данные
PC db  'PC type is PC',0DH,0AH,'$'
PC_XT db 'PC type is PC/XT',0DH,0AH,'$'
AT db  'PC type is AT',0DH,0AH,'$'
PS2_30 db 'PC type is PS2 модель 30',0DH,0AH,'$'
PS2_50_60 db 'PC type is PS2 модель 50 или 60',0DH,0AH,'$'
PS2_80 db 'PC type is PS2 модель 80',0DH,0AH,'$'
PCJR db 'PC type is PCjr',0DH,0AH,'$'
PC_CONVERTIBLE db 'PC type is PC Convertible',0DH,0AH,'$'
PC_OTHER db 'PC type is ',0DH,0AH,'$'

DOS db 'Version MS DOS:  . ',0DH,0AH,'$'
OEM db  'OEM:  ',0DH,0AH,'$'
USER db  'User:      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_1
    or AL,30h
    mov [SI],AL
end_1:
    pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP
;-----
PRINT PROC near
    push AX
    mov AH,09h
    int 21h
    pop AX
    ret
PRINT ENDP

CHECK_PC_TYPE PROC near
    mov ax,0F000h
    mov es,ax
    mov ah,es:[0FFFEh]
    ;mov ah,0EBh

    cmp ah,0FFh
    je _pc

    cmp ah,0FEh
    je _xt

    cmp ah, 0FBh

```

je _xt

cmp ah, 0FCh

je _at

cmp ah, 0FAh

je _ps230

cmp ah, 0FCh

je _ps25060

cmp ah, 0F8h

je _ps280

cmp ah, 0FDh

je _pcjr

cmp ah, 0F9h

je _pcconv

;если нет в таблице

mov di,offset PC_OTHER

add di,11

mov al,ah

call BYTE_TO_HEX

mov [di], ax

mov dx,offset PC_OTHER

jmp _out

_pc:

mov dx,offset PC

jmp _out

_xt:

mov dx,offset PC_XT

jmp _out

_at:

```

    mov dx,offset AT
    jmp _out

_ps230:
    mov dx,offset PS2_30
    jmp _out

_ps25060:
    mov dx,offset PS2_50_60
    jmp _out

_ps280:
    mov dx,offset PS2_80
    jmp _out

_pcjr:
    mov dx,offset PCJR
    jmp _out

_pcconv:
    mov dx,offset PC_CONVERTIBLE
    jmp _out

_out:
    call PRINT
    ret
CHECK_PC_TYPE ENDP

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

    mov si,offset DOS
    add si,16
    call BYTE_TO_DEC
    pop ax
    add si,3

```



```

mov al,ah
call BYTE_TO_DEC
mov dx,offset DOS
call PRINT

mov si,offset OEM
add si,6
mov al,bh
call BYTE_TO_DEC
mov dx ,offset OEM
call PRINT

mov di, offset USER
add di, 10
mov ax, cx
call WRD_TO_HEX
mov al, bl
call BYTE_TO_HEX
sub di,2
mov [di], ax
mov dx, offset USER
call PRINT
ret

CHECK_OS_VERS ENDP

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

```

Название файла: prog_exe.asm

```

; стек
ASTACK SEGMENT stack
    dw 256 dup(?)
ASTACK ENDS

; Данные
DATA SEGMENT
PC db  'PC type is PC',0DH,0AH,'$'
PC_XT db 'PC type is PC/XT',0DH,0AH,'$'
AT db  'PC type is AT',0DH,0AH,'$'
PS2_30 db 'PC type is PS2 модель 30',0DH,0AH,'$'
PS2_50_60 db 'PC type is PS2 модель 50 или 60',0DH,0AH,'$'
PS2_80 db 'PC type is PS2 модель 80',0DH,0AH,'$'
PCJR db 'PC type is PCjr',0DH,0AH,'$'
PC_CONVERTIBLE db 'PC type is PC Convertible',0DH,0AH,'$'
PC_OTHER db 'PC type is ',0DH,0AH,'$'

DOS db 'Version MS DOS:  . ',0DH,0AH,'$'
OEM db  'OEM: ',0DH,0AH,'$'
USER db  'User:      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 PROC near
push AX
mov AH,09h
int 21h
pop AX
ret
PRINT ENDP

CHECK_PC_TYPE PROC near
mov ax,0F000h
mov es,ax
mov ah,es:[0FFFEh]

cmp ah,0FFh
je _pc

cmp ah,0FEh
je _xt

cmp ah, 0FBh
je _xt

```

```

cmp ah, 0FCh
je _at

cmp ah, 0FAh
je _ps230

cmp ah, 0FCh
je _ps25060

cmp ah, 0F8h
je _ps280

cmp ah, 0FDh
je _pcjr

cmp ah, 0F9h
je _pcconv

;если нет в таблице
mov di,offset PC_OTHER
add di,11
mov al,ah
call BYTE_TO_HEX
mov [di], ax
mov dx,offset PC_OTHER
jmp _out

_pc:
mov dx,offset PC
jmp _out

_xt:
mov dx,offset PC_XT
jmp _out

_at:
mov dx,offset AT

```

```

    jmp _out

_ps230:
    mov dx,offset PS2_30
    jmp _out

_ps25060:
    mov dx,offset PS2_50_60
    jmp _out

_ps280:
    mov dx,offset PS2_80
    jmp _out

_pcjr:
    mov dx,offset PCJR
    jmp _out

_pcconv:
    mov dx,offset PC_CONVERTIBLE
    jmp _out

_out:
    call PRINT
    ret
CHECK_PC_TYPE ENDP

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

    mov si,offset DOS
    add si,16
    call BYTE_TO_DEC
    pop ax
    add si,3
    mov al,ah

```

```

call BYTE_TO_DEC
mov dx,offset DOS
call PRINT

mov si,offset OEM
add si,6
mov al,bh
call BYTE_TO_DEC
mov dx ,offset OEM
call PRINT

mov di, offset USER
add di, 10
mov ax, cx
call WRD_TO_HEX
mov al, bl
call BYTE_TO_HEX
sub di,2
mov [di], ax
mov dx, offset USER
call PRINT
ret
CHECK_OS_VERS ENDP

MAIN PROC far
push DS
    xor AX,AX
    push AX
    mov AX,DATA
    mov DS,AX

call CHECK_PC_TYPE
call CHECK_OS_VERS
xor AL,AL
mov AH,4Ch
int 21H
MAIN ENDP

```

CODE ENDS

END MAIN