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

ОТЧЕТ

по практической работе №1

по дисциплине «Операционные системы»

ТЕМА: ИССЛЕДОВАНИЕ СТРУКТУР ЗАГРУЗОЧНЫХ МОДУЛЕЙ

Студентка гр. 7383 _____ Чемова К.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2019

Постановка задачи.

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

Ход работы.

В данной работе были использованы функции, представленные как шаблон в методических указаниях:

- TETR_TO_HEX – переводит число в шестнадцатеричную систему счисления из двоичной;
- BYTE_TO_HEX – переводит байтовое число из регистре AL в шестнадцатеричную систему счисления
- WRD_TO_HEX – переводит число из регистре AX в шестнадцатеричную систему счисления;
- BYTE_TO_DEC – переводит число в двоичную систему счисления;

Также были написаны функции:

- GET_PC_TYPE – определяет тип компьютера;
- GET_SYS_VERS – определяет версию системы;
- GET_OEM_NUM – определяет серийный номер OEM;
- GET_SERIAL_NUM – определяет серийный номер пользователя;
- PRINT_MSG– выводит на экран строку.

Программа определяет тип компьютера, версию системы, серийный номер OEM и серийный номер пользователя, а затем выводит информацию на экран. Исходный код .COM файла представлен в приложении А, код .EXE файла в приложении Б. Результат работы программы представлен на рис. 1-3.

Рисунок 1 – Результат работы .COM файла

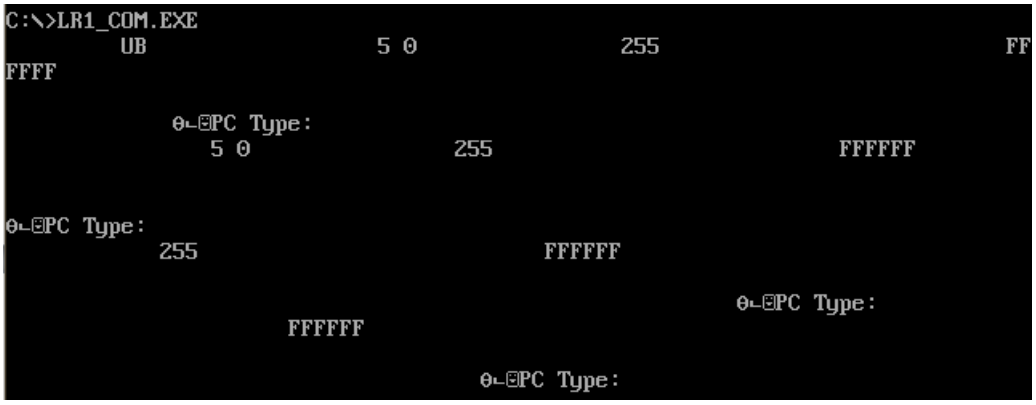

Рисунок 2 – Результат работы «плохого» .EXE файла

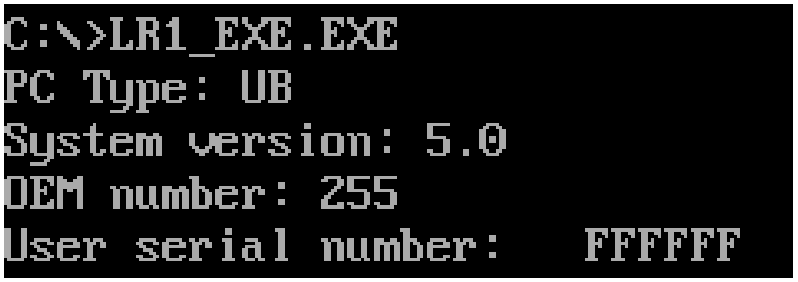

Рисунок 3 – Результат работы «хорошего» .EXE файла

Выводы.

В ходе выполнения лабораторной работы были исследованы различия структур загрузочных модулей типа .COM и .EXE. Было определено, что структура .EXE файла более сложна.

Ответы на контрольные вопросы.

Отличия исходных текстов COM и EXE программ

1. Сколько сегментов должна содержать COM-программа?

Один сегмент.

2. EXE-программа?

Как минимум из одного.

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

Должна быть включена директива ORG 100h, сдвигающая адресацию в программе на 256 байт для расположения PSP. Помимо этого, должна присутствовать директива ASSUME, ставящая в соответствие начало программы сегментам кода и данных. Отсутствие директивы ASSUME не дает программе компилироваться, поскольку невозможно найти начало сегмента кода.

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

Нет, не все. Нельзя использовать дальнюю (far) адресацию, поскольку в этих командах используется таблица настройки в которой содержатся адреса сегментов, содержащаяся только в .EXE файлах. это означает невозможность использования команды вида mov register, segment.

Отличия форматов файлов COM и EXE модулей.

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

Данный вид файлов содержит только машинный код и данные программы. На рис. 4 показано HEX представление файла. Из него видно, код располагается с нулевого адреса.

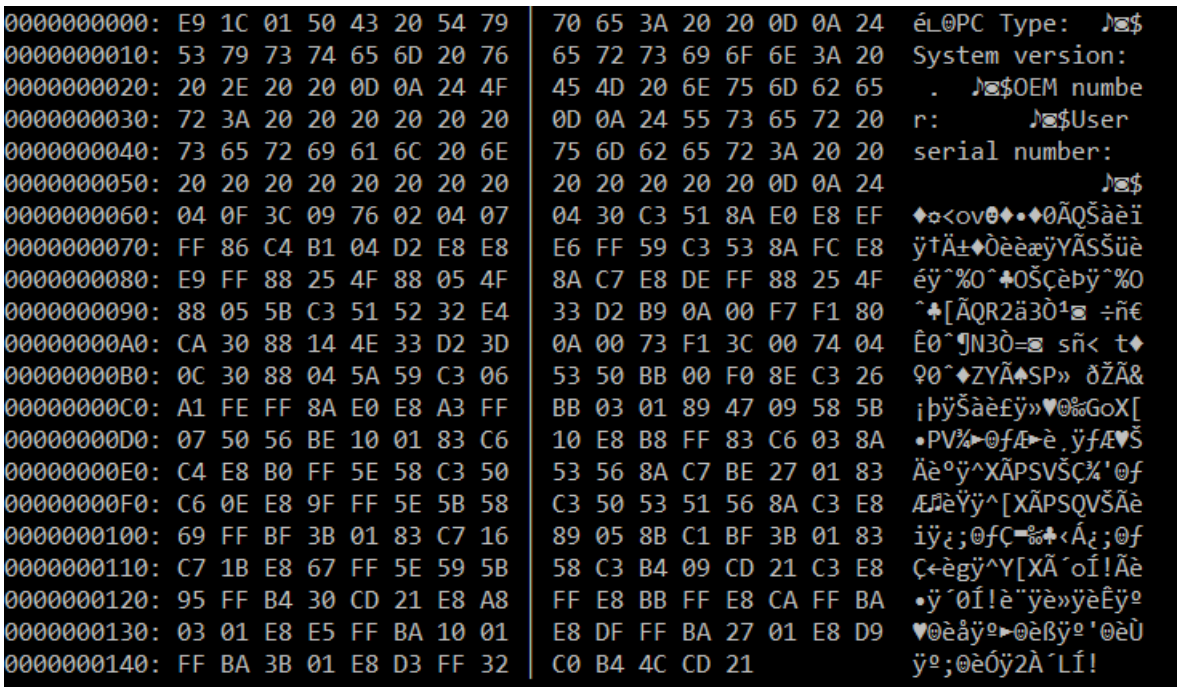

Рисунок 4 – HEX представление .COM файла

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

В «плохом» EXE код и данные не разделены по сегментам, а перемешаны. Код располагается с адреса 300h, так как заголовок занимает 200h байт, а команда ORG 100h сдвигает код еще на 100h. С нулевого адреса располагается заголовок. В первых двух байтах можно увидеть символы MZ, означающие, что формат файла – 16-битный и его следует запускать в соответствии со структурой .EXE файлов. После заголовка идет таблица настройки. Без которой, файл загружался бы в память как .COM файл.

На рисунке 5 можно увидеть HEX представление «плохого» .EXE файла.

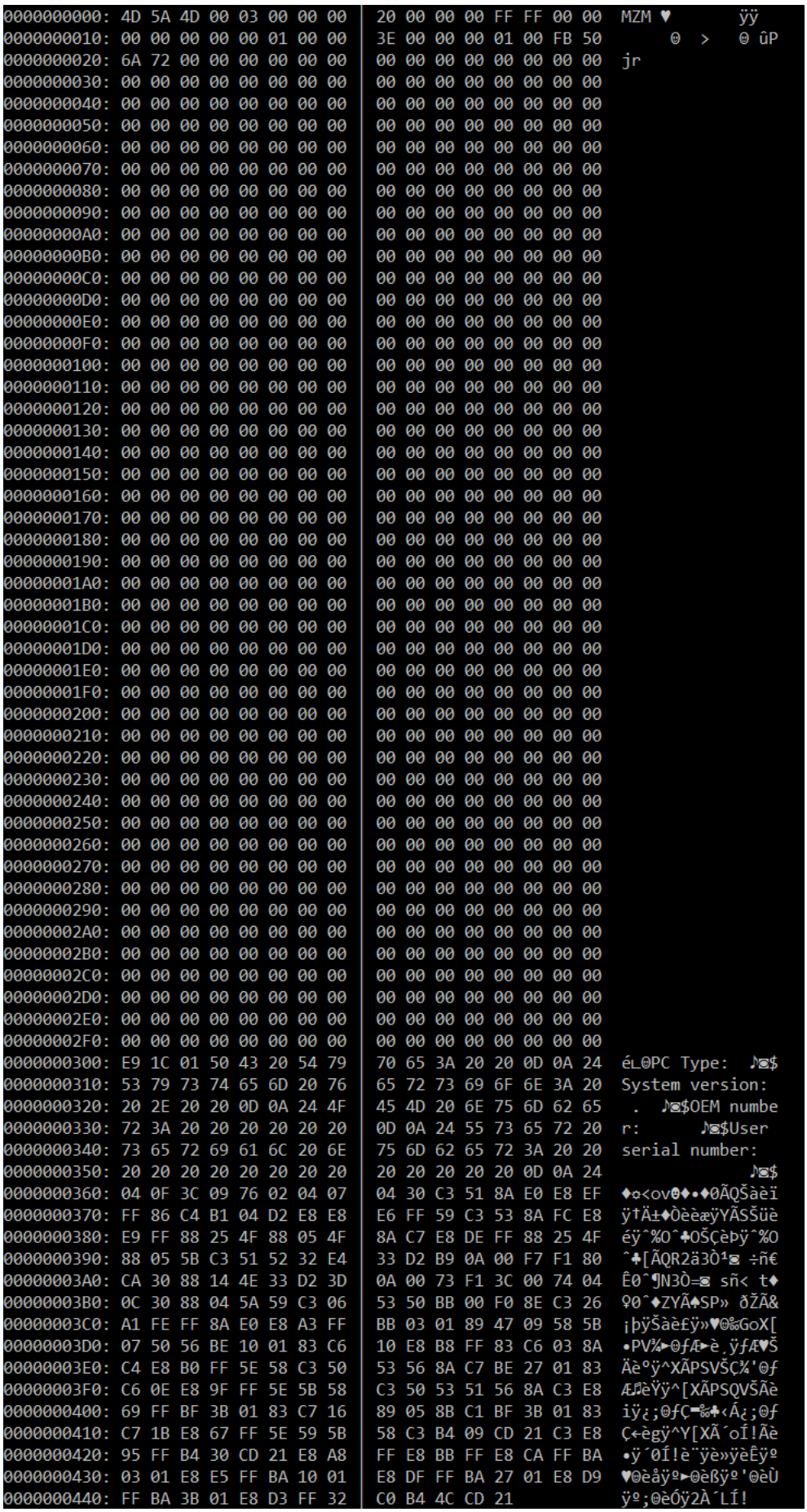

Рисунок 5 – HEX представление «плохого» .EXE файла

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

В отличие от «плохого» .EXE файла, в «хорошем» код, стек и данные выделены в отдельные сегменты. Для «хорошего» .EXE файла в директиве ORG 100h нет необходимости, загрузчик автоматически расположит программу после PSP. Код программы начинается с 200h.

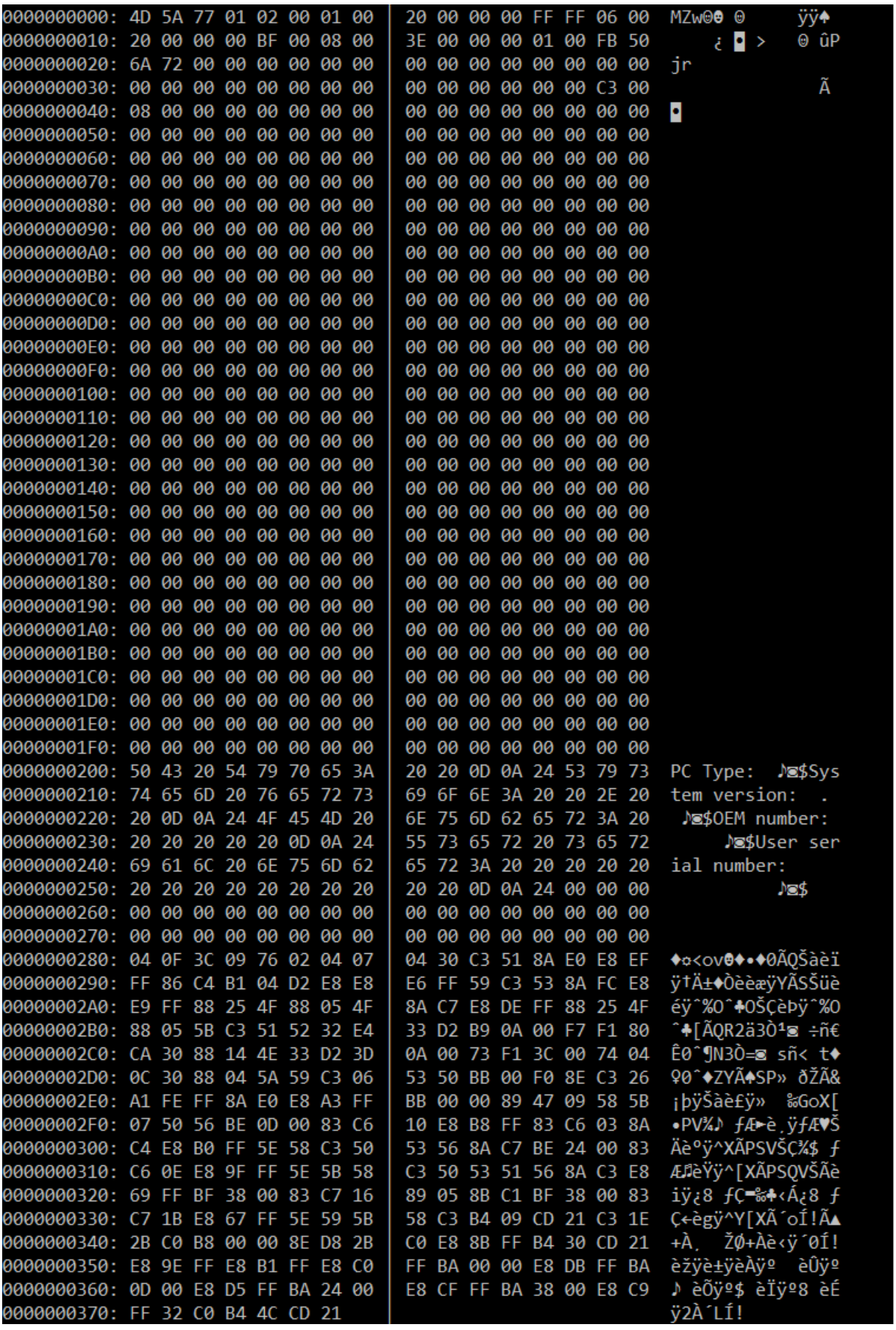

Рисунок 6 – HEX представление «хорошего» .EXE файла

Загрузка COM модуля в основную память.

На рис. 7 показан отладчик, в котором запущен .COM файл.

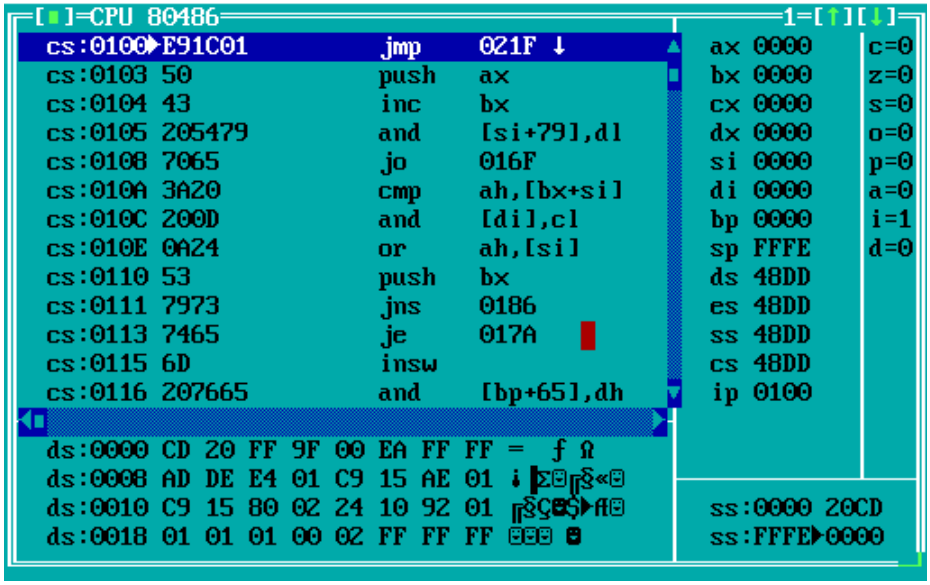

Рисунок 7 – .COM файл в отладчике

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

Сначала система выделяет свободный сегмент памяти, затем заносит его адрес во все сегментные регистры. Потом происходит построение PSP в первых 100h байтах памяти. Загружается содержимое .COM файла и присваивается регистру IP значение 100h. Указатель на стек SP ссылается на конец выделенной памяти, поэтому запись данных стека осуществляется снизу вверх.

Начало кода определяется директивой ORG 100h от начала выделенного фрагмента. Код располагается с адреса 48DD:0100h.

2. Что располагается с адреса 0?

По этому адресу располагается PSP.

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

Все сегментные регистры имеют значения 48DD, это видно из рис. 7. Они указывают на начало PSP.

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

Стек занимает весь сегмент COM-программы, его начало находится в конце сегмента. SS указывает на начало сегмента, а SP=FFFEh – на его конец. Стек может дойти до кода или данных программы при достаточном количестве элементов. 10 Адреса расположены в диапазоне 0000h-FFFEh. Стек растет от больших адресов к меньшим. Это означает, что стек может при большом количестве элементов переписать код или данные программы.

Загрузка «хорошего» EXE модуля в основную память.

На рис. 8 показан отладчик, в котором запущен .EXE файл.

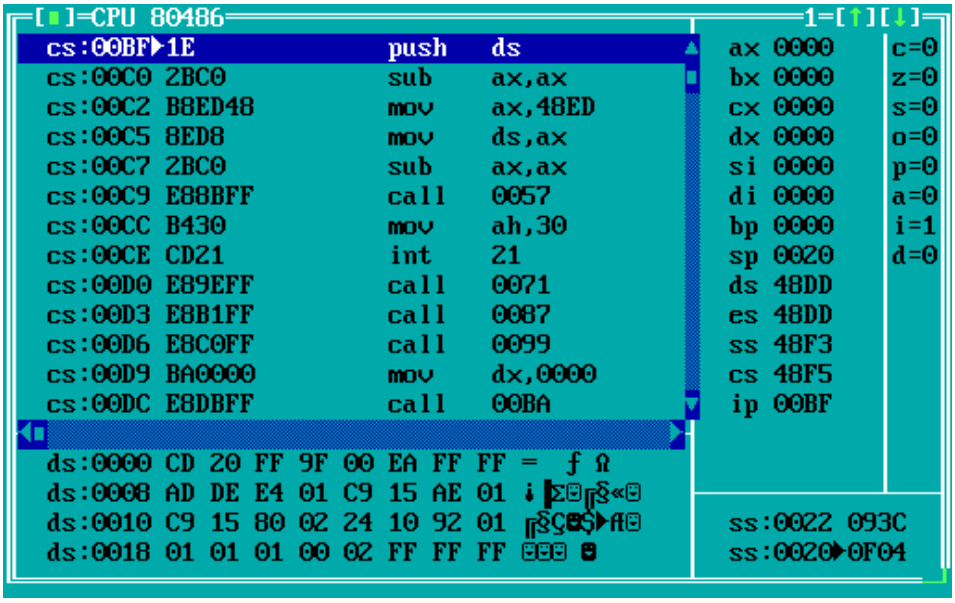

Рисунок 7 – .EXE файл в отладчике

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

Сначала загружается PSP, а дальше устанавливаются сегментные регистры. DS и ES устанавливаются на начало PSP (48DD). SS – на начало сегмента стека (48F3), CS – на начало сегмента кода (48F5).

2. На что указывают регистры DS и ES?

Они указывают на начало PSP.

3. Как определяется стек?

В исходном коде модуля стек определяется при помощи директивы STACK. При запуске программы в SS заносится адрес сегмента стека, его начало, а в SP – адрес его вершины.

4. Как определяется точка входа?

Точка входа в программу определяется с помощью директивы END. После этой директивы указывается метка, куда переходит программа при запуске.

ПРИЛОЖЕНИЕ А

LR1_COM.ASM

```
TESTPC SEGMENT
       ASSUME CS:TESTPC,  DS:TESTPC,  ES:NOTHING,  SS:NOTHING;
       ORG    100H
START: JMP    BEGIN
;ДАННЫЕ
PC_TYPE      db     'PC Type:  ',0DH,0AH,'$'
SYS_VERS     db     'System version:  .  ',0DH,0AH,'$'
OEM_NUM      db     'OEM number:      ',0DH,0AH,'$'
SER_NUM      db     'User serial number:               ',0DH,0AH,'$'
;ПРОЦЕДУРЫ
;-----------------------------------------------------------------
TETR_TO_HEX  PROC  near
       add    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 переводиться в два символа шестнадцатеричного 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 СС 16-ти разрядного числа
;в AX - числоб 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 10 СС, 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
;--------------------------------------------------------------------
GET_PC_TYPE  PROC  near
       push   ES
       push   BX
       push   AX
       mov    BX, 0F000H
       mov    ES, BX
       mov    AX, ES:[0FFFEH]
       mov    AH, AL
       call   BYTE_TO_HEX
       lea    BX, PC_TYPE
       mov    [BX+9], AX
       pop    AX
       pop    BX
       pop    ES
GET_PC_TYPE  ENDP
;--------------------------------------------------------------------
GET_SYS_VERS  PROC  near
       push   AX
       push   SI
       lea    SI, SYS_VERS
       add    SI, 16       ;основная версия
       call   BYTE_TO_DEC
       add    SI, 3        ;модификация
       mov    AL, Ah
       call   BYTE_TO_DEC
```

```
       pop    SI
       pop    AX
       ret
GET_SYS_VERS  ENDP
;--------------------------------------------------------------------
GET_OEM_NUM  PROC  near
       push   AX
       push   BX
       push   SI
       mov    AL, BH
       lea    SI, OEM_NUM
       add    SI, 14
       call   BYTE_TO_DEC
       pop    SI
       pop    BX
       pop    AX
       ret
GET_OEM_NUM  ENDP
;--------------------------------------------------------------------
GET_SERIAL_NUM  PROC  near
       push   AX
       push   BX
       push   CX
       push   SI
       mov    AL, BL
       call   BYTE_TO_HEX
       lea    DI, SER_NUM
       add    DI, 22
       mov    [DI], AX
       mov    AX, CX
       lea    DI, SER_NUM
       add    DI, 27
       call   WRD_TO_HEX
       pop    SI
       pop    CX
       pop    BX
       pop    AX
       ret
GET_SERIAL_NUM  ENDP
;--------------------------------------------------------------------
PRINT_MSG  PROC  near
       mov    AH, 09h
       int    21h
       ret
PRINT_MSG  ENDP
;--------------------------------------------------------------------
BEGIN:
;Определение типа ПК и версии ОС
       call   GET_PC_TYPE
       mov    AH, 30h
       int    21h
       call   GET_SYS_VERS
       call   GET_OEM_NUM
```

```
      call   GET_SERIAL_NUM
;-------------------------------------------------------------------
;Вывод информации
      lea    DX, PC_TYPE
      call   PRINT_MSG
      lea    DX, SYS_VERS
      call   PRINT_MSG
      lea    DX, OEM_NUM
      call   PRINT_MSG
      lea    DX, SER_NUM
      call   PRINT_MSG
;-------------------------------------------------------------------
;Выход в DOS
      xor    AL, AL
      mov    AH, 4Ch
      int    21h
TESTPC ENDS
        END  START  ;Конец модуля, START - точка входа
;-------------------------------------------------------------------
```

ПРИЛОЖЕНИЕ Б

LR1_EXE.ASM

```
DATA SEGMENT
PC_TYPE      db     'PC Type:  ',0DH,0AH,'$'
SYS_VERS     db     'System version:  .  ',0DH,0AH,'$'
OEM_NUM      db     'OEM number:      ',0DH,0AH,'$'
SER_NUM      db     'User serial number:               ',0DH,0AH,'$'
DATA ENDS

AStack SEGMENT STACK
       DW 010H DUP(?)
AStack ENDS
CODE SEGMENT
       ASSUME CS:CODE, DS:DATA, ES:NOTHING, SS:AStack
;ПРОЦЕДУРЫ
;--------------------------------------------------------------------
TETR_TO_HEX  PROC  near
       add    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 переводиться в два символа шестнадцатеричного 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 СС 16-ти разрядного числа
;в AX - числоб 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 10 СС, 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
;--------------------------------------------------------------------
GET_PC_TYPE  PROC  near
       push   ES
       push   BX
       push   AX
       mov    BX, 0F000H
       mov    ES, BX
       mov    AX, ES:[0FFFEH]
       mov    AH, AL
       call   BYTE_TO_HEX
       lea    BX, PC_TYPE
       mov    [BX+9], AX
       pop    AX
       pop    BX
       pop    ES
GET_PC_TYPE  ENDP
;--------------------------------------------------------------------
GET_SYS_VERS  PROC  near
       push   AX
       push   SI
       lea    SI, SYS_VERS
       add    SI, 16       ;основная версия
       call   BYTE_TO_DEC
```

```
       add    SI, 3        ;модификация
       mov    AL, Ah
       call   BYTE_TO_DEC
       pop    SI
       pop    AX
       ret
GET_SYS_VERS  ENDP
;--------------------------------------------------------------------
GET_OEM_NUM  PROC  near
       push   AX
       push   BX
       push   SI
       mov    AL, BH
       lea    SI, OEM_NUM
       add    SI, 14
       call   BYTE_TO_DEC
       pop    SI
       pop    BX
       pop    AX
       ret
GET_OEM_NUM  ENDP
;--------------------------------------------------------------------
GET_SERIAL_NUM  PROC  near
       push   AX
       push   BX
       push   CX
       push   SI
       mov    AL, BL
       call   BYTE_TO_HEX
       lea    DI, SER_NUM
       add    DI, 22
       mov    [DI], AX
       mov    AX, CX
       lea    DI, SER_NUM
       add    DI, 27
       call   WRD_TO_HEX
       pop    SI
       pop    CX
       pop    BX
       pop    AX
       ret
GET_SERIAL_NUM  ENDP
;--------------------------------------------------------------------
PRINT_MSG  PROC  near
       mov    AH, 09h
       int    21h
       ret
PRINT_MSG  ENDP
;--------------------------------------------------------------------
MAIN  PROC  near
       push   DS
       sub    AX, AX
       mov    AX, DATA
```

```
        mov    DS, AX
        sub    AX, AX
;Определение типа ПК и версии ОС
        call   GET_PC_TYPE
        mov    AH, 30h
        int    21h
        call   GET_SYS_VERS
        call   GET_OEM_NUM
        call   GET_SERIAL_NUM
;-------------------------------------------------------------------
;Вывод информации
        lea    DX, PC_TYPE
        call   PRINT_MSG
        lea    DX, SYS_VERS
        call   PRINT_MSG
        lea    DX, OEM_NUM
        call   PRINT_MSG
        lea    DX, SER_NUM
        call   PRINT_MSG
;-------------------------------------------------------------------
;Выход в DOS
        xor    AL, AL
        mov    AH, 4Ch
        int    21h
MAIN    ENDP
CODE     ENDS
        END MAIN
;-------------------------------------------------------------------
```