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 _____ Ивлева О.А.

Преподаватель _____ 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 0F000:0FFFEh, 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:CX - 24-битовый серийный номер пользователя.

Выполнение работы.

Был написан текст исходного .COM модуля, а также текст исходного .EXE модуля. Полученные модули были отлажен, и в результате были получены «плохой» .EXE модуль, «хороший» .COM модуль и «хороший» EXE модуль.

Результат выполнения «плохого» .EXE модуля представлен на рис. 1.

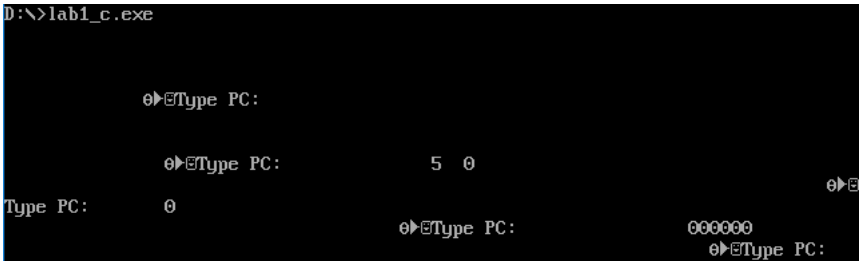

Рисунок 1 – Вывод «плохого» .EXE модуля

Результат выполнения «хорошего» .COM модуля представлен на рис. 2.

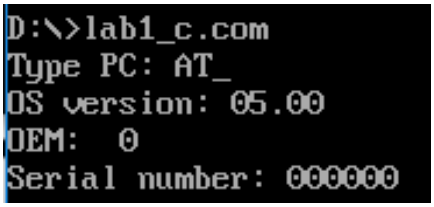

Рисунок 2 – Вывод «хорошего» .COM модуля

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

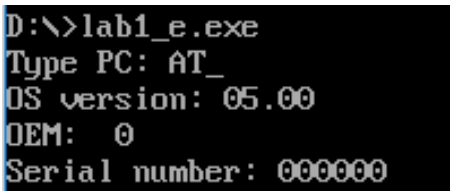

Рисунок 3 – Вывод «хорошего» .EXE модуля

Отличия исходных текстов COM и EXE программ

1. Сколько сегментов должна содержать COM-программа?

COM-программа должна содержать только один сегмент.

2. EXE-программа?

EXE-программа может содержать несколько сегментов. В программе должен содержаться минимум один сегмент данных и один сегмент кода.

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

При запуске COM-программы первые 256 байт необходимо зарезервировать для префикса программного сегмента (PSP). Для этого используется команда ORG 100h, которая устанавливает относительный адрес для начала выполнения программы.

Также обязательна директива ASSUME, которая устанавливает значения регистров CS, DS на адрес сегмента программы.

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

Нельзя использовать команды, связанные с адресом сегмента, так как он неизвестен до загрузки этого сегмента в память. Это связано с тем, что в .COM файле отсутствует таблица настройки с информацией о типе адресов и их местоположении.

Отличия форматов файлов COM и EXE модулей.

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

Вид файла COM представлен на рис. 5.

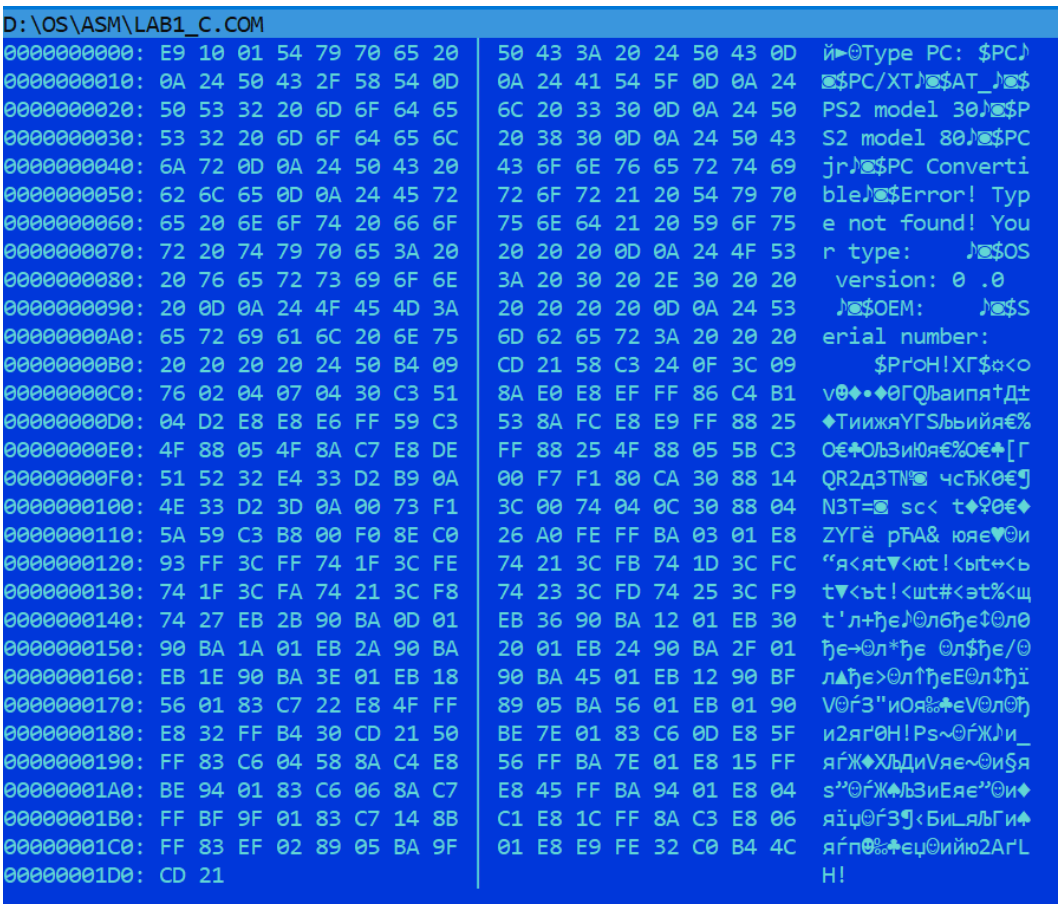

Рисунок 5 – Вид COM файла в шестнадцатеричном виде

COM-файл состоит из одного сегмента, а размер файла не превышает 64 КБ. Код располагается с адреса 0h.

Данные заканчиваются на A0. На B0 начинается сегмент кода, в котором байты отвечают за команды. В строке B0 можно найти коды команд, представленных на рис. 6.

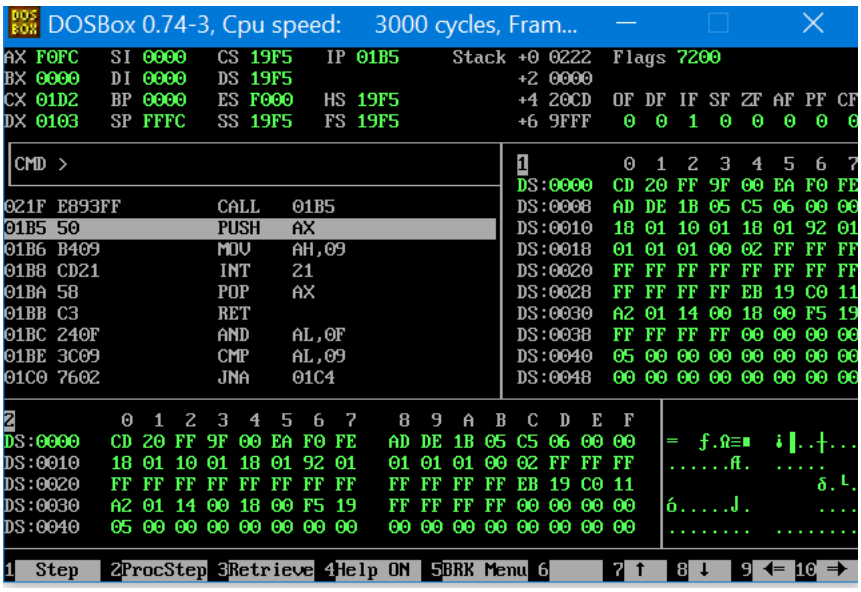

Рисунок 6 – Команды COM модуля

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

Файл «плохого» EXE файла представлен на рис. 7.

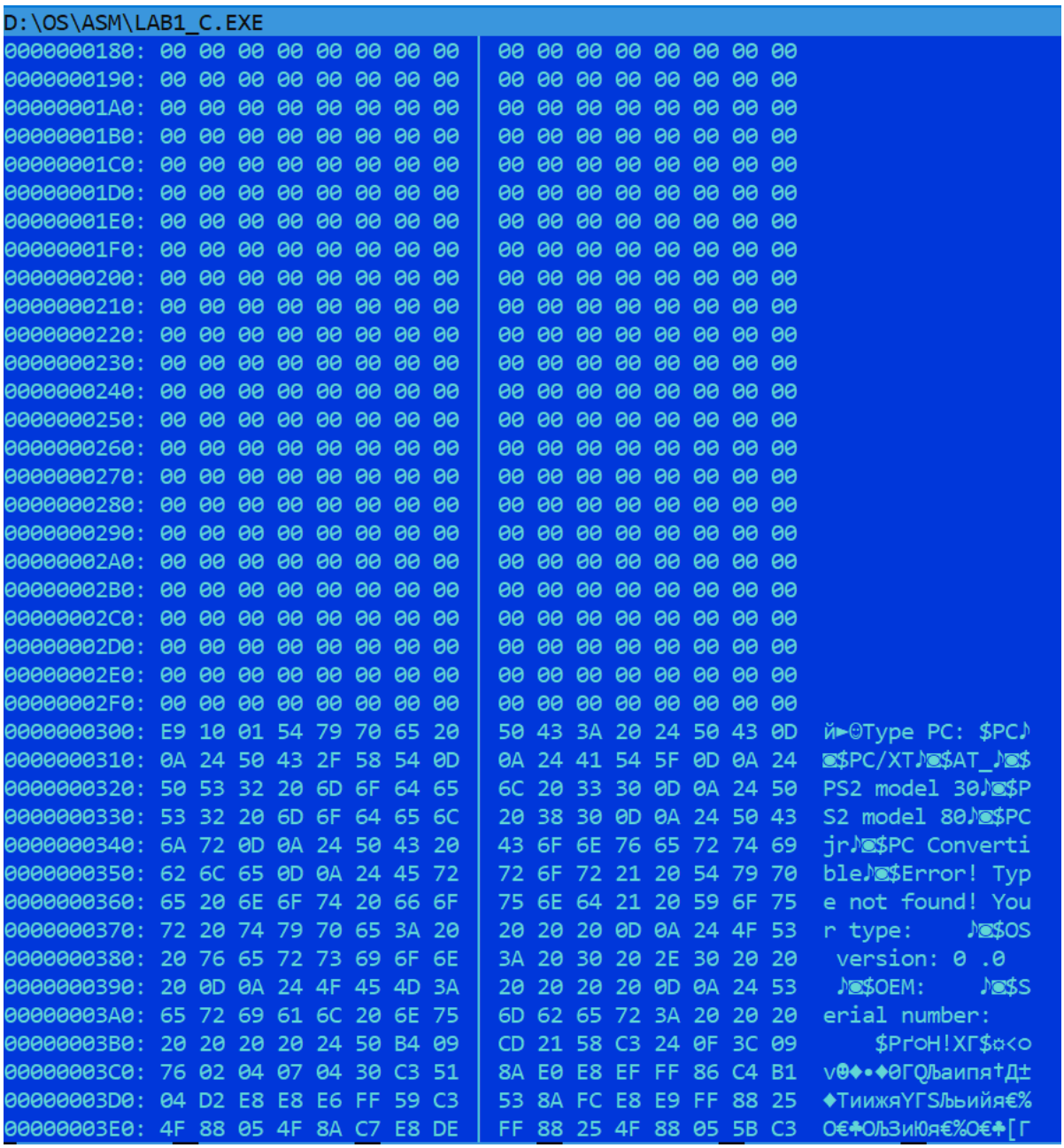

Рисунок 7 – Начало кода «плохого» EXE файла

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.

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

Начало кода в файле «хорошего» EXE представлен на рис. 8.

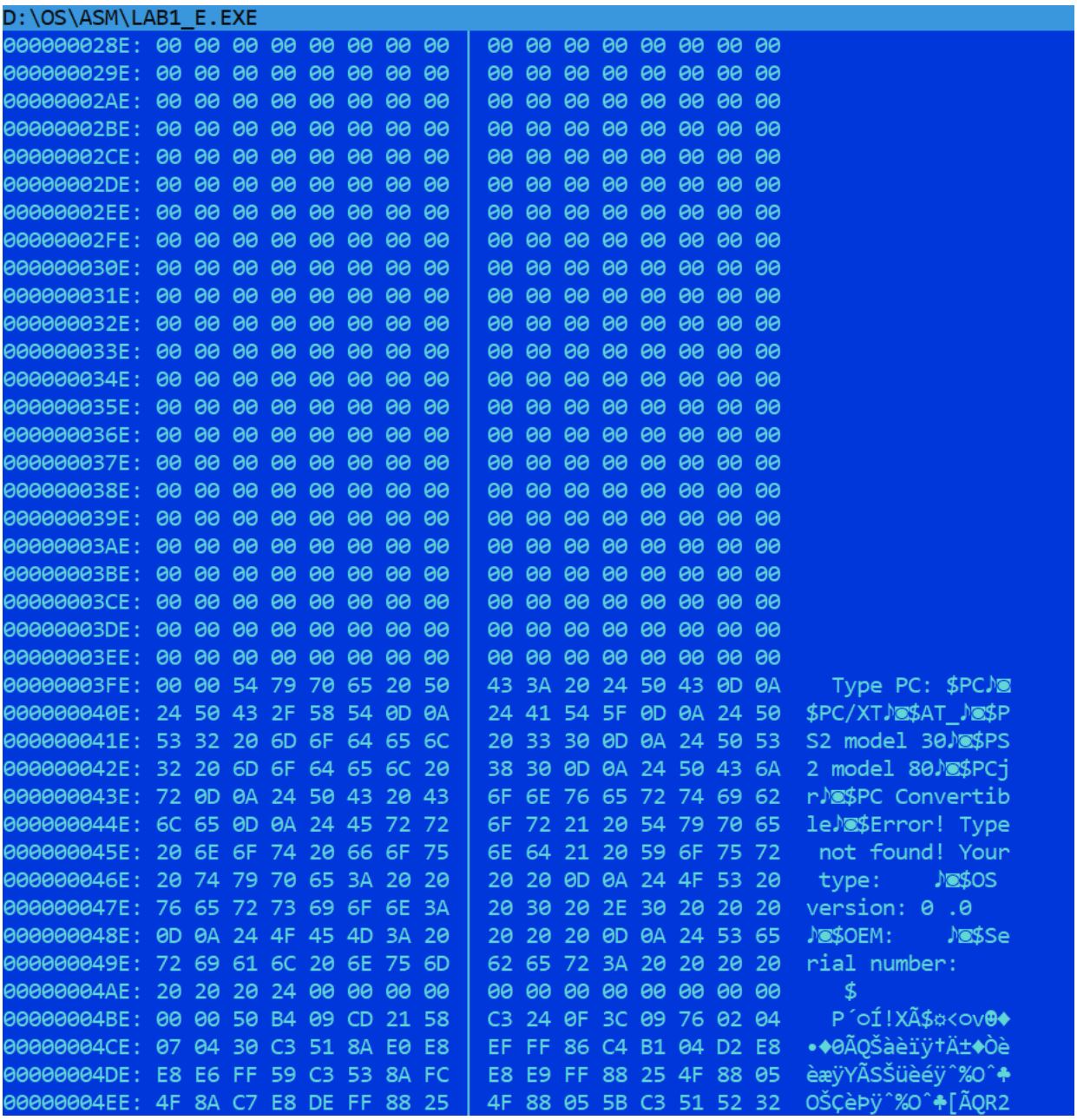

Рисунок 8 – Начало кода «хорошего» EXE файла

В «хорошем» EXE с нулевого адреса также располагается управляющая информация для загрузчика. Также перед кодом располагается сегмент стека. Так, при размере стека 200h код располагается с адреса 400h. Отличие от «плохого» EXE в том, что в «хорошем» не резервируется дополнительно 100h, которые в COM файле требовались для PSP.

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. 9.

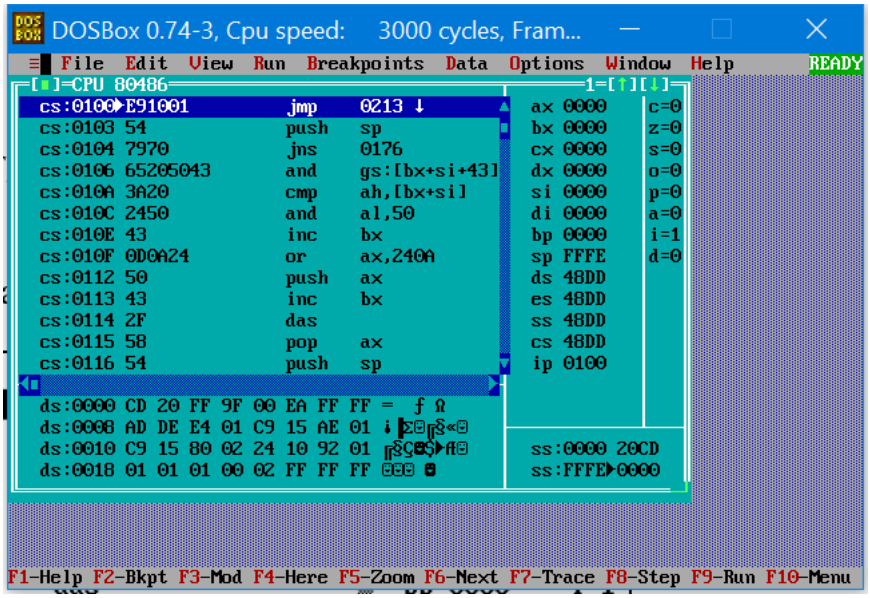

Рисунок 9 – Отладка файла .COM

После загрузки COM-программы в память сегментные регистры указывают на начало PSP. Код располагается с адреса 100h.

2. **Что располагается с адреса 0?**

С адреса 0 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. SP имеет 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. 10.

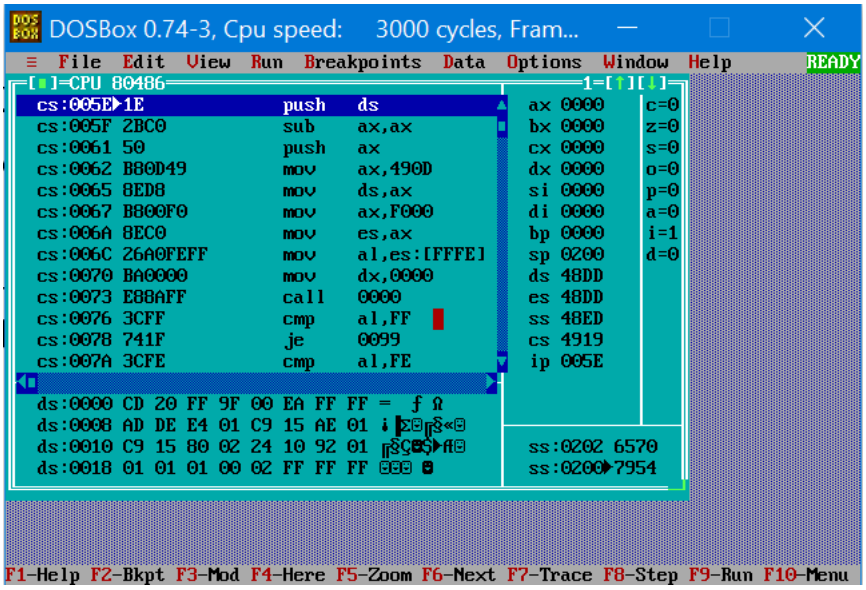

Рисунок 10 – Отладка «хорошего» EXE модуля

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, CS – на начало сегмента команд, а SS – на начало сегмента стека. IP имеет значение отличное от нуля, 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.

4. Как определяется точка входа?

Смещение точки входа в программу загружается в указатель команд IP и определяется операндом (функцией или меткой, с которой необходимо начать программу) директивы END.

Выводы.

В ходе выполнения лабораторной работы были изучены COM и EXE модули и их различия. Так же были отлажены и запущенны «хорошие» COM и EXE модули и «плохой» EXE.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. LAB1_C.ASM

```
CODE        SEGMENT

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

ORG         100H

START:      JMP   BEGIN

STR               DB "Type PC: $"
PC                DB "PC", 13,10, "$"
PC_XT       DB "PC/XT", 13,10, "$"
AT_               DB "AT_", 13,10, "$"
PS2_30            DB "PS2 model 30", 13,10, "$"
PS2_80            DB "PS2 model 80", 13,10, "$"
PCjr        DB "PCjr", 13,10, "$"
PC_Conv           DB "PC Convertible", 13,10, "$"
ERR               DB "Error! Type not found! Your type:    ", 13,10, "$"
VERS        DB "OS version: 0 .0   ", 13,10, "$"
OEM         DB "OEM:    ", 13,10, "$"
NUM         DB "Serial number:       $"


PRINT       PROC  NEAR

            PUSH  AX
            MOV   AH, 09H
         INT 21H
            POP   AX
         RET

PRINT       ENDP


TETR_TO_HEX PROC  NEAR

         AND AL, 0FH
         CMP AL, 09H
         JBE NEXT
            ADD   AL, 07H

      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, 4H
         SHR AL, CL
            CALL  TETR_TO_HEX
         POP CX
         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 10 с/с, SI - адрес поля младшей цифры
            PUSH  CX
            PUSH  DX
            XOR   AH, AH
            XOR   DX, DX
```

```
            MOV   CX, 0AH

            LOOP_BD:
                  DIV   CX
            OR    DL, 30H
            MOV   [SI], DL
                  DEC   SI
            XOR   DX, DX
            CMP   AX, 0AH
            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

      BEGIN:

        MOV AX, 0F000H
            MOV   ES, AX
            MOV   AL, ES:[0FFFEH]

            MOV   DX, OFFSET STR
            CALL  PRINT

            CMP   AL, 0FFH
            JE    TYPE1
            CMP   AL, 0FEH
            JE    TYPE2
            CMP   AL, 0FBH
            JE    TYPE2
            CMP   AL, 0FCH
            JE    TYPE3
            CMP   AL, 0FAH
            JE    TYPE4
            CMP   AL, 0F8H
            JE    TYPE5
            CMP   AL, 0FDH
            JE    TYPE6
            CMP   AL, 0F9H
            JE    TYPE7
```

```
            JMP   ELS


TYPE1:
            MOV   DX, OFFSET PC
            JMP   RES

TYPE2:
            MOV   DX, OFFSET PC_XT
            JMP   RES

TYPE3:
            MOV   DX, OFFSET AT_
            JMP   RES

TYPE4:
            MOV   DX, OFFSET PS2_30
            JMP   RES

TYPE5:
            MOV   DX, OFFSET PS2_80
            JMP   RES

TYPE6:
            MOV   DX, OFFSET PCjr
            JMP   RES

TYPE7:
            MOV   DX, OFFSET PC_Conv
            JMP   RES

ELS:
            MOV   DI, OFFSET ERR
            ADD   DI, 34
            CALL  BYTE_TO_HEX
            MOV   [DI], AX
            MOV   DX, OFFSET ERR
            JMP   RES


RES:
            CALL PRINT

            MOV   AH, 30H
            INT   21H
            PUSH  AX
```

```
                MOV   SI, OFFSET VERS
                ADD   SI, 13
                CALL  BYTE_TO_DEC

                ADD   SI, 4
                POP   AX
                MOV   AL, AH
                CALL  BYTE_TO_DEC
                MOV   DX, OFFSET VERS
                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 NUM
                ADD   DI, 20
                MOV   AX, CX
                CALL  WRD_TO_HEX
                MOV   AL, BL
                CALL  BYTE_TO_HEX
                SUB   DI, 2
                MOV   [DI], AX
                MOV   DX, OFFSET NUM
                CALL  PRINT

                XOR   AL, AL
                MOV   AH, 4CH
                INT   21H

CODE            ENDS
END             START
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. LAB1_E.ASM

```
STACK SEGMENT     STACK

            DW 100h DUP(0)

STACK       ENDS

DAT_A       SEGMENT

STR               DB "Type PC: $"
PC                DB "PC", 13,10, "$"
PC_XT       DB "PC/XT", 13,10, "$"
AT_               DB "AT_", 13,10, "$"
PS2_30            DB "PS2 model 30", 13,10, "$"
PS2_80            DB "PS2 model 80", 13,10, "$"
PCjr        DB "PCjr", 13,10, "$"
PC_Conv           DB "PC Convertible", 13,10, "$"
ERR               DB "Error! Type not found! Your type:    ", 13,10, "$"
VERS        DB "OS version: 0 .0   ", 13,10, "$"
OEM         DB "OEM:    ", 13,10, "$"
NUM         DB "Serial number:       $"

DAT_A       ENDS

CODE        SEGMENT


ASSUME      CS:CODE, DS:DAT_A, SS:STACK

PRINT       PROC  NEAR

            PUSH  AX
      MOV   AH, 09H
      INT   21H
            POP   AX
      RET

PRINT       ENDP


TETR_TO_HEX PROC  NEAR

                  AND   AL, 0FH
      CMP   AL, 09H
      JBE   NEXT
```

```
            ADD   AL, 07H

      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, 4H
     SHR    AL, CL
     CALL   TETR_TO_HEX
     POP    CX
     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 10 с/с, SI - адрес поля младшей цифры

            PUSH  CX
      PUSH  DX
      XOR   AH, AH
      XOR   DX, DX
      MOV   CX, 0AH

    LOOP_BD:
      DIV   CX
      OR    DL, 30H
      MOV   [SI], DL
                  DEC   SI
      XOR   DX, DX
      CMP   AX, 0AH
      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


MAIN        PROC  FAR

            PUSH  DS
    SUB   AX, AX
    PUSH  AX
    MOV   AX, DAT_A
    MOV   DS, AX

    MOV     AX, 0F000H
            MOV   ES, AX
            MOV   AL, ES:[0FFFEH]

            MOV   DX, OFFSET STR
            CALL  PRINT

            CMP   AL, 0FFH
            JE    TYPE1
```

```
            CMP   AL, 0FEH
            JE    TYPE2
            CMP   AL, 0FBH
            JE    TYPE2
            CMP   AL, 0FCH
            JE    TYPE3
            CMP   AL, 0FAH
            JE    TYPE4
            CMP   AL, 0F8H
            JE    TYPE5
            CMP   AL, 0FDH
            JE    TYPE6
            CMP   AL, 0F9H
            JE    TYPE7

            JMP   ELS


TYPE1:
            MOV   DX, OFFSET PC
            JMP   RES

TYPE2:
            MOV   DX, OFFSET PC_XT
            JMP   RES

TYPE3:
            MOV   DX, OFFSET AT_
            JMP   RES

TYPE4:
            MOV   DX, OFFSET PS2_30
            JMP   RES

TYPE5:
            MOV   DX, OFFSET PS2_80
            JMP   RES

TYPE6:
            MOV   DX, OFFSET PCjr
            JMP   RES

TYPE7:
            MOV   DX, OFFSET PC_Conv
            JMP   RES

ELS:
```

```
            MOV   DI, OFFSET ERR
            ADD   DI, 34
            CALL  BYTE_TO_HEX
            MOV   [DI], AX
            MOV   DX, OFFSET ERR
            JMP   RES


RES:
            CALL PRINT

            MOV   AH, 30H
            INT   21H
            PUSH  AX

    MOV     SI, OFFSET VERS
            ADD   SI, 13
            CALL  BYTE_TO_DEC

            ADD   SI, 4H
            POP   AX
            MOV   AL, AH
            CALL  BYTE_TO_DEC
            MOV   DX, OFFSET VERS
            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 NUM
            ADD   DI, 20
            MOV   AX, CX
            CALL  WRD_TO_HEX
            MOV   AL, BL
            CALL  BYTE_TO_HEX
            SUB   DI, 2
            MOV   [DI], AX
            MOV   DX, OFFSET NUM
            CALL  PRINT

RET

MAIN        ENDP
```

```
CODE        ENDS
END         MAIN
```