МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Операционные системы»

Тема: Исследование структур загрузочных модулей

Студент гр. 8383 _____ Бабенко Н.С.

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.

Сведения о функциях и структурах данных управляющей программы.

1) PRINT – процедура печати, вызывает функцию 09h. Выводит содержимое сегмента DS со смещением из регистра DX.

2) TETRTOHEX – процедура вывода байта AL в 16-ричной системе счисления.

3) BYTETOHEX – процедура перевода байта в регистре AL в два символа шестнадцатеричного числа в AX.

4) WRDTOHEX – процедура перевода в 16-ричную систему счисления 16-ти разрядного числа в регистре AX. Из регистра DI берется адрес последнего символа.

5) BYTETODEC – процедура перевода в 10-тичную систему счисления байта в регистре AL. Из регистра SI берется адрес поля младшей цифры.

Последовательность действий, выполняемых утилитой.

Программа читает содержимое предпоследнего байта ROM BIOS и по таблице, сравнивая коды, определяет тип PC. Далее выводит строку с названием модели. Если код не совпал ни с одним значением, то двоичный код переводится в символьную строку, содержащую запись шестнадцатеричного числа, строка выводится. Затем определяется версия системы, используя регистры AL, AH, по которым формируется текстовая строка, формируются строки с серийным номером OEM и серийным номером пользователя. Полученные строки выводятся на экран.

Ход работы.

1) Написан текст исходного .COM модуля. При линковке получаем «плохой» .EXE модуль и наблюдаем предупреждение линковщика о том, что не объявлен сегмент стека. Получим из него «хороший» .COM модуль при помощи EXE2BIN и запустим.

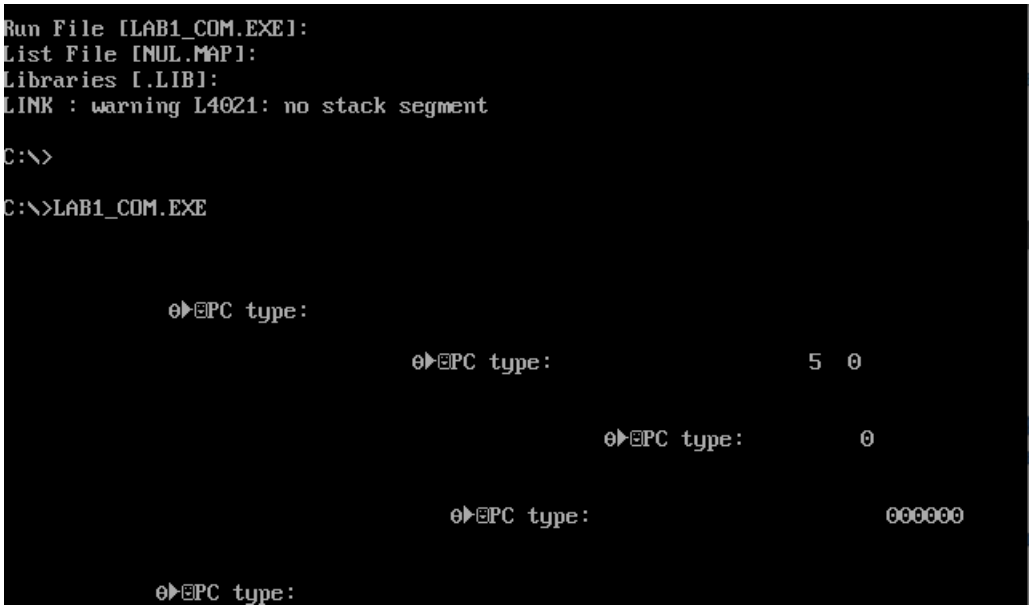

Рисунок 1 – Результат выполнения плохого .EXE модуля

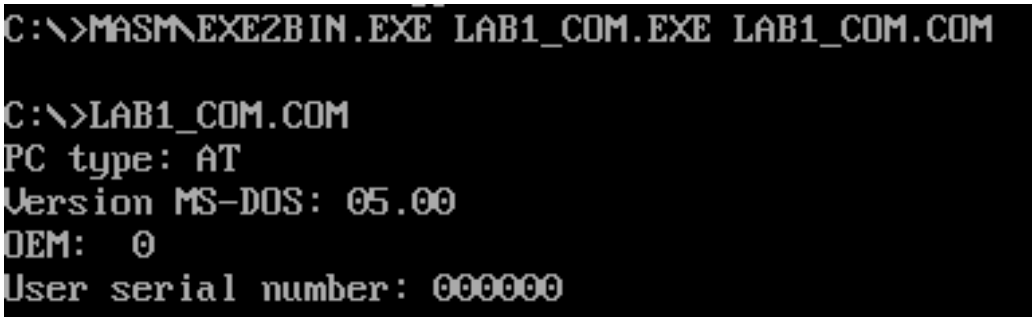

Рисунок 2 – Результат выполнения хорошего **.**COM модуля

2) Написан текст исходного .EXE модуля, который выполняет те же функции, что и модуль в шаге 1. На этот раз линковщик не выдает предупреждений.

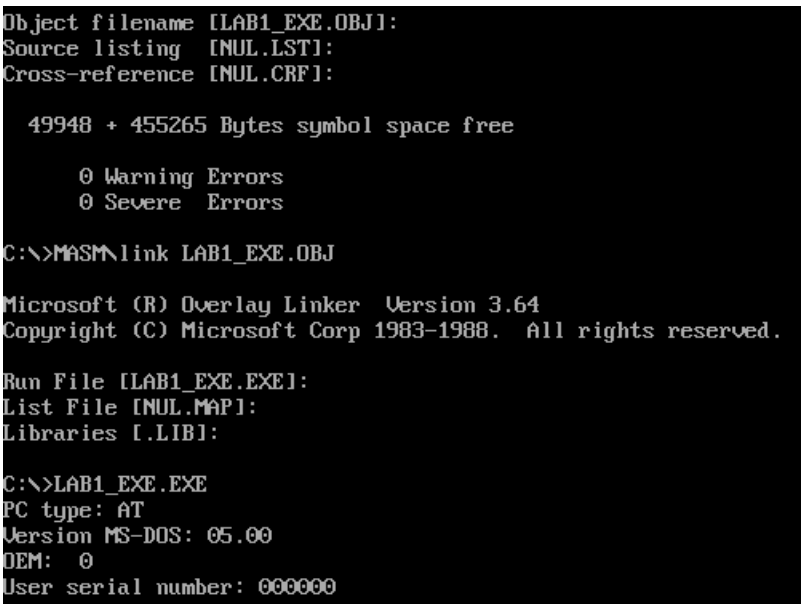

Рисунок 3 – Результат выполнения хорошего **.**EXE модуля

3) Путём сравнения исходных текстов для .COM и .EXE модулей были получены ответы на контрольные вопросы:

Отличия исходных текстов .com и .exe программ

1) COM-программа должна содержать ровно один сегмент.
2) EXE-программа должна содержать хотя бы один сегмент, может содержать и больше одного.

3) В тексте COM-программы должны быть директивы ASSUME (указывает к какому сегментному регистру привязан сегмент), ORG 100h (устанавливает значение IP в 100h - смещение от начала PSP).

4) В COM-программах нельзя использовать команды вида mov <регистр> <сегмент>, так как в такой программе всего один сегмент. Из-за того, что в .COM файлах отсутствует заголовок, в отличие от .EXE файлов, линковщик не может связать смещение до сегмента от адреса загрузки программы и возникает ошибка линковки.

В файловом менеджере FAR были открыты файлы загрузочных модулей .COM, плохого и хорошего .EXE в шестнадцатеричном виде.

Путём сравнения файлов модулей были получены ответы на контрольные вопросы.

Отличия форматов файлов COM и EXE модулей

1) COM - файл содержит только машинный код и данные программы. Код расположен с нулевого адреса.
2) Файл «плохого» EXE модуля содержит машинный код и данные в одном сегменте. Код располагается с адреса 300h.
3) В «хорошем» EXE модуле машинный код, данные и стек находятся в разных сегментах. Отличается от «плохого» EXE несколькими сегментами и наличием стека.

Рисунок 4 – Содержимое файла LAB1_COM.COM

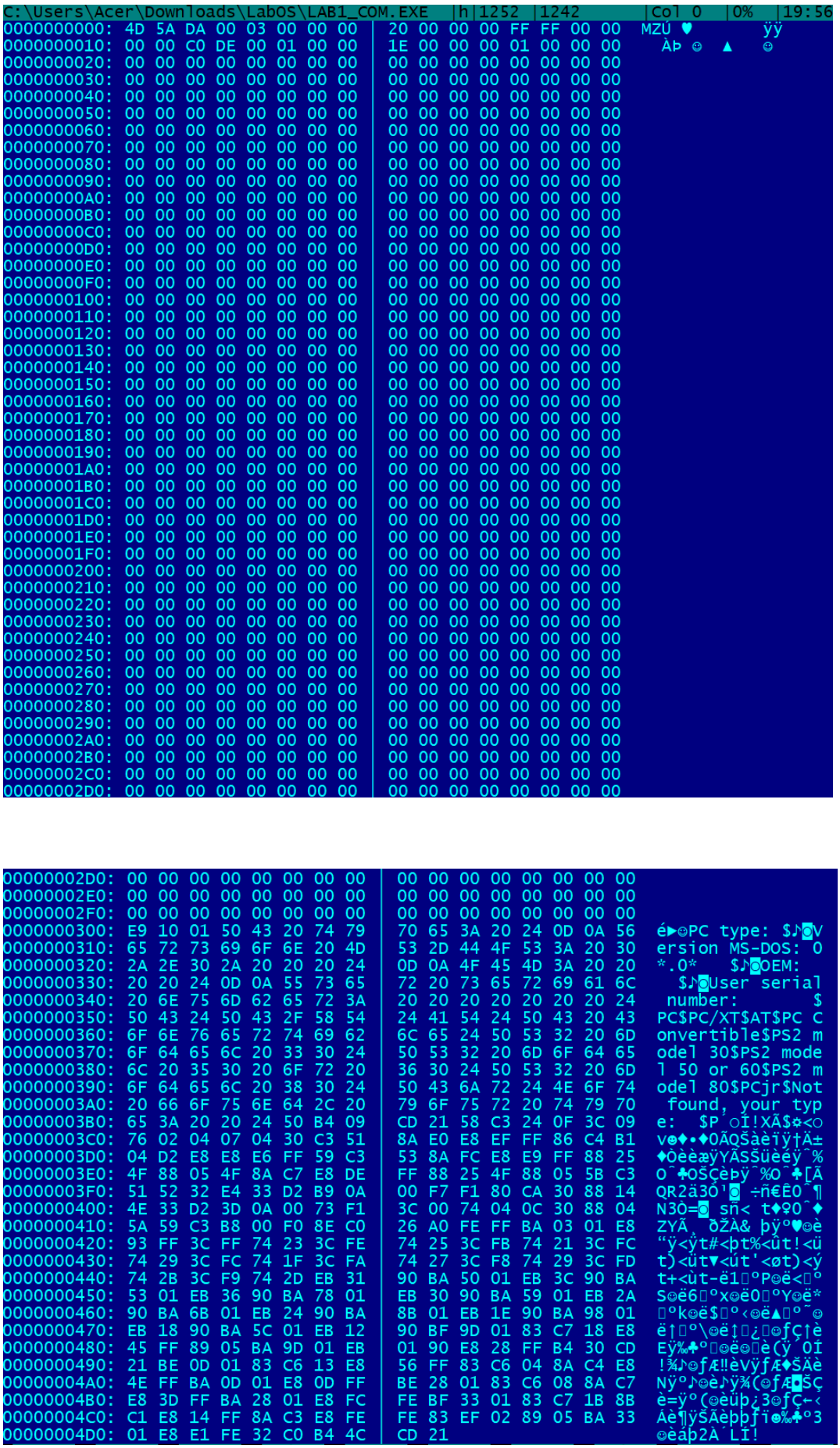

Рисунок 5 – Содержимое файла LAB1_COM.EXE («плохой» EXE)

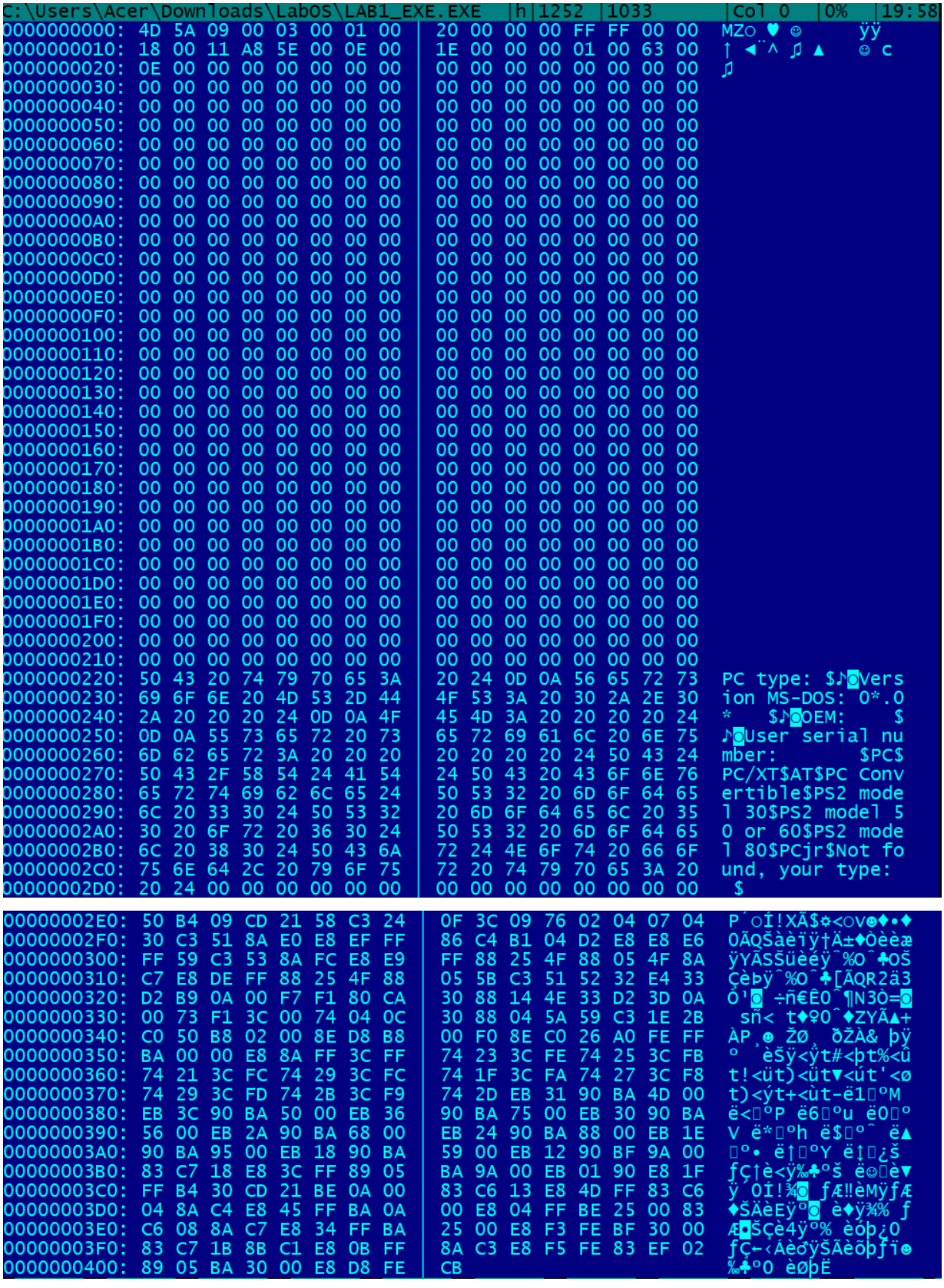

Рисунок 6 – Содержимое файла LAB1_EXE.EXE («хороший» EXE)

5) Был открыт отладчик TD.EXE и загружен .COM. Получены ответы на вопросы:

Загрузка COM модуля в основную память

1) Когда управление передается программе типа .COM, все сегментные регистры указывают на префикс. IP = 0100H. Код располагается с адреса 100H.
2) С адреса 0H располагается PSP.
3) Сегментные регистры имеют одинаковые значения 48DD и указывают на начало PSP.
4) Стек занимает 64кб (все доступное для него место). Регистр SS устанавливается на начало PSP, регистр SP на конец сегмента PSP и указывает на адрес FFFEH.

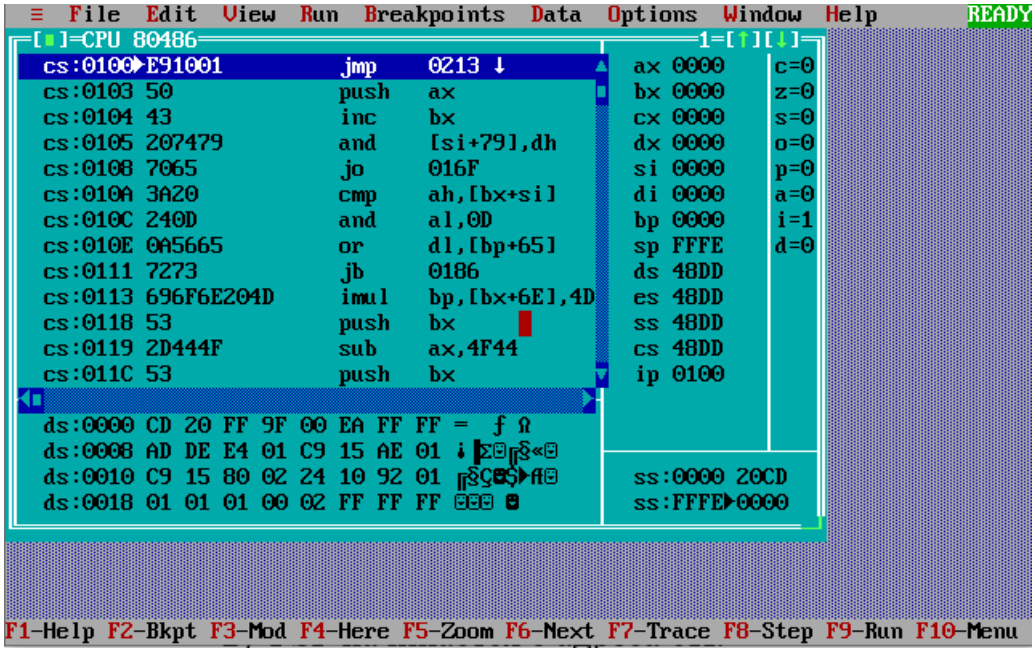

Рисунок 7 – Окно отладчика TD для COM файла

6) Был открыт отладчик TD.EXE и загружен хороший .EXE. Получены ответы на вопросы:

Загрузка «хорошего» EXE модуля в основную память

1) Сегментные регистры ES и DS указывают на PSP = 48DDH. CS указывает на начало сегмента кода, значение 48FBH, SP указывает на начало стека, значение 0018H.
2) Регистры ES и DS указывают на начало PSP.

3) Сегментом стека и выделенным размером под него.

4) Директива END сопоставляет точке входа метку, написанную после директивы.

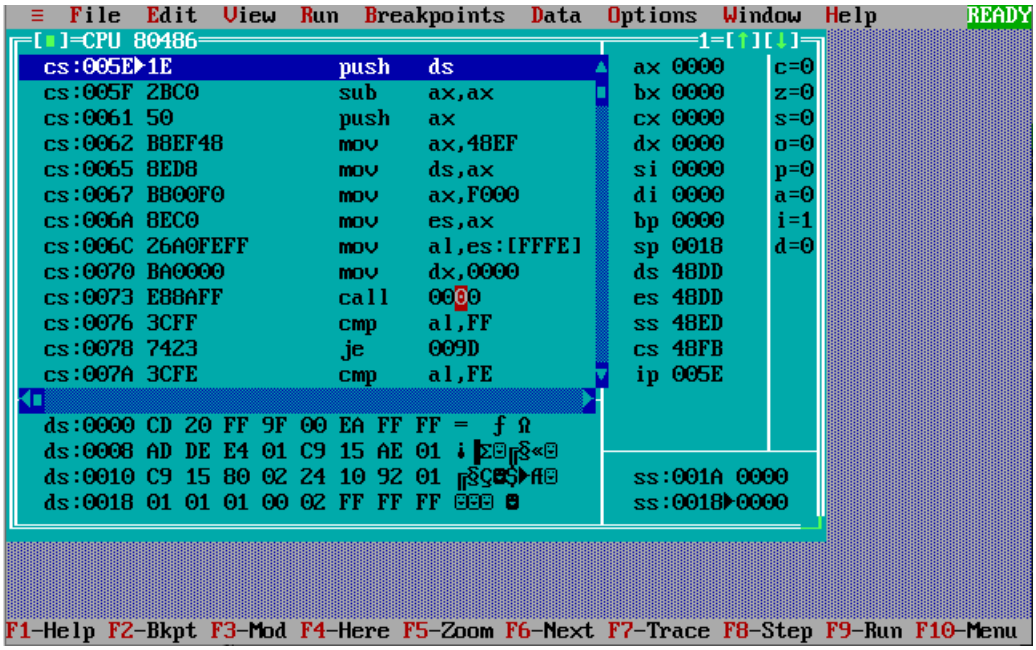

Рисунок 8 – Окно отладчика TD для хорошего EXE файла

Выводы.

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

ПРИЛОЖЕНИЕ А

lab1_exe.asm

```
STACK SEGMENT     STACK

            DW 12 DUP(?)

STACK       ENDS

DATA        SEGMENT

STR_TYPE    DB "PC type: $"
STR_VERSION DB 13, 10, "Version MS-DOS: 0*.0*   $"
STR_OEM     DB 13, 10, "OEM:    $"
STR_NUMBER  DB 13, 10, "User serial number:       $"
STR_PC      DB "PC$"
STR_PCXT    DB "PC/XT$"
STR_AT      DB "AT$"
STR_PCCon   DB "PC Convertible$"
STR_PS2m30  DB "PS2 model 30$"
STR_PS2m50m60     DB "PS2 model 50 or 60$"
STR_PS2m80  DB "PS2 model 80$"
STR_PCjr    DB "PCjr$"
STR_ERROR   DB "Not found, your type:  $"


DATA        ENDS

CODE        SEGMENT


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


PRINT       PROC  NEAR

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

PRINT       ENDP


TETRTOHEX   PROC  NEAR

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

            NEXT:
                  ADD   AL, 30H
                  RET

TETRTOHEX   ENDP


BYTETOHEX   PROC  NEAR

            PUSH  CX
            MOV   AH, AL
```

```
            CALL  TETRTOHEX
            XCHG  AL, AH
            MOV   CL, 4H
            SHR   AL, CL
            CALL  TETRTOHEX
            POP   CX
            RET

BYTETOHEX   ENDP


WRDTOHEX    PROC  NEAR

            PUSH  BX
            MOV   BH, AH
            CALL  BYTETOHEX
            MOV   [DI], AH
            DEC   DI
            MOV   [DI], AL
            DEC   DI
            MOV   AL, BH
            CALL  BYTETOHEX
            MOV   [DI], AH
            DEC   DI
            MOV   [DI], AL
            POP   BX
            RET

WRDTOHEX    ENDP


BYTETODEC   PROC  NEAR

            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

BYTETODEC   ENDP


MAIN        PROC  FAR

            PUSH  DS
```

```
SUB   AX, AX
PUSH  AX
MOV   AX, DATA
MOV   DS, AX

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

MOV   DX, OFFSET STR_TYPE
CALL PRINT

CMP AL, 0FFH
JZ    PC

CMP AL, 0FEH
JZ    PCXT

CMP AL, 0FBH
JZ    PCXT

CMP   AL, 0FCH
JZ    AT

CMP AL, 0FCH
JZ    PC2m50or60

CMP AL, 0FAH
JZ    PC2m30

CMP AL, 0F8H
JZ    PC2m80

CMP AL, 0FDH
JZ    PCjr

CMP AL, 0F9H
JZ    PCCon

JMP   ELS

PC:
      MOV   DX, OFFSET STR_PC
      JMP   PRINT_THIS

PCXT:
      MOV   DX, OFFSET STR_PCXT
      JMP   PRINT_THIS

PC2m50or60:
      MOV   DX, OFFSET STR_PS2m50m60
      JMP   PRINT_THIS

AT:
      MOV   DX, OFFSET STR_AT
      JMP   PRINT_THIS

PC2m30:
      MOV   DX, OFFSET STR_PS2m30
      JMP   PRINT_THIS

PC2m80:
      MOV   DX, OFFSET STR_PS2m80
```

```
            JMP   PRINT_THIS

        PCjr:
            MOV   DX, OFFSET STR_PCjr
            JMP   PRINT_THIS

        PCCon:
            MOV   DX, OFFSET STR_PCCon
            JMP   PRINT_THIS

        ELS:
            MOV   DI, OFFSET STR_ERROR
            ADD   DI, 18H
            CALL  BYTETOHEX
            MOV   [DI], AX
            MOV   DX, OFFSET STR_ERROR
            JMP   PRINT_THIS


        PRINT_THIS:
            CALL PRINT

            MOV   AH, 30H
            INT   21H

            MOV   SI, OFFSET STR_VERSION
            ADD   SI, 13H
            CALL  BYTETODEC
            ADD   SI, 4H
            MOV   AL, AH
            CALL  BYTETODEC
            MOV   DX, OFFSET STR_VERSION
            CALL  PRINT

            MOV   SI, OFFSET STR_OEM
            ADD   SI, 8H
            MOV   AL, BH
            CALL  BYTETODEC
            MOV   DX, OFFSET STR_OEM
            CALL  PRINT

            MOV   DI, OFFSET STR_NUMBER
            ADD   DI, 1BH
            MOV   AX, CX
                  CALL  WRDTOHEX
            MOV   AL, BL
            CALL  BYTETOHEX
            SUB   DI, 2H
            MOV   [DI], AX
            MOV   DX, OFFSET STR_NUMBER
            CALL  PRINT

        RET

MAIN    ENDP
CODE    ENDS
END     MAIN
```

lab1_com.asm

```
LAB1        SEGMENT

ASSUME      CS:LAB1, DS:LAB1, ES:NOTHING, SS:NOTHING

ORG         100H

START:      JMP   START_PROG

STR_TYPE    DB "PC type: $"
STR_VERSION DB 13, 10, "Version MS-DOS: 0*.0*   $"
STR_OEM     DB 13, 10, "OEM:    $"
STR_NUMBER  DB 13, 10, "User serial number:       $"
STR_PC      DB "PC$"
STR_PCXT    DB "PC/XT$"
STR_AT      DB "AT$"
STR_PCCon   DB "PC Convertible$"
STR_PS2m30  DB "PS2 model 30$"
STR_PS2m50m60     DB "PS2 model 50 or 60$"
STR_PS2m80  DB "PS2 model 80$"
STR_PCjr    DB "PCjr$"
STR_ERROR   DB "Not found, your type:  $"


PRINT       PROC  NEAR

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

PRINT       ENDP


TETRTOHEX   PROC  NEAR

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

            NEXT:
                  ADD   AL, 30H
                  RET

TETRTOHEX   ENDP


BYTETOHEX   PROC  NEAR

            PUSH  CX
            MOV   AH, AL
            CALL  TETRTOHEX
            XCHG  AL, AH
            MOV   CL, 4H
            SHR   AL, CL
            CALL  TETRTOHEX
            POP   CX
            RET

BYTETOHEX   ENDP
```

```
WRDTOHEX    PROC  NEAR

            PUSH  BX
            MOV   BH, AH
            CALL  BYTETOHEX
            MOV   [DI], AH
            DEC   DI
            MOV   [DI], AL
            DEC   DI
            MOV   AL, BH
            CALL  BYTETOHEX
            MOV   [DI], AH
            DEC   DI
            MOV   [DI], AL
            POP   BX
            RET

WRDTOHEX    ENDP


BYTETODEC   PROC  NEAR

            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

BYTETODEC   ENDP


            START_PROG:

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

                  MOV   DX, OFFSET STR_TYPE
                  CALL PRINT

                  CMP AL, 0FFH
                  JZ    PC

                  CMP AL, 0FEH
```

```
        JZ    PCXT

        CMP AL, 0FBH
        JZ    PCXT

        CMP   AL, 0FCH
        JZ    AT

        CMP AL, 0FCH
        JZ    PC2m50or60

        CMP AL, 0FAH
        JZ    PC2m30

        CMP AL, 0F8H
        JZ    PC2m80

        CMP AL, 0FDH
        JZ    PCjr

        CMP AL, 0F9H
        JZ    PCCon

        JMP   ELS

PC:
        MOV   DX, OFFSET STR_PC
        JMP   PRINT_THIS

PCXT:
        MOV   DX, OFFSET STR_PCXT
        JMP   PRINT_THIS

PC2m50or60:
        MOV   DX, OFFSET STR_PS2m50m60
        JMP   PRINT_THIS

AT:
        MOV   DX, OFFSET STR_AT
        JMP   PRINT_THIS

PC2m30:
        MOV   DX, OFFSET STR_PS2m30
        JMP   PRINT_THIS

PC2m80:
        MOV   DX, OFFSET STR_PS2m80
        JMP   PRINT_THIS

PCjr:
        MOV   DX, OFFSET STR_PCjr
        JMP   PRINT_THIS

PCCon:
        MOV   DX, OFFSET STR_PCCon
        JMP   PRINT_THIS

ELS:
        MOV   DI, OFFSET STR_ERROR
        ADD   DI, 18H
        CALL  BYTETOHEX
        MOV   [DI], AX
        MOV   DX, OFFSET STR_ERROR
```

```
                JMP   PRINT_THIS


            PRINT_THIS:
                CALL PRINT

                MOV   AH, 30H
                INT   21H

                MOV   SI, OFFSET STR_VERSION
                ADD   SI, 13H
                CALL  BYTETODEC
                ADD   SI, 4H
                MOV   AL, AH
                CALL  BYTETODEC
                MOV   DX, OFFSET STR_VERSION
                CALL  PRINT

                MOV   SI, OFFSET STR_OEM
                ADD   SI, 8H
                MOV   AL, BH
                CALL  BYTETODEC
                MOV   DX, OFFSET STR_OEM
                CALL  PRINT

                MOV   DI, OFFSET STR_NUMBER
                ADD   DI, 1BH
                MOV   AX, CX
                CALL  WRDTOHEX
                MOV   AL, BL
                CALL  BYTETOHEX
                SUB   DI, 2H
                MOV   [DI], AX
                MOV   DX, OFFSET STR_NUMBER
                CALL  PRINT

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

LAB1        ENDS
END         START
```