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 _____ Облизанов А.Д.

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.

Написание работы производилось на базе операционной системы Windows 10 в редакторе Visual Code. Сборка, отладка производились в эмуляторе DOS DOSBox 0.74-3, а также с помощью Far Manager.

Был написан текст исходного .COM модуля, который определяет тип PC и информацию о системе. Полученный модуль был отлажен, и в результате были получены «плохой» .EXE модуль и, с помощью программы EXE2BIN, «хороший» .COM модуль. Во время линковки было выведено предупреждение об отсутствии сегмента стека, представленное на рис. 1.

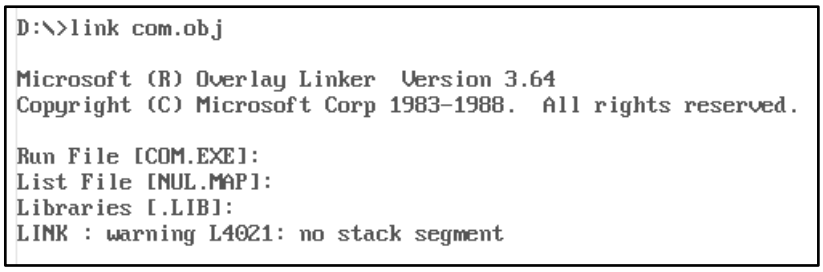

Рисунок 1 – Предупреждение во время линковки

Результат выполнения «плохого» .EXE модуля представлен на рис. 2.

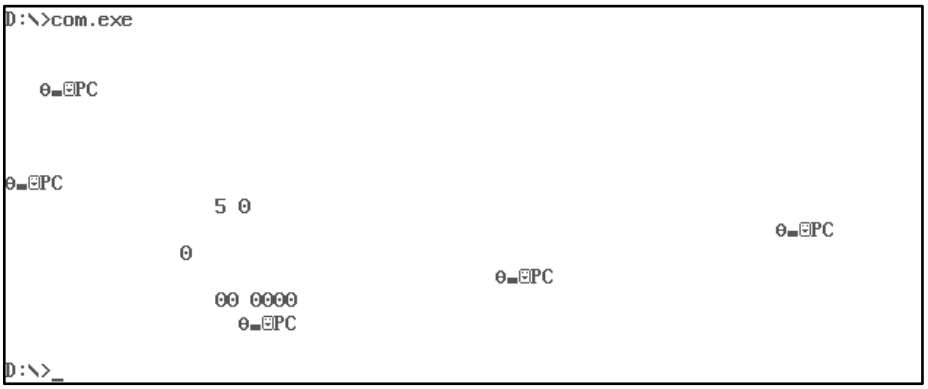

Рисунок 2 – Вывод «плохого» .EXE модуля

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

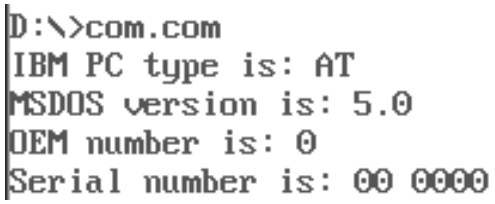

Рисунок 3 – Вывод «хорошего» .COM модуля

Был написан текст исходного .EXE модуля, который выполняет те же функции, что и .COM модуль. В результате постройки и отладки был получен «хороший» .EXE модуль. Результат его выполнения представлен на рис. 4.

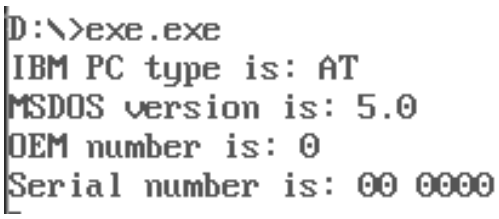

Рисунок 4 – Вывод «хорошего» .EXE модуля

Отличия исходных текстов COM и EXE программ

1. **Сколько сегментов должна содержать COM-программа?**

COM-программа должна содержать только один сегмент (модель памяти tiny)

2. **EXE-программа?**

EXE-программа может содержать несколько сегментов. При использовании модели памяти small, в программе должен содержаться один сегмент данных и один сегмент кода. При других моделях памяти (напр. large) есть возможность использования нескольких сегментов данных и (или) нескольких сегментов кода. Помимо этого, в программе может быть описан сегмент стека – стек в любом случае использует операционная система при обработке прерываний, и создаст его автоматически, если он не был описан.

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

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

При компиляции COM модуля без директивы ASSUME возникают ошибки – эта директива обязательна, чтобы сегментные регистры CS, DS указывали на адрес сегмента программы. Ошибки представлены на рис. 5.

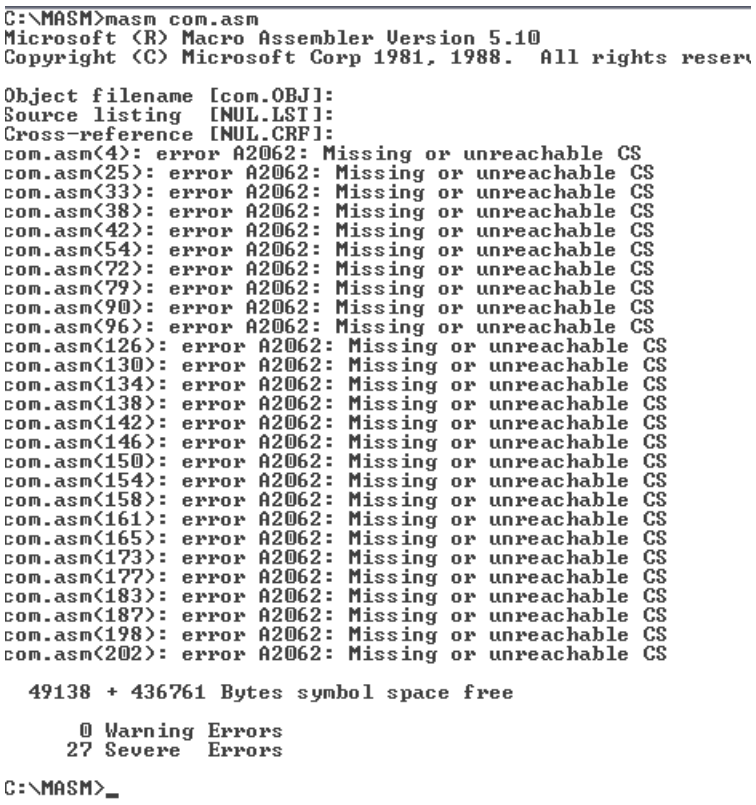

Рисунок 5 – Ошибки при компиляции при отсутствии ASSUME

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

В .COM файле отсутствует таблица настройки с информацией о типе адресов и их местоположении в коде. Поэтому нельзя использовать команды, связанные с адресом сегмента, так как адрес сегмента неизвестен вплоть до загрузки этого сегмента в память. Загрузчику необходима информация о местоположении в файле загрузочного модуля полей адресов.

Отличия форматов файлов COM и EXE модулей.

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

Вид файла COM в шестнадцатеричном формате представлен на рис. 6.

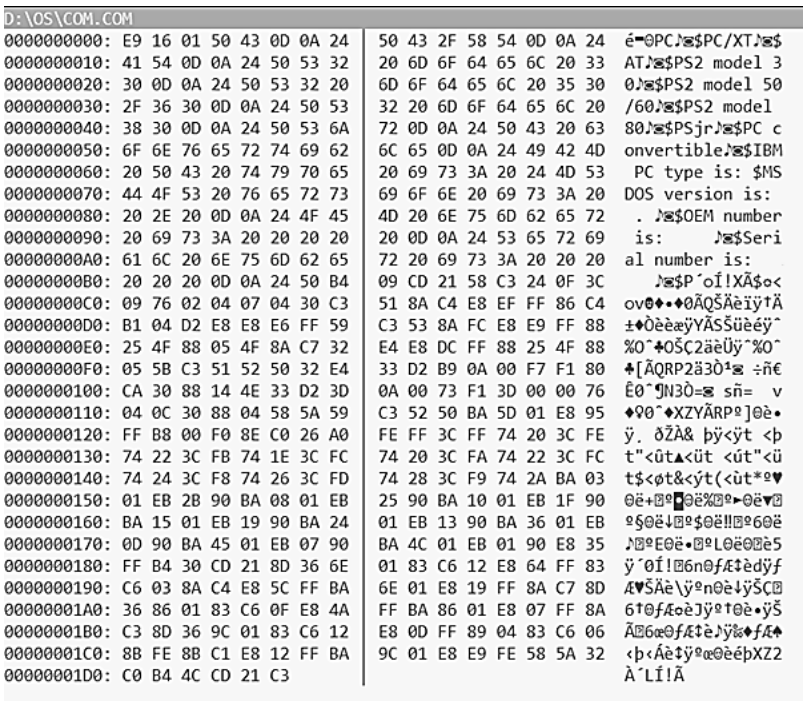

Рисунок 6 – Вид COM файла в шестнадцатеричном виде

COM-файл состоит из одного сегмента, а размер файла не превышает 64 КБ. Код располагается с нулевого адреса, что видно на рис. 6.

Данные заканчиваются на A0. На B0 с 4-го байта начинается сегмент кода. Между сегментами есть 6 байт 20h. Дальше идут байты, отвечающие за код. Например, можно найти коды команд, представленных на рис. 7.

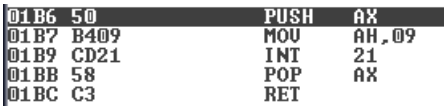

Рисунок 7 – Команды COM модуля

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

Вид «плохого» EXE файла в шестнадцатеричном формате представлен на рис. 8.

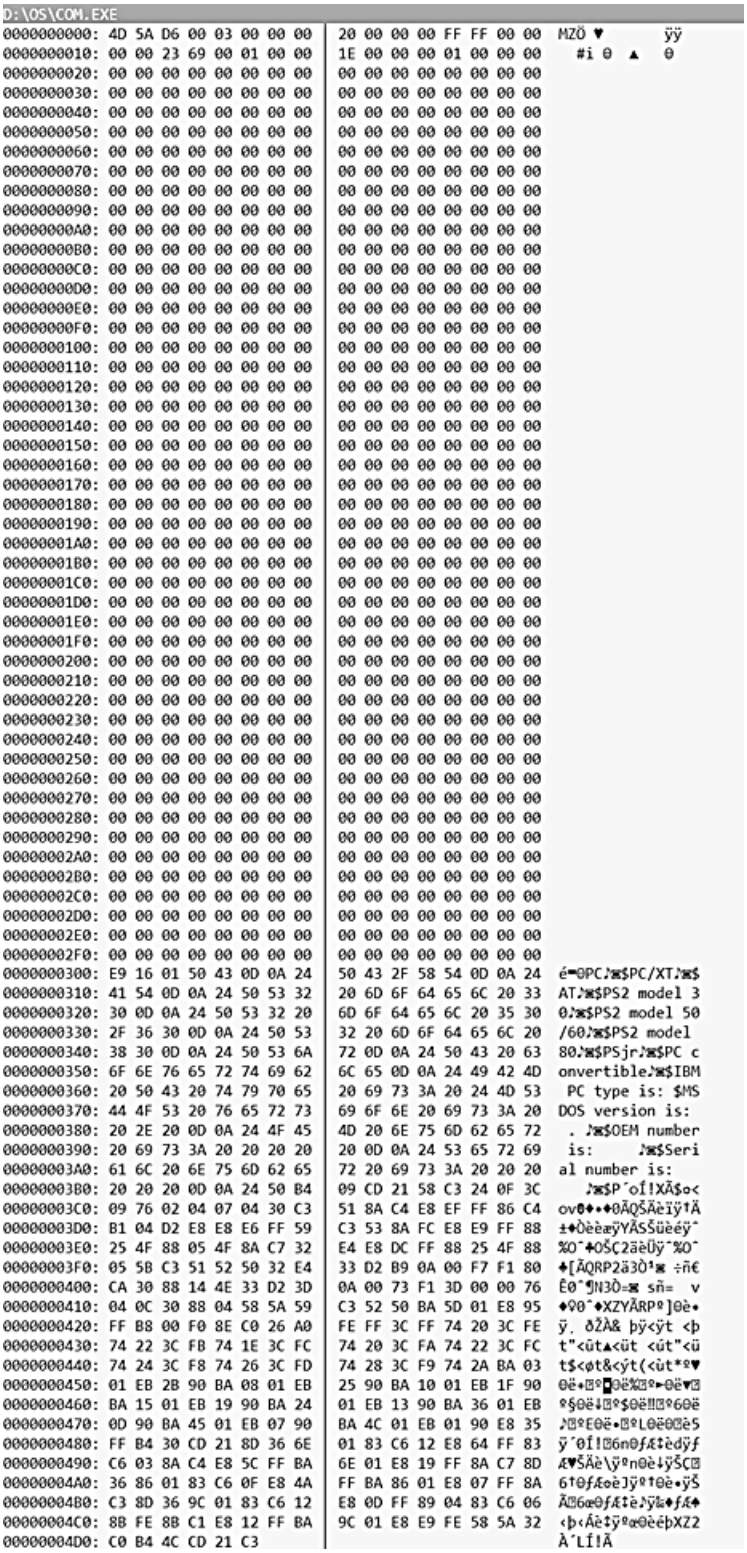

Рисунок 8 – Вид «плохого» EXE файла

В «плохом» EXE-файле 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 (что вызовет отличие в структуре «плохого» и «хорошего» EXE)

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

Вид файла «хорошего» EXE в шестнадцатеричном виде представлен на рис. 9. Ввиду большого числа пустых строк на рисунке представлена только часть файла.

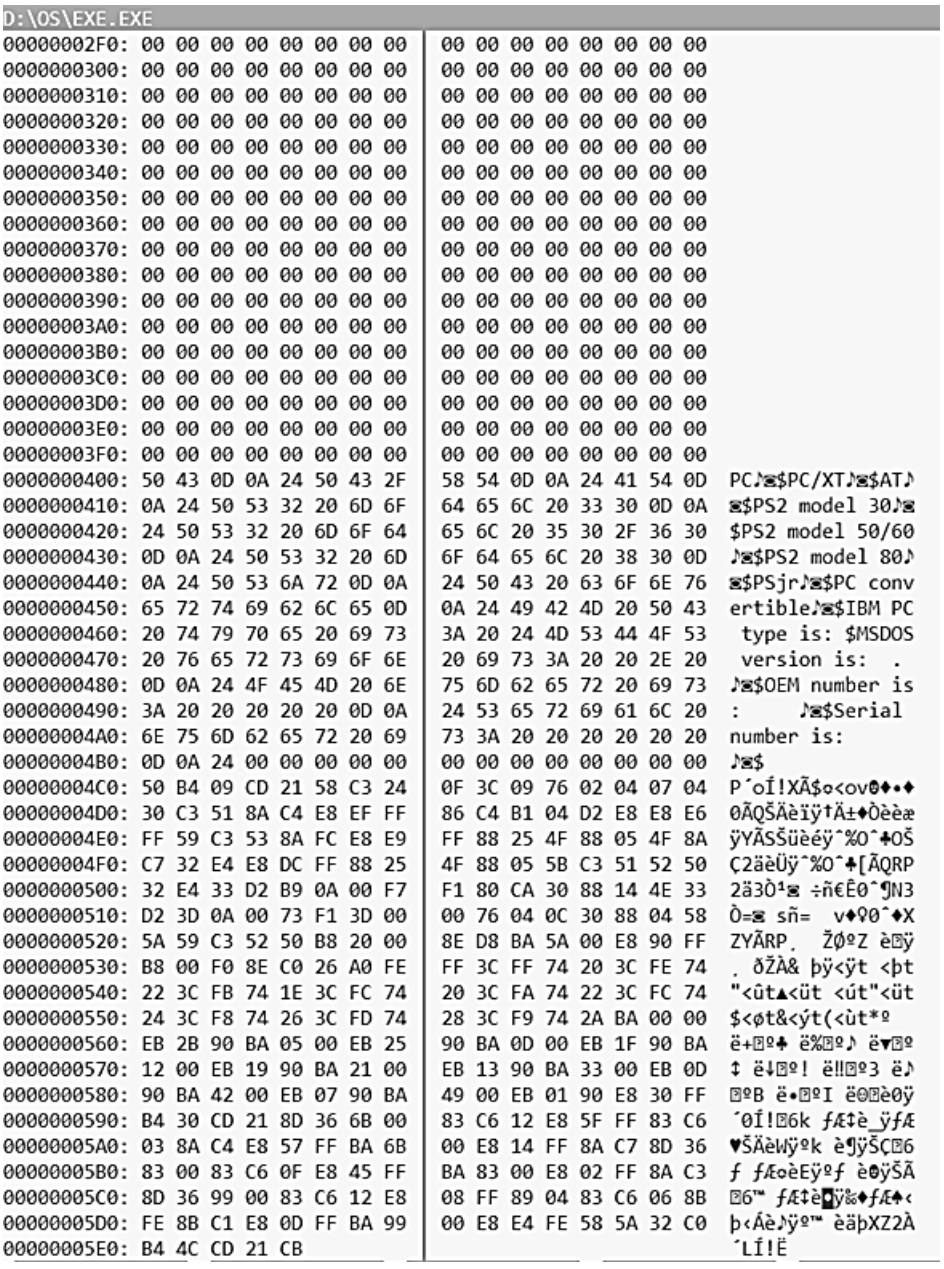

Рисунок 9 – Вид «хорошего» EXE файла

В «хорошем» EXE с нулевого адреса также располагается таблица настроек (управляющая информация для загрузчика). Также перед кодом располагается сегмент стека. Так, при размере стека 200h код располагается с адреса 400h. Если из исходного текста .EXE-программы убрать сегмент стека, то код будет располагаться с адреса 200h. Отличие от «плохого» EXE в том, что в «хорошем» не резервируется дополнительно 100h, которые в COM файле требовались для PSP, поэтому адреса начала кода отличаются на $100h + S$, где $S -$ размер стека.

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. 10.

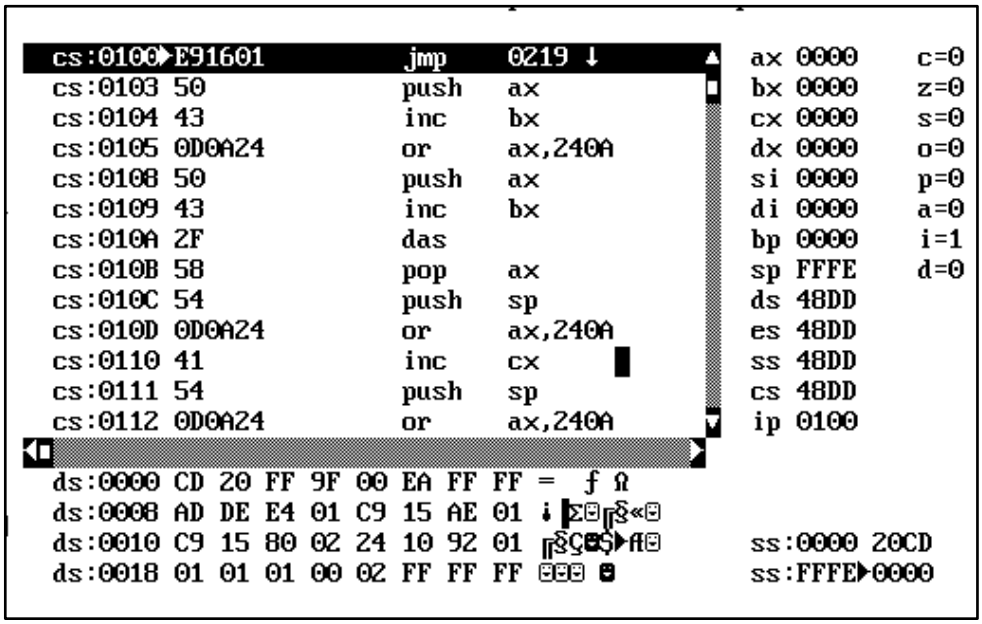

Рисунок 10 – Отладка файла .COM

После загрузки COM-программы в память сегментные регистры указывают на начало PSP. Код располагается с адреса 100h, регистр IP смещения следующей команды также имеет значение 100h.

2. **Что располагается с адреса 0?**

С нулевого адреса 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. На рис. 8 видно, что 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. 11.

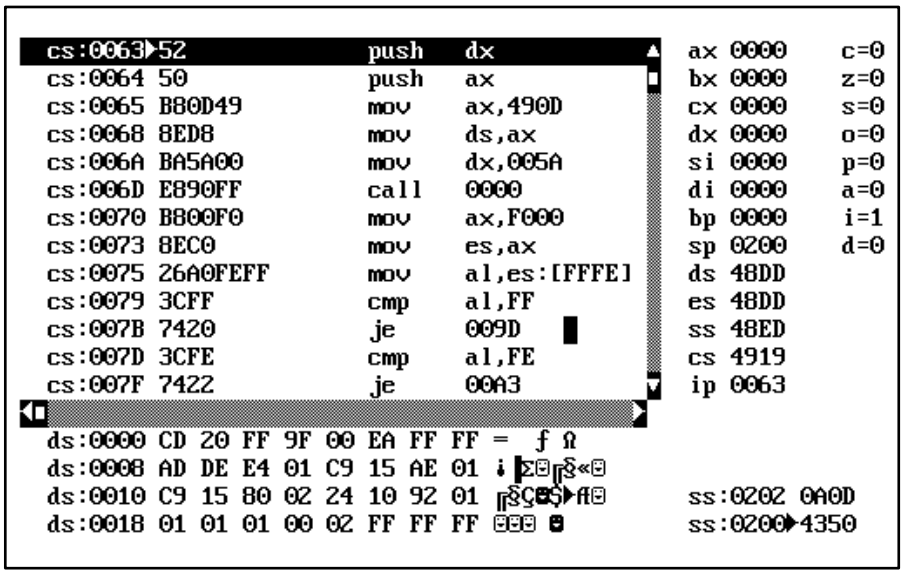

Рисунок 11 – Отладка «хорошего» EXE модуля

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 (48DDh), CS – на начало сегмента команд (4919h), а SS – на начало сегмента стека (48EDh). IP имеет ненулевое значение,

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 таким же образом, как в COM-модуле. Вид программы EXE модуля без объявленного стека после команды push в отладчике представлен на рис. 12.

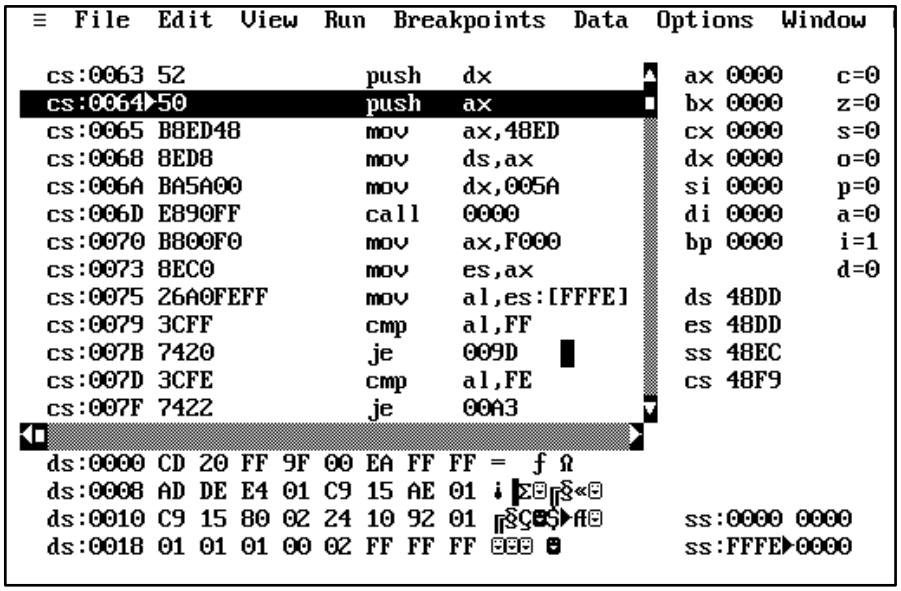

Рисунок 12 – Отладка EXE модуля без объявленного стека

4. Как определяется точка входа?

Смещение точки входа в программу загружается в указатель команд IP и определяется операндом директивы END, который называется точкой входа. Операндом является функция или метка, с которой необходимо начать программу.

Выводы.

В ходе выполнения лабораторной работы были изучены COM и EXE модули и их различия. Так же были получены два «хороших» модуля EXE.EXE и COM.COM, а также один «плохой» модуль COM.EXE.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. COM.ASM

```
TESTPC SEGMENT
            ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
            ORG 100H ; резервирование места для PSP
START:      JMP   BEGIN

;DATA SEGMENT
PC_TYPE                 db    'PC', 0DH, 0AH,'$'
PC_XT_TYPE        db    'PC/XT', 0dh, 0ah,'$'
AT_TYPE                 db    'AT', 0dh, 0ah, '$'
PS2_30_TYPE       db    'PS2 model 30', 0dh, 0ah, '$'
PS2_5060_TYPE     db    'PS2 model 50/60', 0dh, 0ah, '$'
PS2_80_TYPE       db    'PS2 model 80', 0dh, 0ah, '$'
PCjr_TYPE         db    'PSjr', 0dh, 0ah, '$'
PC_CONVERTIBLE    db    'PC convertible', 0dh, 0ah, '$'

IBM_PC_NAME       db    'IBM PC type is: ', '$'
OS_NAME                 db    'MSDOS version is:  . ', 0dh, 0ah, '$'
OEM_NAME          db    'OEM number is:     ', 0dh, 0ah, '$'
SERIAL_NAME       db    'Serial number is:      ', 0dh, 0ah, '$'
;DATA ENDS

;CODE SEGMENT


PRINT_STRING      PROC  near
            push  AX
            mov   AH, 09h
            int         21h
            pop   AX
            ret
PRINT_STRING      ENDP
;-------------------------------------------------------------------------------
--
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
```

```
            push  cx
            mov         al, ah
            call  TETR_TO_HEX
            xchg  al, ah
            mov         cl, 4
            shr         al, cl
            call  TETR_TO_HEX
            pop         cx
            ret
BYTE_TO_HEX       ENDP
;-------------------------------------------------------------------------------
--
WRD_TO_HEX        PROC  near
            push  bx
            mov         bh, ah
            call  BYTE_TO_HEX
            mov         [di], ah
            dec         di
            mov         [di], al
            dec         di
            mov         al, bh
            xor         ah, ah
            call  BYTE_TO_HEX
            mov         [di], ah
            dec         di
            mov         [di], al
            pop         bx
            ret
WRD_TO_HEX        ENDP
;-------------------------------------------------------------------------------
--
BYTE_TO_DEC       PROC  near
            push  cx
            push  dx
            push  ax
            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         ax, 00h
            jbe         end_l
```

```
            or          al, 30h
            mov         [si], al
end_l:      pop         ax
            pop         dx
            pop         cx
            ret
BYTE_TO_DEC       ENDP

BEGIN:
;PC INFO OUT
            push DX
            push AX

            mov DX, offset IBM_PC_NAME
            call PRINT_STRING

            mov AX, 0F000H
            mov ES, AX
            mov AL, ES:[0FFFEH]
            cmp AL, 0FFh
            je PC_WRITE
            cmp AL, 0FEh
            je PC_XT_WRITE
            cmp AL, 0FBh
            je PC_XT_WRITE
            cmp AL, 0FCh
            je AT_WRITE
            cmp AL, 0FAh
            je PS2_30_WRITE
            cmp AL, 0FCh
            je PS2_5060_WRITE
            cmp AL, 0F8h
            je PS2_80_WRITE
            cmp AL, 0FDh
            je PCjr_WRITE
            cmp AL, 0F9H
            je PC_CONVERTIBLE_WRITE

PC_WRITE:
            mov DX, offset PC_TYPE
            jmp TYPE_WRITE

PC_XT_WRITE:
            mov DX, offset PC_XT_TYPE
            jmp TYPE_WRITE

AT_WRITE:
```

```
            mov DX, offset AT_TYPE
            jmp TYPE_WRITE

PS2_30_WRITE:
            mov DX, offset PS2_30_TYPE
            jmp TYPE_WRITE

PS2_5060_WRITE:
            mov DX, offset PS2_5060_TYPE
            jmp TYPE_WRITE

PS2_80_WRITE:
            mov DX, offset PS2_80_TYPE
            jmp TYPE_WRITE

PCjr_WRITE:
            mov DX, offset PCjr_TYPE
            jmp TYPE_WRITE

PC_CONVERTIBLE_WRITE:
            mov DX, offset PC_CONVERTIBLE
            jmp TYPE_WRITE

TYPE_WRITE:
            call PRINT_STRING

OS_INFO_GET:
            mov AH, 30h
            int 21h

OS_VERSION_SET:
            lea         SI, OS_NAME
            add         SI, 18
            call  BYTE_TO_DEC
            add         SI, 3
            mov   AL, AH
            call        BYTE_TO_DEC

OS_VERSION_WRITE:
            mov DX, offset OS_NAME
            call PRINT_STRING

OEM_SET:
            mov   AL, BH
            lea   SI, OEM_NAME
            add   SI, 15
            call  BYTE_TO_DEC
```

```
OEM_WRITE:
            mov   DX, offset OEM_NAME
            call  PRINT_STRING

SERIAL_SET:
            mov         AL, BL
            lea         SI, SERIAL_NAME
            add         SI, 18
            call  BYTE_TO_HEX
            mov         [SI], AX
            add         SI, 6
            mov         DI, SI
            mov   AX, CX
            call  WRD_TO_HEX

SERIAL_WRITE:
            mov   DX, offset SERIAL_NAME
            call  PRINT_STRING

ENDING:
            pop         AX
            pop   DX

            xor         AL, AL
            mov   AH, 4ch
            int         21h
            ret
TESTPC      ENDS
            END   START
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. EXE.ASM

```
AStack      SEGMENT  STACK
                  DW 100h DUP(?)
AStack      ENDS

DATA SEGMENT
      PC_TYPE                 db    'PC', 0DH, 0AH,'$'
      PC_XT_TYPE        db    'PC/XT', 0dh, 0ah,'$'
      AT_TYPE                 db    'AT', 0dh, 0ah, '$'
      PS2_30_TYPE       db    'PS2 model 30', 0dh, 0ah, '$'
      PS2_5060_TYPE     db    'PS2 model 50/60', 0dh, 0ah, '$'
      PS2_80_TYPE       db    'PS2 model 80', 0dh, 0ah, '$'
      PCjr_TYPE         db    'PSjr', 0dh, 0ah, '$'
      PC_CONVERTIBLE    db    'PC convertible', 0dh, 0ah, '$'

      IBM_PC_NAME       db    'IBM PC type is: ', '$'
      OS_NAME                 db    'MSDOS version is:  . ', 0dh, 0ah, '$'
      OEM_NAME          db    'OEM number is:     ', 0dh, 0ah, '$'
      SERIAL_NAME       db    'Serial number is:      ', 0dh, 0ah, '$'
DATA ENDS

CODE SEGMENT
      ASSUME      CS:CODE, DS:DATA, SS:AStack

PRINT_STRING PROC near
            push  AX
            mov   AH, 09h
            int         21h
            pop   AX
            ret
PRINT_STRING      ENDP
;-------------------------------------------------------------------------------
--
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
            push  cx
```

```
            mov         al, ah
            call  TETR_TO_HEX
            xchg  al, ah
            mov         cl, 4
            shr         al, cl
            call  TETR_TO_HEX
            pop         cx
            ret
BYTE_TO_HEX       ENDP
;-------------------------------------------------------------------------------
--
WRD_TO_HEX        PROC  near
            push  bx
            mov         bh, ah
            call  BYTE_TO_HEX
            mov         [di], ah
            dec         di
            mov         [di], al
            dec         di
            mov         al, bh
            xor         ah, ah
            call  BYTE_TO_HEX
            mov         [di], ah
            dec         di
            mov         [di], al
            pop         bx
            ret
WRD_TO_HEX        ENDP
;-------------------------------------------------------------------------------
--
BYTE_TO_DEC       PROC  near
            push  cx
            push  dx
            push  ax
            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         ax, 00h
            jbe         end_l
            or          al, 30h
```

```
            mov         [si], al
end_l:      pop         ax
            pop         dx
            pop         cx
            ret
BYTE_TO_DEC       ENDP

Main  PROC  far
;PC INFO OUT
            push DX
            push AX

            mov ax, DATA
            mov   ds, ax

            mov DX, offset IBM_PC_NAME
            call PRINT_STRING

            mov AX, 0F000H
            mov ES, AX
            mov AL, ES:[0FFFEH]
            cmp AL, 0FFh
            je PC_WRITE
            cmp AL, 0FEh
            je PC_XT_WRITE
            cmp AL, 0FBh
            je PC_XT_WRITE
            cmp AL, 0FCh
            je AT_WRITE
            cmp AL, 0FAh
            je PS2_30_WRITE
            cmp AL, 0FCh
            je PS2_5060_WRITE
            cmp AL, 0F8h
            je PS2_80_WRITE
            cmp AL, 0FDh
            je PCjr_WRITE
            cmp AL, 0F9H
            je PC_CONVERTIBLE_WRITE

PC_WRITE:
            mov DX, offset PC_TYPE
            jmp TYPE_WRITE

PC_XT_WRITE:
            mov DX, offset PC_XT_TYPE
            jmp TYPE_WRITE
```

```
AT_WRITE:
            mov DX, offset AT_TYPE
            jmp TYPE_WRITE

PS2_30_WRITE:
            mov DX, offset PS2_30_TYPE
            jmp TYPE_WRITE

PS2_5060_WRITE:
            mov DX, offset PS2_5060_TYPE
            jmp TYPE_WRITE

PS2_80_WRITE:
            mov DX, offset PS2_80_TYPE
            jmp TYPE_WRITE

PCjr_WRITE:
            mov DX, offset PCjr_TYPE
            jmp TYPE_WRITE

PC_CONVERTIBLE_WRITE:
            mov DX, offset PC_CONVERTIBLE
            jmp TYPE_WRITE

TYPE_WRITE:
            call PRINT_STRING

OS_INFO_GET:
            mov AH, 30h
            int 21h

OS_VERSION_SET:
            lea         SI, OS_NAME
            add         SI, 18
            call  BYTE_TO_DEC
            add         SI, 3
            mov   AL, AH
            call        BYTE_TO_DEC

OS_VERSION_WRITE:
            mov DX, offset OS_NAME
            call PRINT_STRING

OEM_SET:
            mov   AL, BH
            lea   SI, OEM_NAME
```

```
            add   SI, 15
            call  BYTE_TO_DEC

OEM_WRITE:
            mov   DX, offset OEM_NAME
            call  PRINT_STRING

SERIAL_SET:
            mov         AL, BL
            lea         SI, SERIAL_NAME
            add         SI, 18
            call  BYTE_TO_HEX
            mov         [SI], AX
            add         SI, 6
            mov         DI, SI
            mov   AX, CX
            call  WRD_TO_HEX

SERIAL_WRITE:
            mov   DX, offset SERIAL_NAME
            call  PRINT_STRING

ENDING:
            pop         AX
            pop   DX

            xor         AL, AL
            mov   AH, 4ch
            int         21h
            ret

Main  ENDP
CODE  ENDS
            END   Main
```