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 _____ Лисок М.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

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

Задание.

Тип IBM PC хранится в байте по адресу 0F000:0FFFE, в предпоследнем байте ROM BIOS. Соответствие кода и типа в таблице 1.

Таблица 1 – Соответствие типа и кода 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:

```
MOV AH,30h
INT 21h
```

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.

Постановка задачи.

Требуется реализовать текст исходного .COM модуля, который определяет тип PC и версию системы. Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип PC и выводить символьную строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код переводиться в символьную строку, содержащую запись шестнадцатеричного числа и выводиться на экран в виде соответствующего сообщения. Затем определяется версия системы. Ассемблерная программа должна по значениям регистров AL и AH формировать текстовую строку в формате xx.yy, где xx - номер основной версии, а yy - номер модификации в десятичной системе счисления, формировать строки с серийным номером OEM (Original Equipment Manufacturer) и серийным номером пользователя. Полученные строки выводятся на экран.

Далее необходимо отладить полученный исходный модуль и получить «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля.

Затем нужно написать текст «хорошего» .EXE модуля, который выполняет те же функции, что и модуль .COM, далее его построить, отладить и сравнить исходные тексты для .COM и .EXE модулей.

Выполнение работы.

Выполнение работы производилось на базе операционной системы Mac OS в редакторе Sublime. Сборка и отладка модулей производились с помощью компилятора MASM и отладчика AFD. Также в работе был использован компилятор TASM.

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 «хороший» .COM модуль (с помощью пакета Borland Tasm).

Пример сборки «хорошего» .COM модуля показан на рис. 1.

Рисунок 1 – Полученный .COM модуль

Во время линковки «плохого» .EXE модуля было выведено предупреждение об отсутствии сегмента стека, представленное на рис. 2.

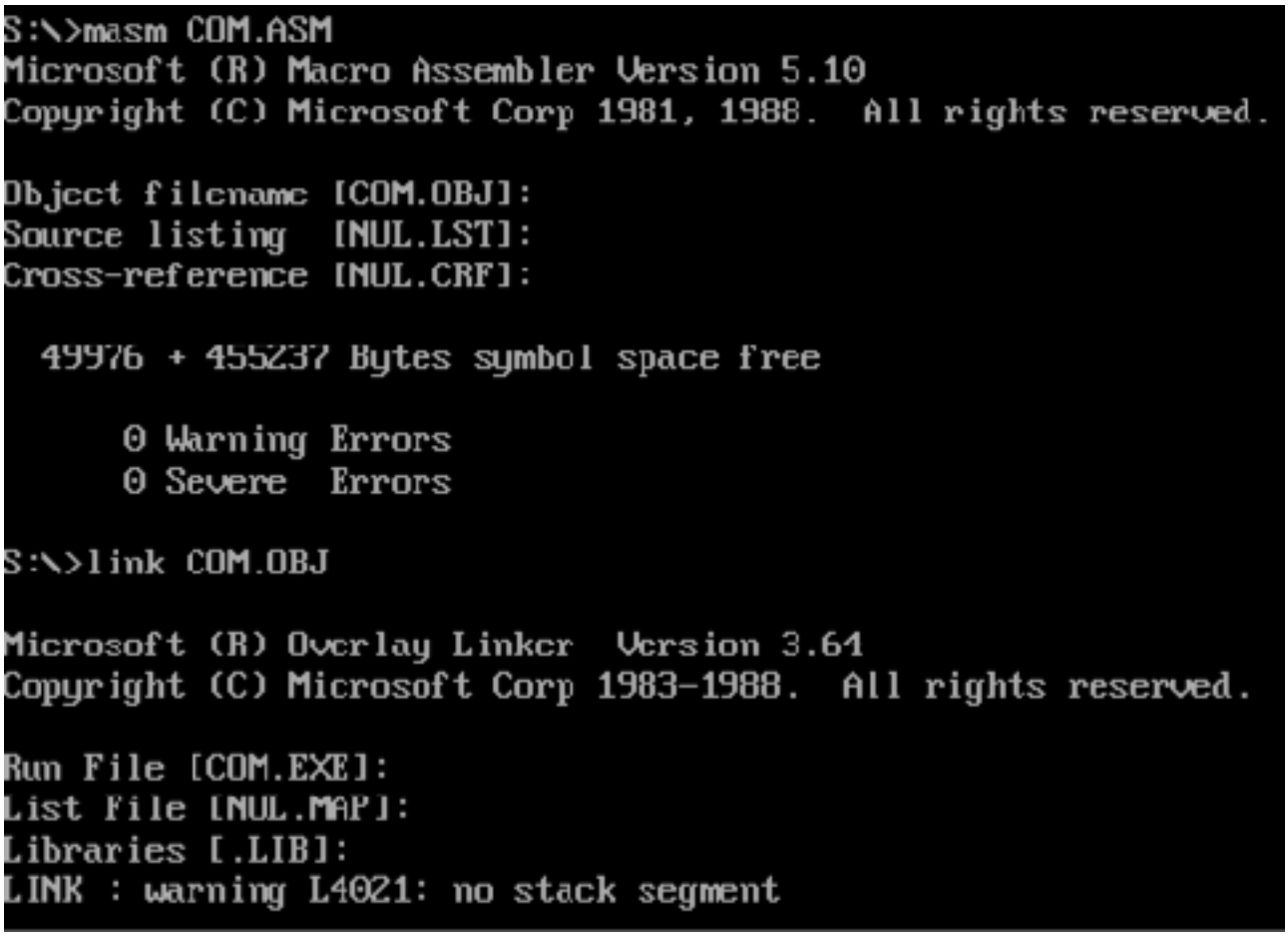


Рисунок 2 – Предупреждение во время линковки

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

Рисунок 3 – Вывод «плохого» .EXE модуля

Результат выполнения «хорошего» .COM модуля представлен на рис. 4.

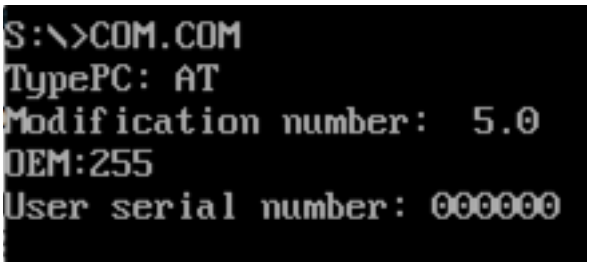


Рисунок 4 – Вывод «хорошего» .COM модуля

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. 5.

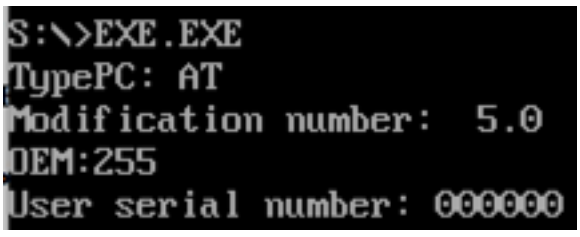


Рисунок 5 – Вывод «хорошего» .EXE модуля

Отличия исходных текстов COM и EXE программ.

1) Сколько сегментов должна содержать COM программа?

.COM - программы содержат только один сегмент. Модель памяти tiny - код, данные и стек объединены в один физический сегмент, максимальный

размер которого не мог превышать 64 Кбайта без 256 байтов (последние требуются для создания префикса программного сегмента (PSP)).

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 в разных физических сегментах и каждый из них не может превосходить 64 Кбайта. При других моделях памяти (например large) есть возможность использования нескольких сегментов данных и (или) нескольких сегментов кода.

Помимо этого, в программе должен быть описан сегмент стека (до 64 Кбайт, содержит адреса возврата как для программы (для возврата в операционную систему), так и для вызовов подпрограмм (для возврата в главную программу), а также используется для передачи параметров в процедуры). Использует операционная система при обработке прерываний. Регистр сегмента стека (SS) адресует данный сегмент. Адрес текущей вершины стека задается регистрами SS:ESP (для 32-х разрядных регистров) и SS:RSP (для 64-х разрядных регистров).

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.

Также обязательной для указания является директива ASSUME. Все сегменты сами по себе равноправны, так как директивы SEGMENT и ENDS не содержат информации о функциональном назначении сегментов. Для того чтобы использовать их как сегменты кода, данных или стека, необходимо предварительно сообщить транслятору об этом, для чего используют специальную директиву ASSUME. Эта директива сообщает транслятору о том, какой сегмент к какому сегментному регистру привязан. В свою очередь, это позволит транслятору корректно связывать символические имена,

определенные в сегментах. Привязка сегментов к сегментным регистрам осуществляется с помощью операндов этой директивы, в которых имя_сегмента должно быть именем сегмента, определенным в исходном тексте программы директивой SEGMENT или ключевым словом nothing. Если данную директиву не прописать, получим ошибки определения программой сегмента кода, как на рис. 6.

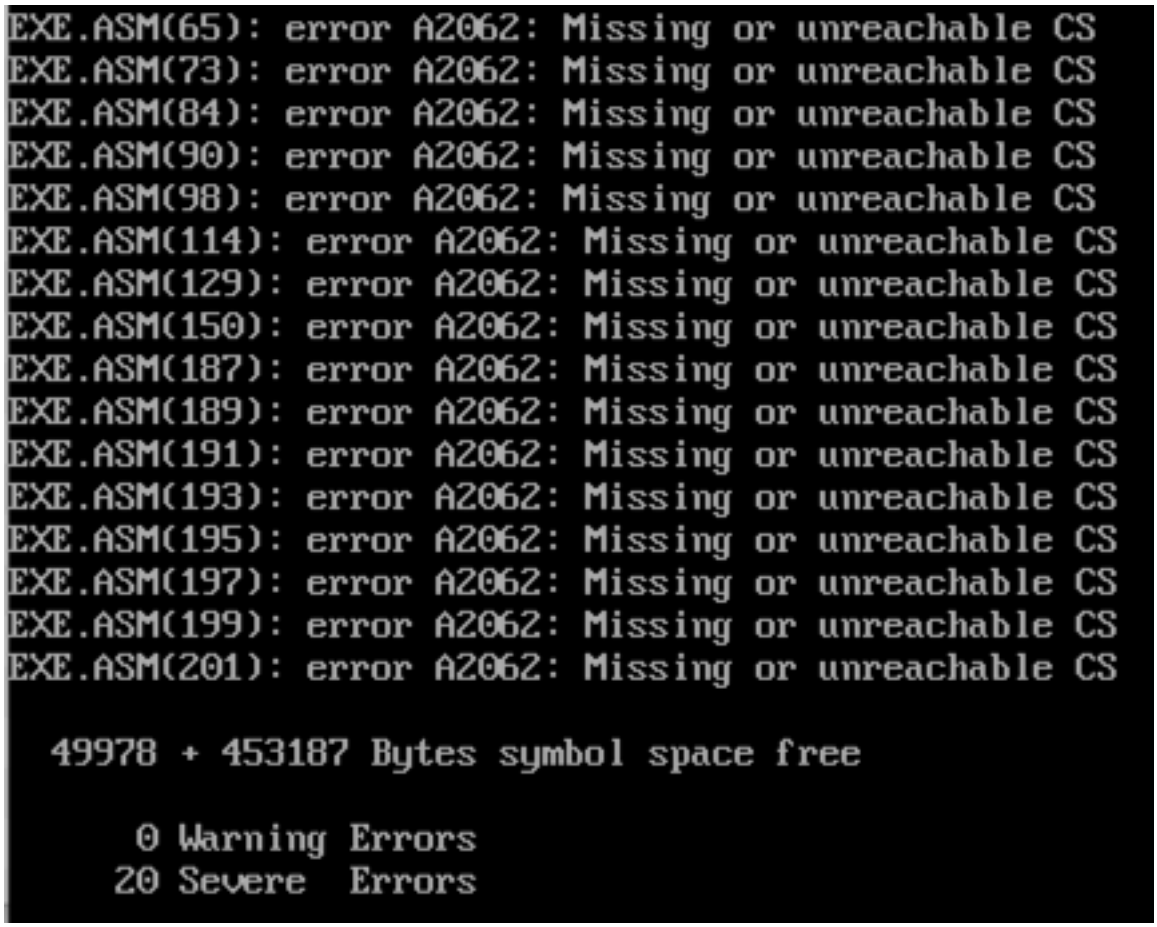


Рисунок 6 – Ошибки при отсутствии директивны ASSUME

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. 7.

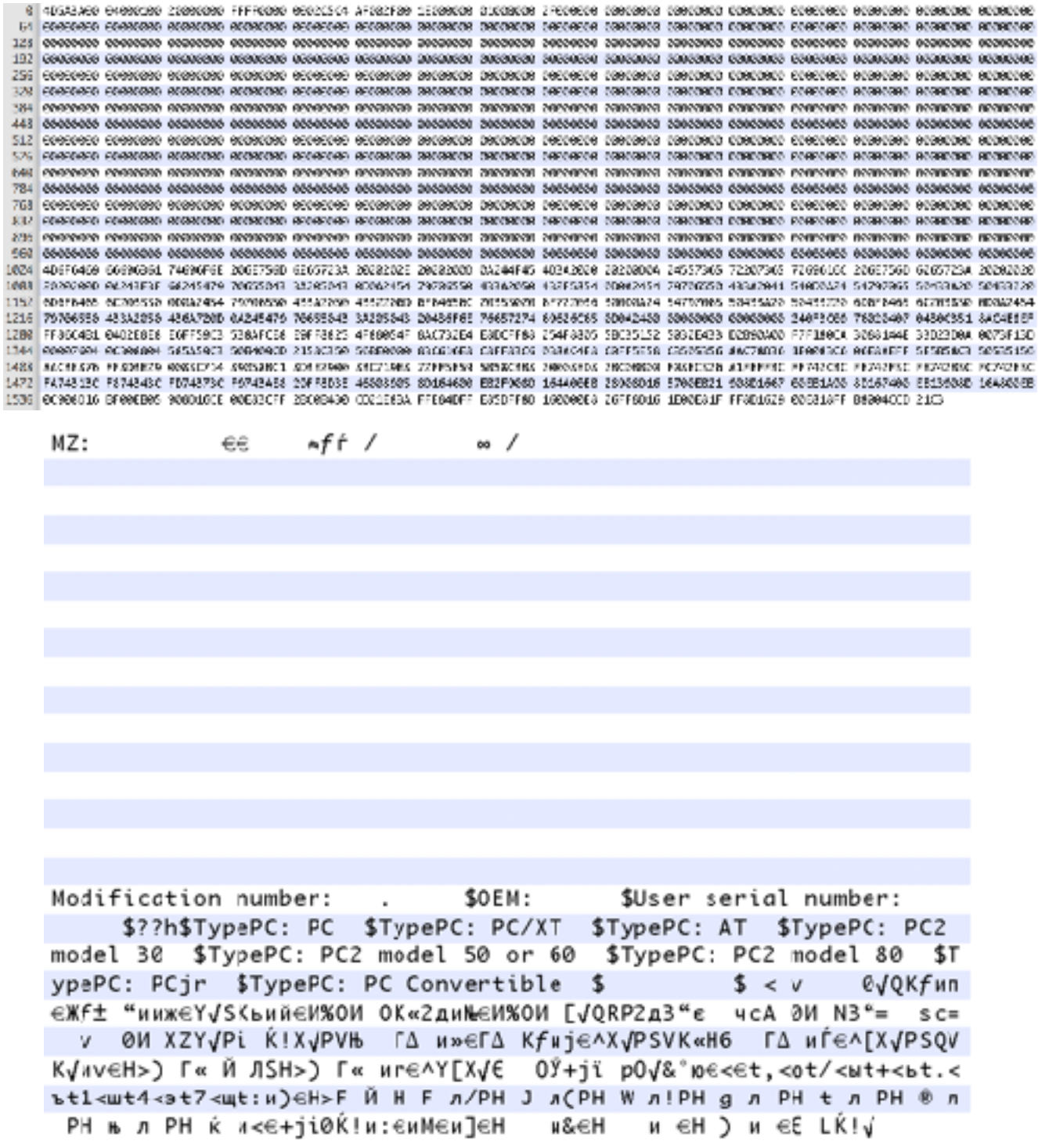


Рисунок 7 – Вид COM файла в шестнадцатеричном виде

COM-файл состоит из одного сегмента, а размер файла не превышает 64 КБ. Код располагается с нулевого адреса, что видно на рис. 7. DATA Segment

заканчиваются на адресе A0. С B0 с 4-го байта начинается CODE Segment. Между сегментами существует промежуток в 6 байт (20h) – связано с GPR и с 6-ти сегментным строением сегмента.

Далее можно отследить команды, отвечающие за исходный код программы. Данный факт изображён на рис. 8 и рис. 9 – соответствия выделены красным и синим цветами.

```
244F454D 3A202020 200D0A24
```

Рисунок 8 – Фрагмент 16-ого представления COM файла

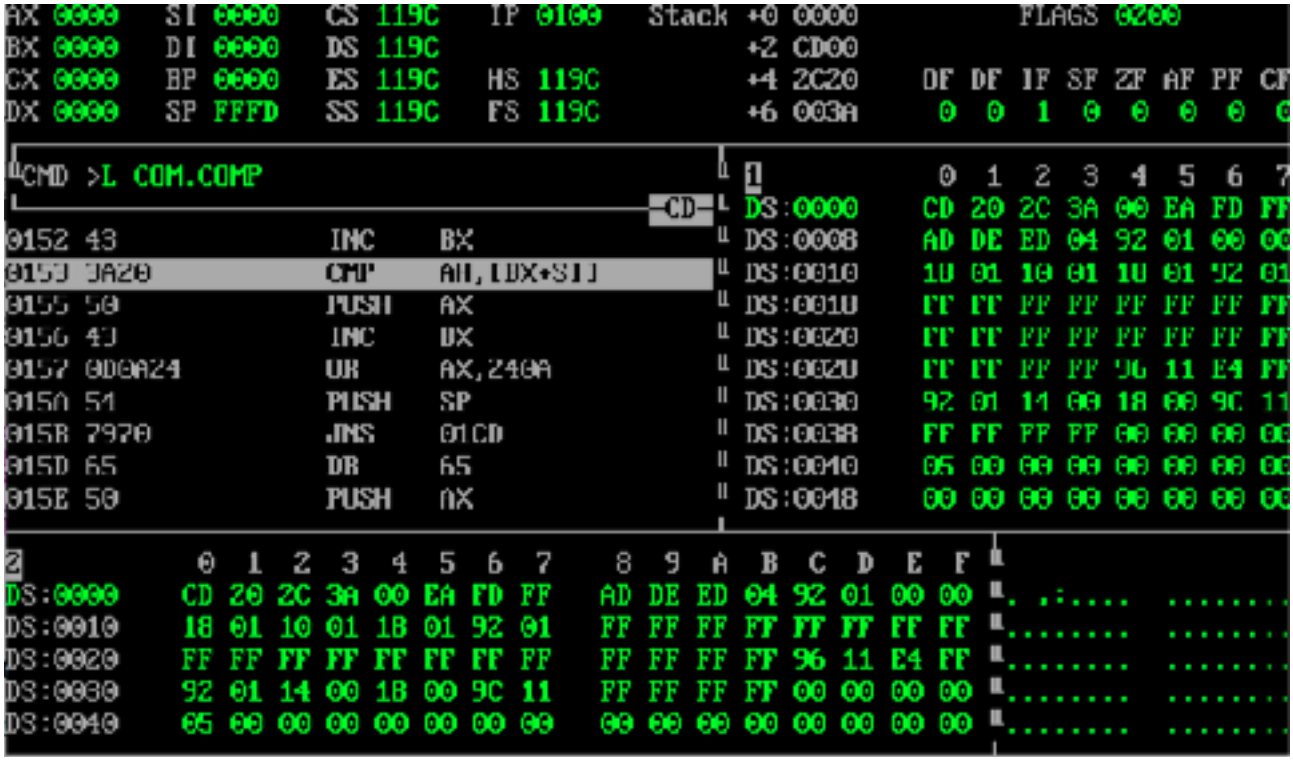
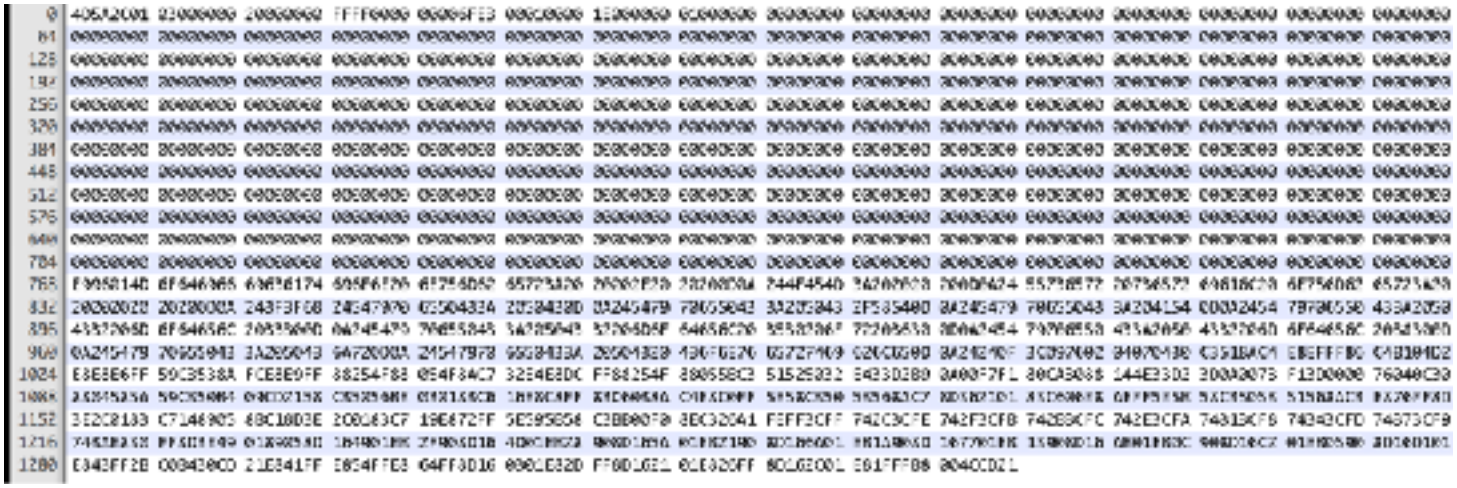


Рисунок 9 – Адреса команд, найденные в отладчике

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. 10.



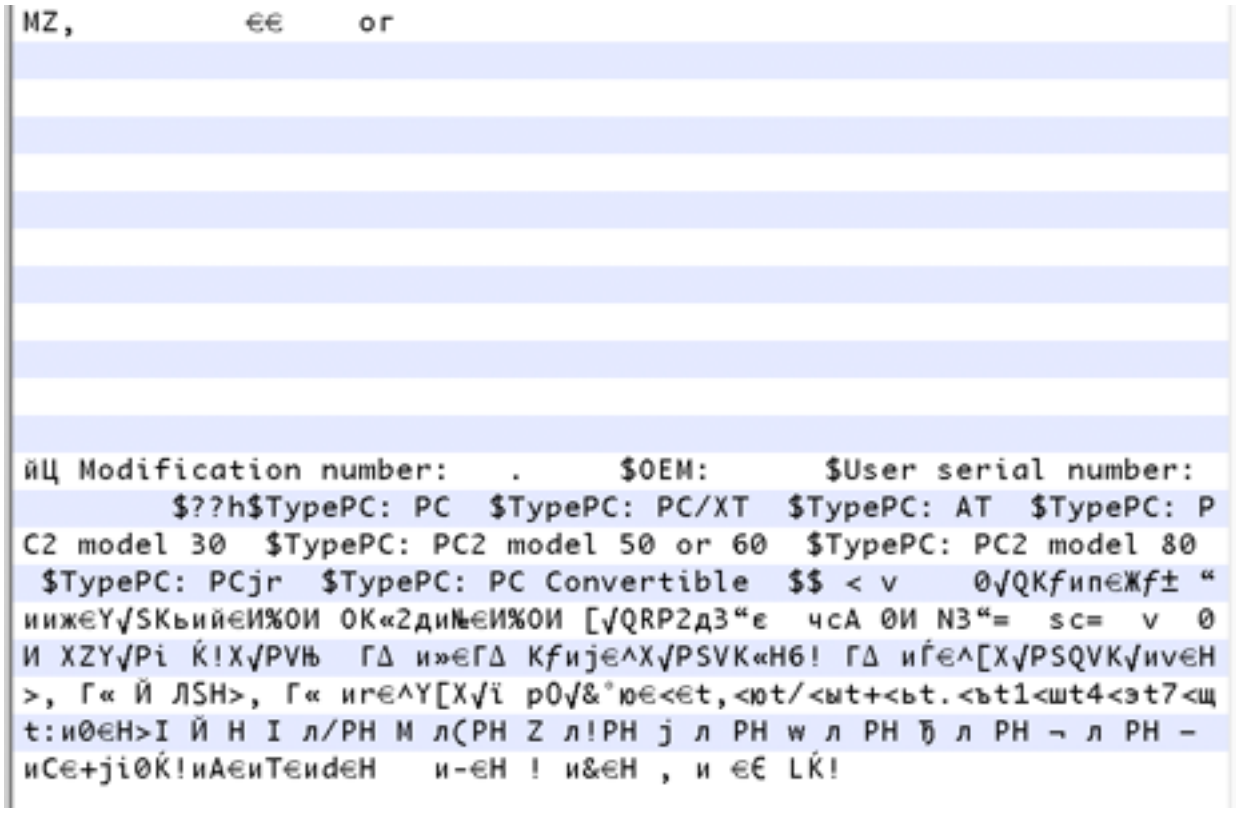


Рисунок 10 – Вид «плохого» EXE файла

В «плохом» EXE-файле код располагается с адреса 300h. С нулевого адреса располагается управляющая информация для загрузчика, образующая заголовок.

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. 11.

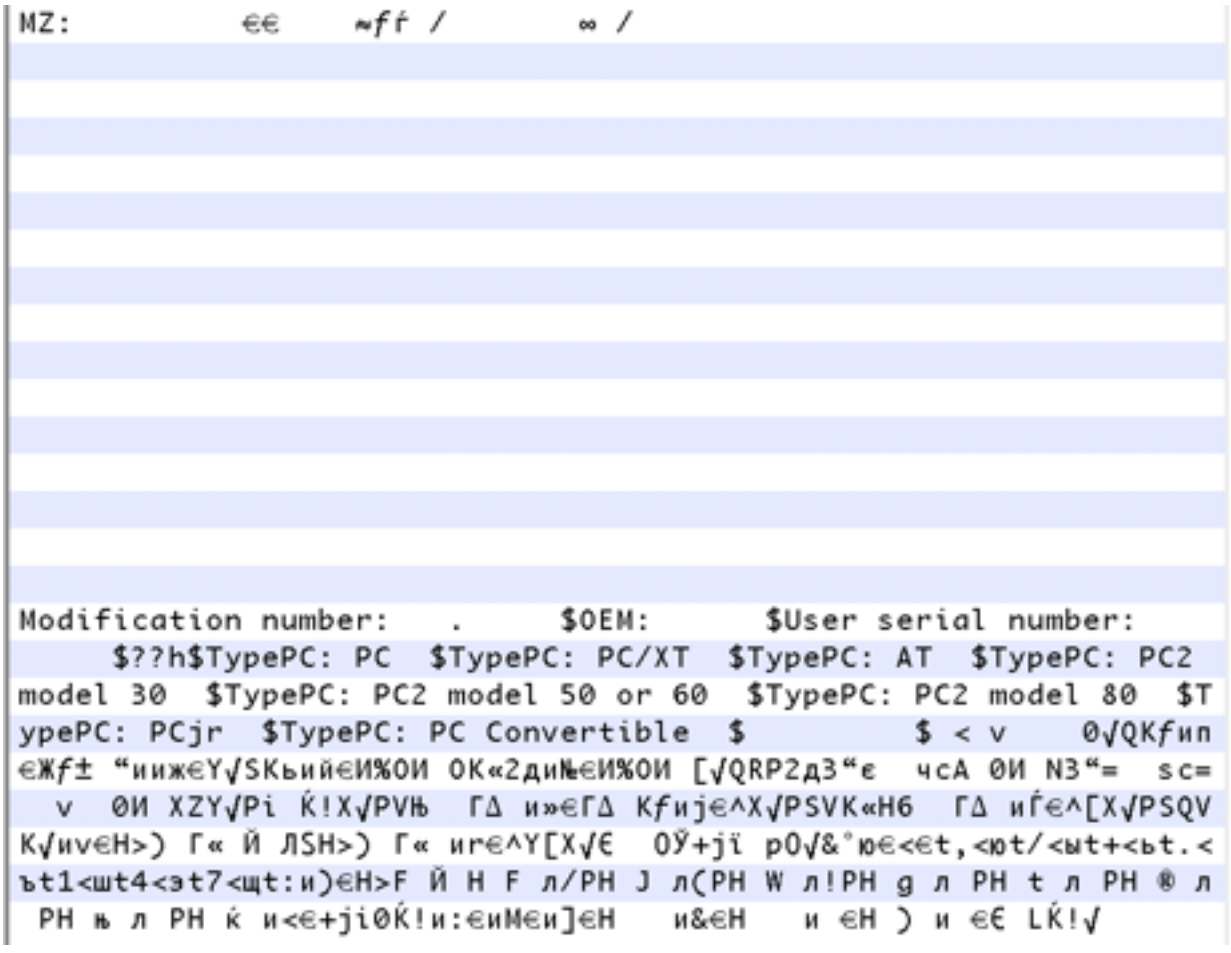


Рисунок 11 – Вид «хорошего» EXE файла

В «хорошем» EXE с нулевого адреса также располагается заголовок (например, 00-01 - 4D5A - сигнатура файла .EXE). 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 AFD.EXE представлен на рис. 12.

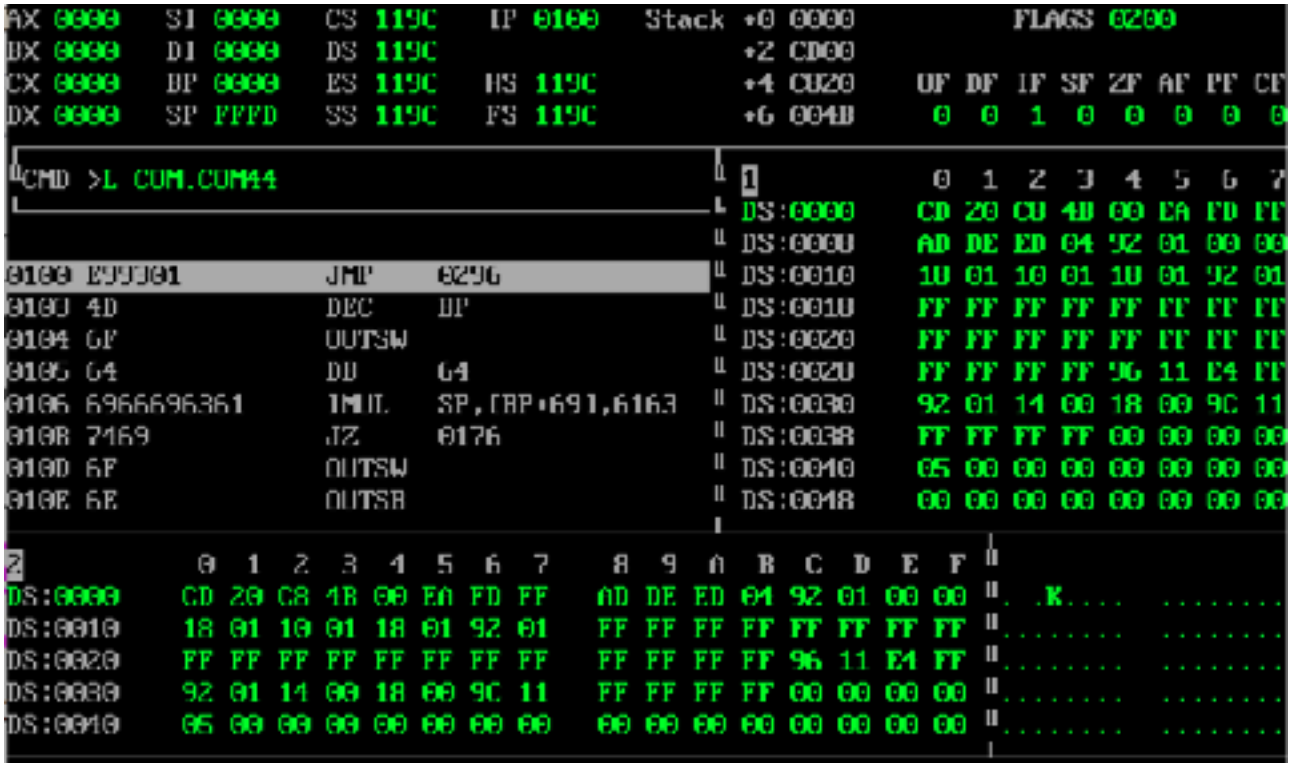


Рисунок 12 – Отладка файла .COM

Определяется сегментный адрес свободного участка ОП, в который можно загрузить программу. Создается блок памяти. В поля PSP заносятся значения. Загружается COM файл со смещением 100h. Сегментные регистры устанавливаются на адрес сегмента PSP, регистр SP указывает на конец сегмента, туда записывается 0000h. С ростом стека значение SP будет уменьшаться. Счетчик команд принимает значение 100h. Программа запускается.

2) Что располагается с адреса 0?

С нулевого адреса (0h) 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 (из 64 Кб), 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 AFD.EXE представлен на рис. 13.

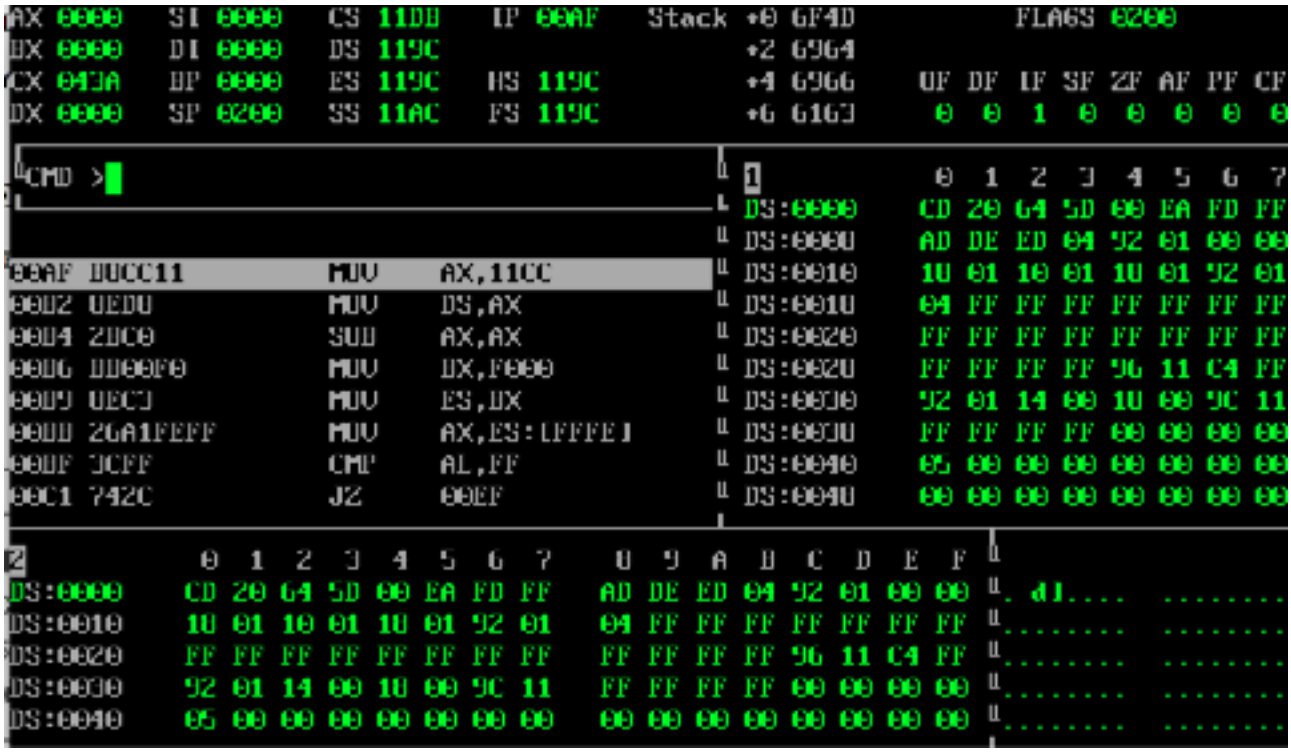


Рисунок 13 – Отладка «хорошего» EXE модуля

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

Определяется сегментный адрес свободного участка ОП, в который можно загрузить программу. Создается блок памяти для PSP и программы. 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.

В PSP заносятся соответствующие значения. В рабочую область загрузчика считывается форматированная часть заголовка файла. Определяется смещение начала загрузочного модуля в EXE файле. Вычисляется сегментный адрес (START_SEG) для загрузки. В память считывается загрузочный модуль. Таблица настройки порциями считывается в рабочую память. Для каждого элемента таблицы настройки к полю сегмента прибавляется сегментный адрес начального сегмента (в результате элемент таблицы указывает на нужное слово в памяти). Управление передается загруженной задаче по адресу из заголовка.

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. 14.

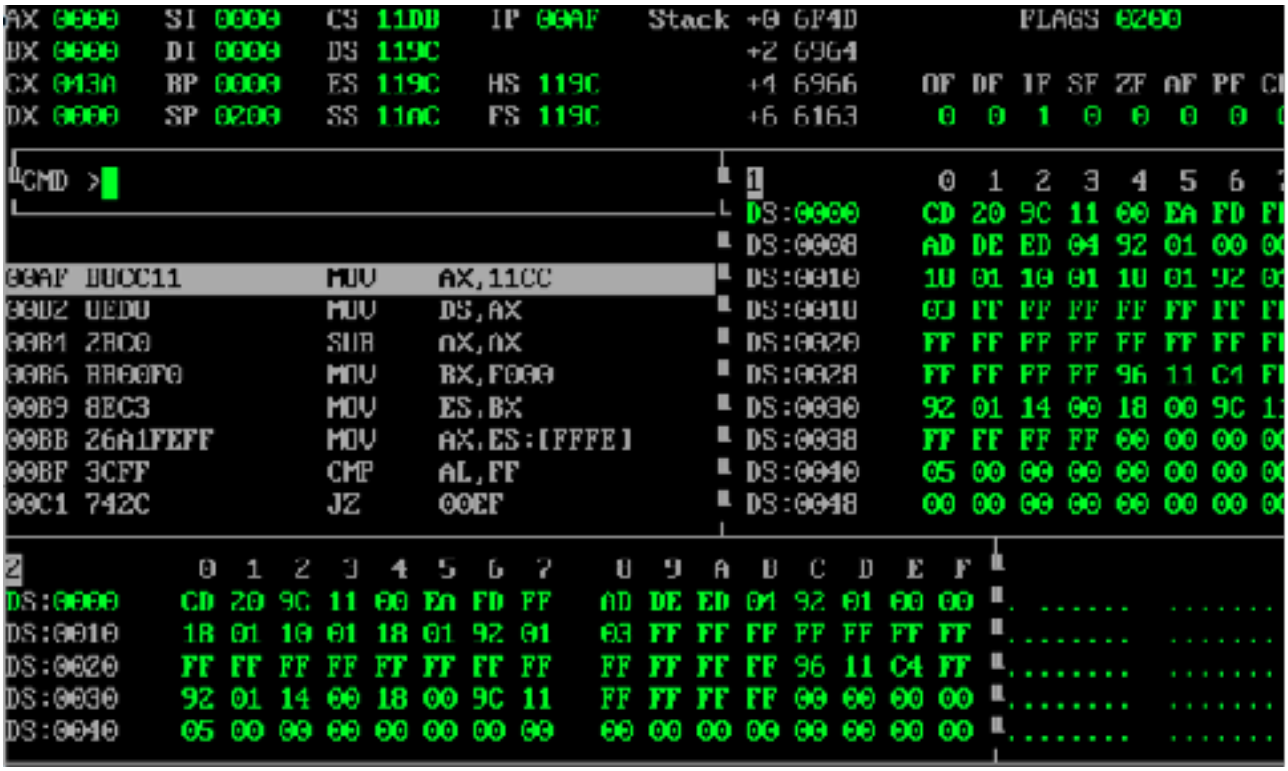


Рисунок 14 – Отладка EXE модуля без объявленного стека

4) Как определяется точка входа?

Смещение точки входа в программу загружается в указатель команд IP и определяется операндом директивы END <метка для входа>, который называется точкой входа.

Операндом является функция или метка, с которой необходимо начать программу.

Выводы.

В ходе выполнения 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.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. EXE.ASM

```
_STACK     SEGMENT    STACK
           db    512   dup(0)
_STACK     ENDS


_DATA SEGMENT
           ModifNum        db         'Modification number:   .   ', 0dh,
0ah, '$'
           OEM                  db         'OEM:    ',0dh,0ah,'$'
           UserSerialNum   db         'User serial number:        ', 0dh,
0ah, '$'

           Type_PC_Other   db         2 dup ('?'), 'h$'
           Type_PC         db         'TypePC: PC', 0DH, 0AH, '$'
           Type_PC_XT           db         'TypePC: PC/XT', 0DH, 0AH, '$'
           Type_AT         db         'TypePC: AT', 0DH, 0AH, '$'
           Type_PS2_30     db         'TypePC:   PC2   model   30',   0DH,
0AH,'$'
           Type_PS2_50     db         'TypePC: PC2 model 50 or 60', 0DH,
0AH,'$'
           Type_PS2_80     db         'TypePC: PC2 model 80', 0DH, 0AH,
'$'
           Type_PCjr       db         'TypePC: PCjr', 0DH, 0AH, '$'
           Type_PC_Conv    db         'TypePC: PC Convertible', 0DH, 0AH,
'$'
_DATA      ENDS
_CODE SEGMENT
           ASSUME CS:_CODE, DS:_DATA, ES:NOTHING, SS:_STACK

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

PRINT PROC  near
            push  ax
            mov   ah,09h
            int         21h
            pop   ax
            ret
PRINT ENDP

MOD_PC      PROC near
            push  ax
            push  si
            mov   si, offset ModifNum
            add   si, 22
            call  BYTE_TO_DEC
            add   si, 3
            mov   al, ah
            call  BYTE_TO_DEC
            pop   si
            pop   ax
            ret

MOD_PC      ENDP


OEM_PC      PROC near
            push  ax
            push  bx
            push  si
            mov   al,bh
            lea         si, OEM
            add         si, 6
            call  BYTE_TO_DEC
            pop         si
            pop         bx
            pop         ax
            ret

OEM_PC      ENDP

SER_PC      PROC near
            push  ax
            push  bx
```

```
            push cx
            push si
            mov  al,bl
            call BYTE_TO_HEX
            lea        di,UserSerialNum
            add        di,20
            mov  [di],ax
            mov  ax,cx
            lea        di,UserSerialNum
            add        di,25
            call WRD_TO_HEX
            pop        si
            pop        cx
            pop        bx
            pop  ax
            ret
SER_PC      ENDP

MAIN  PROC  near
            mov  ax, _DATA
            mov  ds, ax
            sub  ax, ax
            mov  bx, 0F000h
            mov  es, bx
            mov  ax, es:[0FFFEh]
            ;PC
            cmp  al, 0FFh
            je         _PC
            ;PC/XT
            cmp  al, 0FEh
            je         _PC_XT
            cmp  al, 0FBh
            je         _PC_XT
            ;AT
            cmp  al, 0FCh
            je         _AT
            ;PS2 model 30
            cmp  al, 0FAh
            je         _PS2_30
            ;PS2 model 80
            cmp  al, 0F8h
            je         _PS2_80
            ;PCjr
            cmp        al, 0FDh
            je         _PCjr
            ;PC Convertible
            cmp  al, 0F9h
```

```
            je          _PC_Conv
            ;unknown type
            call  BYTE_TO_HEX
            lea   di, Type_PC_Other
            mov   [di], ax
            lea   dx, Type_PC_Other
            jmp   _EndPC

_PC:  lea   dx, Type_PC
            jmp         _EndPC
_PC_XT:     lea   dx, Type_PC_XT
            jmp         _EndPC
_AT:  lea   dx, Type_AT
            jmp         _EndPC
_PS2_30:lea     dx, Type_PS2_30
            jmp         _EndPC
_PS2_80:lea     dx, Type_PS2_80
            jmp         _EndPC
_PCjr:      lea   dx, Type_PCjr
            jmp         _EndPC
_PC_Conv:lea    dx, Type_PC_Conv

_EndPC: call PRINT

            sub   ax, ax
            mov   ah, 30h
            int   21h
            call  MOD_PC
            call    OEM_PC
            call  SER_PC

            ;print results
            lea   dx, ModifNum
            call  PRINT
            lea   dx, OEM
            call  PRINT
         lea           dx, UserSerialNum
            call  PRINT

            mov ax, 4C00h
            int 21h
            ret
MAIN  ENDP
_CODE       ENDS
            END   MAIN
```

Приложение Б

Исходный код программы. COM.ASM

```
TESTPC SEGMENT
```

```
          ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
          ORG 100H
START:    JMP  BEGIN

ModifNum        db        'Modification number:   .   ', 0dh, 0ah, '$'
OEM                  db        'OEM:    ',0dh,0ah,'$'
UserSerialNum   db        'User serial number:       ', 0dh, 0ah, '$'

Type_PC_Other   db        2 dup ('?'), 'h$'
Type_PC         db        'TypePC: PC', 0DH, 0AH, '$'
Type_PC_XT           db        'TypePC: PC/XT', 0DH, 0AH, '$'
Type_AT         db        'TypePC: AT', 0DH, 0AH, '$'
Type_PS2_30     db        'TypePC: PC2 model 30', 0DH, 0AH,'$'
Type_PS2_50     db        'TypePC: PC2 model 50 or 60', 0DH, 0AH,'$'
Type_PS2_80     db        'TypePC: PC2 model 80', 0DH, 0AH, '$'
Type_PCjr       db        'TypePC: PCjr', 0DH, 0AH, '$'
Type_PC_Conv    db        'TypePC: PC Convertible', 0DH, 0AH, '$'


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

PRINT PROC  near
            push  ax
            mov   ah,09h
            int         21h
            pop   ax
            ret
PRINT ENDP

MOD_PC      PROC near
```

```
            push  ax
            push  si
            mov   si, offset ModifNum
            add   si, 22
            call  BYTE_TO_DEC
            add   si, 3
            mov   al, ah
            call  BYTE_TO_DEC
            pop   si
            pop   ax
            ret

MOD_PC      ENDP


OEM_PC      PROC near
            push  ax
            push  bx
            push  si
            mov   al,bh
            lea         si, OEM
            add         si, 6
            call  BYTE_TO_DEC
            pop         si
            pop         bx
            pop         ax
            ret

OEM_PC      ENDP

SER_PC      PROC near
            push  ax
            push  bx
            push  cx
            push  si
            mov   al,bl
            call  BYTE_TO_HEX
            lea         di,UserSerialNum
            add         di,20
            mov   [di],ax
            mov   ax,cx
            lea         di,UserSerialNum
            add         di,25
            call  WRD_TO_HEX
            pop         si
            pop         cx
            pop         bx
```

```
            pop   ax
            ret
SER_PC      ENDP

BEGIN:      mov   bx, 0F000h
            mov   es, bx
            mov   ax, es:[0FFFEh]
            ;PC
            cmp   al, 0FFh
            je          _PC
            ;PC/XT
            cmp   al, 0FEh
            je          _PC_XT
            cmp   al, 0FBh
            je          _PC_XT
            ;AT
            cmp   al, 0FCh
            je          _AT
            ;PS2 model 30
            cmp   al, 0FAh
            je          _PS2_30
            ;PS2 model 80
            cmp   al, 0F8h
            je          _PS2_80
            ;PCjr
            cmp         al, 0FDh
            je          _PCjr
            ;PC Convertible
            cmp   al, 0F9h
            je          _PC_Conv
            ;unknown type
            call  BYTE_TO_HEX
            lea   di, Type_PC_Other
            mov   [di], ax
            lea   dx, Type_PC_Other
            jmp   _EndPC

_PC:  lea   dx, Type_PC
            jmp         _EndPC
_PC_XT:     lea   dx, Type_PC_XT
            jmp         _EndPC
_AT:  lea   dx, Type_AT
            jmp         _EndPC
_PS2_30:lea       dx, Type_PS2_30
            jmp         _EndPC
_PS2_80:lea       dx, Type_PS2_80
            jmp         _EndPC
```

```
_PCjr:     lea   dx, Type_PCjr
           jmp         _EndPC
_PC_Conv:lea     dx, Type_PC_Conv

_EndPC: call PRINT

           sub   ax, ax
           mov   ah, 30h
           int   21h
           call  MOD_PC
           call     OEM_PC
           call  SER_PC

           ;print results
           lea   dx, ModifNum
           call  PRINT
           lea   dx, OEM
           call  PRINT
         lea           dx, UserSerialNum
           call  PRINT

           mov ax, 4C00h
           int 21h

TESTPC     ENDS
           END   START
```